Modellgestützte Analyse und Optimierung

Peter Buchholz
Informatik IV
Technische Universität Dortmund

15. Juni 2012

Vorwort

Bei dem vorliegenden Text handelt es sich um die Version Sommersemester 2012 der Notizen zur Wahlpflichtvorlesung "Modellgestützte Analyse und Optimierung". Im Vergleich zu vorherigen Versionen der Vorlesung wurde das Kapitel über nichtlineare Optimierung gestrichen. Dafür wurde der Teil über Lineare und Dynamische Optimierung etwas erweitert. Der vorliegende Text soll als Ergänzung zu den verfügbaren Power Point-Folien dienen. Es sollte klar sein, dass die Folien und die Notizen allein nicht ausreichen, um den Stoff zu verstehen. Der regelmäßige Besuch der Vorlesung, die Bearbeitung der Übungsaufgaben und das Lesen von Originalliteratur sind unumgänglich, um den Stoff der Vorlesung wirklich zu durchdringen und damit letztendlich auch anwenden zu können.

Trotzdem erschien es mir sinnvoll, diese Notizen zu verfassen, da es momentan kein Lehrbuch gibt, das den gesamten Stoffumfang der Vorlesung abdeckt. Es wären Teile aus drei oder vier Lehrbüchern notwendig, um die unterschiedlichen behandelten Themengebiete zu erfassen. In den Notizen werden deshalb auch konkrete Hinweise zum "Weiterlesen" in der einschlägigen Literatur gegeben. Es soll ferner darauf hingewiesen werden, dass die Vorlesung "Modellgestützte Analyse und Optimierung" eine praktische Veranstaltung ist. In den Übungen wird Gelegenheit gegeben, die theoretischen Erkenntnisse aus der Vorlesung an praktischen Beispielen auszuprobieren und zu vertiefen. Dazu werden beispielhaft mit einem Simulationssystem (z.Z. Arena) einfache Abläufe modelliert und analysiert.

Die Vorlesung baut als weiterführende Vorlesung auf dem Wissen des vorangegangener Veranstaltungen auf. Es müssen Grundkenntnisse in der Programmierung und Softwareentwicklung vorhanden sein, um die Problematik der Erstellung von Simulatoren zu verstehen. Weiterhin sollten Grundkenntnisse in der Wahrscheinlichkeitsrechnung vorhanden sein, da Realität in vielen Fällen nur mit Hilfe von stochastischen Annahmen in Modelle abbildbar ist und für eine adäquate Modellierung und Ergebnisauswertung die Anwendung wahrscheinlichkeitstheoretischer und statistischer Verfahren unumgänglich ist.

Es lässt sich leider nicht ausschließen, dass ein Text wie der vorliegende Text Fehler und Ungenauigkeiten enthält, auch wenn schon mehrere Personen ihn gelesen haben. Ich möchte mich in diesem Zusammenhang bei meinen Mitarbeitern und Studierenden, die frühere Versionen korrigiert haben, herzlich bedanken. Für eventuelle Fehler bin ich natürlich ganz allein verantwortlich und bitte diese zu entschuldigen. Für Hinweise auf Fehler bin ich dankbar (am besten per Email an peter.buchholz@udo.edu). Bei allen, die mich bisher schon auf Fehler hingewiesen und Verbesserungsvorschläge geschickt haben, möchte ich mich herzlich bedanken.

Inhaltsverzeichnis

Einleitung

Viele reale Probleme werden heute mit Hilfe von Modellen analysiert und bewertet. Damit ersetzt die rechnergestützte und modellbasierte Analyse immer mehr das Experimentieren an realen Objekten. Dies gilt in sehr unterschiedlichen Anwendungsgebieten, wie dem Entwurf und Betrieb technischer Systeme, der Analyse ökonomischer Entscheidungen, der Untersuchung physikalischer Phänomene, der Vorhersage des zukünftigen Klimas oder auch der Interaktion in sozialen Gruppen. Auch wenn die einzelnen Anwendungsgebiete stark differieren, basiert ihre rechnergestützte Behandlung doch auf einer formalisierten Darstellung in Form eines mathematischen Modells und der anschließenden Analyse und Optimierung oder Verbesserung des Modells.

Modelle können dabei von sehr unterschiedlicher Art sein, müssen für eine rechnergestützte Behandlung aber formaler Natur sein. Es geht also darum, eine möglichst adäquate formale Darstellung der Realität zu finden, diese zu analysieren und möglichst zu verbessern oder zu optimieren, um dadurch das untersuchte reale Phänomen zu verstehen, zu analysieren oder zu verbessern. Aus dieser sehr kurzen und wohl auch oberflächlichen Darstellung des behandelten Sachverhalts ergibt sich, dass Modellbildung, Analyse und Optimierung in vielen (fast allen) Wissenschaftsbereichen beheimatet ist. Dies gilt unzweifelhaft für die Anwendung von Modellen. Jede Natur- und Ingenieurwissenschaft benutzt heute Modelle zur Planung und Analyse von Systemen. Aber auch in den Sozial- und Wirtschaftswissenschaften spielen Modelle eine immer zentralere Rolle. Etwas eingeschränkter ist die Menge der Disziplinen, die sich mit der Definition von Modelltypen und deren Analyse bzw. Optimierung beschäftigen. In erster Linie ist an dieser Stelle natürlich die Mathematik zu nennen, da praktisch alle formalen Modelle mathematischer Natur sind. Viele Analysemethoden basieren natürlich ebenfalls auf mathematischen Ansätzen. Auf Grund der starken Anwendungsorientierung der Modellierung reicht eine rein mathematische Betrachtung der Probleme nicht aus, es kommen einerseits sehr viele anwendungsspezifische Aspekte hinzu, zum anderen sind Statistik und Informatik als weitere Grundlagenwissenschaften für die modellgestützte Analyse und Optimierung zu nennen. Statistische Methoden sind von zentraler Bedeutung, wenn es darum geht, die Beobachtung der Realität in ein Modell umzusetzen. Dies geschieht in der Regel mit Hilfe von Zufallsvariablen mit vorgegebener Verteilung. Die Informatik spielt eine zentrale Rolle, da Modelle heute rechnergestützt modelliert, analysiert und optimiert werden müssen. Es sind also Methoden notwendig, komplexe Abläufe programmtechnisch zu erfassen. Dies ist ein klassisches Informatikproblem. Es ist darüber hinaus notwendig, die so entstehenden Modelle zu analysieren, was die Umsetzung mathematischer Verfahren in Algorithmen erforderlich macht. Es müssen aber auch ganz neue Lösungsansätze entwickelt werden, wie die stochastische Simulation oder stochastische Optimierungsverfahren, die in der Mathematik sehr lange eher skeptisch bewertet wurden, in der Praxis aber inzwischen eine herausragende Bedeutung haben. Teilweise sind die hier beschriebenen Methoden im Bereich des Operations Research angesiedelt. Operations Research kann als Querschnittsdisziplin sehr unterschiedliche Schwerpunktsetzungen haben und ist in Deutschland in der Regel in den Wirtschaftswissenschaften angesiedelt, während in den angelsächsischen Ländern viele Operations Research Professuren und Zentren im Bereich des Computer Science zu finden sind. Diese unterschiedliche Zuordnung bedingt auch eine unterschiedliche Ausrichtung. So behandelt Operations Research im Bereich Computer Science ähnliche Aspekte, wie wir sie in der Vorlesung kennen lernen werden, während die Ausrichtung in den Wirtschaftswissenschaften oftmals eine andere Schwerpunktsetzung aufweist.

Aus dem bisher gesagten ergibt sich, dass die Thematik der Vorlesung im besten Sinne inter-

disziplinär ist und damit auch sehr unterschiedliche Schwerpunktsetzungen erlaubt. Zwischen den beiden Extremen, einer rein theoretischen Vorlesung über Modellbildung und -analyse und einer rein anwendungsorientierten Darstellung der Methodennutzung, gibt es zahlreiche Facetten. Ich möchte in der Vorlesung eine stark durch die Informatik geprägte Sichtweise vermitteln. Es soll also weniger darum gehen, Modellierung in einem bestimmten Anwendungsgebiet zu vermitteln, es soll auch nicht darum gehen, alle theoretischen Grundlagen und vollständige Beweise der verwendeten Methoden zu vermitteln. Vielmehr soll ein Verständnis der benutzten Methoden und ihre algorithmische Umsetzung gelehrt werden. Erst dadurch ist es möglich, Modelle so zu formulieren, dass sie analysierbar sind und Ergebnisse so zu interpretieren, dass sie signifikante Aussagen über die Realität erlauben. Gerade diese beiden Punkte werden in der Praxis oft vernachlässigt und führen zu vielen Missdeutungen der Realität oder auch der zukünftigen Realität auf Basis von Modellanalysen.

Die vorliegende Vorlesung ist historisch aus den Vorlesungen "Simulation" und "Operations Research" hervorgegangen und umfasst dabei jeweils Teile dieser Vorlesungen plus einiger Ergänzungen. Im wesentlichen gliedert sich die Vorlesung in ein Einführungskapitel und zwei große Teile. Im Einführungskapitel werden die grundlegenden Begriffe definiert und an Hand zahlreicher Beispiele erläutert. Es geht dabei insbesondere um die Begriffsbildung im Bereich der Systeme und Modelle.

Daran anschließend beschäftigt sich der erste Teil der Vorlesung mit der modellgestützten Analyse von Systemen. Es werden dazu ereignisdiskrete und kontinuierliche Modelle unterschieden. Ereignisdiskrete Systeme werden oftmals zur Analyse technischer Systeme eingesetzt, während kontinuierliche Modelle besser zur Beschreibung physikalischer Zusammenhänge geeignet sind. Die Vorlesung legt den Schwerpunkt auf die Modellbildung und Simulation ereignisdiskreter stochastischer Systeme. In diesem Bereich werden neben verschiedenen Modelltypen insbesondere Ansätze zur stochastischen Modellierung und die zugehörige Simulations-/Analysemethodik eingeführt. Darüber hinaus wird ein allgemeines Vorgehen zur Modellbildung und Analyse von Systemen vermittelt.

Der dritte Teil der Vorlesung ist der Optimierung von Systemen gewidmet. Das sehr umfangreiche Gebiet der mathematischen Optimierung kann natürlich nicht vollständig in der zur Verfügung stehenden Zeit behandelt werden. Es wird deshalb eine Auswahl unterschiedlicher Optimierungsprobleme definiert, an Hand von Beispielen motiviert und zugehörige Optimierungsverfahren vorgestellt. Bei der Untersuchung von Optimierungsverfahren beschränken wir uns auf die klassische lineare Optimierung und diskrete Optimierungsprobleme, bei denen aus einer endlichen oder abzählbaren Menge von Möglichkeiten eine optimale Konfiguration ausgewählt wird. Diskrete Optimierung ist ein klassisches Gebiet der Informatik, das in zahlreichen Veranstaltungen weiter vertieft werden kann. Nicht behandeln werden wir das umfangreiche Gebiet der kontinuierlichen Optimierung im nichtlinearen Fall.

Die Vorlesung "Modellgestützte Analyse und Optimierung" kann als Grundlage für weitere Vertiefungsveranstaltungen verwendet werden. So bietet der Lehrstuhl regelmäßig eine weitergehende Vertiefungsveranstaltung im Bereich Modellierung und Simulation an. Weitere Vertiefungsveranstaltungen des Lehrstuhls ergänzen und vertiefen einzelne der vermittelten Aspekte. In der Informatik gibt es in verschiedenen anderen Bereichen Vertiefungsvorlesungen, die als Ergänzung zu sehen sind. So ist existieren mehrere Spezialveranstaltungen im Bereich Optimierung und auch modellgestützte Analyse. Neben Vorlesungen bieten wir Seminare, Fachpraktika, Projektgruppen und natürlich auch Bachelor-, Master- und Diplomarbeiten zum Thema an. Die vermittelte Methoden zur Systemanalyse und -optimierung können in vielen Gebieten der Informatik im Studium und auch im späteren Berufsleben Verwendung finden.

Kapitel 1

Systeme und Modelle

Bevor wir uns mit der Modellierung, Analyse und Optimierung beschäftigen, sollen die grundlegenden Begriffe definiert werden und an Hand von Beispielen die Konzepte erläutert werden. Die Inhalte dieses Kapitel beruhen auf den Einführungskapitel der Lehrbücher Law and Kelton [2000, Kap. 1.2], und Cellier [1991, Kap. 1] sowie der Vorlesung Beilner [2002]. Ergänzt werden diese Quellen durch einige Originalartikel, die im Text genannt werden.

1.1 Systeme

Der Begriff des Systems ist sehr schillernd und nur schwer fassbar. In Buch von Cellier [1991] wird die folgende Definition gegeben: "*Ein System ist das, was als System erkannt wird*". Dies hört sich erst einmal nicht wie eine übliche Definition an, trifft aber durchaus das, was sehr allgemein unter einem System verstanden wird. Also kann ein System im Prinzip alles sein. Andererseits bedeutet das Erkennen eines Systems aber auch, dass das System in irgend einer Weise von seiner Umwelt abgegrenzt bzw. abgrenzbar ist. Damit gibt es für jedes System ein "Innen" und ein "Außen". Im Inneren können weitere Strukturen erkannt werden, um das System zu verstehen und zu beschreiben. Da wir als Menschen zweckorientiert und zweckbestimmt handeln, existiert ein System unter einem bestimmten Systemzweck. Insgesamt bleibt der Begriff des Systems damit immer noch vage, die folgenden Aspekte können aber als zentral angesehen werden:

- Struktur im Inneren
- Abgrenzung nach Außen
- Systemzweck als Existenzgrundlage

Wenn wir etwas weitergehen und die interne Struktur des Systems und seine Beziehung zur Umwelt mit einbeziehen, so kann man ein System als eine Anzahl in Beziehung stehender Teile ansehen, die zu einem gemeinsamen Zweck interagieren. Diese Darstellung wird in Abbildung 1.1 deutlich. Das System besteht im Inneren aus einer Menge interagierender Komponenten. Interaktionen werden durch Pfeile angedeutet. Falls ein Zyklus aus Interaktionsbeziehungen existiert, so spricht man auch von einer Rückkopplung. Darüber hinaus gibt es eine Systemgrenze, die das System von der Umwelt abgrenzt. Zwischen System und Umwelt finden eine Interaktion in Form von Ein- und Ausgaben statt, die jeweils Einfluss auf Komponenten im Inneren des Systems haben. Ein- und Ausgaben können dabei natürlich mehr als eine Systemkomponente beeinflussen.

Aus der bisherigen Definition folgt natürlich, dass fast alles aus der realen Umwelt als System aufgefasst werden kann. Dies ist sogar gewünscht, um einen möglichst allgemeinen Begriff zu bekommen. Wenn wir Beispiele betrachten, so wäre in der Physik unser gesamtes Universum ein System, wobei wir bei diesem Beispiel je nach Lebensauffassung Probleme mit der Definition der Umgebung bekommen. Es ist aber genauso ein Planet oder ein Atom ein System. In der Biologie ist ein Mensch genauso ein System, wie etwa ein Insektenvolk oder eine einzelne Zelle. Wenn wir

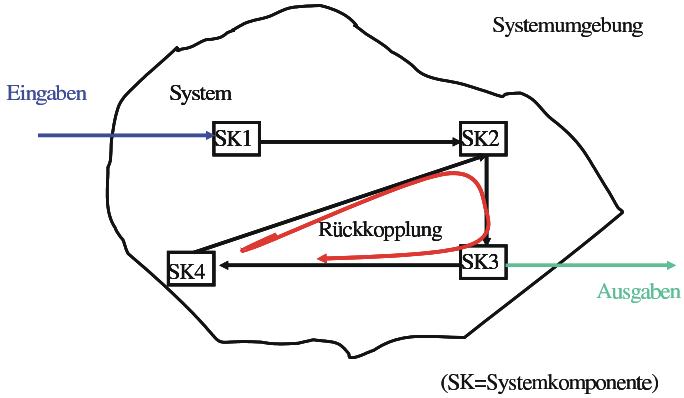


Abbildung 1.1: Struktur eines Systems

den Bereich der Sozialwissenschaften betrachten, so ist dort ein Mensch ebenfalls ein System, wie auch in der Biologie. Der Systemzweck und damit auch die Wahrnehmung des Systems Mensch ist aber in beiden Wissenschaften unterschiedlich. Während in der Biologie die biologischen und biochemischen Prozesse im Mittelpunkt stehen, befassen sich die Sozialwissenschaften mit dem Verhalten. Der Systemzweck beeinflusst also die Wahrnehmung und Darstellung des Systems.

Die systematische Untersuchung und Planung von Systemen ist Gegenstand der *Systemanalyse* und wird vom *Systemanalytiker* durchgeführt. Systemanalyse ist ein interdisziplinäres Wissensgebiet, zu dem auch die Informatik wesentliche Beiträge liefert.

Wir wollen den Begriff des Systems nun etwas stärker eingrenzen, da wir an einer Unterklasse von Systemen und deren Analyse interessiert sind und keine allgemeine Systemtheorie betreiben wollen. Dazu betrachten wir unser System als eine Menge von interagierenden *Komponenten*, die jeweils wieder aus Komponenten bestehen können, so dass eine hierarchische Beziehung entsteht. Die Komponenten können *Eigenschaften* besitzen, sofern diese Eigenschaften veränderlich sind, werden sie als *Zustandsvariablen* bezeichnet. Jede Zustandsvariable hat, wie in der Informatik für Variablen üblich, einen *Wertebereich*, der sich aus einer Codierung der jeweiligen Eigenschaften ergibt. Der *Zustand des Systems* ist damit definiert als der Wert aller Zustandsvariablen zu einem Zeitpunkt. Da sich der Wert der Zustandsvariablen mit der Zeit ändert, ergibt sich ein zeitliches Verhalten, welches als *Dynamik des Systems* bezeichnet wird. Grundsätzlich gibt es auch Systeme, die keine Dynamik aufweisen bzw. über den Beobachtungszeitraum keine Dynamik aufweisen. Diese Systeme wollen wir nicht weiter betrachten, da das Ziel gerade die Bewertung von Systemen ist, die verschiedene Zustände annehmen können, analysierbar und oftmals auch beeinflussbar und damit optimierbar sind.

Eine zentrale Rolle spielt natürlich die *Komplexität* eines Systems, da durch sie die Möglichkeit der Wahrnehmung und des Verständnisses für Systeme entscheidend beeinflusst wird. In unserer formaleren Darstellung ergibt sich die Komplexität aus der Zahl und Interaktion der Komponenten, die das System bilden. Da unsere Wahrnehmung aber subjektiv und auch selektiv ist, gibt es kein absolutes Maß für die Komplexität eines Systems, sie hängt von der *Abstraktion* des Benutzers bei der Wahrnehmung des Systems ab. Abstraktion ist Teil jeder menschlichen Wahrnehmung (z.B. können wir nur ein bestimmtes Lichtspektrum sehen oder nur ein bestimmtes Frequenzspektrum hören), kann aber auch bewusst herbeigeführt werden, um nicht mit Details überfrachtet zu werden. Insbesondere ist die Wahrnehmung immer systemzweckbezogen geprägt, wie am Beispiel des Systems Mensch in der Biologie und in den Sozialwissenschaften angedeutet.

Als nächstes sollen Systeme klassifiziert werden. Man kann Systeme in *natürliche* und *künstliche* Systeme klassifizieren. Natürliche Systeme entstehen und existieren ohne menschlichen Einfluss, wie zum Beispiel ein Biotop. Künstliche Systeme (engl. artifical systems) werden durch den Menschen erstellt und betrieben. Durch den starken Einfluss des Menschen auf die Umwelt ergibt sich eine Vielzahl von Systemen, die weder rein natürlich noch künstlich sind. Eine weitere Möglichkeit ist die Unterscheidung in *offene* und *geschlossene* Systeme. Ein offenes System interagiert

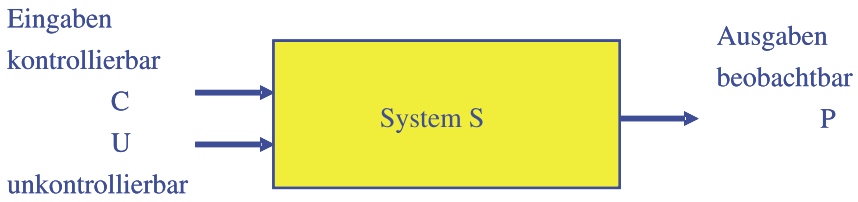


Abbildung 1.2: Abstrakte Systemdarstellung.

mit seiner Umwelt durch Ein- und Ausgaben. Als Beispiel könnte eine Volkswirtschaft dienen, die durch Import- und Export-Beziehungen mit anderen Volkswirtschaften interagiert. Geschlossene Systeme weisen keine (oder kaum) Interaktionen mit der Umwelt auf. Ein typisches Beispiel für ein geschlossenes System ist das Biosphere II Projekt in den USA, das ursprünglich dazu dienen sollte zu untersuchen, wie sich eine Biosphere in Isolation entwickelt. Offene Systeme können geschlossen werden, indem man Teile der Umgebung zum System dazu nimmt. So kann die Volkswirtschaft durch Vereinigung mit anderen Volkswirtschaften zur Weltwirtschaft ergänzt werden. Letzteres Beispiel zeigt auch, dass auch geschlossene Systeme nicht vollständig geschlossen sein müssen. So kann die Weltwirtschaft natürlich durch externe Ereignisse, wie den Einschlag eines großen Asteroiden, beeinflusst werden. Diese Ereignisse werden aber als so selten bzw. unwahrscheinlich eingestuft, dass man sie nicht beachtet und trotzdem von einem geschlossenem System spricht. Daneben gibt es noch die Unterteilung in statische und dynamische Systeme. Dynamische Systeme ändern ihren Zustand mit der Zeit, während statische Systeme keine Zustandsänderung über die Zeit erfahren.

In der Vorlesung werden wir uns primär mit künstlichen dynamischen Systemen beschäftigen, da diese einen großen Teil der Simulations- und Optimierungsanwendungen ausmachen. Uns interessiert ein System bzgl. seiner Funktion (d.h. dem Systemzweck). Bei künstlichen Systemen kann man in der Regel davon ausgehen, dass sie zweckbestimmt entworfen wurden. Ziel der Systemanalyse ist die Bereitstellung von Information über das Systemverhalten als Basis der Entscheidungsfindung. Bei künstlichen Systemen ergeben sich insbesondere Fragen nach der Korrektheit, der Zuverlässigkeit, der Leistungsfähigkeit und den Kosten eines Systems. Neben der Frage wie ein System sich verhält, stellt sich die Frage, was zu ändern ist, dass ein System sich in einer vorbestimmten Art und Weise verhält.

Wir formalisieren das Vorgehen weiter und benutzen dazu die Systemdarstellung in Abbildung 1.2. In dieser Darstellung betrachten wir ein System bzgl. seiner Ein- und Ausgaben, also bzgl. seiner Interaktion mit der Umgebung. Wenn Interna des Systems beobachtet werden sollen, so können diese als Ausgaben kodiert werden. Sei P die Menge der Ausgaben mit Wertebereich W_P. Bei den Eingaben unterscheiden wird zwischen kontrollierbaren Eingaben, also solchen die vom Betreiber/Betrachter beeinflusst werden können und unkontrollierbaren Eingaben, also solchen die nicht vom Betreiber/Betrachter beeinflusst werden können. Sei C die Menge der kontrollierbaren Eingaben mit Wertebereich W_C und U die Menge der unkontrollierbaren Eingaben mit Wertebereich W_U. In der einfachsten Form reagiert ein System auf bestimmte Eingaben mit beobachtbaren Ausgaben. Diese Reaktion wird beschrieben durch eine Funktion $f : W_C \times W_U \to W_P$ oder eine Relation $f \subset W_C \times W_U \times W_P$.

Meistens betrachten wir Systeme, die über der Zeit agieren. Sei t der aktuelle Zeitpunkt, dann sind $C(t)$, $U(t)$ und $P(t)$ die Ein- und Ausgaben zum Zeitpunkt t und $C(t_0, t)$, $U(t_0, t)$ $(t_0 \leq t)$ ist die Sequenz der Eingaben von einem Startzeitpunkt t_0 bis zum Zeitpunkt t. Man kann annehmen, dass vor dem Zeitpunkt t_0 das System nicht existierte oder die Eingaben vor t_0 keinen Einfluss auf die Ausgaben zum Zeitpunkt t haben. Damit ist f eine Funktion oder Relation, die die Eingaben im Intervall $[t_0, t]$ auf die Ausgaben $P(t)$ abbildet. Ziel der Systemanalyse ist es, f zu verstehen und damit den Einfluss von $C(t_0, t)$ und $U(t_0, t)$ auf $P(t)$ zu verstehen und zu nutzen.

Wenn man $C = C(t_0, t)$ und $U = U(t_0, t)$ definiert, kann man mit der zeitunabhängigen Darstellung aus Abbildung 1.2 auch den zeitabhängigen Fall abdecken, erhält aber deutlich komplexere Eingaben.

Um f zu verstehen muss der Einfluss der Eingaben auf die Ausgaben untersucht werden. Da für ein reales System S f durch S repräsentiert wird, ist es nahe liegend S zu beobachten, um Informationen über f zu erhalten. Dazu muss der Einfluss der Eingaben auf die Ausgaben untersucht werden. Die Untersuchung des Systems für verschiedene Eingaben nennt man *experimentieren*. Eine einzelne Beobachtung heißt *Experiment*. Es muss dazu beachtet werden, dass die kontrollierbaren Größen einstellbar und beobachtbar sind, unkontrollierbare Größe aber nicht beeinflusst und in manchen Fällen auch nicht beobachtet werden können. Die Ausgaben P sind in der Regel beobachtbar. Also kann ein Experiment durchgeführt werden indem C eingestellt, P und U beobachtet werden. Das Vorgehen sollte möglichst systematisch und geplant erfolgen, um f zu verstehen.

Bei diesem Vorgehen können eine Reihe praktischer Probleme auftreten, die wir hier kurz erwähnen und in den folgenden Abschnitten genauer analysieren wollen. So ist für praktisch alle realen Systeme eine Beobachtung nur mit eingeschränkter Genauigkeit möglich. Je nach Anwendung werden Beobachtungen von mehr oder weniger großen Fehlern überlagert, so dass Werte nur "ungefähr" bekannt sind. Gleichzeitig ist bei fast allen Systemen ein Teil der unkontrollierbaren Eingaben nicht beobachtbar und führt dazu, dass bei identischer Beobachtungslage unterschiedliche Ausgaben P beobachtet werden.

Das Experimentieren mit Systemen ist in vielen Disziplinen weit verbreitet und wird zum Beispiel in den Naturwissenschaften seit langem genutzt, um Systeme zu verstehen und Theorien zu validieren oder falsifizieren. Die Problematik der variierenden Beobachtungen bei gleichen Eingaben ist dort seit langem bekannt und wird dadurch gelöst, dass mehrmals beobachtet wird und die Beobachtungen statistisch ausgewertet werden.

Neben der Messungenauigkeit gibt es weitere Aspekte, die das Experimentieren mit Systemen erschweren. So sind manche Phänomene aus Zeitgründen nicht beobachtbar, da sie entweder zu schnell sind, wie der Atomzerfall, oder zu langsam sind, wie die Geburt einer Galaxie. Andere Phänomene können nur für einige Eingabekombinationen beobachtet werden, da gewisse Eingabekombinationen fatale Folgen für das System haben können, etwa die Zerstörung des Systems. Als Beispiel sei hier die Kernschmelze in einem Atomreaktor genannt. Darüber hinaus müssen Systeme existieren, damit mit ihnen experimentiert werden kann. Dies ist aber gerade bei künstlichen Systemen in der Planungsphase nicht der Fall. Die Probleme beim Experimentieren mit Systemen zeigen, dass ein großer Bedarf an einer Alternative besteht, die breiter und einfacher einsetzbar ist. Die Herleitung solcher Alternativen ist Inhalt dieser Vorlesung.

1.2 Modelle

Wir haben bereits im letzten Abschnitt gesehen, dass Systeme in vielen Fällen und Situationen nicht experimentell untersucht werden können, da eine solche Untersuchung zu teuer (z.B. Untersuchung unterschiedlicher Abläufe in einem Fertigungsprozess), zu aufwändig (z.B. Messung von Gesteinsschichten zur Exploration von Goldvorkommen), zu langsam (z.B. Beobachtung von Planetenbewegungen), zu gefährlich (z.B. Untersuchung der Sicherungssysteme in einem Atomkraftwerk), zu ungenau (z.B. Atomzerfall) oder prinzipiell nicht möglich sind, da S nicht real existiert. Als Alternative bleibt damit nur die Untersuchung eines Ersatzsystems S', welches die jeweiligen Nachteile nicht hat. S' muss damit einfacher zu beobachten sein als S und für ein Experiment E ähnliches Verhalten wie S bzgl. des Untersuchungsziels zeigen. Wir werden die Forderung ähnliches Verhalten bzgl. eines Untersuchungsziels nicht genau definieren, da dazu einige weitere Grundlagen notwendig sind. Abschnitt 8 widmet sich dann dem Problem einer genügend genauen Repräsentation von S durch S'. Hier soll nur allgemein bemerkt werden, dass das Verhalten S'die Funktion oder Relation f, also letztendlich S bzgl. einer Teilmenge $P' \subseteq P$ nachbilden soll. P' ist die Menge der relevanten Beobachtungsgrößen. Ein Ersatzsystem S' für ein System S bzgl. eines Analyseziels bezeichnet man als ein Modell M. Die folgende Definition nach Niemeyer fasst die Aussagen zusammen.

Definition 1 *Modelle sind materielle oder immaterielle Systeme, die andere Systeme so darstellen, dass eine experimentelle Manipulation der abgebildeten Strukturen und Zustände möglich ist.*

Eine andere einschränkendere Definition in Anlehnung an Cellier [1991] lautet wie folgt.

Definition 2 *Ein Modell M für ein System S und ein Experiment E ist ein System S', auf das E angewendet werden kann und Aussagen über die Anwendung von E auf S macht.*

Die erste Definition ist deutlich allgemeiner, da Modelle nicht direkt an ein Experiment gebunden sind. Der zweite Ansatz konstruiert für ein System und ein Experiment ein Modell. Die meisten realen Modelle liegen zwischen den beiden Definitionen. Sie erlauben die Analyse für eine Menge von Experimenten und eine eingegrenzte Menge von Beobachtungswerten. Es können und werden für ein System S je nach Fragestellung verschiedene Modelle existieren. So wird sich das Modell eines Pkws zur Ermittlung der Schadstoffemission von einem Modell zur Ermittlung des Verhaltens bei Unfällen deutlich unterscheiden. Alle Modelle sollten aber die folgenden beiden Anforderungen erfüllen:

1. Um Ergebnisse vom Modell auf das Originalsystem zu übertragen, ist eine ausreichend genaue Abbildung bzgl. der relevanten Merkmale notwendig.

2. Um Modelle handhabbar zu halten, müssen Details weggelassen werden (durch Abstraktion oder Idealisierung).

Offensichtlich existiert ein Zielkonflikt zwischen den beiden Punkten. Wenn ein System möglichst detailgetreu abgebildet wird, so enthält es zahlreiche Details und ist damit weniger handhabbar als ein abstrakteres Modell. Darüber hinaus ergibt sich das Problem, dass die relevanten Merkmale eines Systems in den meisten Fällen nicht bekannt sind und damit auch nicht klar ist, welche Auswirkungen ein Idealisierung oder Abstraktion des Systems auf das Verhalten des Modells hat.

Der Begriff des Modells ist nach den bisherigen Ausführungen sehr weit gefasst und legt in keiner Weise fest, wie Modelle auszusehen haben. Als Beispiele seien die folgenden Modelltypen kurz genannt:

- Mentale Modelle, die als Gedankenmodelle beim Betrachter entstehen. Jedes vom Menschen entworfene Modell basiert auf einem mentalen Modell. Gleichzeitig sind mentale Modelle sehr schwer fassbar, da sie in der Vorstellung jedes Einzelnen existieren.

- Verbale Modelle, die eine umgangssprachliche Beschreibung eines Systems repräsentieren und auf Grund der fehlenden Präzision menschlicher Sprache in der Regel Raum für Interpretationen lassen.

- Grafische Modelle, die eine Abbildung der sichtbaren Eigenschaften eines Systeme sind. Typische Beispiele sind Fotos und Filme, aber auch Flussdiagramme als semi-formale Beschreibungsmittel.

- Materielle Modelle, die eine Nachbildung der Form des jeweiligen Systems sind. Beispiele sind Modellautos oder Globen.

- Formale Modelle, die aus der mathematischen-logischen Verknüpfung von Symbolen nach einer vorgegebenen Syntax entstehen. Das Verhalten des Modells wird durch die zugehörige Semantik beschrieben.

Wir betrachten in der Vorlesung formale/symbolische Modelle, die sich in einem formalen System nach festgelegten Regeln beschreiben lassen und sich mittels einer Programmiersprache in ein auf einem Computer analysierbares Modell transformieren lassen. In der Regel erfordert die Erstellung eines formalen Modells eines realen Systems ein mehrstufiges Vorgehen, das in Abbildung 1.3 dargestellt wird. Da Systeme nicht zweckfrei definiert sind und auch Modelle mit einem bestimmten Ziel erstellt werden, erfolgt die Modellierung nicht zweckfrei. Vielmehr existiert ein reales Problem. Dieses Problem wird wahrgenommen, entsteht also aus einem mentalen Modell eines Ausschnitts der realen Welt. Meist erfolgt dann eine formalere Beschreibung des Sachverhalts in Form eines deskriptiven Modells. Die genaue Struktur des deskriptiven Modells ist nicht

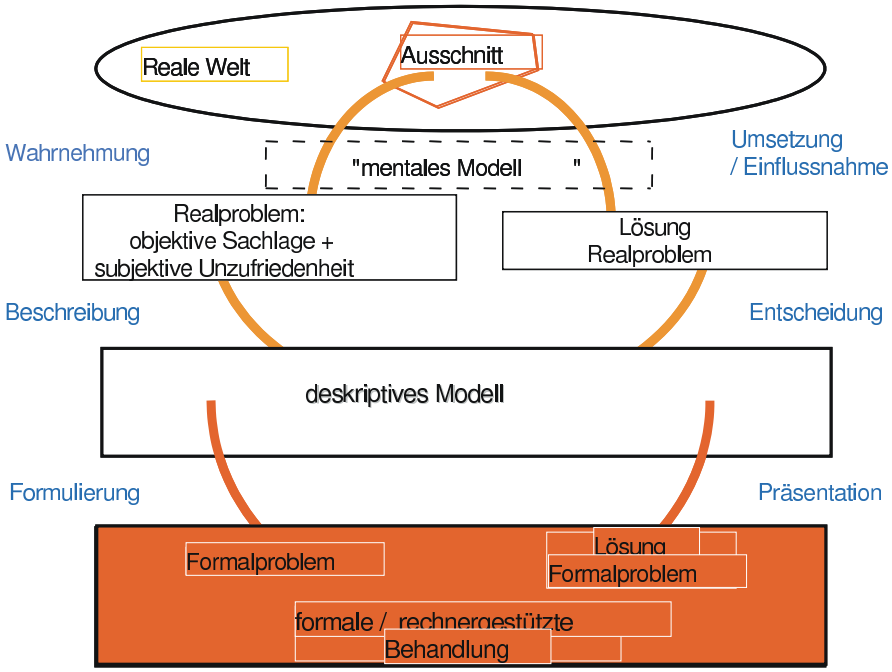


Abbildung 1.3: Vorgehen bei der Modellierung

festgelegt, sie kann von einer umgangssprachlichen Beschreibung bis hin zu einer semiformalen Spezifikation reichen. Durch weitere Formalisierungen entsteht ein Formalproblem, welches sich einfach in eine Darstellung transformieren lässt, die rechnergestützt analysiert werden kann. Mit Hilfe der rechnergestützten Analyse lässt sich das Formalproblem lösen. Damit diese Lösung für das reale Problem relevant wird, muss die Lösung transformiert werden, so dass man schließlich eine Umsetzung der im formalen Modell ermittelten Lösung in praktische Handlungsanweisungen erreicht.

Das hier sehr abstrakte beschriebene Vorgehen der Modellierung und Analyse enthält natürlich, wenn es konkret umgesetzt wird, eine Reihe anwendungsspezifischer Komponenten. So unterscheidet sich das Vorgehen bei der Analyse einer Fertigungsstraße zur Erhöhung des Durchsatzes, von der Analyse des Weltklimas zur Minderung des Treibhauseffektes. Beide Vorgehensweisen lassen sich unter das abstrakte Schema fassen. Unterschiede treten bei der Wahl der konkreten Modelle und deren Aussagegenauigkeit auf und natürlich treten auch Unterschiede bei der Umsetzbarkeit von konkreten Handlungsanweisungen auf.

Formale Modelle sollen nun klassifiziert werden. Dazu betrachten wir zuerst die folgenden Kriterien zur Klassifizierung:

- *Statisch - dynamisch*: Bei statischen Modellen bleibt der Zustand konstant, während dynamische Modelle sich dadurch auszeichnen, dass sich der Zustand mit der Zeit ändert.

- *Deterministisch - stochastisch*: In einem deterministischen Modell erfolgt auf eine Eingabe (d.h. eine feste Belegung der Eingaben C und U) immer eine eindeutige Reaktion am Ausgang, d.h. $f(C, U)$ ist eine Funktion. In stochastischen Modellen können auch bei identischen Eingaben unterschiedliche Werte am Ausgang angenommen werden.[1]

- *Kontinuierlich - diskret*: In kontinuierlichen Modellen ändern sich die Zustandsvariablen kontinuierlich mit der Zeit, während in diskreten Modellen Zeitpunkte existieren, zu denen eine Zustandsänderung auftritt. Zwischen diesen Zeitpunkten bleibt der Zustand unverändert.

[1] Wir werden später stochastische Einflüsse durch bestimmte Parameter in der Menge U kodieren, um so zu erreichen, dass Modelle ein eindeutiges und reproduzierbares Verhalten zeigen.

Die Kombination von kontinuierlichen und diskreten Modellen bezeichnet man als *hybride* Modelle.

Uns interessieren in Teil I primär dynamische Modelle, deren Dynamik auf einem Rechner nachgebildet werden kann. Die Kapitel 2-7 betrachten diskrete und stochastische Modelle. Dieser Modelltyp ist der in der Informatik primär untersuchte Modelltyp, wenn es um Simulation geht und eignet sich besonders für die Abbildung von künstlichen Systemen wie Fertigungsprozessen, Kommunikationsnetzen oder Computersystemen, kann aber auch für populationsbasierte Systeme in der Natur eingesetzt werden. Kapitel 8 ist ein allgemeines Kapitel, das sich dem Nachweis der Modellgültigkeit widmet. Auch hier stehen diskrete stochastische Modelle im Mittelpunkt, die beschriebenen Methoden sind aber breiter anwendbar. Kontinuierliche Modelle, die in vielen Anwendungsszenarien in der Simulation analysiert werden, sind nicht Teil dieser Vorlesung. Sie werden in einer weiterführenden Vorlesung über Modellierung und Simulation behandelt und sind auch Inhalt von Vorlesungen in der (numerischen) Mathematik oder der Elektrotechnik. Teil II beschäftigt sich mit der Optimierung von statischen Modellen, deren Analyse auf einem Rechner relativ schnell auszuführen ist, während die Analyse der Modelle aus Teil I an sich schon relativ aufwändig ist. Die Optimierung der dynamischen Modelle aus Teil I wird in Teil II nur kurz angesprochen und ist Inhalt einer weiterführenden Vorlesung im Wintersemester.

Wir haben bereits gesehen, dass Modelle zielgerichtet erstellt werden. Trotzdem bleibt die zentrale Frage der Modellbildung: *Was ist in einem Modell zu berücksichtigen?* Im Prinzip kann mit heutigen Modellierungsansätzen fast jedes Detail der Realität im Modell abgebildet werden. Dabei ist allerdings zu beachten, dass zusätzliche Details Modelle unübersichtlicher machen. Weiterhin erfordert die Berücksichtigung eines Details dessen Codierung im Modell in Form von Parametern und/oder Zustandsvariablen. Für eine solche Darstellung sind Eingabedaten notwendig, die entweder im realen System gemessen werden müssen und auf andere Weise zu ermitteln sind (siehe auch Kapitel 4). Damit erhöht jedes zusätzliche Detail den Erstellungsaufwand des Modells. Darüber hinaus erhöht sich mit der Hinzunahme von weiteren Details der Analyseaufwand und manchmal auch die Genauigkeit und Allgemeingültigkeit der ermittelten Resultate. Damit ist die zentrale Aufgabe der Modellierung die Unterscheidung der wesentlichen von unwesentlichen Faktoren bzgl. des Analysezieles und das Erkennen von möglichen Vereinfachungen. Man muss zu möglichst verständlichen, leicht analysierbaren und bzgl. der Zielsetzung genügend wirklichkeitsgetreuen Modellen gelangen. Die Modellerstellung wird von folgenden Faktoren beeinflusst:

- Der Zielsetzung der Modellierung,
- der Kenntnis über das System,
- den Möglichkeiten der Parametermessung oder -schätzung,
- den verfügbaren Modellierungformalismen und
- dem vertretbaren Aufwand.

Bei einem ingenieurmäßigen Vorgehen wird angestrebt, Techniken bereitzustellen, die eine möglichst automatisierbare Erstellung von Modellen erlauben. Es zeigt sich aber, dass die Modellierung realer Systeme allein mit vorgegebenen Techniken nicht realisierbar ist. Ein mehr oder weniger großes Maß an Kreativität ist zusätzlich erforderlich. Damit ist die Modellbildung in großem Maße *Kunst* gepaart mit der Verwendung von verfügbaren *Techniken*. Unterstützung bei der Modellierung gibt es heute durch vorhandene Richtlinien, Softwareumgebungen und -werkzeuge und natürlich durch Kenntnis in der Modellierungsmethodik und im Anwendungsgebiet.

Es lassen sich die folgenden Zielstellungen bei der Modellierung unterscheiden:

- *Erklärungsmodelle* bilden den Ist-Zustand eines Systems ab und dienen dazu das Verhalten deutlicher zu machen. Dazu werden Visualisierungstechniken eingesetzt. In heutiger Zeit spielt die Animation (2-D oder 3-D) dynamischer Abläufe eine zentrale Rolle. Die Modelle beschreiben teilweise nur das qualitative Verhalten, d.h. quantitative Aspekte wie Zeiten werden nicht berücksichtigt. Erst wenn Verhalten auch bewertet werden soll, müssen quantitative Aspekte zusätzlich berücksichtigt werden.

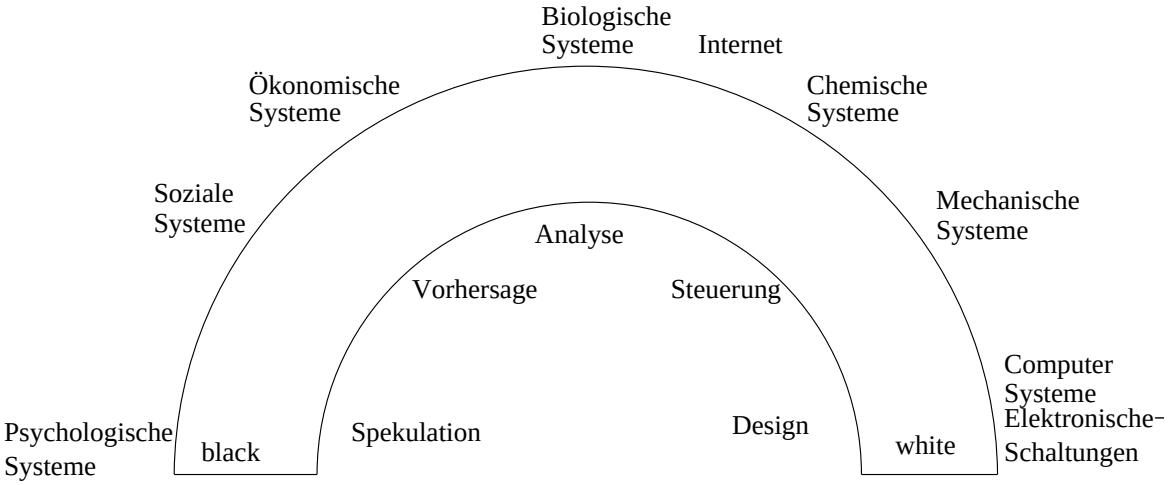


Abbildung 1.4: Regenbogen von Karplus (nach Cellier [1991]).

- *Prognosemodelle* dienen zur Vorhersage zukünftigen Verhaltens. Dazu muss das aktuelle Verhalten in die Zukunft extrapoliert werden.
- *Gestaltunsgmodelle* werden zur Analyse und zum Vergleich von Systemalternativen benutzt. Sie bieten damit eine Experimentierumgebung für ein System.
- *Optimierungsmodelle* haben die Suche nach einer optimalen Konfiguration oder Steuerung als Zielsetzung. Sie werden mit Optimierungsmethoden gekoppelt. Der Begriff Optimierung wird dabei nicht immer ganz korrekt eingesetzt, da es oft um das Finden einer guten aber nicht zwangsläufig optimalen Lösung geht.

Klarerweise bestimmt die Zielsetzung den verwendeten Modelltyp. Darüber hinaus gibt es zwei grundsätzlich unterschiedliche Arten der Modellbildung, die als black-box und white-box oder induktives und deduktives Vorgehen bezeichnet werden. Beim induktiven Vorgehen (black-box-Sichtweise) wird das Verhalten des Systems am Ausgang in Abhängigkeit der Parameterwerte am Eingang analysiert. Es wird also $f(C, U)$ oder auch nur $f(C)$ (falls U unbeobachtbar) an einigen Punkten beobachtet. Dann wird eine Funktion $g(C, U)$ bzw. $g(C)$ gesucht, so dass für die beobachteten Situationen $g(.)$ und $f(.)$ ähnliche Werte liefern. In der Regel wird die Funktion $g(.)$ aus einer Klasse leicht zu parametrisierender Funktionen gewählt. Funktion $g(.)$ wird anschließend als Modell verwendet. Beim deduktiven Vorgehen (white-box-Sichtweise) erfolgt die Modellbildung auf Basis der Systemstruktur. Es wird also Kenntnis über die Struktur des Modells, die auftretenden Wechselbeziehungen und das Verhalten der Komponenten vorausgesetzt. Falls möglich, sollte deduktiv modelliert werden, da die so entstehenden Modelle in fast allen Fällen deutlich bessere Abbildungen der Realität sind als induktiv erstellte Modelle.

Die Verwendung der Modellierungsansätze hängt natürlich vom Modellierer aber auch vom Anwendungsgebiet ab. Abbildung 1.4 zeigt den berühmten Regenbogen von Karplus Cellier [1991, Kap. 1], der für verschiedene Anwendungen die Analyseziele und Modellierungsmethodik zusammenfasst. An dieser Stelle sollen die einzelnen Punkte nicht im Detail erläutert werden. Aus der Abbildung wird aber deutlich, dass sich künstliche Systeme in den meisten Fällen sehr viel detaillierter modellieren lassen, da sie besser verstanden werden als natürlich Systeme. Konsequenterweise ist für die Modellierung künstlicher Systeme ein deduktives Vorgehen in fast allen Fällen angebracht, während für viele natürliche Systeme, mangels Alternativem, induktiv modelliert werden muss.

Eine wesentliche Unterscheidung im Bereich der dynamischen Modelle, die auch in der Simulation verwendet werden, ist die Unterscheidung in diskrete und kontinuierliche Modelle. Die wesentlichen Unterschiede werden in Abbildung 1.5 noch einmal deutlich gemacht, wobei dort diskrete Modelle zusätzlich in zeitdiskrete und ereignisdiskrete Modelle unterschieden werden.

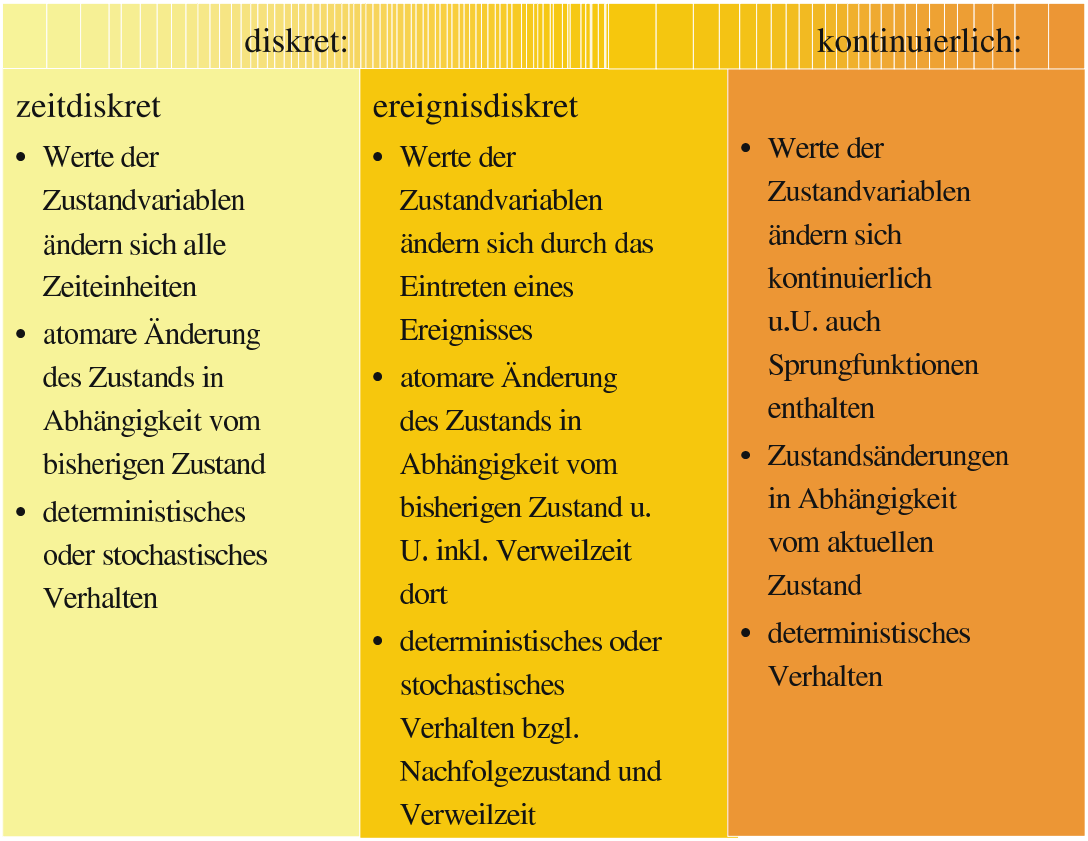


Abbildung 1.5: Unterteilung dynamischer Modelle

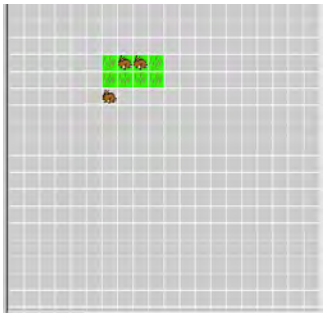


Abbildung 1.6: Modellwelt des Beispielmodells (aus dem Tool WorldMaker).

Es werden nun einige Beispiele für dynamische Modelle vorgestellt, die wir zum Teil auch später zur Simulation verwenden werden. Basismodelle zur Darstellung von dynamischen Abläufen sind einfache regelbasierte und zeitdiskrete Modelle. Modelle dieser Art gibt es in verschiedenen Ausprägungen. Wir betrachten exemplarisch eine einfache Variante, die ursprünglich zum Einsatz im schulischen Bereich entwickelt wurde. Die Modellwelt besteht aus einem endlichen zweidimensionalen Gitter von Zellen. Zellen können bestimmte Zustände haben. Der Zustand des Modells ist der Zustand aller Zellen. Das Modell ist zeitdiskret. Dynamik wird dadurch modelliert, dass zu jedem Zeitpunkt, zu dem ein Zustandswechsel stattfindet, Regeln angewendet werden, nach denen sich der Zustand des Systems im nächsten Zeitintervall aus dem Zustand zum jetzigen Zeitpunkt ergibt. Da die Spezifikation allgemeiner Regeln sehr aufwändig und kaum beherrschbar wäre, werden Regeln lokal definiert. D.h. der Zustand einer Zelle im nächsten Zeitintervall ergibt sich aus dem aktuellen Zustand der Zelle und der Nachbarzellen.

Als Beispiel soll eine Kaninchenpopulation auf einer Weise dienen. Das Modell wird mit dem Tool WorldMaker Law [1999] modelliert[2]. Jede Zelle hat einen Hintergrundzustand, in unserem Fall unbewachsen (grau/braun) oder bewachsen (grün). Weiterhin existieren Spezies, die sich auf Zellen befinden können. Zu einem Zeitpunkt kann sich maximal ein Individuum in einer Zelle befinden. Im Beispiel betrachten wir als einzige Spezies die Kaninchen. Jede Zelle kann damit in einem von vier Zuständen sein, nämlich bewachsen-unbewachsen, mit oder ohne Kaninchen. Die Dynamik im Modell ergibt sich durch Anwendung der Regeln aus Abbildung 1.7. Regeln beziehen sich jeweils auf die aktuelle Zelle und eine beliebige Nachbarzelle. Es bleibt damit noch festzulegen, welche Regel ausgewählt wird, wenn mehrere Regeln angewendet werden können. Für diese Festlegung gibt es verschiedene Möglichkeiten. So kann die Auswahl zufällig erfolgen oder es kann eine Prioritätisierung unter den Regeln vorgenommen werden. Im ersten Fall zeigt das Modell ein stochastisches Verhalten, im letzteren ein deterministisches Verhalten.

Wenn man das Modell mit dem in Abbildung 1.6 gezeigten Startzustand laufen lässt, so beobachtet man ein oszillierendes Verhalten. Wenn viele Zellen bewachsen sind und wenige Kaninchen da sind, so wächst die Kaninchenpopulation stark, dadurch wird viel Gras gefressen und die Kaninchen leiden an Nahrungsmangel, wodurch die Population abnimmt und das Gras wachsen kann. Auch wenn dieses Modell sehr einfach ist und viele Details realer Ökosysteme vernachlässigt, kann man das am Modell beobachtbare qualitative Verhalten auch in der Natur beobachten.

Es ist leicht nachvollziehbar, dass die vorgestellte Art der Modellierung einige Schwächen aufweist. Insbesondere ist die Darstellung bei aller Einfachheit doch sehr detailliert und damit aus Komplexitätsgründen kaum für größere Populationen geeignet. Wenn das Ziel der Modellierung die Analyse der Kaninchen und des Grases ist, so spielt die räumliche Ausdehnung offensichtlich keine Rolle und man kann sich auf die Interaktion zwischen Gras und Kaninchen beziehen. Dazu abstrahieren wir von den einzelnen Individuen und drücken die Menge des Grases und die Zahl der Kaninchen durch Variablen $x_G, x_K \in \mathbb{R}_{\geq 0}$ aus. Die Population ist damit nicht ganzzahlig sondern eine nicht negative reelle Zahl. Bei entsprechend großen Populationen ist die Abstraktion gerechtfertigt. Die Veränderung in der Populationen kann nun wieder in diskreter oder kontinu-

[2]Eine Java-Version des Tools ist unter `http://worldmaker.cite.hku.hk/worldmaker/pages/` zu finden.

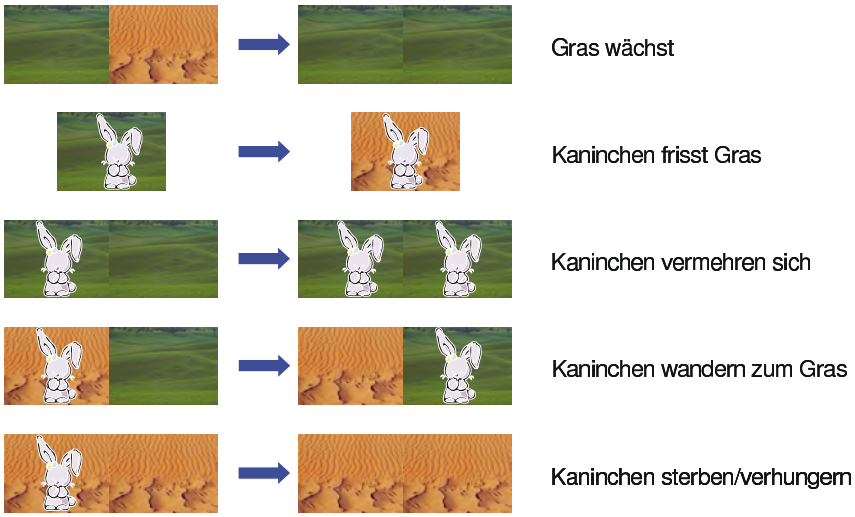


Abbildung 1.7: Regeln zur Zustandsänderung (aus dem Tool WorldMaker).

ierlicher Zeit erfolgen und ergibt sich aus der Interaktion zwischen den beteiligten Spezies. Heute wird üblicherweise eine kontinuierliche Zeitskala verwendet (siehe auch Cellier [1991]) und die Dynamik mittels Differentialgleichungen beschrieben. Für unser Beispiel könnte man folgendes Differentialgleichungssystem verwenden.

$$\dot{x}_K = -a \cdot x_K + k \cdot b \cdot x_K \cdot x_G \quad \text{und} \quad \dot{x}_G = c \cdot x_G - b \cdot x_K \cdot x_G$$

mit $a, b, c > 0$ und $0 < k < 1$. x_K ist die Kaninchenpopulation, x_G die Menge an Gras. Kaninchen sterben mit Rate a unabhängig vom Futterangebot. Gleichzeitig nimmt die Population zu, wenn Futter vorhanden ist. Die Zuwachsrate ist proportional zum Produkt der Kaninchenpopulation und der Grasmenge, dem Wachstumsfaktor b und einer Konstante k, die den Energieanteil modelliert, der vom Gras zu die Kaninchen übergeht. Das Gras wächst von selbst mit Rate c und wird mit Rate bx_Gx_K gefressen. Insgesamt ergibt sich damit das gezeigte Differentialgleichungssystem. Das resultierende Modell bezeichnet man als Lotka-Volterra-Modell Cellier [1991, Kap. 10]. Ähnlich wie bei dem einfachen zellenbasierten Modell können wir auch hier eine oszillierende Population beobachten. Der Aufwand der Berechnung hängt nun aber nicht mehr von der Population ab. Da die mathematische Beschreibung relativ komplex werden kann und Abhängigkeiten kaum erkennbar sind, gibt es zahlreiche graphische Ansätze zur Beschreibung von Differentialgleichungen. Ein in der Modellierung von Ökosystemen weit verbreiteter Ansatz sind die Modelle der System Dynamics Cellier [1991, Kap. 11]. Eine mögliche Darstellung wird in Abbildung 1.8 gezeigt. Wir wollen an dieser Stelle nicht weiter auf die Details der Modellwelt eingehen.

Neben der Modellwelt der System Dynamics gibt es zahlreiche weitere Ansätze zur grafischen Spezifikation von Differentialgleichungssystemen. Abbildung 1.9 zeigt eine blockorientierte Darstellung der Differentialgleichung $dK/dt = l(t) - a \cdot K(t)$ und Abbildung 1.10 zeigt eine anwendungsspezifische Darstellung von Differentialgleichungen, hier am Beispiel elektrischer Schaltkreise.

Viele diskrete Systeme lassen sich auf Warteschlangenmodelle abbilden. Die Idee der Warteschlangenmodelle besteht darin, dass Kunden oder Aufträge an Ressourcen oder Stationen Bedienung verlangen. Diese Bedienung dauert eine gewisse (oft stochastische) Zeit und nach Abschluss der Bedienung verlässt der Auftrag die Station und betritt u. U. eine andere Station. Es gibt viele unterschiedliche Ausprägungen von Warteschlangennetzen. An dieser Stelle soll kurz eine einfache Ausprägung der Modellwelt vorgestellt werden. Eine einzelne Station wird in Abbildung 1.11 gezeigt. Die Station besteht aus einer Warteschlange und einem Bediener. Der Bediener bedient Kunden nach der Strategie First Come First Served (FCFS). Ist der Bediener belegt, so müssen ankommende Kunden in der Warteschlange warten. Nach Bedienende verlassen die Kunden das System wieder. Die Objekte der Systembeschreibung sind Kunden, Bedieneinrichtung und War-

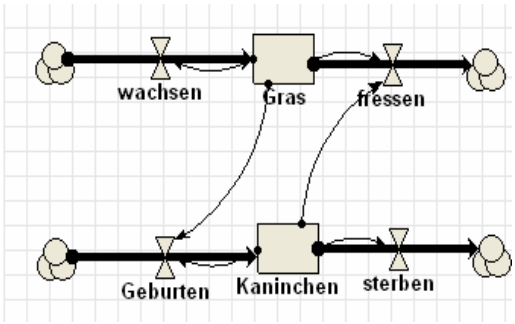


Abbildung 1.8: Beschreibung des Beispiels mit einem Modell der System Dynamics-Welt (aus dem Werkzeug DynaSys).

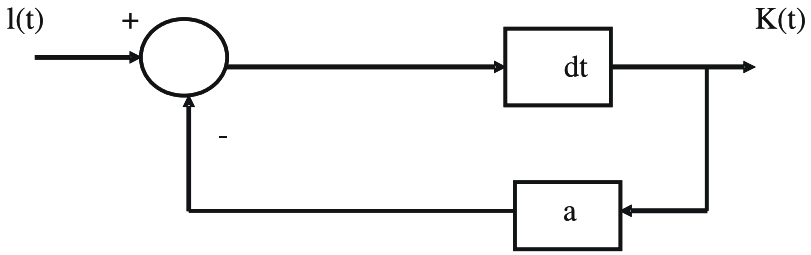


Abbildung 1.9: Blockorientierte Darstellung einer Differentialgleichung.

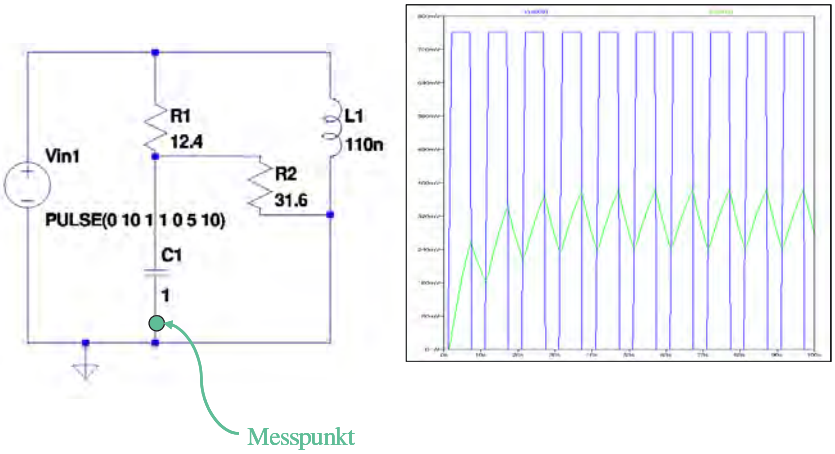


Abbildung 1.10: Modell einer elektronischen Schaltung und die Ergebnisse eines Simulationslaufs (aus dem Werkzeug SwitcherCAD).

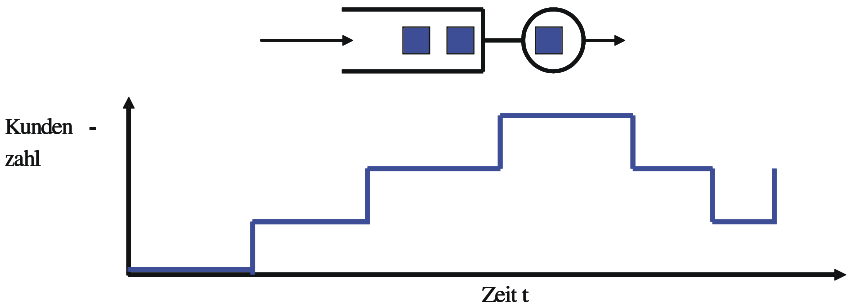


Abbildung 1.11: Station in einem Warteschlangennetz.

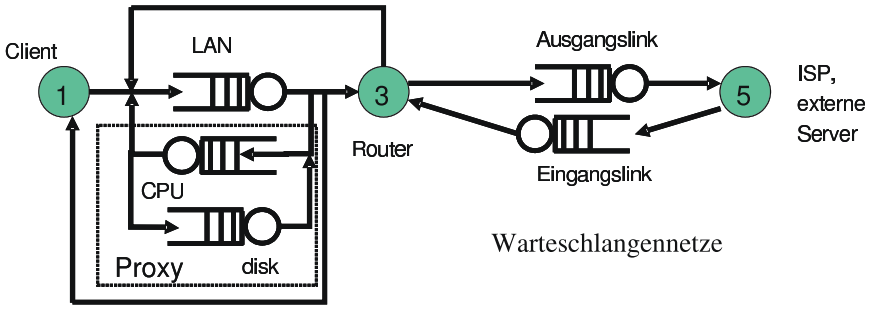


Abbildung 1.12: Warteschlangenmodell eines Web-Servers.

teschlange. Jeder Kunde hat eine Ankunftszeit und einen Bedienbedarf. Der Zustand des Systems ist gegeben durch die Zahl der Kunden im System und deren Bedienzeiten und der Ankunftszeit des nächsten Kunden. Attribute ändern sich bei Ankunft und Bedienende der Kunden (ereignisdiskrete Betrachtung). Zu diesen Zeitpunkten wird die Kundenzahl im System erhöht/reduziert und u.U. der Status des Bedieners geändert (belegt - frei).

Durch die Kombination von Stationen entstehen Warteschlangennetze, die in unterschiedlichen Gebieten zur Systemanalyse eingesetzt werden. Abbildung 1.12 zeigt als Beispiel das einfache Modell des Zugriffs auf einen Web-Server mit zwischen geschaltetem Proxy-Server. Kunden sind in diesem Fall Anfragen an den Web-Server, die Komponenten des Systems werden als Stationen modelliert. Auch dieses System beschreibt ein ereignisdiskretes Verhalten. Der Zustand des Modells ändert sich sobald eine Bedienung endet oder beginnt.

Neben der grafischen Darstellung von Modellen wurden Modelle über lange Zeit in programmiersprachlicher Form beschrieben. Vor der Verfügbarkeit grafischer Eingabeschnittstellen war dies die einzige Möglichkeit Dynamik zu beschreiben. Auch bei heutigen grafischen Modellierungsparadigmen ist es in der Regel noch notwendig, die grafische Darstellung mit programmiersprachlichen Beschreibungen zu ergänzen, um komplexes Verhalten darzustellen. Darüber hinaus dienen grafische Modelle oftmals als Front-End für die Beschreibung von Simulationsprogrammen in einer Simulationssprache. Zur Darstellung von Dynamik in Programmiersprachen muss eine entsprechende Unterstützung vorhanden sein. Für diskrete Systeme benötigt man:

- Eine Darstellung des zeitlichen Ablaufs,
- Unterstützung bei der Ereignisverwaltung,
- Unterstützung bei der Realisierung von Zufallseinflüssen und
- Unterstützung bei der Systembeobachtung und -auswertung.

Beispiele für Sprachen, die diese Unterstützung bieten sind GPSS, Simula und Simscript. Für kontinuierliche Systeme muss zumindest

- die Möglichkeit der Formulierung mathematischer Zusammenhänge und
- eine Unterstützung bei der Lösung von Differentialgleichungssystemen

vorhanden sein. Beispiele für solche Sprachen sind Mathematica, Matlab, Octave oder Scilab. Wir werden uns in Kapitel 6 etwas näher mit Programmiersprachen zur diskreten Modellierung beschäftigen.

Im Informatikumfeld existieren eine Reihe weiterer Modelltypen zur Beschreibung von diskreten dynamischen Systeme, die sich grob in formale und semi-formale Modelltypen einordnen lassen. Im Bereich der formalen Modelle sind endliche Automaten, Automatennetze und Petri-Netze von zentraler Bedeutung. Beide Modelltypen vereinen eine grafische Darstellung (siehe Abbildung 1.13) mit einer formalen Semantik, die das Modellverhalten festlegt. Die Modelle beinhalten in ihrer ursprünglichen Form keine Darstellung der Zeit oder keine Wahrscheinlichkeiten zur Konfliktlösung,

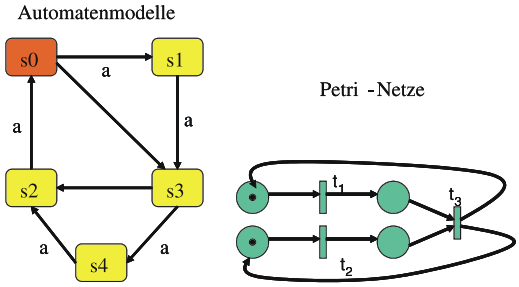


Abbildung 1.13: Endlicher Automat und Petri-Netz als diskrete Modelle.

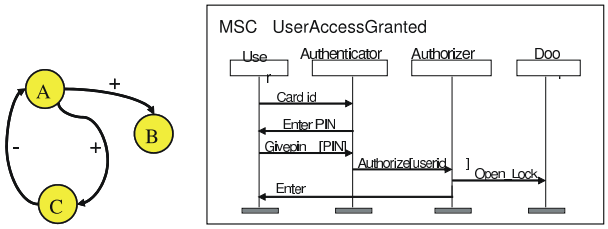


Abbildung 1.14: Wirkungsgraph und MSC als semiformale Modelle.

sondern beschreiben die kausalen Abhängigkeiten zwischen Ereignissen. Es gibt aber inzwischen zahlreiche Erweiterungen, die quantitative Aspekte und damit auch Zeit integrieren.

In der Praxis werden oftmals semiformale Modelle eingesetzt, da sie keine strikte Formalisierung erfordern. Dies hat den Vorteil, dass gewisse Details nicht vollständig berücksichtigt werden müssen, hat aber gleichzeitig den Nachteil, dass dynamisches Verhalten nicht vollständig modelliert wird. Zwei typische Beispiele für semiformale Modelle sind in Abbildung 1.14 dargestellt. Wirkungsgraphen beschreiben den Einfluss einzelner Variablen aufeinander. Dabei wird nur qualitative Information dargestellt, z.B. A wird größer $\Rightarrow$ B wird größer. Sie beinhalten keine Information über die tatsächlichen mathematischen Zusammenhänge und können deshalb auch nicht als ausführbare Modelle dienen. In den frühen Phasen der Modellbildung erlauben Wirkungsgraphen die Darstellung kausaler Zusammenhänge. Message Sequence Charts (MCSs) beschreiben exemplarische Abläufe. Sie enthalten keine Aussagen über alle möglichen Abläufe und auch keine Quantifizierung der Abläufe.

Die bisher vorgestellten Modelle sind auf einer relativ abstrakten Ebene angesiedelt und erlauben eine oftmals problemnahe Modellierung. Sofern es sich um formale Modelle handelt lassen sich die Modelle in der Regel auf ein mathematisches Modell abbilden, das dann analysiert werden kann. Für diskrete Modelle ist das mathematische Basismodell meist ein bewertetes Transitionssystem oder eine Formel einer temporalen Logik. Beide Ansätze können um deterministische Zeiten erweitert werden. Im stochastischen Fall hat man es in der Regel mit einem stochastischen Prozess zu tun. In den letzten Jahren ist es gelungen, stochastische Prozesse und bewertete Transitionssysteme zu kombinieren, um so eine allgemeine Modellklasse zu erhalten. Im Bereich der kontinuierlichen Systeme arbeitet man heute primär mit Differentialgleichungen.

In der Praxis werden Modelle in einer abstrakteren Darstellung spezifiziert, etwa als Warteschlangennetz. Die Analyse erfolgt dann automatisch entweder durch Transformation in das zugehörige mathematische Modell und dessen Analyse oder durch Nachspielen des dynamischen Verhaltens direkt im abstrakteren Modell.

1.3 Analyse, Simulation und Optimierung

Bisher haben wir die Beschreibung eines realen oder geplanten Systems durch ein formales Modell betrachtet, ohne die Analyse genauer zu untersuchen. Ziel ist es, Experimente am realen System

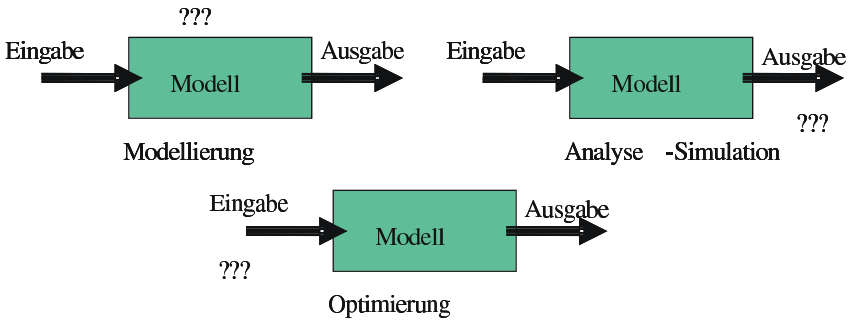


Abbildung 1.15: Schematische Darstellung der modellgestützten Analyse.

durch Experimente mit dem Modell zu ersetzen und dabei zu identischen Resultaten zu gelangen. Dabei stellt sich oft die Frage nach Veränderungen am System, um dessen Leistung oder Zuverlässigkeit zu verbessern. Die verschiedenen Schritte werden an der einfachen Graphik in Abbildung 1.15 dargestellt. Kontrollierbare und unkontrollierbare Eingaben sind dort zusammengefasst. Bei dem gezeigten Modell kann es sich sowohl um ein black-box als auch um ein white-box-Modell handeln. Das Ziel der Modellierung ist es, aus vorhandenen Ein- und Ausgabedaten unter Umständen mit zusätzlicher Strukturinformation ein Modell zu generieren. Bei der Analyse werden die Ausgaben eines gegebenen Modells zu vorgegebenen Eingaben ermittelt. Bei der Optimierung geht es schließlich darum, zu einem gegebenem Modell und vorgegebenen Anforderungen an die Ausgaben (z.B. möglichst klein, größer als, etc.) Eingaben zu finden, die den Anforderungen genügen.

Zur Analyse von Modellen, also zur Auswertung von f, gibt es unterschiedliche Möglichkeiten, die von der Struktur des Modells abhängen. Der einfachste und oftmals ideale Fall ist, dass f als explizite Formel vorliegt. Ein typisches Beispiel ist das Ohmsche Gesetz $R = U/I$ ($\Omega = V/A$). In solchen einfachen Fällen kann die Analyse und Optimierung prinzipiell per Hand erfolgen. Allgemein spricht man immer dann von einer *analytischen* Lösung, wenn sich die Ergebnisse in geschlossener Form prinzipiell darstellen lassen. Dies kann durchaus bedeuten, dass z.B. große lineare Gleichungssysteme gelöst werden müssen. In der Praxis wird man solche Gleichungssysteme nicht symbolisch lösen wollen oder können, prinzipiell ist aber eine geschlossene Lösung algorithmisch herleitbar.

Die nächst komplexere Lösung wäre die *numerische* Analyse. In diesem Fall liegt f nur implizit als Formel vor, für die keine geschlossene Formel existiert. So lässt sich das Integral der standardisierten n-dimensionalen Normalverteilung
$((2\pi)^n \det V)^{-1/2} \int_{-\infty}^{b_1} \dots \int_{-\infty}^{b_n} \exp\left(-1/2 x^T V^{-1} x\right) dx_1 \dots dx_n$ mit $V \in \{\mathbb{R}\}_{\geq 0}^{n \times n}$ und $x \in \{\mathbb{R}\}_{\geq 0}^{n}$
nicht in geschlossener Form darstellen. Vielmehr muss durch punktweises Abtasten für jeweils feste Werte von x die Funktion analysiert werden. Methoden zum Abtasten findet man in der numerischen Mathematik.

Falls f nur als eine Menge von Zusammenhängen und Abhängigkeiten bekannt ist, so kann das Verhalten des Modells nur durch schrittweises Durchspielen der Abhängigkeiten analysiert werden. Dies ist die schwierigste und aufwändigste Form der Analyse, die auch als *simulatives* Vorgehen bezeichnet wird. Ein Beispiel ist die Beschreibung einer Ampelkreuzung. Die Schaltzeiten der Ampel, die Ankunftszeiten, Geschwindigkeiten und Beschleunigungswerte der Fahrzeuge sind bekannt. Zur Ermittlung der Anzahl der wartenden Fahrzeuge zu einem vorgegebenem Zeitpunkt müssen alle Fahrzeugbewegungen nachgespielt werden.

Vom Standpunkt einer effizienten Analyse ist eine analytische Lösung anzustreben, da nur so Lösungen über dem gesamten Parameterraum charakterisierbar sind. Ziel ist es, das einfachste adäquate Modell zu wählen. Um eine analytische Lösung zu erhalten ist es aber oft notwendig das Systemverhalten im Modell stark zu vereinfachen. Dies erfordert eine höhere Abstraktion, die vom Modellierer geleistet werden muss. Außerdem sind für viele Problemstellungen analytische Modelle zu abstrakt, da sie wesentliche Verhaltensdetails nicht berücksichtigen. Aus diesem Grund wird in der Systemanalyse oft auf simulative Modelle zurückgegriffen.Allerdings sind analytische Modelle,

wenn sie verfügbar sind, vorzuziehen, da sie in der Regel deutlich einfacher zu handhaben und verstehen sind.

Bei der Simulation werden im Gegensatz zum analytischen Vorgehen Resultate punktweise durch Nachspielen des Systemverhaltens ermittelt. Oft beinhaltet das Systemverhalten stochastische Elemente. Es existieren zahlreiche Definitionen des Begriffs Simulation. Hier sind drei Beispiele:

- Durchführung von Experimenten an einem Modell, das anstelle des Originalsystems tritt. (nach Krüger [1975])
- Nachbildung eines dynamischen Prozesses in einem Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind. (VDI Richtlinie)
- Prozess der Modellbeschreibung eines realen Systems und anschließendes Experimentieren mit diesem Modell mit der Absicht, entweder das Systemverhalten zu verstehen oder verschiedene Strategien für Systemoperationen zu gewinnen. (nach Shannon [1975])

Gegenüber analytischen Modellen erlauben simulative Modelle eine realitätsnähere Modellierung. So können beliebige Verteilungen abgebildet werden, es können komplexe Synchronisationen dargestellt werden und es besteht die Möglichkeit, auch komplexe Abhängigkeiten zu modellieren. Verhalten, das in der Realität erkannt und beobachtet wird, kann prinzipiell auch in der Simulation nachgebildet werden. Grenzen sind durch die Komplexität der resultierenden Modelle und den Simulationsaufwand auf natürliche Weise gegeben. Insofern besteht auch bei der Simulation ein Zwang zur Abstraktion und Idealisierung. Ein weiterer Vorteil der Simulation besteht darin, dass sie es als einzige Technik erlaubt reale Systeme und Modelle zu koppeln. So können in der Realität aufgezeichnete Traces als Teil eines Simulationsmodells verwendet werden. Beim obigen Ampelbeispiel können Ankunftszeiten von Fahrzeugen an einer realen Ampel gemessen und anschließend in der Simulation verwendet werden, um dort identische Fahrzeugströme zu erzeugen und auf dieser Basis verschiedene Strategien der Ampelsteuerung zu vergleichen. Weiterhin ist es sogar möglich, Modell und System direkt zu koppeln. Beispiele für eine solche Kopplung sind men-in-the-loop oder hardware-in-the-loop Simulatoren. Im ersten Fall wird Simulation als Teil der Realität benutzt, mit der ein Mensch interagiert. Flugsimulatoren sind ein Beispiel für eine solche Kopplung. In hardware-in-the-loop Simulatoren werden Ereignisse und Zustände der Simulation in elektrische Impulse umgesetzt und umgekehrt elektrische Impulse in Ereignisse und Zustände der Simulation. Auf diese Art lassen sich elektronische Bauteile testen. Men-in-the-loop und hardware-in-the-loop sind Beispiele für Echtzeitsimulatoren, d.h. die Zeit im Modell muss mit der realen Zeit übereinstimmen. Echtzeitsimulatoren können auch benutzt werden, um Prozesse zu steuern oder verteilte Softwaresysteme zu testen.

Auch wenn die Verwendung simulativer Modelle eine Vielzahl von Möglichkeiten eröffnet, so sollte doch immer das einfachste adäquate Modell verwendet werden. Dies bedeutet, dass falls möglich, analytischen Modellen der Vorzug vor Simulationsmodellen gegeben werden sollte. Die wesentlichen Nachteile der Simulation sind ein hoher Erstellungsaufwand verbunden mit einem großen Datenbedarf und damit einem hohen Aufwand der Datenerhebung. Da die Simulation keinen Zwang zur Abstraktion liefert, werden Modelle oft komplexer als notwendig und sind damit unverständlich und kaum validierbar. Ein anderer Aspekt ist die oft notwendige Verwendung von Stochastik, die zu schwankenden Ergebnissen und damit zur Notwendigkeit statistischer Ergebnisauswertung führt. Ein weiterer Nachteil von Simulationsmodellen ist das Fehlen von Strukturinformation, die für viele Optimierungsmethoden notwendig ist.

Ziel der Modellbildung ist es, Aussagen über ein System zu machen und oftmals auch Entscheidungen über die Konfiguration des Systems zu treffen, so dass möglichst gute oder optimale Resultate erzielt werden. Allgemein gefasst bedeutet dies, dass nach dem "besten" System für eine Aufgabenstellung gesucht wird. Eine solche Zielstellung ist für die meisten realen Probleme per se unlösbar. Deshalb wird nach kontrollierbaren Eingaben C gesucht, so dass die Ausgaben P optimal sind. Im Gegensatz zur allgemeinen Formulierung bedeutet dies, dass die Struktur des Systems vorgegeben wird und nur einzelne Parameter geändert werden. Je nach Interpretation der Parameter können in C aber durchaus vorher geplante Strukturalternativen kodiert sein.

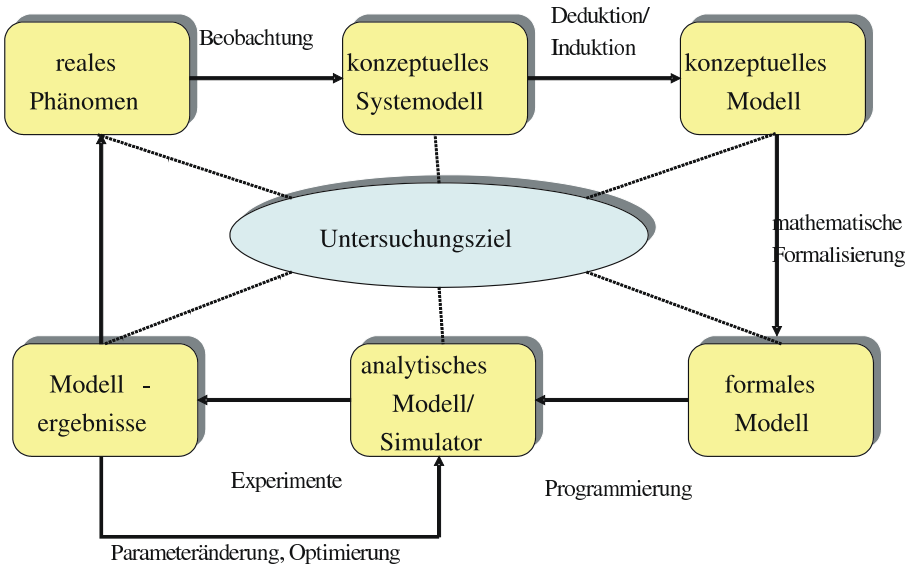


Abbildung 1.16: Zyklus der modellgestützten Analyse und Optimierung.

Die konkret angewendete Optimierungsmethode hängt vom konkreten Modell und dessen Struktur ab. Bei der Verwendung von Simulationsmodellen wird die Optimierung dadurch erschwert, dass f nicht explizit vorliegt sondern nur punktweise abgetastet werden kann und damit auch keine Information über die Ableitungen von f vorhanden ist. Falls f auch von U abhängt, so ändert sich P bei festem C durch variierende Werte von U. Damit muss Optimalität für alle Werte von U in Abhängigkeit von deren Auftretenswahrscheinlichkeit definiert werden. Weitere Erschwernisse können darin liegen, dass C nicht frei wählbar ist, sondern gewissen Nebenbedingungen gehorchen muss und P mehrdimensional ist und sich nicht sinnvoll auf eine skalare Zielfunktion abbilden lässt. Im letzteren Fall existiert kein Optimum, sondern nur eine Menge von Punkten, die sich nicht mehr in allen Komponenten der Zielfunktion verbessern lassen.

Zum Abschluss dieses Kapitels wird der abstrakte Ansatz der Modellierung aus Abbildung 1.3 unter Berücksichtigung der Analyse und Optimierung verfeinert. Abbildung 1.16 zeigt den Zyklus der modellgestützten Analyse. Zentral ist das Untersuchungsziel, welches bei allen Schritten im Auge behalten werden muss. Über die angegebenen Zwischenschritte gelangt man vom realen Phänomen zum Modell mit dem experimentiert werden kann und das die Grundlage für die Optimierung ist. Die Ergebnisse der Analyse und Optimierung fließen in das reale System ein.

Ein zentraler Punkt jeder Art von modellbasierter Analyse ist die prozessbegleitende Validierung. Es ist festzustellen, inwieweit ein Modell das Verhalten des Systems bzgl. des Untersuchungsziels adäquat wiedergibt. Dies ist streng mathematisch nicht beweisbar, sondern es kann nur eine graduelle Validität festgestellt werden. Die folgenden Aspekte der Validität werden unterschieden:

- Bei der Untersuchung der Strukturgültigkeit wird festgestellt, inwieweit System und Modell strukturell übereinstimmen.
- Die Verhaltensgültigkeit bewertet, inwieweit des Verhalten für relevante Anfangsbedingungen und äußere Einflüsse übereinstimmt.
- Die empirische Gültigkeit stellt die genügende Ähnlichkeit der im System gemessen Daten und der aus dem Modell ermittelten Daten fest.
- Die Anwendungsgültigkeit weist schließlich nach, dass die Modellbildung der Zielsetzung entspricht und damit die durchgeführten Experimente mit dem Modell zu den gleichen Entscheidungen führen, die auch bei Experimenten mit dem System getroffen worden wären.

Aus der kurzen Beschreibung folgt, dass Validität immer abhängig vom Ziel und auch nur graduell feststellbar ist. Abschnitt 8 beschäftigt sich mit Methoden zur Feststellung der Validität auf unterschiedlichen Ebenen.

1.4 Literaturhinweise

Grundlagen der Modellierung und Simulation werden in praktisch allen Büchern zur Simulation vermittelt. Die Basis dieses Kapitel bilden Kapitel aus Beilner [2002], Law and Kelton [2000], Banks et al. [2000], Cellier [1991]. Ein speziell auf das Simulationswerkzeug bzw. die Simulationsumgebung Arena ausgerichteter Ansatz zur Modellierung wird in Kelton et al. [2004] gegeben. Eine sehr allgemeine Übersicht über die Modellierung mit ereignisdiskreten Systemen bietet Cassandras and Lafortune [2007]. Darüber hinaus existieren zahlreiche grundlagenorientierte Bücher, die sich mit der Modellierung von natürlichen und ökologischen Systemen beschäftige Bossel [1989], Imboden and Koch [2003]. Der dabei verwendete Modelltyp sind allerdings im Wesentlichen kontinuerliche Modelle, die in dieser Vorlesung nicht von Interesse sind.

Teil I

Modellierung und Analyse

Als zentrale Modellklasse werden in der Vorlesung diskrete Systeme betrachtet. Diese Systeme ändern zu diskreten Zeitpunkten ihren Zustand. In den meisten Fällen wird das Verhalten der Systeme teilweise durch Zufallsvariablen beschrieben. Dies bedeutet, dass Zufallsvariablen festlegen, wie viel Zeit zwischen zwei Ereignissen vergeht und welche Entscheidung aus einer Menge von möglichen Entscheidungen getroffen wird. Modelle dieser Art werden per Simulation analysiert. Dabei wird das dynamische Verhalten nachgebildet und beobachtet. Dazu muss der Modellzustand im Programm dargestellt werden, die auftretenden Ereignisse müssen zeitgerecht ausgeführt werden und das Modell muss beobachtet werden, um auf diese Weise zu Resultaten zu gelangen.

Es gibt zahlreiche Lehrbücher zum Thema diskrete Simulation mit recht unterschiedlichen Schwerpunktsetzungen. Allerdings existiert momentan kein aktuelles deutschsprachiges Buch, welches die hier behandelte Thematik abdeckt. Das Kapitel beruht auf einzelnen Kapitel der englischsprachigen Lehrbücher Law and Kelton [2000] und Banks et al. [2000]. Beide Bücher liefern eine recht vollständige Darstellung der Thematik, die durch einige spezifische Informationen über spezielle Werkzeuge und Methoden ergänzt wird. Ein weiteres Lehrbuch über diskrete Simulation, welches Simulation auf Basis des in den Übungen verwendeten Tools Arena einführt ist Kelton et al. [2004]. Ebenfalls zu empfehlen ist ein Blick in die auf der Winter Simulation Conference präsentierten Arbeiten. Neben aktuellen Forschungsarbeiten sind dort auch zahlreiche Tutorien zur Einführung verfügbar.

Die folgenden Kapitel geben eine Einführung in das Gebiet der diskreten Simulation. Die dabei verwendeten Methoden sind der Informatik, der Statistik und den Anwendungswissenschaften zuzuordnen. Das erste Kapitel beschäftigt sich mit den programmtechnischen Voraussetzungen zur Realisierung von Simulatoren und ist damit der Informatik zuzuordnen. Die dann folgenden Kapitel 3 bis 5 betrachten die Einbeziehung von Zufall in Simulationsmodelle. Dazu sind Methoden aus der Wahrscheinlichkeitsrechnung und Statistik notwendig. Daran anschließend wird ein kurzer Überblick über Simulationssoftware gegeben und es werden die Grenzen der Simulation aufgezeigt. Die dabei betrachteten Methoden sind der Informatik und dem jeweiligen Anwendungsgebiet zuzurechnen. Kapitel 8 stellt Ansätze zur Modellvalidierung vor. Die dabei verwendete Methodik stammt aus der Informatik und der Statistik.

Kapitel 2

Konzepte ereignisdiskreter Simulation

Wir wollen in diesem Kapitel damit beginnen den grundsätzlichen Ablauf diskreter Simulation kennen zu lernen. Diese Grundlagen sind in Banks et al. [2000, Kap. 3] und Law and Kelton [2000, Kap. 2.1, 2.2] sowie in vielen anderen Lehrbüchern über Simulation zu finden.

Wie bereits herausgearbeitet, soll mittels Simulation der dynamische Ablauf in einem realen System nachgebildet werden. Da ereignisdiskrete Systeme untersucht werden, ändert sich der Zustand jeweils zu Ereigniszeitpunkten. Dies bedeutet, dass

- der Systemzustand bekannt sein muss und
- der Zeitpunkt und die Auswirkungen der nächsten Zustandsänderung ebenfalls bekannt sein müssen.

2.1 Struktur und Ablauf der Simulation

Dieser Abschnitt beschreibt die Basisstrukturen, die zum Verständnis der Abläufe in ereignisdiskreten Simulatoren notwendig sind. Man unterscheidet dabei den statischen Aufbau, der den Zustand des Modells beschreibt, und die dynamischen Abläufe, die den Zustand im Ablauf der Zeit ändern.

2.1.1 Statische Struktur eines Simulators

Der Zustand des Simulators ist durch die Werte der Variablen im Programm gegeben. Die nächste Zustandsänderung erfolgt durch ein Programmsegment, auf das der Programmzeiger zeigt. Damit wird der Zustand des Realsystems im Simulator durch eine entsprechende Datenstruktur repräsentiert, die für relevante Zustandsmerkmale Variablen enthält. Man spricht von der *statischen Struktur* des Modells. Die Abbildung vom realen System erfolgt nach dem in der Informatik üblichen Vorgehen. Das reale System wird zuerst in einem Gedankenmodell strukturiert. Dadurch entstehen Objekte mit Attributen, die jeweils in Zustandsvariablen erfasst werden. Die Struktur des Gedankenmodells dient zur Strukturierung der Zustandsvariablen in entsprechenden Datenstrukturen. Das Ergebnis dieses Prozesses ist dann eine minimale Simulator-Datenstruktur.

2.1.2 Dynamische Struktur eines Simulators

Zustandsänderungen im Simulator erfolgen bei Werteänderungen der Zustandsvariablen. Dies bedeutet, dass neue Werte zugewiesen werden müssen, was nur innerhalb des Simulationsprogramms durch die Ausführung von Simulator-Code geschehen kann. Damit muss zu jedem Ereigniszeitpunkt ein Stück Simulator-Code abgearbeitet werden und in diesem Code-Segment müssen die

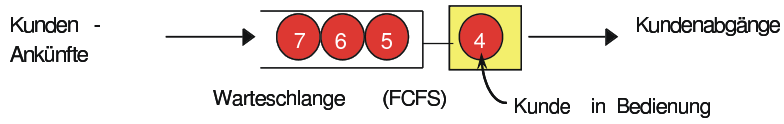


Abbildung 2.1: Station in einem Warteschlangennetz (nach Kelton et al. [2004]).

Auswirkungen des Ereignisses auf den Systemzustand realisiert sein. Um das Vorgehen zu strukturieren werden im Simulator *Ereignistypen* festgelegt. Ein Ereignistyp ist durch eine Menge von Zustandsänderungen, d.h. Wertzuweisungen an Variablen charakterisiert. Zu jedem Ereignistyp gibt es eine *Ereignisroutine*, die die Zustandsänderungen durch den zugehörigen Ereignistyp im Programm umsetzt.

Um die Simulation durchzuführen fehlt damit nur noch ein Mechanismus zur zeitgerechten Ausführung der Ereignisse. Dieser Aspekt soll nun etwas genauer beschrieben werden. Wenn man den Zeitablauf im Modell betrachtet, so fällt auf, dass die Abarbeitung von Ereignisroutinen Zeit benötigt, da Code ausgeführt wird. Aus Sicht des Modells sind Ereignisse dagegen atomar und finden in Nullzeit zu einem Zeitpunkt statt. Andererseits vergeht im Modell und damit auch in der modellierten Realität Zeit zwischen zwei Ereignissen. Wenn zum Zeitpunkt t ein Ereignis und zum Zeitpunkt $t + \Delta$ das nächste Ereignis ausgeführt wird, so vergeht die Zeit Δ. Im Simulator wird in diesem Fall das erste Ereignis ausgeführt und nach Abarbeitung der zugehörigen Ereignisroutine die Ereignisroutine für das nächste Ereignis aufgerufen. Dies bedeutet, dass unabhängig von der Größe von Δ praktisch keine Zeit zwischen den Ereignissen vergeht. Wir werden uns diesem Phänomen der unterschiedlichen Zeiten in Simulationsmodellen später noch etwas genauer widmen.

2.1.3 Ein einfaches Beispiel

An Hand eines einfachen Beispiels soll zuerst der grundsätzliche Ablauf einer Simulation näher erläutert werden. Wir betrachten als Beispiel einen Schalter an dem Bedienungen stattfinden. Es existiert genau ein Bediener, der zu jedem Zeitpunkt maximal einen Kunden bedienen kann und immer arbeitet, d.h. Kunden bedient, wenn Kunden im System sind. Kunden betreten das System (d.h. kommen am Schalter an) und verlangen nach Bedienung. Falls der Schalter belegt ist, so warten die Kunden in einem Warteraum. Wir nehmen an, dass der Warteraum eine potenziell unendliche Kapazität hat und das Kunden nach der Strategie *first come first served* (FCFS) bedient werden. Nach Beendigung der Bedienung verlassen die Kunden das System. Die Ankunftszeiten und die Bedienzeiten der Kunden seien bekannt. Sie können entweder aus Messungen an einem realen System oder aus einer stochastischen Verteilung resultieren. Dieser Punkt soll uns erst einmal nicht interessieren, da für den Ablauf der Simulation nur wichtig ist, dass die Werte bekannt sind.

Das beschriebene System stellt eine Station aus einem Warteschlangennetz dar. Warteschlangennetze sind eine weit verbreite Modellklasse zur Modellierung diskreter Systeme. Abbildung 2.1 zeigt die graphische Darstellung unseres Beispiels. Die Darstellung ist so oder ähnlich an vielen Stellen zu finden (siehe auch Abb. 1.11 oder 1.12).

Beginnen wir mit der statischen Struktur des Modells. Es gibt natürlich unterschiedliche Möglichkeiten den Systemzustand zu beschreiben. Je nachdem, welche Details relevant sind, muss eine mehr oder weniger komplexe Datenstruktur erzeugt werden. Wir wollen hier ein Minimalmodell betrachten, um die Abläufe zu verdeutlichen. Deshalb benutzen wir nur die beiden folgenden Variablen zur Modellbeschreibung:

- Die Anzahl der Kunden Q im Warteraum (als integer-Variable kodiert) und
- den Zustand des Bedieners B (der als Boolsche-Variable kodiert ist, 0 = frei, 1 = belegt).

Zusätzlich zu den beiden genannten Variablen gibt es eine Variable t, die in jeder Simulation vorhanden ist und die aktuelle Modellzeit beinhaltet. Wir nehmen an, dass die Simulation zum

Kunde	Ankunftszeit	Zwischenankunftszeit	Bedienzeit
1	0.4	0.4	2.0
2	1.6	1.2	0.7
3	2.1	0.5	0.2
4	3.8	1.7	1.1
5	4.0	0.2	3.7
6	5.6	1.6	1.5
7	5.8	0.2	0.4
8	7.2	1.4	1.1

Tabelle 2.1: Ereigniszeiten für die Simulation im Zeitintervall $[0, 10]$.

Zeitpunkt $t = 0$ beginnt, der Zustand (d.h. die Werte von Q und B) zu diesem Zeitpunkt bekannt ist und die Simulation bis zum Zeitpunkt T dauert. $Q(t)$ und $B(t)$ beschreiben den Zustand zum Zeitpunkt t. Der initiale Zustand sei $Q(0) = 0$ und $B(0) = 0$.

Die dynamische Struktur des Modells ist durch zwei Ereignisse charakterisiert, nämlich die Ankunft eines Kunden und das Bedienende eines Kunden.

- Bei der Ankunft eines Kunden:
  Wird B auf 1 gesetzt, falls der Wert 0 ist. Damit startet implizit die Bedienung. Falls B bereits 1 ist, wird Q auf $Q + 1$ gesetzt.

- Bei einem Bedienende:
  Wird B auf 0 gesetzt, falls $Q = 0$ (d.h. kein Kunde wartet), ansonsten wird Q auf $Q - 1$ gesetzt, womit implizit eine neue Bedienung startet.

Es sollte beachtet werden, dass dies nur eine Möglichkeit ist, Ereignisse zu definieren. Zusätzlich zu den Ereignisroutinen muss noch dafür gesorgt werden, dass Ereignisse zeitgerecht ausgeführt werden. Die zugehörigen Mechanismen werden etwas später eingeführt. Weiterhin muss die Simulation das Modellverhalten beobachtbar machen, damit Aussagen über das Systemverhalten möglich sind. Dies geschieht in der Regel dadurch, dass Resultatgrößen zur Auswertung definiert werden. Beispiele für Resultatgrößen sind

- P, die Anzahl der Kunden, die ihre Bedienung im Intervall $[0, T]$ beendet haben
- $\bar{Q} = \int_0^T Q(t)dt/T$, die mittlere Population in der Warteschlange,
- $Q_{\max} = \max_{0 \le t \le T} Q(t)$, die maximale Population in der Warteschlange,
- $\bar{B} = \int_0^T B(t)dt/T$, die mittlere Auslastung des Bedieners,
- $\bar{V} = \sum_{p=1}^{P} D_p/P$ die mittlere Verweilzeit von Kunden, wobei D_p die Verweilzeit des p-ten Kunden ist und
- $TH = P/T$, der Kundendurchsatz.

Diese Resultatgrößen müssen während der Simulation berechnet werden. Immer dann, wenn sich der Zustand ändert, also zu Ereigniszeitpunkten, müssen die Variablen zur Resultatberechnung modifiziert werden. Bei der nachfolgenden detaillierten Beschreibung des Simulationsablaufs wird die Resultatberechnung exemplarisch für einige Größen vorgestellt.

Tabelle 2.1 beinhaltet die Ankunfts- und Bedienzeiten für einen Beispielablauf im Intervall $[0, 10]$. Ankunftszeiten sind jeweils absolute Zeiten, während Zwischenankunftszeiten und Bedienzeiten Dauern beschreiben. Für einen beliebigen Kunden i gilt offensichtlich

- Bedienende i = Bedienanfang i + Bedienzeit i
- Bedienanfang i = max(Ankunftszeit i, Bedienende $i - 1$), wobei Bedienende $0 = 0$ gilt.

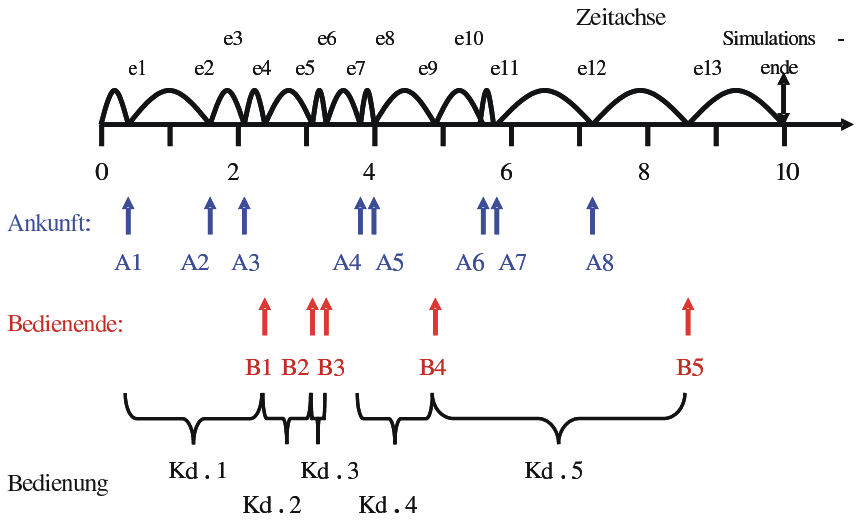


Abbildung 2.2: Ablauf der Beispielsimulation.

Die beschriebenen Ereignisse müssen in der Simulation zeitgerecht ausgeführt werden. Ein zeitgerechter Ablauf erfordert, dass zu den angegebenen Ereigniszeitpunkten die zugehörige Ereignisroutine ausgeführt wird. Abbildung 2.2 zeigt den Ablauf der Beispielsimulation. Es wird nach Ankunft und Bedienende unterschieden. Auf der Zeitachse sieht man das Springen der Simulation von Ereigniszeitpunkt zu Ereigniszeitpunkt. Die unterste Zeile fasst jeweils die zur Bedienung eines Kunden gehörenden Ereignisse zusammen.

Zur Resultatauswertung muss die Simulation beobachtet werden. Dies soll an zwei Beispielen erläutert werden, nämlich der mittleren Population in der Warteschlange und der mittleren Verweilzeit der Kunden. Zur Berechnung der mittleren Population muss $\int_0^T Q(t)dt/T$ ausgewertet werden. Da wir es mit einer ereignisdiskreten Simulation zu tun haben, bleibt $Q(t)$ zwischen den Ereigniszeitpunkten konstant (der Zustand des Systems ändert sich nicht) und springt u.U. zu Ereigniszeitpunkten auf einen anderen Wert. Abbildung 2.3 zeigt den Verlauf von $Q(t)$ für unser Beispiel. Der erste Kunde kommt nicht in den Warteraum, da er direkt bedient wird, der zweite und dritte Kunde müssen dagegen warten usw. Zur Berechnung der mittleren Population muss der Flächeninhalt der grauen Fläche durch die Länge des Beobachtungsintervalls dividiert werden. Zur Bestimmung des Integrals muss damit nur der Flächeninhalt seit der letzten Zustandsänderung in einer Variablen kumuliert werden. Sei Qt diese Variable, die mit 0 initialisiert wird. Bei Ankunft des dritten Kunden wird der Flächeninhalt des Rechtecks beginnend mit der Ankunft des zweiten Kunden berechnet und zu Qt addiert. In unserem Fall beträgt der Abstand zwischen der Ankunft des zweiten und des dritten Kunden 0.5 und die Höhe des Rechtecks ist 1, damit wird 0.5 zu Qt addiert. Beim Bedienende des ersten Kunden ändert sich $Q(t)$ wieder, so dass der zwischenzeitliche Flächeninhalt zu Qt addiert werden muss. Die Breite des neuen Rechtecks ist 0.3 $(= 2.4 - 2.1)$ und die Höhe beträgt 2, so dass 0.6 zu Qt addiert wird. Zur Ermittlung der mittleren Population muss am Ende der Simulation nur noch Qt durch $T = 10$ dividiert werden. Die Ermittlung der mittleren Verweilzeit der Kunden funktioniert anders, da es sich um ein anderes Maß handelt. Abbildung 2.4 zeigt die Messungen der Verweilzeiten. Insgesamt 5 Kunden durchlaufen das System und für jeden Kunden wird eine Verweilzeit gemessen. Der Mittelwert ergibt sich aus der Summe der Kundenverweilzeiten dividiert durch die Anzahl der Kunden, die das System durchlaufen haben. Die Verweilzeit eines Kunden ist ein kundenspezifisches Maß, grundsätzlich müsste für jeden Kunden die Ankunftszeit gespeichert werden und beim Verlassen des Systems müsste von der aktuellen Zeit die Ankunftszeit abgezogen werden, um so an die Verweilzeit zu gelangen. Dies erfordert offensichtlich eine deutlich komplexere Datenstruktur und kann nicht in einer einzelnen Variable gespeichert werden. Allgemein muss also eine Liste von Ankunftszeiten gespeichert werden und jeder Kunde muss eine Identität bekommen, damit beim Verlassen des Systems auf seine

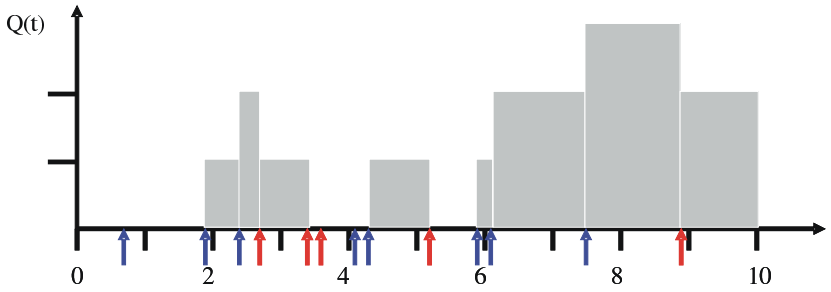


Abbildung 2.3: Verlauf von $Q(t)$ für das Beispielmodell.

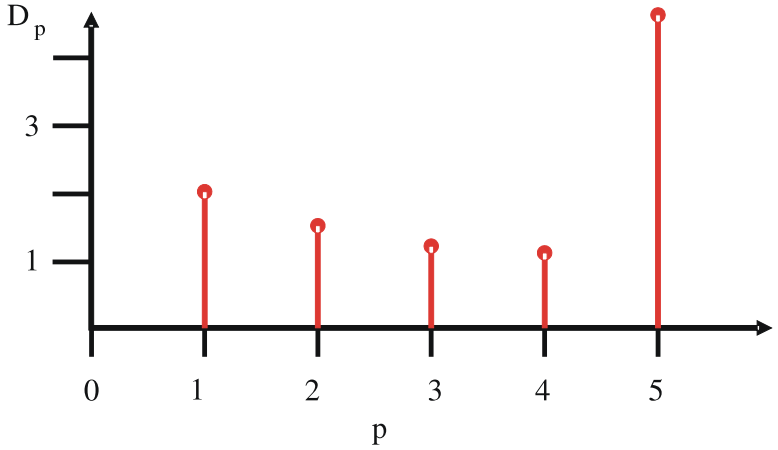


Abbildung 2.4: Kundenverweilzeiten im Beispielmodell.

Ankunftszeit zugegriffen werden kann. Da die Anzahl der Kunden im System vorab nicht bekannt ist, muss eine dynamische Datenstruktur verwendet werden. In unserem einfachen Modell können wir uns zu Nutze machen, dass Kunden in der Reihenfolge das System verlassen, in der sie es betreten haben. Dadurch reicht es aus, die Liste der Ankunftszeiten zu speichern und wenn ein Kunde das System verlässt, jeweils die erste noch nicht benutzte Ankunftszeit von der aktuellen Zeit abzuziehen, um so die Verweilzeit zu ermitteln.

2.2 Dynamischer Ablauf von Simulatoren

Die Simulation läuft dadurch ab, dass die Ereignisse nacheinander abgearbeitet werden. In unserem Beispiel sind alle Ereigniszeiten als Liste vorgegeben. Das zugehörige Vorgehen bezeichnet man auch als *trace-getriebene* Simulation. Alternativ werden die Zeiten während der Simulation aus vorgegebenen Verteilungen generiert. Damit sind die Zeiten nicht vorab bekannt, sondern werden zur Laufzeit des Simulators erzeugt. Dies kann nur in den Ereignisroutinen erfolgen, da nur an dieser Stelle während der Simulation Programmcode abgearbeitet wird. Die Ereignisroutinen haben damit die folgenden Aufgaben:

1. Modifikation des Systemzustands,
2. Speicherung der Resultatwerte und
3. Einplanung zukünftiger Ereignisse.

Der gesamte Ablauf einer Simulation wird durch die so genannte Simulationshauptroutine gesteuert. In Abbildung 2.5 wird dieser Ablauf dargestellt. Der zentrale Punkt ist eine Ausführung der

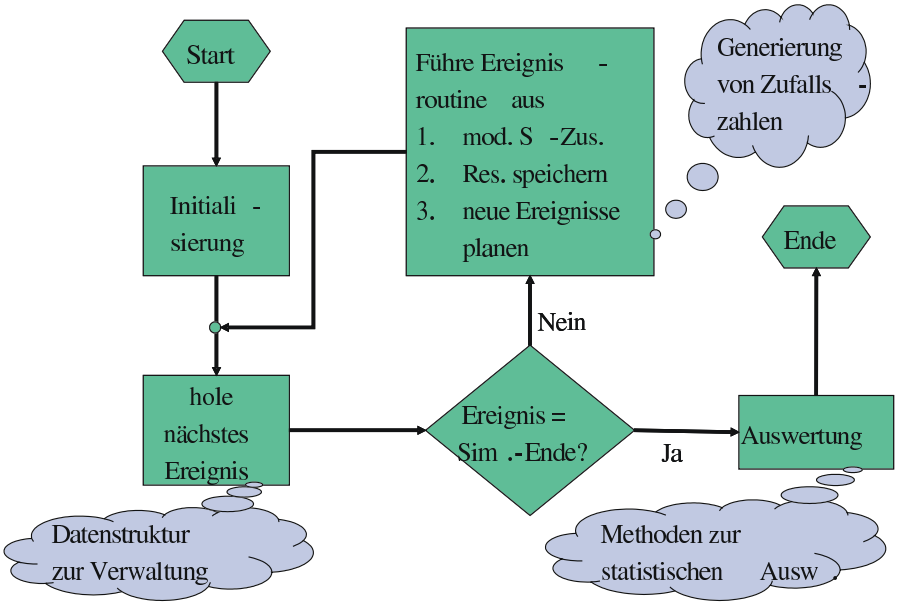


Abbildung 2.5: Ablauf der Simulationshauptroutine.

t_i	t_1	t_2	t_3	t_4	t_5	...
tp_i	tp_1	tp_2	tp_3	tp_4	tp_5	...

Abbildung 2.6: Beispiel für eine Ereignisliste (es gilt $t_i \leq t_{i+1}$).

Ereignisroutinen in der Simulationsschleife. In den Ereignisroutinen werden die drei angegebenen Aufgaben erledigt. Vor Beginn der Simulationsschleife müssen die Variablen initialisiert und nach Beendigung die Resultatgrößen ausgewertet werden. Neben der Beschreibung der Ereignisroutinen sind die drei in der Abbildung angegebenen Aspekte von Bedeutung, nämlich die Verwaltung zukünftiger Ereignisse in einer adäquaten Datenstruktur, die Generierung von Zufallszahlen und die statistische Auswertung der Simulation. Wir werden uns den beiden letzten Punkten in späteren Abschnitten widmen.

2.2.1 Ereignisverwaltung und Ereignisspeicherung

Ereignisse sind charakterisiert durch ihre Eintrittszeit t_i und einen Ereignistyp tp_i. Da sie konsekutiv nach Eintrittszeit abgearbeitet werden, müssen sie nach Eintrittszeit geordnet in einer Liste gespeichert werden. Bei Erzeugung eines neuen Ereignisses muss dieses an die entsprechende Stelle in der Liste eingeordnet werden. Auslesen des nächsten Elements bedeutet, dass das erste Element der Liste ausgelesen wird. Zusätzlich muss es noch die Möglichkeit geben Ereignisse zu löschen. Dies kommt in dem einfachen Beispiel nicht vor, ist aber bei komplexeren Modellen üblich. Als einfaches Beispiel dazu kann man einen Schalter mit zwei Kundenklassen unterschiedlicher Priorität betrachten. Ein Kunde höherer Priorität unterbricht die Bedienung eines Kunden niedrigerer Priorität. Wenn nun das Ereignis Bedienende für den Kunden niedriger Priorität eingeplant wurde und ein Ereignis Ankunft Kunde mit hoher Priorität eintritt, so wird in der Ereignisroutine für die Ankunft das Bedienende des Kunden höherer Priorität geplant und das bereits geplante Bedienende des Kunden niedriger Priorität gelöscht. Die Situation mit priorisierten Kunden mag für Bedienschalter wie Supermarktkassen oder ähnliche Systeme künstlich erscheinen, ist aber ein übliches Verhalten, wenn man den Schalter als CPU und die Kunden als Jobs interpretiert. In diesem Fall unterbricht ein Betriebssystem-Job einen Benutzer-Job.

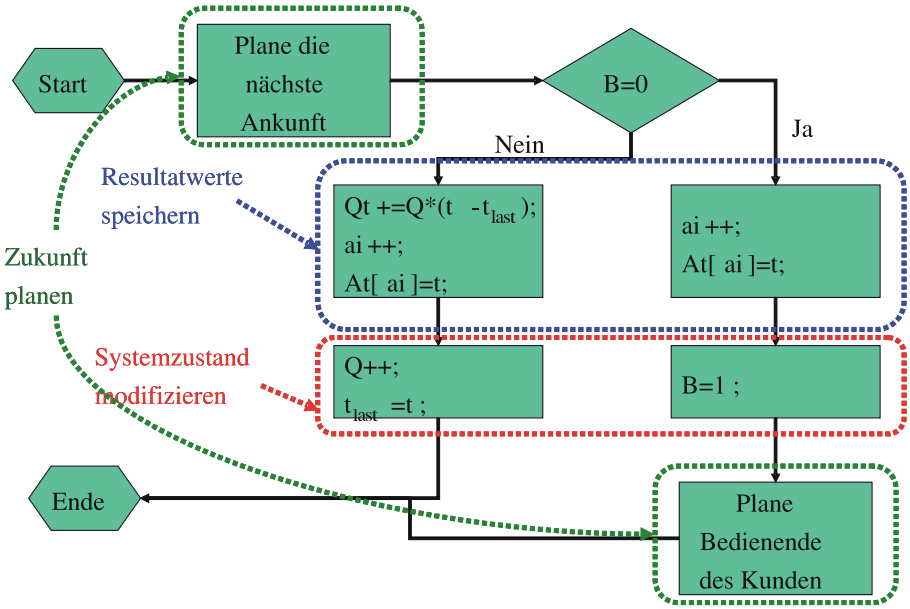


Abbildung 2.7: Ereignisroutine für die Kundenankunft.

Die Datenstruktur zur Verwaltung der Ereignisse bezeichnet man als *Ereignisliste*. Der Inhalt der Ereignisliste umfasst die bisher geplante Modellzukunft. Die aktuelle Zeit der Simulation ergibt sich aus der Zeit des nächsten Ereignisses. In der in Abbildung 2.6 gezeigten Situation wäre dies t_1. In diesem Fall können neue Ereignisse nur zu Zeitpunkten $t \geq t_1$ eingefügt werden. Die Simulation kann nur in die Zukunft planen und nicht in der Vergangenheit etwas verändern. Dies entspricht unserem realen Zeitablauf. Nicht definiert ist die Reihenfolge von Ereignissen, die zu gleicher Zeit eintreten (d.h. $t_i = t_j$). Prinzipiell ist damit eine beliebige Reihenfolge möglich. Es gibt Beispiele, bei denen eine Reihenfolge gleichzeitiger Ereignisse vom Modell vorgegeben wird. Z.B. wenn ein Ereignis ein anderes Ereignis zur gleichen Zeit initiiert, so wird das initiierte Ereignis immer später ausgeführt werden. Im allgemeinen Fall existieren solche Abhängigkeiten nicht und gleichzeitige Ereignisse können in beliebiger Reihenfolge angeordnet werden. Die Anordnung kann aber das spätere Modellverhalten verändern. Das Modellverhalten ist für gleichzeitige Ereignisse nicht eindeutig spezifiziert und ist damit indeterministisch. Dieser Indeterminismus ist vom Indeterminismus durch die bewusste Einführung von Stochastik zu unterscheiden. In praktischen Implementierungen wird oftmals eine feste Strategie der Einordnung gleichzeitiger Ereignisse gewählt (z.B. zuerst eingeordnet - zuerst in der Ereignisliste oder zuerst eingeordnet - zuletzt in der Ereignisliste).

2.2.2 Dynamischer Ablauf der Beispielsimulation

Wir wollen zu unserem Beispielmodell zurückkehren und das konkrete Modell und einen exemplarischen Ablauf beschreiben. Die folgenden Zustandsvariablen werden benutzt:

- Q Anzahl Kunden in der Warteschlange,
- B Status des Bedieners.

Weiterhin werden die folgenden Variablen zur Resultataufzeichnung und -auswertung benötigt.

- Qt der kumulierte Wert der Auftragszahl in der Warteschlange (initialer Wert 0),
- Ft der kumulierte Wert der Verweilzeiten (initialer Wert 0),
- t_{last} der Zeitpunkt der letzten Änderung von Q (initialer Wert 0),

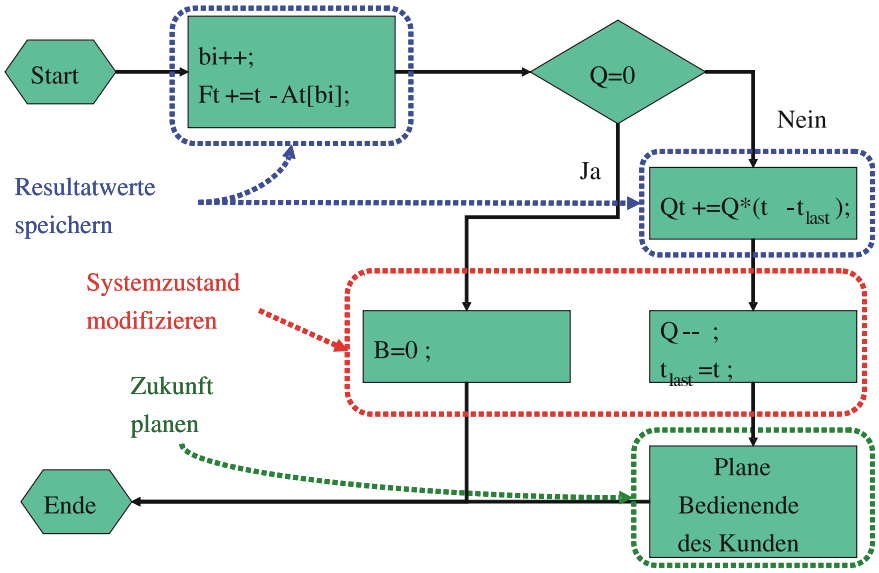


Abbildung 2.8: Ereignisroutine für das Bedienende.

- ai Nummer des letzten Auftrags, der angekommen ist (initialer Wert 0),
- bi Nummer des letzten Auftrags, der das System verlassen hat (initialer Wert 0),
- $At[i]$ Ankunftszeit des i-ten Auftrags (in der Beispielsimulation sind alle Werte vor Simulationsbeginn bekannt, in der Realität werden die Werte oft erst während der Simulation erzeugt).

Die Werte werden jeweils mit 0 initialisiert.

Darüber hinaus dient t zur Darstellung der aktuellen Simulationszeit, diese entspricht der aktuellen Ereigniszeit. Die Ereignisroutine für eine Kundenankunft wird in Abbildung 2.7 gezeigt. Die Beschreibung wird als Flussdiagramm mit der üblichen Semantik gegeben. Die Ereignisroutine umfasst die drei genannten Aufgaben: Planen der Modellzukunft, Modifizieren des Systemzustandes und Auswerten der Resultate. Die jeweiligen Komponenten sind im Flussdiagramm markiert. Das Planen der Modellzukunft besteht einfach darin, dass ein zugehöriges Ereignis in die Ereignisliste eingehängt wird. Die anderen beiden Schritte bestehen aus den Modifikationen der Variablen. In ähnlicher Form lässt sich die Ereignisroutine für das Bedienende formulieren (siehe Abbildung 2.8). Als letzte Ereignisroutine wird eine Routine für das Ereignis Simulationsende benötigt. Diese Routine wird in Abbildung 2.9 gezeigt. In der Ereignisroutine werden die Ergebnisse, nämlich die mittlere Population im Warteraum und die mittlere Verweilzeit der Kunden, ausgegeben. Die Simulation ermittelt die Mittelwerte für Population und Verweilzeit im Intervall $[0, T]$ für eine vorgegebene Sequenz von Ankunfts- und Bedienzeiten. Im allgemeinen Fall würden die Ankunfts- und/oder Bedienzeiten aus einer stochastischen Verteilung generiert und ein Durchlauf liefert eine mögliche Realisierung. Um in diesem Fall Aussagen über das Modellverhalten zu treffen (d.h. alle möglichen Realisierungen des Zufalls) ist eine statistische Auswertung notwendig, auf die in Abschnitt 5 eingegangen wird. In unserem Beispiel wird die Simulation nach einer vorgegebenen Zeit abgebrochen. Es wird also für einen festen Zeithorizont eine Simulation durchgeführt. Es gibt auch andere Abbruchbedingungen. So kann abgebrochen werden, wenn ein bestimmtes Ereignis eintritt. Dies wird dadurch realisiert, dass bei Eintreten des Ereignisses ein neues Ereignis Simulationsende zum aktuellen Zeitpunkt eingefügt wird. Weiterhin wird oftmals nach Laufzeit oder Genauigkeit der Ergebnisse abgebrochen. Im ersteren Fall wird das Ereignis Simulationsende dann eingefügt, wenn die vorgegebene CPU-Zeit durch das Simulationsprogramm verbraucht wurde, im zweiten Fall muss die statistische Auswertung mit der Simulation interagieren.

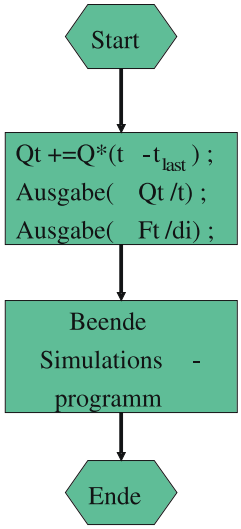


Abbildung 2.9: Ereignisroutine für das Simulationsende.

Wir betrachten nun exemplarisch den Ablauf der Simulation für die in Tabelle 2.1 gezeigten Zeiten. Zusätzlich wird das Simulationsende für den Zeitpunkt 10 und eine weitere Ankunft für den Zeitpunkt 10.2 geplant. Die Ereignisse werden als A Ankunft, B Bedienende und E Simulationsende kodiert. Der Ablauf der Simulation wird hier sehr detailliert beschrieben, da er die zentralen Elemente enthält, die in jeder Simulation vorkommen und für das Verständnis diskreter Simulation essentiell sind.

In Abbildung 2.10 werden die Initialisierung der Simulation und die ersten drei Ereignisse dargestellt. Zu Beginn der Simulation werden alle Zustandsvariablen und die Resultatvariablen initialisiert. Im Beispiel werden sämtliche Werte mit 0.0 initialisiert. Die Simulationszeit startet bei 0 und die Ereignisliste enthält die erste Ankunft, die für den Zeitpunkt 0.4 eingeplant wurde und das Simulationsende, das gleich zu Beginn für den Zeitpunkt 10 eingeplant wurde. In der Simulationshautproutine wird nun das erste Ereignis aus der Ereignisliste geholt, wodurch die Zeit t auf den Ereigniszeitpunkt 0.4 gesetzt wird. Das erste Ereignis ist eine Ankunft, die durch Ausführung der entsprechenden Ereignisroutine im Modell realisiert wird. Der zweite Block in Abbildung 2.10 zeigt den Modellzustand nach Abarbeitung der Ereignisroutine. Zum Zeitpunkt 0.4 findet der atomare Zustandswechsel in Nullzeit statt. Der Bediener ist anschließend belegt ($B = 1$), die erste Ankunft bekommt die Nummer 1 ($ai = 1$) und die Ankunftszeit ist ($At = (0.4)$). Die restlichen Variablen werden nicht verändert, da die Warteschlange weiterhin leer ist und bisher auch noch kein Kunde das System verlassen hat. In der Ereignisliste wird durch die Ankunft des ersten Kunden das Bedienende des Kunden und die Ankunft des zweiten Kunden eingeplant. Im nächsten Schritt erfolgt die Ankunft des zweiten Kunden. Da der erste Kunde noch in Bedienung ist, muss der zweite Kunde warten. Dies wird dadurch realisiert, dass Q auf 1 gesetzt wird. Weiterhin muss t_{last} auf 1.6 gesetzt werden, da sich der Zustand des Warteraums zum Zeitpunkt 1.6 geändert hat. Ansonsten sind die Änderungen analog zu den Änderungen bei Ankunft des ersten Kunden. Das nächste Ereignis ist die dritte Ankunft, die ebenfalls die entsprechenden Änderungen an den Modellvariablen durchführt. Bei der dritten Ankunft ändert sich der Zustand des Warteraums. Zur späteren Auswertung des Integrals über die Population im Warteraum muss deshalb der bisher akkumulierte Wert gespeichert werden. Dieser Wert lautet $(2.1 - 1.6) \cdot 1 = 0.5$.

Abbildung 2.11 zeigt die nächsten vier Schritte der Simulation. Das nächste Ereignis ist das Bedienende des ersten Kunden zum Zeitpunkt $t = 2.4$. Der Bediener bleibt danach belegt, da noch zwei weitere Kunden im Warteraum warten und die Bedienung des nächsten Kunden direkt beginnt. Dadurch reduziert sich die Zahl der Kunden im Warteraum auf $Q = 1$. Zur Berechnung

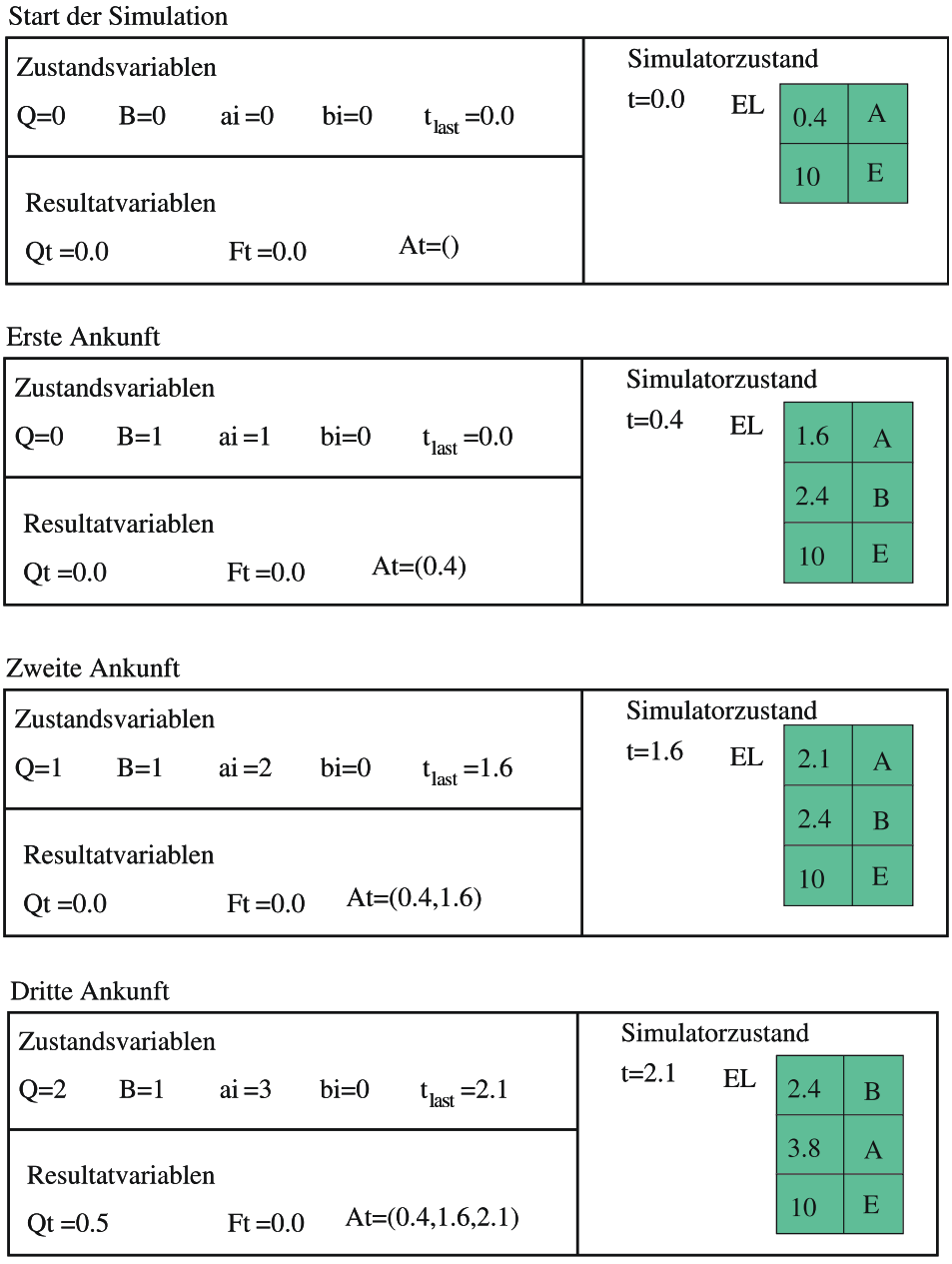


Abbildung 2.10: Erster Teil des Beispielsimulationslaufs.

Erstes Bedienende

Zustandsvariablen	Simulatorzustand
Q=1 B=1 ai =3 bi=1 t_{last} =2.4	t=2.4 EL: 3.1 B; 3.8 A; 10 E
Resultatvariablen	
Qt =1.1 Ft =2.0 At=(0.4,1.6,2.1)	

Zweites Bedienende

Zustandsvariablen	Simulatorzustand
Q=0 B=1 ai =3 bi=2 t_{last} =3.1	t=3.1 EL: 3.3 B; 3.8 A; 10 E
Resultatvariablen	
Qt =1.8 Ft =3.5 At=(0.4,1.6,2.1)	

Drittes Bedienende

Zustandsvariablen	Simulatorzustand
Q=0 B=0 ai =3 bi=3 t_{last} =3.1	t=3.3 EL: 3.8 A; 10 E
Resultatvariablen	
Qt =1.8 Ft =4.7 At=(0.4,1.6,2.1)	

Vierte Ankunft

Zustandsvariablen	Simulatorzustand
Q=0 B=1 ai =4 bi=3 t_{last} =3.8	t=3.8 EL: 4.0 A; 4.9 B; 10 E
Resultatvariablen	
Qt =1.8 Ft =4.7 At=(0.4,1.6,2.1, 3.8)	

Abbildung 2.11: Zweiter Teil des Beispielsimulationslaufs.

der mittleren Füllung des Warteraums wird $(2.4 - 2.1) \cdot 2 = 0.6$ zu Qt addiert. Da der erste Kunde das System verlässt wird $bt = 1$ gesetzt, die Verweilzeit des Kunden ergibt sich aus der aktuellen Zeit minus der Ankunftszeit des Kunden, als $t - At(bt) = 2.4 - 0.4 = 2.0$. Dieser Wert wird zu Ft addiert. Diese Art der Berechnung der Verweilzeit funktioniert nur, weil Kunden in der Reihenfolge das System verlassen, in der sie es betreten haben. Im allgemeinen Fall müsste der Kunde sich seine Ankunftszeit merken. Dies bedeutet, dass für jeden Kunden eine Variable vorhanden sein muss. Dies wird meist dadurch realisiert, dass Kunden Inkarnationen spezieller Datentypen sind. Dieser Aspekt wird später detailliert behandelt. Durch das Bedienende des ersten Kunden wird das Bedienende des zweiten Kunden planbar und für den Zeitpunkt 3.1 in der Ereignisliste vorgemerkt. Nach dem ersten Bedienende folgen zwei weitere Bedienenden, die ähnlich behandelt werden. Nach dem Bedienende des dritten Kunden ist das System leer, d.h. $Q = B = 0$. Die nächste Ankunft ist aber für den Zeitpunkt 3.8 eingeplant. Die Ankunft des vierten Kunden setzt den Wert von B wieder auf 1 und plant die fünfte Ankunft ein. Der Wert Qt ändert sich durch die fünfte Ankunft nicht, da Q im Intervall vor der Ankunft 0 war.

Die nächsten vier Ereignisse sind in Abbildung 2.12 dargestellt. Das Vorgehen entspricht dem der vorherigen Ereignisse.

Abbildung 2.13 zeigt schließlich die letzten Schritte der Simulation. Zuerst erfolgt die achte Ankunft, die gleichzeitig eine neunte Ankunft für den Zeitpunkt 10.2 einplant. Dieser Zeitpunkt liegt nach dem Ereignis Simulationsende. Dies ist aber nicht vorab feststellbar, deshalb wird das Ereignis einsortiert. Zum Zeitpunkt 10 wird das Ereignis Simulationsende ausgeführt. Dieses Ereignis sorgt dafür, dass die Simulation beendet wird und die Auswertung erfolgt. Zur Auswertung wird Ft, die Summe der Verweilzeiten aller Kunden, die das System durchlaufen haben, durch die Anzahl der Kunden, die das System verlassen haben, dividiert. Kunden, die noch im System sind, zählen nicht mit. Zur Berechnung der mittleren Warteraumbelegung wird Qt, die kumulierte Kundenzahl im Warteraum (also die in Abbildung 2.3 gezeigte Fläche) durch die Länge der Simulation dividiert. In diesem Fall zählen Kunden mit, die sich zu Simulationsende noch im Warteraum befinden. Zum Abbruch der Simulation werden die zugehörigen Daten freigegeben, weitere Ereignisse in der Ereignisliste werden also gelöscht und nicht mehr ausgeführt.

Das einfache Beispiel enthält in seiner jetzigen Codierung redundante Information, die aber zum Verständnis hilfreich ist. So gilt $Q = \max(0, ai - bi - 1)$ und $B = \delta(ai > bi)$, wobei $\delta(b) = 1$ falls b true ist und 0 sonst. Weiterhin fällt auf, dass die Ereignisliste maximal 3 Elemente enthält, da immer genau eine Ankunft und ein Bedienende voraus geplant wird. Dazu kommt noch das Simulationsende. Außerdem werden keine Ereignisse aus der Ereignisliste gelöscht. Im Beispiel könnte die Ereignisliste damit sehr einfach realisiert werden. Für komplexere Modelle kann eine Maximalzahl von Elementen in der Ereignisliste nicht garantiert werden und Funktionen zum Löschen von Ereignissen werden benötigt. Die Variable At speichert alle bisherigen Ankunftszeiten, die Länge wächst mit steigender Simulationszeit und Ankunftszahl. Man kann sich überlegen, dass eigentlich nur die Ankunftszeiten für die Kunden $bi, \ldots, ai$ benötigt werden, da die übrigen Kunden das System bereits verlassen haben. Trotzdem ist schon für das einfache Beispiel die Zahl der Werte, die gespeichert werden müssen, nicht vorab bekannt. Damit muss eine dynamische Datenstruktur zur Speicherung der Ankunftszeiten verwendet werden.

Das Vorgehen, das für das einfache Beispiel beschrieben wurde, lässt sich prinzipiell auch auf komplexere Modelle übertragen. Insbesondere ist der Zeitablauf in allen ereignisdiskreten Systemen so wie im Beispiel beschrieben. Unterschiede ergeben sich aber bei den Datenstrukturen, die zur Zustandsdarstellung benötigt werden. Insbesondere reicht es oft nicht aus, Kunden implizit durch eine integer-Variable zu kodieren. Kunden haben oft Eigenschaften, die z.B. Bedienzeiten beeinflussen und für jeden Kunden gespeichert werden müssen. Damit muss für jeden ankommenden Kunden eine neue Variable instantiiert werden. Die Population im Modell ist eine Menge von Kunden, im Simulationsprogramm dargestellt durch eine Liste von Variablen vom Typ Kunden. Verlässt ein Kunde das System, so werden seine Daten nicht mehr benötigt und der belegte Speicherplatz sollte wieder freigegeben werden, da ansonsten die Simulation mit steigender Simulationszeit immer mehr Speicher benötigt. Die dynamische Belegung und Freigabe ist ein zentraler Aspekt der Realisierung von Simulationsprogrammen.

Fünfte Ankunft

Zustandsvariablen	Simulatorzustand
Q=1 B=1 ai =5 bi=3 t_{last} =4.0	t=4.0 EL: 4.9 B; 5.6 A; 10 E
Resultatvariablen Qt =1.8 Ft =4.7 At=(0.4,1.6,2.1, 3.8,4.0)	

Viertes Bedienende

Zustandsvariablen	Simulatorzustand
Q=0 B=1 ai =5 bi=4 t_{last} =4.9	t=4.9 EL: 5.6 A; 8.6 B; 10 E
Resultatvariablen Qt =2.7 Ft =5.8 At=(0.4,1.6,2.1, 3.8,4.0)	

Sechste Ankunft

Zustandsvariablen	Simulatorzustand
Q=1 B=1 ai =6 bi=4 t_{last} =5.6	t=5.6 EL: 5.8 A; 8.6 B; 10 E
Resultatvariablen Qt =2.7 Ft =5.8 At=(0.4,1.6,2.1, 3.8,4.0,5.6)	

Siebte Ankunft

Zustandsvariablen	Simulatorzustand
Q=2 B=1 ai =7 bi=4 t_{last} =5.8	t=5.8 EL: 7.2 A; 8.6 B; 10 E
Resultatvariablen Qt =2.9 Ft =5.8 At=(0.4,1.6,2.1, 5.8,3.8,4.0, 5.6)	

Abbildung 2.12: Dritter Teil der Beispielsimulation.

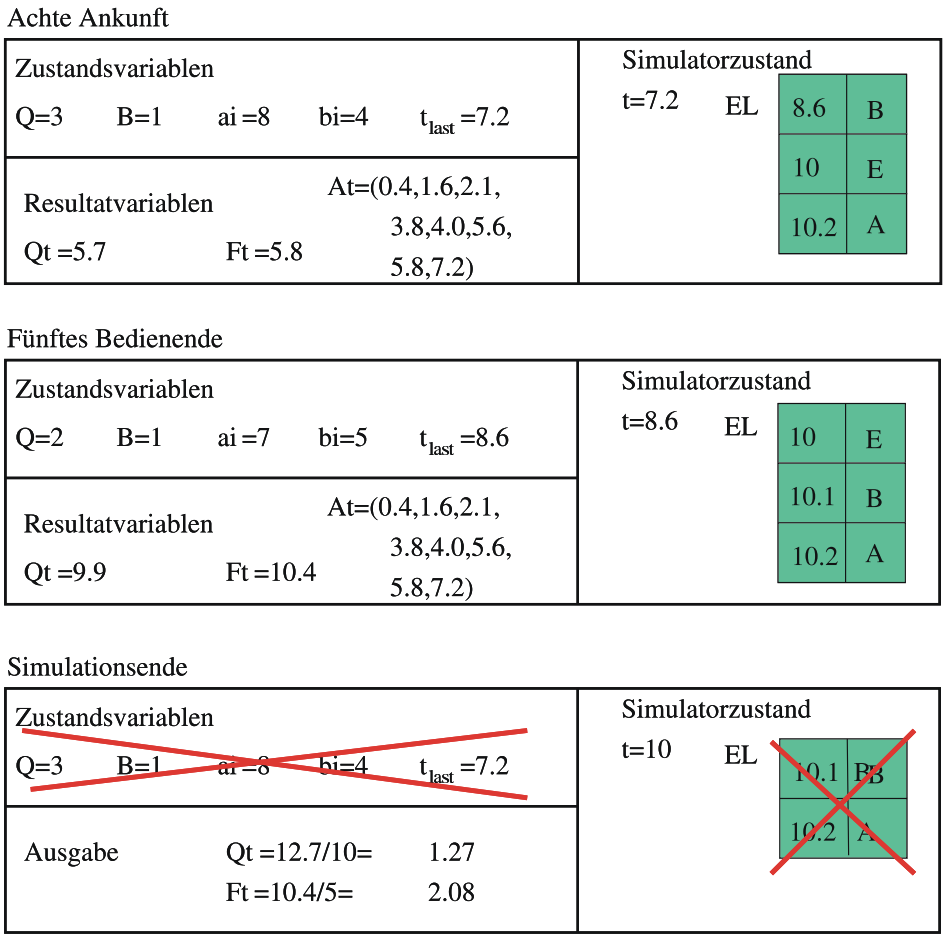


Abbildung 2.13: Letzter Teil der Beispielsimulation.

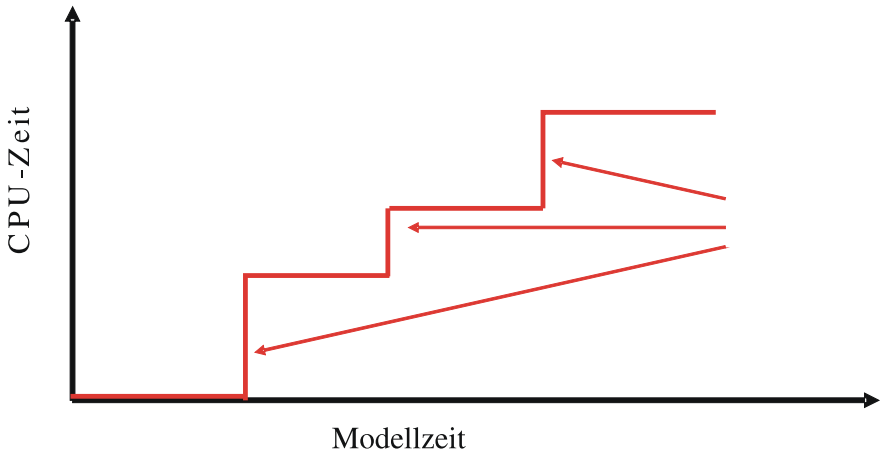


Abbildung 2.14: Zusammenhang zwischen CPU-Zeit und Modellzeit.

2.2.3 Zeitablauf in der Simulation

Betrachten wir noch einmal den Zeitablauf als zentralen Aspekt der Simulation dynamischer Systeme. Die Zeit läuft diskontinuierlich in Sprüngen ab. Der Zustand des Modells ist nur zu Ereigniszeitpunkten direkt nach Abarbeitung der Ereignisroutine definiert. Ereignisse sind atomar und verbrauchen damit keine Zeit und können auch nur vollständig ausgeführt werden. Es gibt also keinen Zustand während des Auftretens eines Ereignisses. Gleichzeitig ist der Zustand zwischen Ereigniszeitpunkten nicht vollständig definiert. So ist z.B. zwischen zwei Ereignissen der Wert einiger Variablen nicht korrekt gesetzt. Als Beispiel sei auf den Wert von Qt in dem einfachen Beispielmodell verwiesen, der nur zu Ereigniszeitpunkten angepasst wird. Wenn also der Modellzustand zu einem Zeitpunkt ermittelt werden soll, so ist für diesen Zeitpunkt ein Ereignis vorzumerken. Dies kann auch ein spezielles Beobachtungsereignis sein, das dann allerdings in der zugehörigen Ereignisroutine die Variablenwerte anpassen muss. Es ist ferner zu beachten, dass mehrere Ereignisse zu einem Zeitpunkt stattfinden können, so dass ein Modell zu einem Zeitpunkt mehrere Zustände haben kann.

Den Ablauf eines Simulators über ein Zeitintervall $[0, T]$ bezeichnet man als *Trajektorie*. Für ein Modell mit vorgegebenen Ereigniszeiten ist die Trajektorie dann eindeutig definiert, wenn keine gleichzeitigen Ereignisse vorkommen. Bei gleichzeitigen Ereignissen, die nicht kausal abhängig sind, gibt es mehrere Trajektorien. In stochastischen Simulatoren, bei den Zeiten oder Entscheidungen durch Zufallsvariablen definiert sind, können meist unendliche viele Trajektorien auftreten. Entsprechend erfordert die Auswertung stochastischer Simulationen Aussagen über alle möglichen Trajektorien auf Basis der Beobachtung einiger Trajektorien. Diese Beobachtungslage erlaubt natürlich nur statistische Aussagen.

Zentraler Aspekt jeder Simulation sind die unterschiedlichen Zeitbegriffe der Simulation. Wir unterscheiden die folgenden Zeitbegriffe.

- Die *Objektzeit* ist die Zeit, in der das reale System, zumindest hypothetisch, abläuft.
- Die *Modell-* oder *Simulationszeit* wird im Simulationsprogramm manipuliert indem Ereignisse abgearbeitet werden und damit t gesetzt wird. Die Modellzeit imitiert die Objektzeit und ist damit bis auf Translation (z.B. durch Skalierung, Start bei 0 etc.) identisch zur Objektzeit.
- Das Simulationsprogramm läuft in *Realzeit* ab. Diese Zeit ist für die Analyse natürlich belanglos.
- Zur Ausführung benötigt das Simulationsprogramme *Ausführungszeit* oder *CPU-Zeit*. Die CPU-Zeit ist neben dem Speicherbedarf der zentrale Teil des Ressourcenbedarfs. Simulationsprogramme zur Abbildung realer Systeme sind oft notorische Langläufer. In den meisten

Fällen ist auf heutigen Rechnern der CPU-Zeitbedarf ein größeres Problem als der Speicherbedarf.

Abbildung 2.14 stellt noch einmal den Zusammenhang zwischen CPU-Zeit und Modellzeit dar. Immer dann, wenn Modellzeit verstreicht, verstreicht keine CPU-Zeit und umgekehrt. Dies liegt daran, dass endliche Modellzeitintervalle zwischen Ereignissen in verschwindender CPU-Zeit übersprungen werden und verschwindende Modellzeitintervalle zu Ereigniszeitpunkten endliche CPU-Zeit benötigen, da Ereignisroutinen abgearbeitet werden. Damit ist der CPU-Zeitbedarf eines Simulators von der Anzahl der Ereignisse und der Komplexität der Ereignisroutinen und nicht allein von der Modellzeit abhängig. Die Modellzeit ist in den meisten Fällen festgelegt, da eine gewisse Anzahl Beobachtungen notwendig ist oder das System für ein vorgegebenes Zeitintervall analysiert werden soll. Eine Erhöhung der Effizienz von Simulatoren muss damit an der Ereigniszahl und an der Komplexität der einzelnen Ereignisse ansetzen. Dies bedeutet, dass am Abstraktionsniveau der Modellierung und damit meist ursächlich am Abstraktionsniveau des mentalen Modells angesetzt werden muss. Ein höheres Abstraktionsniveau führt zu weniger Ereignissen und zu weniger komplexen Ereignissen. Die Kunst der Modellbildung, die die Laufzeit des resultierenden Simulators maßgeblich beeinflusst, ist die Erstellung eines möglichst abstrakten Modells, das alle für die Ergebnisermittlung wesentlichen Systemaspekte abbildet.

2.3 Spezifikation von Simulatoren

Nachdem im letzten Abschnitt das grundsätzliche Vorgehen der ereignisdiskreten Simulation eingeführt wurde, soll in diesem Abschnitt die Umsetzung der vorgestellten Konzepte in ein Simulationsprogramm behandelt werden. Das vorgestellte Prinzip der ereignisdiskreten Simulation ist strukturiert und verständlich, die konkrete Umsetzung in ein Simulationsprogramm erfordert aber Abstraktion und ist damit komplex. Grundsätzlich ist zu beantworten, welche sprachlichen Konstrukte notwendig sind, um Simulatoren zu implementieren. Wir betrachten hier eine programmiersprachliche Realisierung und orientieren uns an gängigen Programmiersprachen. Später, in Kapitel 6, werden andere Arten der Spezifikation, wie graphische oder menübasierte Techniken, vorgestellt und spezielle Simulationssprachen eingeführt. Die abstrakteren graphischen Spezifikationsansätze dienen in den meisten Fällen als Eingabeschnittstelle für eine Simulationssprache. D.h. die graphische Spezifikation wird automatisch in ein Simulationsprogramm in einer gängige Programmiersprache oder speziellen Simulationssprache transformiert und dieses Programm wird dann für die Zielarchitektur übersetzt. Aus diesem Grund macht es Sinn, die Realisierung von Simulationsprogrammen in Programmiersprachen und die Syntax und Semantik notwendiger Konstrukte einzuführen. Darüber hinaus soll kurz erläutert werden, welche Voraussetzungen eine Programmiersprache erfüllen muss, um die Konstrukte zu realisieren.

2.3.1 Anforderungen an Programmiersprachen

Die folgenden Anforderungen an Programmiersprachen durch spezielle, simulationsspezifische Konstrukte wurden herausgearbeitet:

1. Höhere rekursive Datenstrukturen zur Speicherung von homogenen aber auch inhomogenen Variablen oder Objekte. Dazu notwendig sind Funktionen zum

   (a) Einfügen nach Schlüssel, am Anfang oder am Ende,

   (b) Herauslesen am Anfang, am Ende oder nach Schlüssel,

   (c) Löschen nach Schlüssel.

2. Die Verfügbarkeit eines Kalenders zukünftiger Ereignisse (d.h. einer Ereignisliste) geordnet nach Eintretenszeit. Funktionen zum Einfügen von Elementen (nach Schlüssel), zum Herauslesen des ersten Elements und zum Löschen eines Elements nach Schlüssel werden benötigt.

Dies ist im Prinzip ein Spezialfall von 1, der aber hier gesondert behandelt wird, da die effiziente Realisierung der Zugriffe auf die Ereignisliste von zentraler Bedeutung für die Effizienz der Simulationsprogramme ist.

3. Die Verfügbarkeit von Methoden zur dynamischen Belegung und Freigabe von Speicherplatz, um damit Variablen und Objekte zur Laufzeit zu erzeugen und zu zerstören. Diese Anforderung ist zentral, da in komplexen Simulationsprogrammen eine Vielzahl von Objekten generiert und nur temporär benötigt wird.

4. Schließlich müssen Methoden zur Generierung von Zufallszahlen vorhanden sein und statistische Methoden zur Simulationsauswertung unterstützt werden.

Im Licht der genannten Anforderungen betrachten wir gängige Programmiersprachen wie C, C++ und Java. Auch heute noch werden viele Simulationsprogramme in diesen Sprachen oder auch in Fortran realisiert (siehe auch Law and Kelton [2000, Kap. 1]), auch wenn dies aus Sicht des Software-Entwurfs nicht befriedigend ist.

Höhere Datenstrukturen, wie sie für die Simulation benötigt werden, sind heute in praktisch allen modernen Programmiersprachen vorhanden. Sie werden z.B. als *structure*, *record* oder *class* bezeichnet. Als Beispiel soll die Definition einer Datenstruktur zur Beschreibung eines Kunden mit den Attributen *Name* (als Zeichenkette kodiert), *Ankunftszeit* (als reelle Zahl kodiert) und *Priorität* (als ganze Zahl kodiert) dienen.

In C:

```
typedef struct {
     char *name ;
    float azeit ;
      int prio ;
} Kunde ;
```

In Java:

```
class Kunde {
      char[] name ;
      float azeit ;
      int prio ;
// Methodendefinitonen
}
```

Die wesentlichen Unterschiede der beiden Ansätze liegen darin, dass für Java, als objektorientierte Sprache, die Methoden zur Datenmanipulation und -ausgabe direkt mit der Datenstruktur spezifiziert werden. In C müssen Datenmanipulation und -ausgabe durch Funktionen realisiert werden, die unabhängig von der Datenstruktur spezifiziert werden. Weitere Unterschiede ergeben sich bei der Freispeicherverwaltung.

Ebenso bieten die meisten modernen Programmiersprachen die Funktionalität zur dynamischen Speicherverwaltung. Dazu sind Funktionen zur Allokation eines Speicherbereichs zur Laufzeit notwendig. Dies geschieht in der Regel durch Aufruf einer entsprechenden Funktion oder des Konstruktors eines Objekts. Damit der allokierte Speicher, wenn er nicht mehr benötigt wird, wieder benutzt werden kann, muss er freigegeben werden. Dies geschieht entweder explizit durch Aufruf einer Funktion zur Freigabe oder implizit per *garbage collection*. Letzteres bedeutet, dass von Zeit zu Zeit oder bei Speichermangel automatisch eine Funktion aufgerufen wird, die überprüft, auf welche allokierten Speicherbereiche keine Referenzen mehr existieren. Diese Überprüfung kann relativ aufwändig sein, verhindert aber, dass Speicher zu früh freigegeben wird. Grundsätzlich ist die dynamische Speicherbelegung eines der fehleranfälligsten Teile eines Programms. So kann Speicher nicht oder nicht in ausreichender Größe allokiert worden sein, der allokierte Bereich kann nicht korrekt initialisiert worden sein, die Speicherfreigabe kann zu früh oder gar nicht erfolgen. Diese Probleme sind nur sehr eingeschränkt statisch durch einen Compiler überprüfbar und führen oft zu Fehlern, die zur Laufzeit nur schwer lokalisierbar sind, da sie unter Umständen nicht lokale Effekte haben. Deshalb ist eine weitgehende Benutzerunterstützung notwendig, wie sie z.B. bei der Entwicklung moderner objekt-orientierter Programmiersprachen als grundlegende Anforderung diente. Gleichzeitig muss die Speicherverwaltung aber auch effizient realisiert sein, da sie die Effizienz des Simulationsprogramms stark beeinflusst.

Als einfaches Beispiel sollen die verfügbaren Funktionen bzw. Methoden zur Freispeicherverwaltung in C und Java kurz vorgestellt werden. Wir beginnen mit der Sprache C, die ein sehr niedriges Abstraktionsniveau besitzt. In C gibt es unter anderem die folgenden Funktionen zur Speicherallokation und -freigabe:

```
void *malloc (size_t size) ; // Allokiert size Bytes, ohne Initialisierung
void *calloc (size_t nelem, size_t elsize) ;
          // Allokiert nelem * elsize Bytes und initialisiert diese mit 0
void *realloc (void *prt, size_t size) ;
          // Der Speicherplatz, auf den prt zeigt, wird auf Länge size verlängert/verkürzt
void free (void *prt) ; // Speicherfreigabe des Bereichs, auf den prt zeigt
```

Die Zuweisung des allokierten Speichers an eine Variable erfolgt durch Umformung in den gewünschten Typ, also

```
Kunde *meine_kunden = (Kunde *) calloc(10, sizeof(Kunde)) ;
```

allokiert Speicherplatz für 10 Kunden. Der Zugriff auf den i-ten Kunden erfolgt durch `meine_kunden[i]`. Die Freigabe erfolgt durch `free(meine_kunden)`. C erlaubt eine sehr flexible Handhabung der Speicherverwaltung, da die Sprache nicht streng getypt ist und Adressarithmetik möglich ist. Gleichzeitig bedeutet dies aber auch, dass die Konstrukte extrem fehleranfällig sind und Fehler sehr schwer zu lokalisieren sind.

In Java erfolgt die Allokation durch Aufruf von new, wodurch implizit ein Konstruktor der Klasse aufgerufen wird. Es kann mehrere Konstruktoren geben, die durch unterschiedliche Parameterlisten unterschieden werden. Die folgenden Beispiele zeigen zwei mögliche Konstruktoren für den Kunden.

```
Kunde {                          Kunde (int my_prio, float my_azeit, char[] my_name) {
  prio = 1 ;                          prio = my_prio ;
 azeit = 0.0 ;                        azeit = my_azeit ;
name = ‘‘Mustermann’’ ;               name = my_name ;
} // ein Konstruktor               } // ein anderer Konstruktor
```

Konstruktoren haben jeweils den Namen der zugehörigen Klasse. C++ kann als objektorientierte Sprache ähnlich wie Java verwendet werden.

Zusammenfassend kann man sagen, dass die Freispeicherverwaltung in C sehr effizient und flexibel, aber auch extrem fehleranfällig ist. Die Freispeicherverwaltung in Java ist deutlich weniger effizient, insbesondere das garbage collection ist aufwändig, und auch weniger flexibel. Durch die vorgenommenen Einschränkungen werden viele Fehlermöglichkeiten zwar nicht beseitigt, aber doch deutlich eingeschränkt. Beispiele sind der Zugriff auf nicht allokierten Speicher oder Speicherlecks.

Da in der Simulation meistens sehr viele temporäre Objekte benötigt werden, muss auf eine effiziente Realisierung geachtet werden. Gleichzeitig sollte die Allokation und Freigabe von Speicherplatz nicht manuell erfolgen, sondern automatisch durch das System vorgenommen werden. Es sollten also Simulationssprachen oder Werkzeuge benutzt werden, die eine entsprechende Unterstützung bieten.

2.3.2 Datenstrukturen für die Ereignisliste

Als zentrale Datenstruktur der Simulation soll die Ereignisliste näher betrachtet und verschiedene Realisierungsalternativen verglichen werden. Die einfachste Form ist die Realisierung als Array fester Länge. Zwei Variablen dienen dazu das erste und das letzte Element der Ereignisliste zu referenzieren. Sei m die Länge des Arrays und n die Anzahl der Elemente in der Ereignisliste. Abbildung 2.15 zeigt die Struktur dieser Realisierung und das Anfügen von Elementen am Ende. Das Auslesen des ersten oder letzten Elements ist unproblematisch, da nur ein Zugriff erfolgt und ein Variablenwert verändert wird. Der Aufwand für das Aus- oder Einfügen des ersten oder letzten Elements liegt damit in $O(1)$. Problematischer ist das Aus- oder Einfügen eines Elements in der Liste. Durch binäres Suchen kann das gesuchte Element bzw. die Stelle an der das Element eingefügt wird in $O(\lg n)$ gefunden werden. Anschließend müssen allerdings im Mittel $n/2$ Elemente verschoben werden (siehe Abbildung 2.16). Damit beträgt der Aufwand für das Ein- und Ausfügen $O(n)$. Da man davon ausgehen kann, dass in einer Simulation ungefähr gleich viele Operationen neue Elemente einfügen und das erste Element auslesen und relativ wenige Operationen Elemente aus der Liste ausfügen, werden im Mittel die Hälfte der Operationen mit Aufwand $O(1)$ durchgeführt und der Rest in $O(n)$. Ein weiterer Nachteil der Array-Variante besteht darin, dass nicht mehr als m Elemente in der Ereignisliste sein dürfen. Da die Anzahl der Elemente in der Ereignisliste

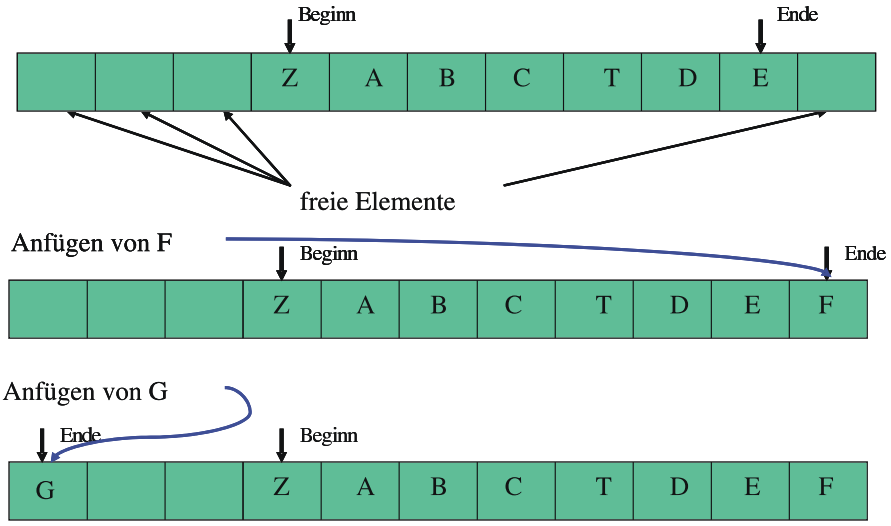


Abbildung 2.15: Ereignisliste als Array realisiert.

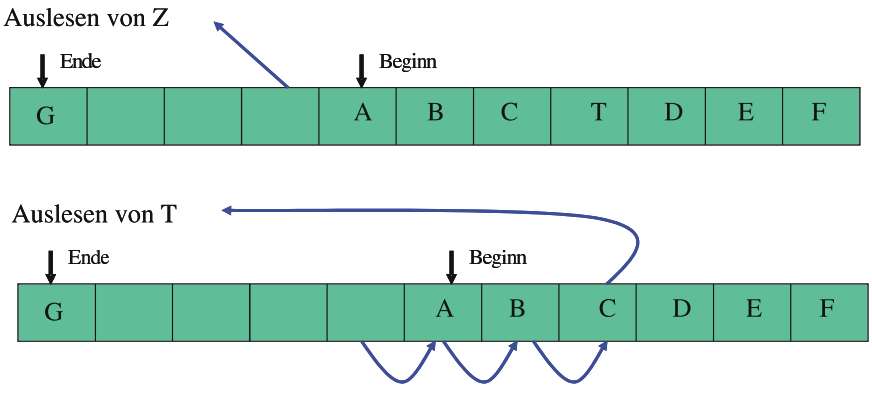


Abbildung 2.16: Ausfügen eines Elements aus dem Array.

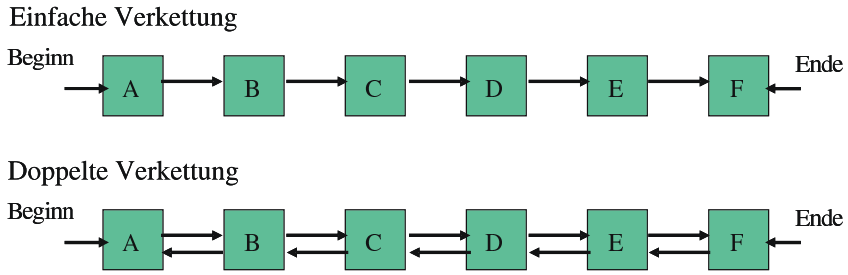


Abbildung 2.17: Ereignisliste als Listenstruktur.

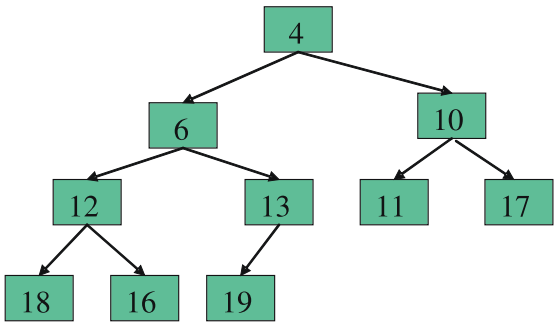


Abbildung 2.18: Ereignisliste als Heap kodiert.

aber bei den meisten Simulationsprogrammen a priori unbekannt ist, muss ein relativ großer Wert für m gewählt werden und falls $n >$m auftritt, muss entweder das Programm mit Laufzeitfehler abgebrochen werden oder das Array zur Laufzeit vergrößert werden.

Als natürliche Alternative zur einem Array fester Länge bietet sich eine Listenstruktur zur Speicherung der Ereignisliste an. Diese kann als einfach oder doppelt verkettete Liste realisiert werden (siehe Abbildung 2.17). Dabei besteht insbesondere die Möglichkeit auch inhomogene Elemente zu speichern. Dies ist für die Ereignisliste nicht notwendig, da Ereignisse in der Regel durch einen einfachen Typ referenziert werden. Die Listen werden durch einen Zeiger auf das erste und das letzte Element komplettiert. Der Aufwand für das Auslesen des ersten Elements ist wieder in $O(1)$. Das Einfügen oder Löschen eines Elements erfordert dagegen einen Aufwand in $O(n)$, da nun zwar die eigentliche Einfüge- oder Löschoperation in $O(1)$ durchgeführt wird, das Suchen aber einen linearen Durchlauf durch die Liste notwendig macht. Entscheidender Vorteil der Listenrealisierung ist der Speicherplatzbedarf von $O(n)$, so dass die Länge der Ereignisliste nur durch den physikalisch verfügbaren Speicherplatz beschränkt ist.

In der Informatik gibt es neben der linearen Liste weitere Datenstrukturen, die ein effizienteres Suchen ermöglichen. Ein Beispiel sind die so genannten Heaps. Ein Heap kann als Binärbaum dargestellt werden, dessen Wurzel das minimale Element beinhaltet und in dem kein Vorgänger kleiner als sein Nachfolger ist. Üblicherweise werden Heaps in Arrays gespeichert, was wir hier aber nicht näher betrachten wollen. Die unterste Ebene eines Heaps wird von links gefüllt. Abbildung 2.18 zeigt ein Beispiel für einen Heap, bei dem für die Elemente nur die Werte von t und nicht die Ereignistypen angegeben sind. Auf einem Heap sind die benötigten Operationen wie im Folgenden beschrieben zu realisieren.

- Entfernen der Wurzel: Der Wurzelknoten wird entfernt und der letzte Knoten der unteren Ebene wird als neue Wurzel eingesetzt. Anschließend wird die Wurzel so lange mit dem kleinsten Nachfolger vertauscht, bis kein kleinerer Nachfolger mehr existiert.

- Einfügen eines Elements: Das neue Element wird als letztes Element der untersten Ebene eingehängt und anschließend solange mit seinem Vorgänger vertauscht, bis dieser kleiner ist.

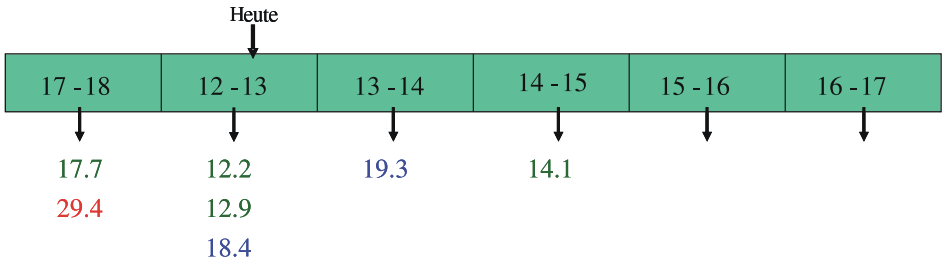


Abbildung 2.19: Ereignisliste als Kalender-Warteschlange.

Die beiden Operationen können in einem ausgeglichenen Heap mit logarithmischen Aufwand ausgeführt werden. Das Löschen eines beliebigen Schlüssels erfordert dagegen einen linearen Suchaufwand. Es zeigt sich, dass für viele Verteilungen der Ereigniszeiten der Heap mit großer Wahrscheinlichkeit fast ausgeglichen bleibt, so dass aufwändige Operationen zum Ausgleichen nur selten notwendig wären.

Als letzte Datenstruktur für die Ereignisliste soll kurz die Kalender-Warteschlange vorgestellt werden, die sich speziell als Datenstruktur für Ereignislisten eignet. Die Idee entspricht der eines Kalenders, der in Intervalle fester Größe eingeteilt ist. Ein Beispiel ist in Abbildung 2.19 zu sehen. Es liegt also eine Anzahl von Zellen fester Breite vor. Im Beispiel ist die Zeit des nächsten Ereignisses 12.2. Da die Zellenbreite 1 ist, befinden sich alle Ereignisse mit Zeitstempeln aus dem Intervall $[12, 13)$ in der aktuellen Zelle. Ereignisse aus dem Intervall $[13, 14)$ sind in der nachfolgenden Zelle usw. Da im Beispiel 6 Zellen vorhanden sind, umfasst die letzte Zelle das Intervall $[17, 18)$. Tritt nun ein Ereignis mit einem neuen Zeitstempel auf, der größer gleich 18 ist, so wird der Kalender zyklisch benutzt. Man kann dies damit vergleichen, dass in einen Kalender des aktuellen Jahres Ereignisse, die in einem der nächsten Jahre eintreten, für den entsprechenden Tag vorgemerkt werden. Formal sei $T_{\max}$ die rechte Grenze der letzten Zelle, T_{min} die linke Grenze der aktuellen Zelle und Δ die Zellenbreite, dann wird t $(\geq t_{jetzt} \geq T_{min})$ in die Zelle $\lceil((t - T_{min}) \mod (T_{max} - T_{min}))/\Delta\rceil$ eingeordnet, wobei die Zellen konsekutiv beginnend mit der aktuellen Zelle nummeriert werden. Innerhalb einer Zelle sind die Elemente linear geordnet. Wenn alle Elemente des aktuellen Intervalls aus einer Zelle ausgelesen wurden, wird solange zur nächsten Zelle übergegangen, bis das kleinste Element in der Zelle einen Zeitstempel im aktuellen Intervall hat. Beim Übergang werden die Grenzen der letzten Zelle jeweils um $T_{max} - T_{min}$ vergrößert. Im Beispiel würden also die Elemente 12.2 und 12.9 ausgelesen, dann würde zur nächsten Zelle gewechselt und dabei die Grenzen der alten Zelle auf 18 und 19 gesetzt. In der nächsten Zelle befindet sich nur das Element 19.3, das nicht zum aktuellen Intervall gehört. Deshalb werden die Grenzen auf 19 und 20 gesetzt und in der nächsten Zelle nachgeschaut. Dort befindet sich das Element 14.1, welches als nächstes Ereignis abgearbeitet wird.

Es ist einsichtig, dass die Effizienz der Zugriffsoperationen von der Menge der Elemente in einer Zelle und der Anzahl der Zellen ohne Elemente im aktuellen Intervall abhängt. Da diese von der Struktur der Ereigniszeiten abhängen, werden die Parameter der Datenstruktur dynamisch angepasst. Die Anzahl Zellen wird dynamisch angepasst, indem die Zahl verdoppelt wird, wenn die Anzahl der Elemente in der Datenstruktur doppelt so groß wie die Zahl der Zellen ist und sie wird halbiert, wenn die Anzahl der Elemente halb so groß wie die Anzahl der Zellen ist. Der Aufwand der Verdopplung oder Halbierung ist linear in der Anzahl Elemente in der Datenstruktur. Die Zellenbreite Δ sollte so gewählt werden, dass ungefähr 75% aller Einfügeoperationen im aktuellen Jahr stattfinden und damit keinen Überlauf erzeugen. Die Zellenbreite wird dann angepasst, wenn die Zellenzahl modifiziert wird.

Die gesamte Datenstruktur und die zugehörigen Operationen sind relativ komplex, so dass eine Aufwandsanalyse schwierig ist. Insbesondere macht natürlich eine worst case Analyse wenig Sinn. Zahlreiche Experimente zeigen aber, dass sich unter realistischen Bedingungen linearer Aufwand für das Auslesen des nächsten Elements und das Einfügen beliebiger Elemente erreichen lässt.

Abbildung 2.20 zeigt die Laufzeiten für ein Beispiel, bei dem zuerst n zufällige Werte in die

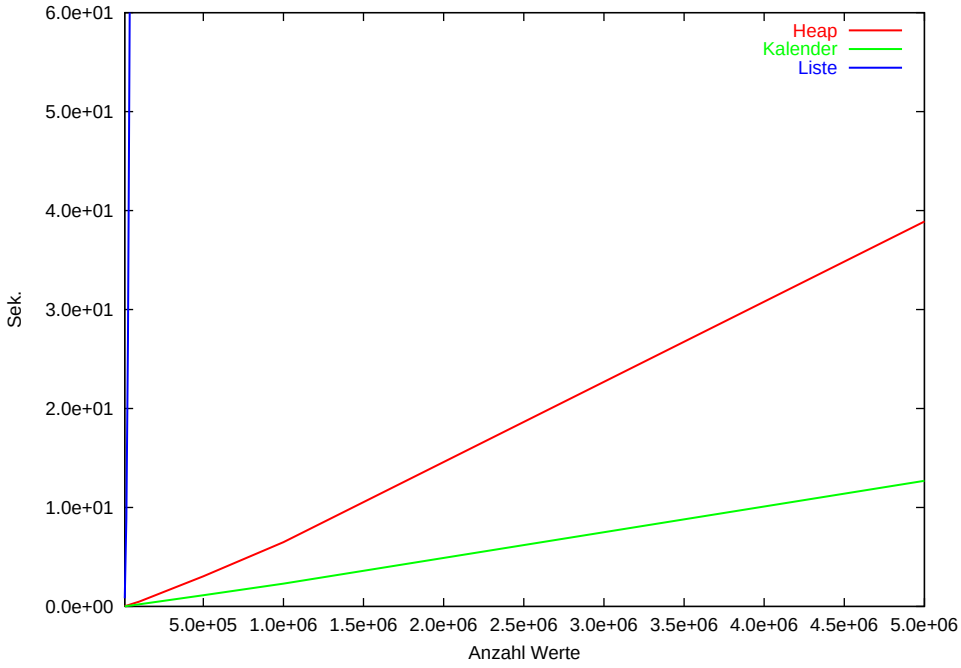


Abbildung 2.20: Laufzeiten für das Einfügen und sequentielle Auslesen von Ereignissen aus der Ereignisliste.

Datenstruktur eingefügt werden und anschließend die Werte nach ihrer Größe ausgelesen werden. Es wird deutlich, dass die lineare Liste dem Heap oder der Kalender-Warteschlange klar unterlegen ist. Gleichwohl kommen die Vorteile komplexerer Datenstrukturen erst bei relativ großen Ereigniszahlen in der Ereignisliste zum Tragen. So werden in dem Beispiel zwischen 10000 und 5 Millionen Ereignisse in der Ereignisliste gespeichert. In vielen Simulationsmodellen sind aber meist weniger als Hundert Ereignisse gleichzeitig in der Liste, so dass eine lineare Liste als Datenstruktur ausreicht. Wenn man neben dem Einlesen und sequentiellen Auslesen von Elementen auch das zufällige Auslesen bzw. Löschen von Elementen der Liste untersucht, so sind die Laufzeitunterschiede deutlich kleiner. Abbildung 2.21 zeigt die Laufzeiten, wenn die Elemente nicht mehr sequentiell, sondern in der Reihenfolge des Einfügens, unabhängig von ihrer Größe ausgelesen werden. In diesem Fall ist die lineare Liste zwar immer noch die schlechteste Wahl, der Heap ist aber besser als die Kalender-Warteschlange, die beim Ausfügen der Elemente mehrfach aufwändig umstrukturiert wird.

2.3.3 Operationen zur Realisierung von Simulationsprogrammen

Zur Formulierung der folgenden Simulationsabläufe soll eine Pseudoprogrammiersprache benutzt werden, die sich an gängige Programmiersprachen anlehnt, diesen aber nicht entspricht. Für diese Programmiersprache definieren wir jetzt einige Basisfunktionen zur Simulation, die ähnlich in vielen realen Simulationssprachen existieren. Die Funktion

```
atime = ziehe_zz (Verteilung, Parameter) ;
```

generiert ein Zufallszahl aus einer gegebenen Verteilung mit bekannten Parametern. Die konkrete Realisierung der Funktion wird in Abschnitt 3 beschrieben. Weiterhin realisiert

```
plane (Ereignistyp, Ereigniszeit) ;
```

das Einhängen eines Ereignisses vom Typ `Ereignistyp` für den Zeitpunkt `Ereigniszeit` in die Ereignisliste. Dabei spielt es keine Rolle, wie die Ereignisliste realisiert ist. Die Funktion

```
ereignis = erstes_ereignis() ;
```

liefert das nächste Ereignis aus der Ereignisliste und

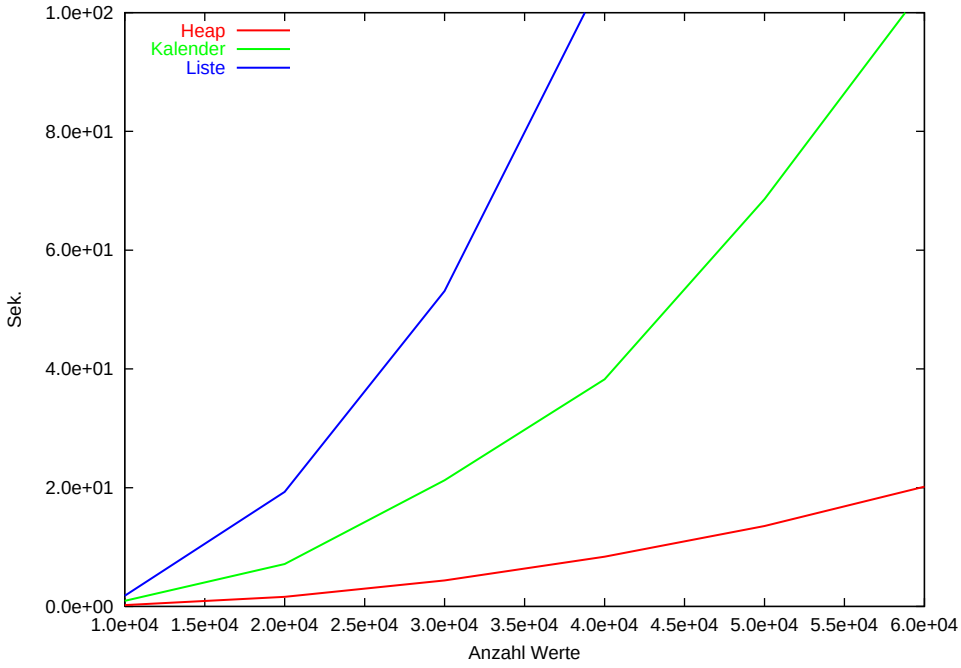


Abbildung 2.21: Laufzeiten für das Einfügen und zufällige Auslesen von Ereignissen aus der Ereignisliste.

`zeit = t ;`
liefert die Zeit des nächsten Ereignisses.

2.4 Struktur von Simulatoren

Mit den vorgestellten Konstrukten können ereignisdiskrete Simulatoren auf dem bisher verwendeten Niveau realisiert werden. Es müssen dazu

- die Datenstrukturen zur Abbildung der statischen Struktur aufgebaut werden und
- die Ereignisroutinen für alle Ereignisse definiert werden.

Den zugehörigen Ansatz bezeichnet man als *event scheduling*. Das Vorgehen beschreibt Modellverhalten auf einem sehr niedrigen Abstraktionsniveau, da

- sämtliche Ereignisse und deren (globale) Auswirkungen erkannt und kodiert werden müssen und damit
- flache aber stark vernetzte und damit unübersichtliche Modelle entstehen.

Dies bedingt, dass event scheduling für komplexe Probleme sehr aufwändig und damit fehleranfällig ist. Insbesondere entspricht die unstrukturierte Darstellung weder den Prinzipien des modernen Softwareentwurfs noch der üblicherweise verwendeten Methodik zum Entwurf komplexer technischer Systeme. In beiden Fällen wird versucht, durch die Einführung von Struktur die Komplexität beherrschbar zu machen. Damit stellt sich natürlich die Frage, ob dies nicht auch in der Simulation möglich ist.

Da die Basis der Modellierung das mentale oder semi-formale Modell ist, mit dem die Modellierung beginnt, müssen einfachere Simulationsmodelle näher am mentalen Modell sein. In unserer Vorstellung sind komplexe Zusammenhänge strukturiert, sonst wären sie nicht verständlich oder

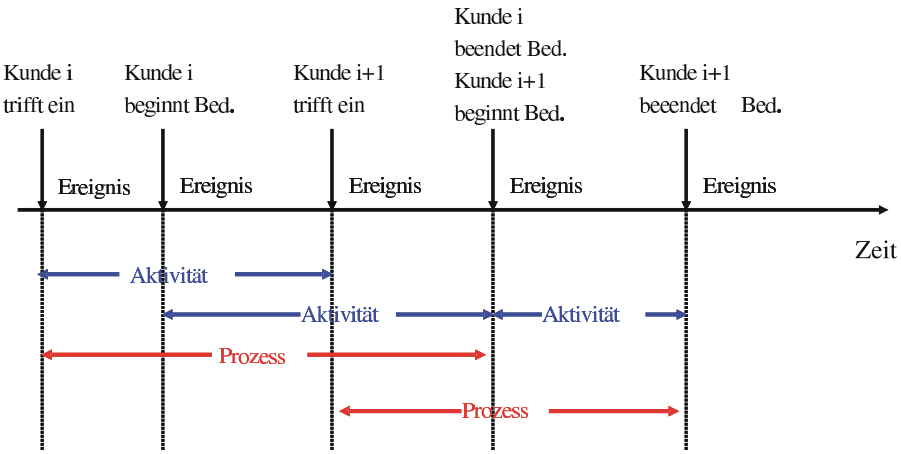


Abbildung 2.22: Unterschiedliche Sichtweisen der Zeitsteuerung.

überschaubar. Die Strukturierung wird dadurch unterstützt, dass in praktisch allen realen Systemen Ereignisse im Wesentlichen *lokale* Auswirkungen haben, die nur einzelne Zustandsvariablen betreffen. Damit besteht die Kunst einer übersichtlicheren Modellierung darin, Ereignisse mit ähnlichen lokalen Auswirkungen geeignet zusammenzufassen. Eine solche Zusammenfassung ist natürlich abhängig vom zu untersuchenden System und der Zielstellung der Modellierung, kann aber durch geeignete Strukturierungsmechanismen unterstützt werden. In der Literatur existieren zahlreiche Strukturierungsmethoden aber relativ wenig Systematik. Wir betrachten die drei folgenden Ansätze, die recht weit verbreitet sind:

- event scheduling
- activity scanning
- process interaction

Die drei Ansätze entsprechen drei Sichten auf ein System und definieren den zugehörigen konzeptuellen Rahmen der Zeitsteuerung mit steigender Abstraktion. Die Ansätze sollen an einfachen Beispielen eingeführt werden. Dazu betrachten wir zuerst die Abläufe an dem einfachen Schalter. Abbildung 2.22 zeigt einen Ausschnitt aus dem Modellverhalten. Oberhalb der Zeitachse sind die einzelnen Ereignisse eingetragen und der Ablauf in der bisher verwendeten event scheduling-Sichtweise beschrieben. Eine abstraktere Darstellung wird dadurch erreicht, dass zusammengehörige Ereignisse zu *Aktivitäten* zusammengefasst werden. Ein typische Sichtweise wäre die Definition der Aktivitäten Bedienung und Ankunft. Durch die Ankunftsaktivität trifft jeweils ein neuer Kunde ein und die Zeit zwischen dem Eintreffen von Kunden ist gerade die Zwischenankunftszeit. Die Aktivität Bedienung beschreibt den Ablauf vom Beginn der Bedienung bis zu deren Ende. Wie man in Abbildung 2.22 sehen kann, laufen die Aktivitäten Ankunft und Bedienung parallel ab. Eine höhere Abstraktionsstufe erreicht man, wenn man die Ereignisse *handelnder Individuen* zusammenfasst. Im Beispiel sollen Kunden gewählt werden. Wir betrachten also das Leben eines Kunden im Modell. Dieses ist charakterisiert durch drei Ereignisse, nämlich seine Ankunft, seinen Bedienbeginn und sein Bedienende. Man kann schon bei diesem einfachen Beispiel deutliche Unterschiede zwischen den Strukturierungsansätzen erkennen. So gibt es beim event scheduling eine endliche Menge von Ereignissen und beim activity scanning eine endliche Menge von Aktivitäten. Beim process interaction können potenziell unendlich viele Kunden gleichzeitig im System sein und diese Kunden können eine potenziell unendliche Zahl von Ereignissen ausführen. Auch wenn sich im Beispiel die Definition von Aktivitäten und Prozessen sehr natürlich ergab, ist diese keineswegs eindeutig. In einem komplexen Modell gibt es eine Vielzahl von Möglichkeiten Aktivitäten oder Prozesse zu definieren. Je nach Auswahl wird das resultierende Modell übersichtlicher oder

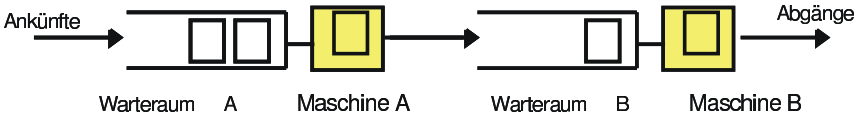


Abbildung 2.23: Beispielmodell mit zwei Stationen.

komplexer. Letztendlich ist die Strukturierung vom mentalen Modell abhängig und damit auch ein Stück weit subjektiv. Trotzdem gibt es Regeln, wie strukturiert werden sollte.

Bevor die einzelnen Modellierungsansätze an einem etwas komplexeren Modell erläutert werden, sollen die zentralen Begriffe festgelegt werden:

- Ein *Ereignis* ist die Veränderung der Zustandsvariablen zu bestimmten Zeitpunkten.
- Eine *Aktivität* ist eine Menge von Operationen während eines Zeitintervalls.
- Ein *Prozess* ist eine Folge von Aktivitäten eines Objekts über eine Zeitspanne.

Ein zentraler Freiheitsgrad gerade bei der prozessorientierten Modellierung ist die Festlegung aktiver und passiver Elemente. Man kann dies schon am einfachen Schalterbeispiel erläutern. In der bisherigen Sichtweise waren Kunden aktiv und Bediener passiv. Kunden bewegen sich zwischen Bedienern und belegen diese für ihre Bedienzeit. Man kann auch eine andere Sichtweise einnehmen und Bediener als aktiv und Kunden als passiv interpretieren. Dann übernimmt und übergibt ein Bediener einen Kunden und bedient diesen für eine Zeit. Als dritte Möglichkeit können sowohl Kunden als auch Bediener aktiv sein. In diesem Fall müssen sich beide auf eine Bedienung und deren Dauer einigen. Die am Beispiel beschriebenen Möglichkeiten bestehen in allen warteschlangenorientierten Modellen. Man bezeichnet sie als *kunden-* (bzw. *material-*) und *ressourcen-orientierte* Sichtweise. Je nach Wahl der aktiven Elemente lassen sich manche Verhaltensweisen besser modellieren und andere unter Umständen nur mit Schwierigkeiten oder gar nicht modellieren. Gleiches gilt für die Möglichkeiten der Systembeobachtung und Auswertung. Wir werden später noch einmal zu den Vor- und Nachteilen der beiden bzw. drei Ansätze zurückkommen.

Die eingeführten Konzepte der Modellbeschreibung sollen nun an einem etwas komplexeren, aber immer noch sehr elementaren Beispiel detailliert beschrieben werden. Das Beispiel (siehe Abbildung 2.23) besteht aus 2 Stationen, die als Maschinen interpretiert werden können. Werkstücke oder Aufträge (dies waren im vorherigen Beispiel die Kunden) betreten das Modell indem sie den Warteraum der ersten Maschine betreten. Nachdem sie an der ersten Maschine bearbeitet wurden, gehen sie zu Maschine 2 über, werden dort bearbeitet und verlassen danach das System. Die Aufträge werden an den Maschinen nach FCFS bearbeitet und die Puffer besitzen eine potenziell unendliche Kapazität. Der Zeitabstand T_I zwischen zwei Ankünften ist eine Zufallsvariable mit unabhängiger Verteilung. Die Bearbeitungszeiten an den Maschinen T_A und T_B seien ebenfalls Zufallsvariablen mit unabhängigen Verteilungen. Das einfache Modell kann zur Ermittlung unterschiedlicher Ergebnisse simuliert werden. Als Beispiele seien hier genannt:

- Durchlaufzeit von Aufträgen (kundenorientiert)
- Anzahl Aufträge im Warteraum (ressourcenorientiert)
- Auslastung der Maschinen (kostenorientiert)

2.4.1 Modellierung nach dem event scheduling-Ansatz

Die statische Struktur ist charakterisiert durch zwei Maschinen A und B, die als permanent existierende Objekte vorhanden sind. Der Zustand der Maschinen ist *frei* oder *belegt*. Die Warteräume vor jeder Maschine werden als FIFO-Warteschlangen (First In First Out) realisiert. Die Warteschlangen müssen Aufträge aufnehmen. Wir wollen keine detaillierte Definition eines Auftrags geben, man kann aber annehmen, dass Aufträge durch eine entsprechende Datenstruktur beschrieben sind. Im Gegensatz zum einfachen Schalterbeispiel reicht es nicht mehr aus, nur die Anzahl

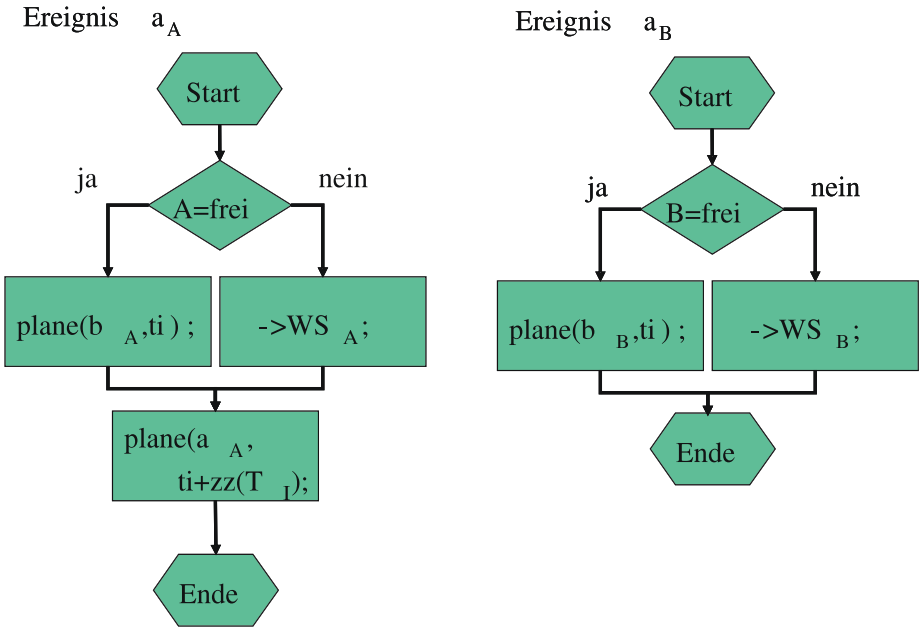


Abbildung 2.24: Ereignisroutine für die Auftragsankunft an Maschine A und B.

Aufträge im System zu zählen, sondern für jeden Auftrag wird ein neues Auftragsobjekt erzeugt. Aufträge sind damit temporäre Objekte, von denen sich potenziell unendlich viele im Modell befinden können.

Die Dynamik des Modells wird durch die folgenden Ereignisse realisiert:

1. Ankunft an Maschine A (a_A)
2. Ankunft an Maschine B (a_B)
3. Bearbeitungsbeginn an Maschine A (b_A)
4. Bearbeitungsbeginn an Maschine B (b_B)
5. Bearbeitungsende an Maschine A (e_a)
6. Bearbeitungsende an Maschine B (e_B)

Für die einzelnen Ereignisse werden die Ereignisroutinen wieder mit Hilfe von Flussdiagrammen spezifiziert. In den Diagrammen werden die folgenden Abkürzungen verwendet. ti beschreibt die aktuelle Simulationszeit, $-> WS_A$ und $WS_A ->$ beschreiben das Ein- und Ausfügen von Aufträgen in die/aus der Warteschlange vor Maschine A (analog für Maschine B) und $zz(T)$ generiert eine Zufallszahl gemäß T (T beschreibt die Verteilungsfunktion). Abbildung 2.24 zeigt die Ereignisroutinen für die Auftragsankünfte an den beiden Maschinen. Es wird jeweils zuerst getestet, ob die Maschine frei ist. Falls die Maschine frei ist, so kann direkt ein Ereignis Bedienbeginn für die aktuelle Zeit geplant werden. Bei belegter Maschine wird der ankommende Auftrag in die jeweilige Warteschlange einsortiert. In der Ankunftsroutine für Maschine A muss anschließend noch die nächste Ankunft für den Zeitpunkt $ti + zz(T_I)$ geplant werden. Eine solche Planung ist für Maschine B nicht notwendig, da dort Ankünfte durch das Bedienende an Maschine A initiiert werden.

Die restlichen Ereignisroutinen sind in Abbildung 2.25 dargestellt. Beim Bedienbeginn wird jeweils der Zustand der Maschine auf *belegt* gesetzt und anschließend das Ende der Bedienung geplant. Beim Bedienende wird der Zustand der Maschine auf frei gesetzt und dann getestet, ob die Warteschlange noch weitere Aufträge beinhaltet. Falls dies der Fall ist, so wird der nächste

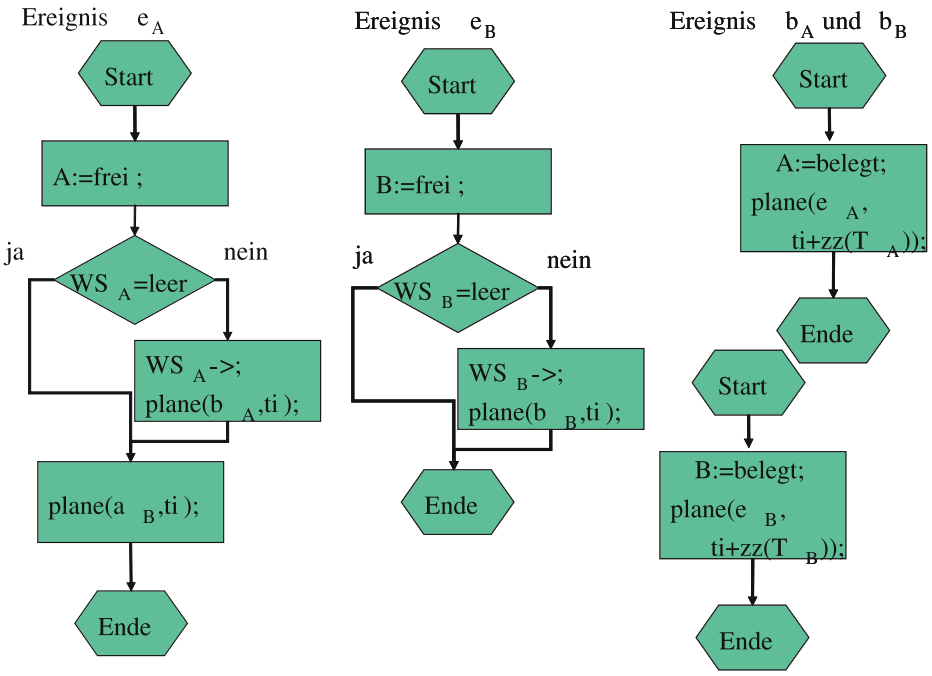


Abbildung 2.25: Restliche Ereignisroutinen für das Beispielmodell.

Auftrag aus der Warteschlange geholt und sein Bedienbeginn für den aktuellen Zeitpunkt geplant. Bei Maschine A muss zusätzlich noch die Ankunft des Auftrags, der gerade seine Bedienung beendet hat, an Maschine B für den aktuellen Zeitpunkt geplant werden. Es ist dabei zu beachten, dass der Ansatz so nur funktioniert, wenn ausgeschlossen werden kann, dass Bedienende und Ankunft gleichzeitig stattfinden können, da ansonsten zwei gleichzeitige Bedienungen initiiert werden könnten (Ablauf nachvollziehen!).

Offensichtlich beschreibt die vorgenommene Strukturierung der Ereignisse nur eine mögliche Realisierung. Es wäre genauso gut möglich, die Ereignisse Bedienbeginn an Maschine A und B mit in den Ereignissen Ankunft und Bedienende zu kodieren, wie es für den einfachen Schalter gemacht wurde.

Zum Start der Simulation müssen die Variablen initialisiert werden. Naheliegend ist es $ti := 0$ zu wählen, die Warteschlangen leer zu initialisieren und die Maschinen auf frei zu setzen. Wenn zu Beginn die erste Ankunft an Maschine A für den Zeitpunkt $zz(T_I)$ eingeplant wird, dann läuft die Simulation von da an weiter. Zur Beendigung der Simulation kann entweder ein Ereignis Simulationsende zu Beginn für einen vorgegebenen Zeitpunkt in die Ereignisliste per $plane(Simulationsende, T_{Ende})$ eingehängt oder die Simulation kann durch andere Bedingungen abgebrochen werden. Z.B. könnte nach einer bestimmten Anzahl Bedienungen abgebrochen werden. In diesem Fall müsste in der Ereignisroutine für das Bedienende an Maschine B nachgehalten werden, wie viele Aufträge bereits ihre Bedienung beendet haben und falls die gewünschte Anzahl erreicht wurde, muss für den aktuellen Zeitpunkt ein Ereignis Simulationsende eingeplant werden.

Damit die Simulation nicht spurlos abläuft, müssen Daten aufgezeichnet und am Ende ausgewertet werden. Wie bereits erwähnt, ist dies Teil der Aufgaben der Ereignisroutinen. Dieser Teil wurde aus Übersichtlichkeitsgründen in den Beispielroutinen weggelassen. Es soll kurz das Konzept zur Ergebnisermittlung skizziert werden. Man unterscheidet dabei zwischen auftrags- und stationsspezifischen Maßen. Auftragsspezifische Maße wie Verweilzeiten oder Wartezeiten müssen in der Auftragsdatenstruktur gespeichert werden. Dazu sind entsprechende Variablen vorzusehen. Dies soll kurz am Beispiel der Wartezeiten erläutert werden. Die Wartezeit jedes Auftrags soll für jeden Auftrag und beide Maschinen ermittelt werden. Dazu sind zwei reelle Variablen t_in und t_wait notwendig, die für jeden Auftrag mit 0 initialisiert werden. Bei Ankunft des Auftrags (d.h.

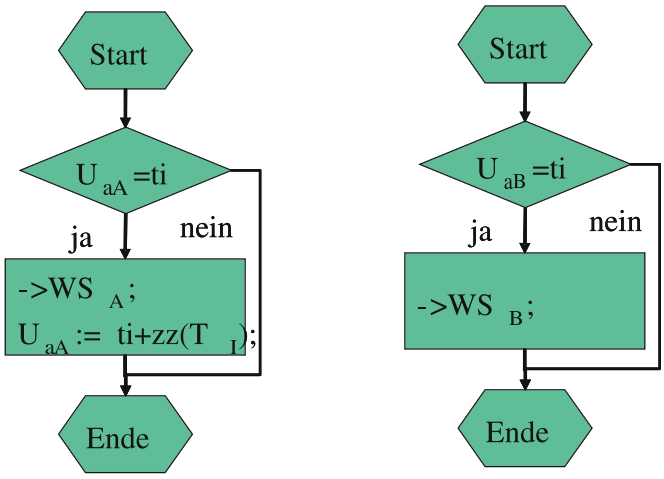


Abbildung 2.26: Aktivitätsroutinen für Auftragsankünfte.

in der zugehörigen Ereignisroutine) wird die Ankunftszeit t_in auf die aktuelle Zeit ti gesetzt. Bei Bedienbeginn wird zu t_wait die Differenz $ti - t\_in$ (d.h. die Wartezeit) addiert. Bei Bedienbeginn an Maschine B muss anschließend noch der Wert von t_wait zur Auswertung gespeichert werden. Dies geschieht durch Aufruf der entsprechenden Funktion oder Methode. Wie die einzelnen Werte gespeichert werden hängt vom Ziel der Auswertung ab (z.B. nur Mittelwertermittlung, Ermittlung von Quantilen etc.). Bei stationsspezifischen Maßen kann man davon ausgehen, dass die Station für einen spezifischen Zeitraum in einem Zustand ist (z.B. frei-belegt, mit x Aufträgen). Der Zustandswert und der Zeitraum müssen gespeichert werden. Für eine Mittelwertbestimmung kann der bereits für den einfachen Schalter beschriebene Ansatz verwendet werden.

2.4.2 Modellierung nach dem activity scanning-Ansatz

Das nächst höhere Abstraktionsniveau ist der activity scanning-Ansatz. Es wird von einer Menge von Aktivitäten ausgegangen, die im Gegensatz zu Ereignissen wissen, wann sie beginnen oder enden können. Aktivitäten entscheiden also lokal, ob sie eintreten oder nicht. In der Simulation werden nach jeder Zustandsänderung, d.h. nach Ausführung einer Aktivität, alle Aktivitäten angestoßen und jede entscheidet lokal für sich, ob sie eintreten kann oder nicht. Die Implementierungsidee besteht in einer Menge lokaler Aktivitätsuhren. Bei Aktivierung testet eine Aktivität, ob die eigene Uhr abgelaufen ist. Im Unterschied zum event scheduling, bei dem alle Folgeereignisse von einem Ereignis generiert werden müssen, werden beim activity scanning nur die unmittelbaren Folgen des Ereignisses beschrieben, da Folgeereignisse selbst testen, ob sie ausgeführt werden müssen.

Bei der Realisierung stehen die Uhren einer Aktivität auf der lokalen Aktivitätszeit und die jeweils nächste lokale Aktivitätszeit tritt ein. Wie beim event scheduling ist bei identischen Aktivitätszeiten keine Reihenfolge vorgegeben und das Modellverhalten hängt von der Reihenfolge ab, in der Aktivitäten angestoßen werden. Der Vorteil des activity scanning ist eine kompaktere Beschreibung und damit ein höheres Abstraktionsniveau. Gleichzeitig ergibt sich auch ein Nachteil, nämlich oft eine mangelnde Effizienz. In Modellen mit vielen Aktivitäten werden Aktivitäten oft angestoßen, obwohl sie nicht ausführbar sind. Im Programm bedeutet dies in der Regel jeweils einen Funktions- und Methodenaufruf, der Zeit benötigt, auch wenn der eigentliche Test auf Ausführbarkeit sehr effizient realisiert ist.

Wir betrachten nun das einfache Beispiel im activity scanning Ansatz. U_{xY} wird zur Bezeichnung der Uhr für Aktivitätsroutine x_Y verwendet. Die lokale Uhr wird in der zugehörigen Aktivitätsroutine gesetzt und gelesen. Ein globaler Prozess springt jeweils zur nächsten Uhrzeit. Ähnlich zu den Ereignissen wird das Modellverhalten durch 6 Aktivitäten und die zugehörigen Aktivitäts-

routinen beschrieben. Auch in den Aktivitätsroutinen wird die Aufzeichnung von Resultaten nicht betrachtet. Diese funktioniert genauso wie im event scheduling-Ansatz. Abbildung 2.26 zeigt die Aktivitätsroutinen für die Auftragsankunft an Maschine A und B. Die einzige Bedingung für das Eintreten ist, dass die lokale Uhr auf der aktuellen Zeit steht. Ansonsten sind die Aktivitätsroutinen recht ähnlich zu den Ereignisroutinen, was an den relativ einfachen Ereignisstrukturen liegt, die im Wesentlichen nur lokale Änderungen hervorrufen.

Abbildung 2.27 zeigt die restlichen Aktivitätsroutinen. Bei den Aktivitätsroutinen für den Bedienbeginn wurde auf die explizite Angabe der lokalen Uhren verzichtet, da das Abprüfen, ob die Maschine frei ist und die Warteschlange nicht leer ist, ausreicht. Die Synchronisation zwischen den Aktivitäten findet zum Teil dadurch statt, dass eine Aktivität die Uhr einer anderen Aktivität setzt.

Da das Problem des vielfachen Aufrufs von Aktivitätsroutinen gerade bei komplexeren Modellen auftaucht, ist man bestrebt den Ansatz effizienter zu realisieren. Eine solche effizientere Realisierung ist der folgende 3 Phasen-Ansatz, der zwischen B- und C-Aktivitäten unterscheidet. B-Aktivitäten sind solche, deren Vorbedingungen immer erfüllt sind, bei denen also nur die lokale Uhrzeit eine Rolle spielt. Im Beispiel wären dies Auftragsankunft und Bedienende. B-Aktivitäten werden in einer Liste nach Eintretenszeit geordnet gespeichert. Die restlichen Aktivitäten werden als C-Aktivitäten bezeichnet. Die Simulation läuft dann wie folgt ab:

1. Zeitfortschreibung (A-Phase)
2. Ausführung der B-Aktivitäten, deren Uhren auf der aktuellen Zeit stehen (dazu muss maximal ein Test durchgeführt werden, bei dem keine Aktivität ausgeführt wird).
3. Test und evtl. Ausführung der C-Aktivitäten.

Durch diese Unterscheidung werden B-Aktivitäten praktisch zu Ereignissen und die Unterschiede zwischen event scheduling und activity scanning verschwimmen. Im Gegensatz zum event scheduling-Ansatz, der von mehreren Simulationssprachen unterstützt wird, gibt es kaum Sprachen, die activity scanning unterstützen. Trotzdem beschreibt der Ansatz einen wichtigen Abstraktionsschritt.

2.4.3 Modellierung nach dem process interaction-Ansatz

Die bisher vorgestellte Abstraktionsebene ist für viele Anwendungen zu niedrig und entspricht auch nicht dem in der Informatik heute üblichen Abstraktionsniveau. Es gibt dort und in vielen anderen Ingenieursdisziplinen den Trend zum Denken und Strukturieren in Prozessen. Dieser Ansatz wird in der Simulation seit langem verwendet, was auch daran deutlich wird, dass die Simulationssprache Simula67 als erste objektorientierte Sprache mit einem Koroutinenkonzept vor fast 40 Jahren entwickelt wurde.

Das Strukturieren in Prozessen soll nun am Beispiel entwickelt werden. Man hat dabei zwei grundsätzliche Möglichkeiten Prozesse zu definieren. Die erste Möglichkeit ist die Sichtweise der Aufträge als Prozesse. Das Leben eines Auftrages im Modell ist gekennzeichnet durch die folgenden 6 Ereignisse.
Jeder Auftrag

- trifft an Maschine A ein und stellt sich an,
- kommt dran,
- ist fertig und geht weg,
- trifft an Maschine B ein und stellt sich an,
- kommt dran,
- ist fertig und geht weg.

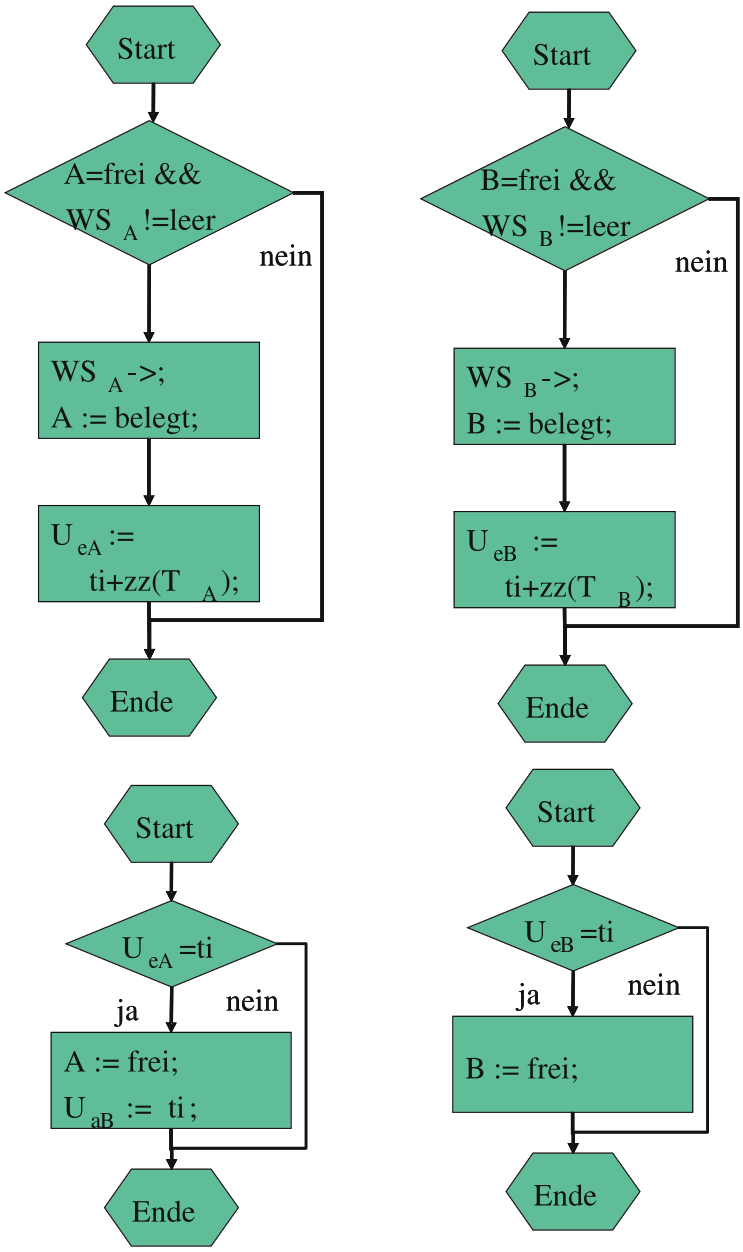


Abbildung 2.27: Restliche Aktivitätsroutinen für das Beispiel.

Damit überhaupt Aufträge in das System gelangen, müssen diese generiert werden. Dies kann einmal durch die Aufträge selbst geschehen oder durch einen separaten Prozess. Hier soll die zweite Möglichkeit betrachtet werden, auch wenn später meistens die erste Möglichkeit verwendet werden wird. Die Prozess Umwelt zur Auftragsgenerierung ist durch die folgende Sequenz realisiert

- schickt einen Auftrag in das System,
- schickt später einen weiteren Auftrag in das System,
- ...

Die Strukturierung in Aufträge ist offensichtlich nur eine Möglichkeit der Strukturierung, nämlich genau die, die dem bisherigen Ansatz entspricht, der auch beim event scheduling oder activity scanning eingesetzt wurde. Alternativ können Maschinen und nicht Aufträge als Prozesse angesehen werden. Auch in diesem Fall wird neben den beiden Prozessen für die Maschinen ein Prozess Umwelt benötigt, der die Maschinen mit Aufträgen versorgt. Der Ablauf eines Prozesses zur Modellierung einer Maschine ist durch die folgende Sequenz beschrieben. Die Maschine

- beginnt mit einer Bedienung,
- beendet diese später,
- beginnt die nächste Bedienung,
- ...

Jeder der Schritte der Maschinen und der Umwelt kann als Ereignis angesehen werden und sowohl Maschine als auch Umwelt können eine potenziell unendliche Anzahl von Ereignissen ausführen. Im Gegensatz zur Strukturierung nach Auftragsprozessen, in dem eine potenziell unendliche Zahl von Prozessen generiert und prinzipiell auch gleichzeitig im System sein kann, benötigt der zweite Ansatz nur 3 Prozesse. Aufträge sind *temporäre Prozesse*, da sie während der Simulation generiert werden und das System auch wieder verlassen. Umwelt und Maschinen sind *permanente Prozesse*, die zu Anfang generiert werden und während der gesamten Simulationszeit im System sind.

Nach dieser kurzen informellen Einführung soll der Ansatz weiter formalisiert werden. Ziel ist es, "Objekt-Leben" als Strukturierungsmittel zu nutzen, womit die Einzelereignisse Unterstukturen werden. Damit ist ein dynamisches System beschrieben durch eine Menge dynamischer Objekte (Prozesse), die sich entsprechend festgelegter Vorschriften (Prozessmuster) verhalten. Es liegt also ein *System paralleler Prozesse* vor. Dieser Ansatz liegt der Sichtweise der Informatik sehr nahe, da diese sich an verschiedenen Stellen mit parallelen Prozessen beschäftigt und diese als Strukturierungsmittel einsetzt. Die Sichtweise kann aber auch auf natürliche Weise auf viele andere Gebiete übertragen werden.

Prozesse laufen unabhängig, müssen aber von Zeit zu Zeit interagieren. Im Beispiel tritt eine solche Interaktion beim "Bearbeitetwerden" im Auftrags-Prozess und beim "Bearbeiten" im Maschinen-Prozess auf. Also *Prozesse müssen sich synchronisieren*, d.h. gewisse Aktionen gleichzeitig ausführen oder nacheinander ausführen. Um ein prozessorientiertes Modell zu beschreiben müssen die folgenden Bedingungen erfüllt sein.

1. Prozessmuster müssen notiert werden.
2. Prozesse bestimmter Muster müssen gestartet werden.
3. Prozesse müssen interagieren können.

Ein Prozess läuft nach folgendem Muster ab

- Operation(en) ausführen
- Warten (d.h. nichts tun, Zeit verstreichen lassen)
- Operation(en) ausführen

- ...

Für das Beispiel Auftrag existiert folgende Teilsequenz

- Stell dich an
- Warte bis du dran bist (1)
- Warte bis Bedienzeit abgelaufen (2)

(1) und (2) beschreiben mögliche Wartebedingungen und sie beschreiben abstrakt gesehen sogar alle möglichen Wartebedingungen:

(1) Es wird auf eine Bedingung gewartet. Der Prozess kann erst fortfahren, wenn diese Bedingung erfüllt ist.

(2) Es wird bis zu einem Zeitpunkt gewartet. Der Prozess kann fortfahren, wenn die Simulationszeit bis zum Ende der Wartezeit fortgeschritten ist.

Um diese beiden Wartebedingungen auszudrücken definieren wir die beiden folgenden Befehle für unsere Pseudoprogrammiersprache.

- wait_until (Boolsche-Bedingung)
  - Formuliert Bedingung (1).
  - Der Prozess wird inaktiv und bleibt so lange inaktiv, bis die Bedingung erfüllt ist.
  - Die Bedingung wird durch Zustandsänderungen erfüllt. Damit kann sie nur durch Ereignisse in anderen Prozessen erfüllt werden.
- pause_for (t)
  - Formuliert Bedingung (2).
  - Der Prozess wird für die Dauer t inaktiv.
  - Nach Ablauf der Zeit wird er von selbst wieder aktiv.

In beiden Fällen fährt der Prozess, nachdem er wieder aktiv wird, dort fort wo er gewartet hat, also nach der wait_until- oder pause_for-Anweisung.

Mit den beiden Anweisungen können die Prozesse spezifiziert werden. Wie bisher werden dazu Flussdiagramme verwendet. Abbildung 2.28 zeigt das Prozessmuster für einen Auftrag. Jeder Auftrag durchläuft beide Maschinen und fasst alle dazu notwendigen Ereignisse zusammen. Die Maschinen werden in diesem Ansatz durch zwei Boolsche-Variablen A und B und durch zwei Warteschlangen WS_A und WS_B realisiert. Wenn ein Auftrag an einer belegten Maschine eintrifft, so ordnet er sich selbst in die Warteschlange ein und wartet anschließend bis er der erste in der Warteschlange ist (1. WS) und die Maschine frei ist. Diese Bedingung kann nur erfüllt werden, wenn alle Aufträge vor ihm ihre Bedienung abgeschlossen haben und die Maschine nach Bedienende auf frei gesetzt haben. Das Vorrücken in der Warteschlange geschieht automatisch, was mit einer entsprechenden Datenstruktur (z.B. Liste) einfach realisiert werden kann. Um eine Bedienung auszuführen belegt ein Prozess die Maschine, pausiert für seine Bedienzeit und gibt anschließend die Maschine wieder frei. Wie schon in den vorherigen Beispielen wurden die Operationen zur Simulationsbeobachtung weggelassen. Genauso wie in den anderen Beispielen muss die Systembeobachtung als Teil des Prozessmusters notiert werden.

Abbildung 2.29 zeigt den Prozess Umwelt, der eine sehr einfache Struktur hat. Im Gegensatz zu den Auftragsprozessen gibt es im Umweltprozess kein Ende, sondern nur einen Start, da der Prozess permanent läuft. Zum Starten der Simulation wird ein Umwelt Prozess generiert, der dann die Auftragsprozesse generiert, die sich mit Hilfe der Boolschen-Variablen und der Warteschlangen an den Maschinen synchronisieren.

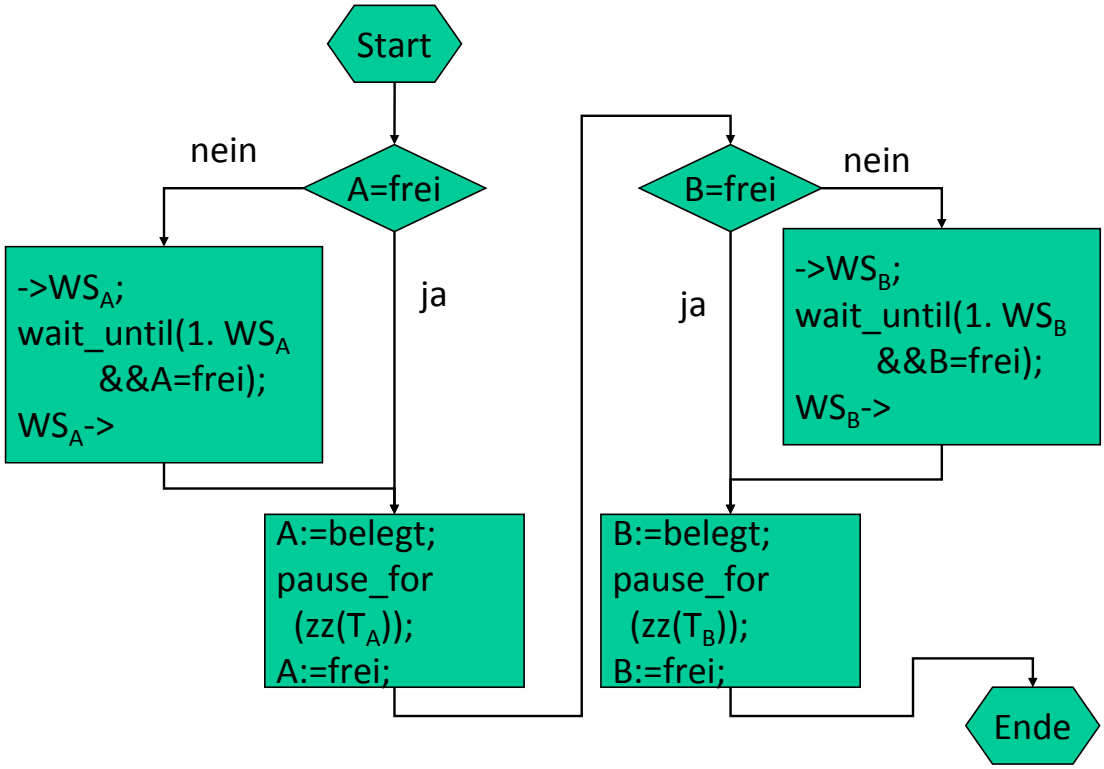


Abbildung 2.28: Prozessmuster eines Auftrags.

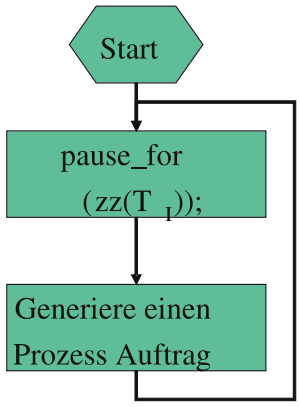


Abbildung 2.29: Umweltprozess für die auftragsorientierte Modellierung.

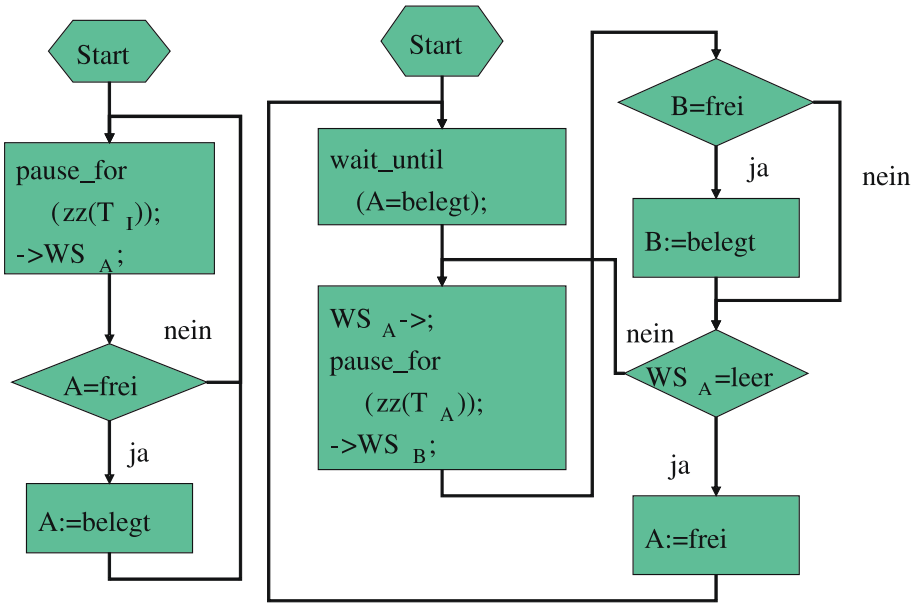


Abbildung 2.30: Umweltprozess und Prozess für Maschine A in der maschinenorientierten Sicht.

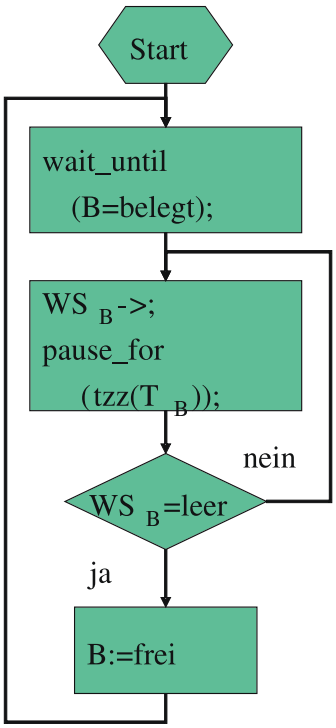


Abbildung 2.31: Prozess für Maschine B.

Nach der Darstellung des Beispiels als prozessorientiertes Modell mit aktiven Aufträgen, soll nun als Alternative die prozessorientierte Sichtweise mit aktiven Maschinen vorgestellt werden. Die Abbildungen 2.30 und 2.31 zeigen die drei benötigten Prozesse zur Modellierung der Umwelt und der beiden Maschinen. Alle Prozesse laufen in dieser Darstellung permanent. Aufträge werden durch den Umweltprozess erzeugt und in den Puffer der ersten Maschine abgelegt. Falls die erste Maschine frei ist, wird sie durch den Umweltprozess bei Ankunft eines Kunden dadurch aktiviert, dass sie belegt wird. Dadurch wird die wait_until Bedingung der Maschine erfüllt und die Bedienung beginnt. Falls ein Auftrag bedient wurde und noch andere in der Warteschlange vorhanden sind, so beginnt sofort die nächste Bedienung. Die erste Maschine übergibt Aufträge der zweiten Maschine und aktiviert diese, falls nötig. Durch die Erzeugung der drei Prozesse läuft die Simulation an, da der Umwelt-Prozess die Aufträge nacheinander generiert. Aufträge tauchen in dieser Modellierung nur implizit auf, könnten aber durchaus als komplexere Datenstruktur zur Speicherung von Parametern dargestellt werden. Die Warteschlangen vor den Maschinen müssen nur die entsprechenden Datenstrukturen aufnehmen können. Wie schon in den vorangegangenen Beschreibungen wurden die Teile zur Resultatauswertung in den Beschreibungen weggelassen.

2.4.3.1 Bewertung der material- und maschinenorientierter Sicht

Grundsätzlich können Modelle material- oder maschinenorientiert spezifiziert werden, die Art der Beschreibung hat aber Auswirkungen auf die Ausdrucksmöglichkeiten. In einer maschinenorientierten Sicht können die folgenden Aspekte auf sehr natürliche Weise beschrieben werden:

- Unterschiedliche Scheduling-Strategien, bei denen die Maschine Aufträge nach ihren Eigenschaften auswählt. So könnte der Auftrag mit der geringsten Bearbeitungszeit immer zuerst gewählt werden. Diese Strategie nennt shortest job first (SJF). Aufträge könnten in Klassen eingeteilt werden und jede Klasse eine bestimmte Priorität bekommen, wobei Aufträge höherer Priorität vor Aufträgen niedriger Priorität bedient werden. Die Implementierung der Scheduling-Strategie im Auftragsprozess würde sehr komplex, da sich alle auf eine Maschine wartenden Aufträge synchronisieren müssten.
- Maschinenorientierte Leistungsmaße sind im Maschinenprozess ebenfalls auf natürlich Weise auswertbar. Beispiele für solche Leistungsgrößen sind die Auslastung der Maschine oder die mittlere Population im Puffer.
- In der Realität fallen Maschinen aus oder bringen auf Grund von temporären Fehlern kurzzeitig eine geringere Leistung. Auch dieses Verhalten ist im Maschinenprozess einfach beschreibbar, kann aber nur sehr aufwändig und unnatürlich im Auftragsprozess beschrieben werden.

Für andere Abläufe ist die materialorientierte Sichtweise die natürliche Beschreibungsform.

- Wenn Aufträge mehrere Maschinen gleichzeitig belegen, so kann dies im Auftrag einfach spezifiziert werden, würde aber auf Maschinenebene eine komplexe Synchronisation erfordern.
- Materialorientierte Leistungsmaße wie Durchlaufzeit sind auf Ebene der Maschine kaum analysierbar, können aber im Auftragsprozess einfach akkumuliert werden.
- Wenn Material synchronisiert wird, so kann dies auf Maschinenebene nur in einfachen Fällen beschrieben werden. Komplexere Synchronisationen erfordern Prozesse für die einzelnen Materialien.

Es zeigt sich deutlich, dass je nach Modellstruktur die eine oder die andere Sichtweise vorzuziehen ist. In vielen komplexen Modellen treten Strukturen auf, die beides, eine material- und eine maschinenorientierte Sichtweise erfordern. Prinzipiell ist eine solche Beschreibung möglich und wird auch von Simulationswerkzeugen unterstützt. Es muss nur darauf geachtet werden, dass die Beschreibung konsistent bleibt. So kann die Bedienzeit vom zu Bedienenden und auch vom Bediener festgelegt werden. Es ist aber nicht möglich, dass beide jeweils eine eigene Bedienzeit festlegen, da der Bediener genau so lange belegt sein muss, wie der zu Bedienende bedient wird.

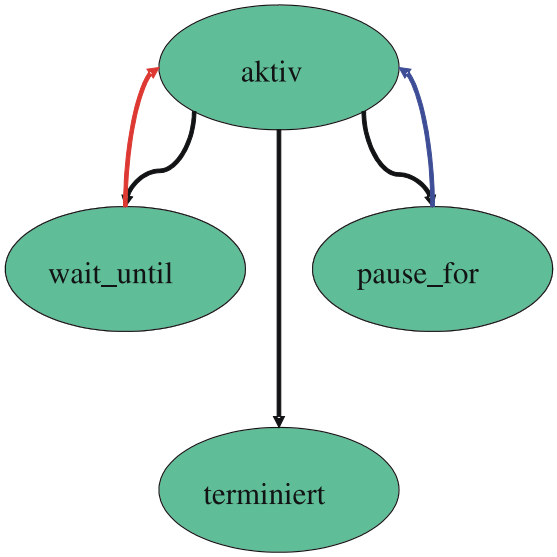


Abbildung 2.32: Prozesszustandsdiagramm der prozessorientierten Simulation.

2.4.4 Realisierung des process interaction-Ansatzes

Der generelle Ablauf prozessorientierter Simulationen und deren Realisierung soll nun näher erläutert werden. Die einzelnen Prozesse können in vier verschiedenen Zuständen sein, wie in Abbildung 2.32 gezeigt wird. Prozesse wechseln zwischen aktiv und einer der beiden Wartebedingungen wait_until oder pause_for. Schließlich terminiert ein Prozess und kann danach nicht wieder aktiv werden. In einem Simulationsmodell können mehrere Prozesse gleichzeitig im Zustand aktiv sein. Da wir davon ausgehen, dass unsere Simulation auf einem Einzelprozessor abläuft, kann von den aktiven Prozessen nur einer zu einer Zeit wirklich bearbeitet werden. Die zeitgerechte Ausführung zeitgleich aktiver Prozesse ist ein zentraler Aspekt der prozessorientierten Simulation. Bevor dieser Punkt näher untersucht wird soll die Frage beantwortet werden, wie Prozesse in einer Programmiersprache realisiert werden. Nahe liegend wäre eine Beschreibung als Prozeduren und Funktionen, wie sie in allen Programmiersprachen vorhanden sind. Prozessmuster und -abläufe lassen sich in Prozeduren grundsätzlich beschreiben. Wenn man allerdings etwas näher darüber nachdenkt, so wird schnell deutlich, dass Prozeduren nicht ausreichen. Das Problem wird in Abbildung 2.33 verdeutlicht. Eine Prozedur würde beim Eintreten einer Wartebedingung verlassen und beim nächsten Aufruf, d.h. beim nächsten Wechsel in den Zustand aktiv, wieder von vorn beginnen. Dies ist offensichtlich nicht das gewünschte Verhalten, sondern bei der nächsten Aktivierung soll nach der Wartebedingung weitergemacht werden. Dieses Verhalten wird durch Koroutinen oder Threads beschrieben, die in manchen Programmiersprachen vorhanden sind. So besitzt Simula67 ein Koroutinenkonzept und Java ein Thread-Konzept. C oder C++ hat standardmäßig kein solches Konzept, es gibt allerdings Zusatzbibliotheken zur Realisierung von Threads und/oder Koroutinen.

Kommen wir nun zur Ausführung gleichzeitiger Ereignisse zurück. Beim event scheduling-Ansatz wurde davon ausgegangen, dass gleichzeitige Ereignisse in beliebiger Reihenfolge ausgeführt werden können. Dies ist im process interaction-Ansatz nicht möglich, wie das folgende Beispiel zeigt. Wir betrachten dazu wieder einen Schalter und einen Kunden. Der Schalter besitzt eine Warteschlange *WS* für Kunden, eine Boolsche Variable belegt und eine Referenz *Kd* auf einen Kunden, der gerade bedient wird. Der Kunde besitzt eine Boolsche Variable *dran*, eine Referenz *Sc* auf den Schalter und eine Referenz *Current* auf sich selbst. Schalter und Kundenprozess sind wie folgt realisiert.

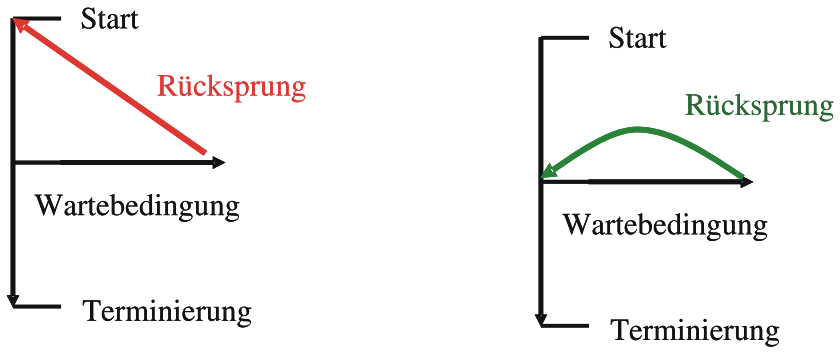


Abbildung 2.33: Verhalten von Prozeduren und Koroutinen.

```
Schalter                                        Kunde
Repeat
    wait_until(WS nicht leer) ;          (S1)   dran := false ;                  (K1)
    Repeat                                      Current->Sc.Ws ;                (K2)
        Kd := WS.first ;                 (S2)   wait_until (dran) ;             (K3)
        belegt := true ;                 (S3)   pause_for (Bedienzeit) ;        (K4)
        WS.first.dran := true ;          (S4)   Sc.belegt := false ;            (K5)
        wait_until (not belegt) ;        (S5)   Sc.WS-> ;                       (K6)
    Until WS leer ;                      (S6)
Until (Simulationsende) ;
```

Die folgende Festlegung der Semantik würde dem bisherigen Vorgehen im event scheduling-Ansatz entsprechen.

- Die Reihenfolge der Ereignisse eines Prozesses muss beachtet werden.
- Zeitgleiche Ereignisse in unterschiedlichen Prozessen können beliebig angeordnet werden.

Ein möglicher Ablauf, der mit der Ankunft des ersten Kunden startet wäre der folgende:

K1, K2, S1, S2, S3, S4, S5, K3, K4, K5, S6, S2, S3, S4, S5, K6

Nach Abarbeitung dieser Schritte hätte der Kunde das System verlassen, wäre also terminiert. Der Schalter hätte eine Referenz auf den Kunden, der das System gerade verlassen hat und würde nun warten, dass der Kunde seine Bedienung beendet. Wenn nun nachfolgende Kunden eintreffen, würden sie sich selbst in die Warteschlange einordnen und warten, dass der Schalter ihren Bedienbeginn initiiert. Dies würde aber nicht passieren, da der Schalter auf das Bedienende des Kunden wartet, der bereits weggegangen ist. Offensichtlich liegt eine klassische Deadlock-Situation vor.

Was ist bei dem Ablauf falsch gelaufen? Der Grund liegt darin, dass die Atomizität der Ereignisse nicht beachtet wurde. Wie im event scheduling festgelegt wurde, müssen Ereignisse ganz oder gar nicht ausgeführt werden. Nun existieren im process interaction keine Ereignisse im ursprünglichen Sinne. Trotzdem sind unter Umständen mehrere Zuweisungen notwendig, um den Systemzustand zu ändern. Der Zustand ist undefiniert oder nicht vollständig definiert, solange nicht alle notwendigen Zuweisungen ausgeführt wurden. Im obigen Ablauf hatte der Kunde K6 noch nicht ausgeführt und der Zustand war nicht definiert, als der Schalter S2 ausführte.

Um solche Probleme zu vermeiden, werden in allen prozessorientierten Programmiersprachen die folgenden Vereinbarungen getroffen:

- Ein Prozess läuft solange, bis er eine hemmende Bedingung erreicht oder terminiert.
- Die Auswahl des nächsten Prozesses aus einer Menge aktiver Prozesse bleibt willkürlich und kann das Verhalten des Modells und damit die Semantik beeinflussen.

Kommen wir nun zur Realisierung der beiden Wartebedingungen:

- pause_for (t)
  - Der Prozess pausiert für t Zeiteinheiten und wird danach wieder aktiv.
  - Das Ereignis der Aktivierung kann damit für den Zeitpunkt $ti + t$ eingeplant werden.
  - Die Realisierung erfolgt über eine Ereignisliste, die alle Prozesse enthält, die pausieren.
- wait_until (b)
  - Bedingung b kann über beliebige Zustandsvariablen formuliert werden.
  - Damit kann jede Wertzuweisung jedes wait_until eines Wartenden ändern.
  - Damit muss nach jedem Anhalten eines Prozesses jede Wartebedingung überprüft werden.
  - Dies ist extrem aufwändig und führt zu ineffizienten Simulatoren.

Aus Effizienzgründen ist wait_until in allgemeiner Form in kaum einer Simulationssprache oder einem Simulationswerkzeug realisiert. Trotzdem wird neben dem reinen Warten für einen Zeitraum, ein Warten auf Bedingungen benötigt, da sonst keine wirkliche Synchronisation zwischen Prozessen möglich ist. Es gibt zwei unterschiedliche Möglichkeiten das ineffiziente wait_until zu ersetzen.

Man kann näher zur Implementierung gehen und damit weiter weg vom mentalen Modell. In diesem Fall werden zwei Anweisungen *passivate* und *activate* eingeführt. Der Aufruf von passivate sorgt davor, dass der aufrufende Prozess anhält und damit praktisch in den Zustand von wait_until gerät. Typischerweise wird erst die Bedingung b getestet und falls diese nicht gilt, wird der Prozess passivate aufrufen.
if not b then passivate ;
Im Gegensatz zu wait_until wird der Prozess nicht automatisch aufgeweckt, wenn b gilt. Das Aufwecken muss explizit durch den Aufruf von activate(P) für einen wartenden Prozess P erfolgen. Der Aufruf kann natürlich nur durch einen anderen Prozess erfolgen, idealerweise durch den Prozess, der b erfüllt hat. Diese Art der Beschreibung sorgt für effiziente Simulatoren, bedingt aber auch, dass der Simulator-Code länger und komplexer wird. So muss der aufweckende Prozess eine Referenz auf den aufzuweckenden Prozess besitzen und der eigentlich Vorteil der prozessorientierten Simulation, nämlich die weitgehende Unabhängigkeit und lokale Beschreibung von Prozessen wird teilweise eingeschränkt. Trotzdem sind activate und passivate in vielen Simulationssprachen, z.B. in Simula67 verfügbar.

Die zweite Möglichkeit orientiert sich mehr am mentalen Modell und ist damit weiter weg von der Implementierung. Das Warten auf Bedingungen wird dahingehend eingeschränkt, dass b nicht mehr eine beliebige Bedingung sein darf, sondern nur auf bestimmte Bedingungen gewartet werden kann. Dieser Ansatz ist insbesondere für Szenario-Sprachen von Interesse. So ist in Warteschlangennetzen das Warten auf Bediener üblich. Falls es nur eine endliche Anzahl möglicher Bedingungen gibt, auf die gewartet werden kann, so wird für jede Bedingung eine Warteschlange gebildet, in die Prozesse, die auf die zugehörige Bedingung warten, eingeordnet werden. Wird nun eine Bedingung erfüllt, so braucht nur in der zugehörigen Warteschlange nach Prozessen zur Aktivierung gesucht werden. Dieses Konzept ist effizient und stellt keine zusätzlichen Anforderungen an den Modellierer. Es ist allerdings eine Einschränkung gegenüber dem allgemeinen wait_until.

Kapitel 3

Generierung und Bewertung von Zufallszahlen

In der diskreten Simulation spielt die Stochastik eine große Rolle. Es gibt nur wenige Modelle, in denen keine Stochastik auftritt. Man kann sicherlich lange und intensiv darüber diskutieren, ob Zufall oder Stochastik Teil unserer Umwelt ist oder ob es nur ein künstliches Phänomen ist, das vom Menschen aus Unkenntnis eingeführt wurde. Diese Diskussion soll hier aber nicht geführt werden, für Interessierte sei auf die beiden populärwissenschaftlichen Bücher Beltrami [1999], Klein [2005] zum Weiterlesen verwiesen.

Unabhängig davon, ob Zufall als gegeben vorausgesetzt wird oder als ein Instrument angesehen wird, um reales Verhalten in einem Modell kompakt zu beschreiben, ist unsere Wahrnehmung der Realität in den meisten Fällen nicht deterministisch. So treten Ausfälle technischer Geräte zufällig auf und auch die Reparatur erfordert eine nicht exakt bestimmbare Zeit. Ankünfte von Fahrzeugen an einer Ampel oder von Kunden in einem Supermarkt sind ebenfalls zufällig. Auch viele Vorgänge in der Natur, wie etwa die Einschlagstellen von Blitzen oder der Zerfall von Atomen treten zufällig auf. Gleichzeitig kann Zufall auch benutzt werden, um komplexe deterministische Vorgänge kompakter zu beschreiben.

Im Modell wird Zufall benutzt, um Komplexität zu vermeiden oder wenn eine deterministische Beschreibung nicht möglich ist, weil Details und Zusammenhänge nicht bekannt sind. Grundlage von Zufallsprozessen sind Wahrscheinlichkeitsrechnung und Statistik. Bevor nun die Realisierung von Zufallsprozessen im Rechner beschrieben wird, sollen in einer sehr kurzen Einführung auf die benötigten Grundlagen eingegangen werden (siehe auch Banks et al. [2000, Kap. 5.1], Law and Kelton [2000, Kap. 4.2] oder Montgomery and Runger [2007]).

3.1 Grundlagen der Wahrscheinlichkeitsrechnung

Das mathematische Modell zur Behandlung zufälliger Ereignisse liefert die Wahrscheinlichkeitsrechnung.

Ein *Zufallsexperiment* ist ein Prozess, dessen Ausgang wir nicht mit Gewissheit vorhersagen können. Die Menge aller möglichen einander ausschließenden Ausgänge bezeichnen wir mit S, die Menge der *Elementarereignisse*. Eine Menge E ist eine *Ereignismenge*, falls gilt

- $\emptyset \in E$ und $S \in E$
- $A \in E \Rightarrow S \setminus A \in E$
- $A_i \in E \ (i \in \mathcal{I}) \Rightarrow \cup_{i\in\mathcal{I}} A_i \in E$ und $\cap_{i\in\mathcal{I}} A_i \in E$

Ein weiterer zentraler Begriff ist das *Wahrscheinlichkeitsmaß*. Ein Wahrscheinlichkeitsmaß $P[A]$ bildet $A \in E$ auf reelle Zahlen ab, so dass

- $0 \leq P[A] \leq 1$, $P[S] = 1$ und $P[\emptyset] = 0$
- falls $A \cap B = \emptyset \Rightarrow P[A \cup B] = P[A] + P[B]$

Typische Beispiele für Zufallsexperimente sind das Werfen einer Münze und eines Würfels. Um den Zusammenhang zwischen verschiedenen Ereignissen herzustellen, kann man bedingte Wahrscheinlichkeiten betrachten. Man schreibt üblicherweise $A|B$ mit der Interpretation A unter der Bedingung B. Es gilt

$$P[A|B] = \frac{P[A \cap B]}{P[B]} \tag{3.1}$$

Sei $S = A_1 \cup A_2 \cup \ldots \cup A_K$ und alle A_k seien disjunkt, dann gilt (Satz von der Totalen Wahrscheinlichkeit)

$$P[B] = \sum_{k=1}^{K} P[B|A_k] \cdot P[A_k]$$

Von großer Bedeutung ist auch der Satz von Bayes, der eine Beziehung zwischen $A|B$ und $B|A$ herstellt.

$$P[A|B] = P[B|A] \cdot \frac{P[A]}{P[B]}$$

Zwei Ereignisse A und B heißen unabhängig, falls eine der folgenden Bedingungen gilt:

- $P[A|B] = P[A]$
- $P[B|A] = P[B]$
- $P[A \cap B] = P[A] \cdot P[B]$

Eine *Zufallsvariable* (ZV) ist eine reellwertige Variable, deren Wert durch den Ausgang eines Zufallsexperiment bestimmt ist. Eine Zufallsvariable bildet also die Ausgänge eines Zufallsexperiments auf reelle Zahlen ab. Zufallsvariablen werden im Folgenden mit Großbuchstaben $X, Y, Z, \ldots$ bezeichnet, während Kleinbuchstaben $x, y, z, \ldots$ den konkreten Wert einer Zufallsvariablen bezeichnen. Man unterscheidet zwischen *diskreten* und *kontinuierlichen Zufallsvariablen*. Eine diskrete Zufallsvariable nimmt Werte aus einer endlichen oder abzählbaren Menge an. Typische Beispiele sind der Münzwurf (Werte $0, 1$), das Würfeln (Werte $1, \ldots, 6$) oder die Anzahl eingehender Telefonanrufe an einer Vermittlungsstelle innerhalb einer Stunde (da keine obere Schranke a priori bekannt ist, werden theoretisch Werte aus $\mathbb{N}$ betrachtet). Kontinuierliche Zufallsvariablen können Werte aus einer überabzählbaren Menge annehmen. Beispiele sind die Zwischenankunftszeiten von Kunden an einem Schalter, die Bedienzeiten der Kunden oder die Regenmenge, die an einem Tag fällt. Wenn man davon ausgeht, dass diese Werte beliebig genau messbar wären, so können in diesen Fällen beliebige nicht negative Werte angenommen werden.

Zufallsvariablen können durch verschiedene Maßzahlen charakterisiert werden. Von zentraler Bedeutung ist die *Verteilungsfunktion* (Vfkt) $F(x) = P[X \leq x]$ für $-\infty \leq x \leq \infty$. Die Verteilungsfunktion charakterisiert die Zufallsvariable vollständig und es gilt

- $0 \leq F(x) \leq 1$
- $x_1 < x_2 \Rightarrow F(x_1) \leq F(x_2)$
- $\lim_{x \to -\infty} F(x) = 0$ und $\lim_{x \to \infty} F(x) = 1$

Oft kann die Zufallsvariable auch nur Werte aus einem Intervall annehmen. In diesem Fall ist $F(x) = 0$, falls x kleiner als der kleinste Wert ist, der angenommen werden kann und $F(x) = 1$, falls x größer als der größte Wert ist, der angenommen werden kann. Abbildung 3.1 zeigt zwei Beispiele für den Verlauf der Verteilungsfunktion. Für diskrete Zufallsvariablen ist $F(x)$ immer eine Treppenfunktion. Für kontinuierliche Zufallsvariablen kann die Verteilungsfunktion ebenfalls Sprungstellen aufweisen.

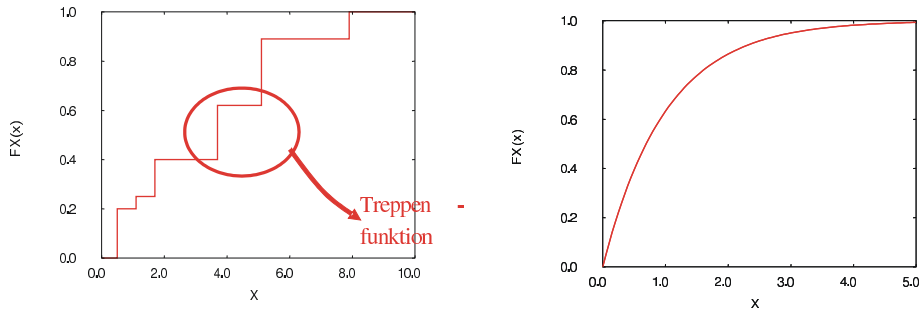


Abbildung 3.1: Verteilungsfunktion einer diskreten und einer kontinuierlichen Zufallsvariablen.

3.1.1 Diskrete ZV X mit Wertebereich W_X

Sei $p(x) = P[X = x]$ die Wahrscheinlichkeit, dass X den Wert x annimmt. Es gilt

- $p(x) = 0$ für $x \notin W_X$
- $0 \leq p(x) \leq 1$ für $x \in W_X$
- $\sum_{x \in W_X} p(x)$=1.0
- $\sum_{x \in W_X \wedge x \leq y} p(x) = F(y)$

Neben der Verteilungsfunktion dienen die *Momente* zur Charakterisierung von Zufallsvariablen. Das $i$te Moment einer diskreten Zufallsvariablen ist definiert als

$$E(X^i) = \sum_{x \in W_X} p(x) \cdot x^i$$

Von besonderer Bedeutung ist das erste Moment $E(X) = E(X^1)$, der *Erwartungswert*. Der Erwartungswert gibt die Summe der mit ihren Wahrscheinlichkeiten gewichteten Werte an. Beim einem fairen Würfel wäre der Erwartungswert 3.5. Das zweite Moment beschreibt die mögliche Variation der Werte der Zufallsvariablen. Als Maßzahl ist die *Varianz*

$$\sigma^2(X) = E((X - E(X))^2) = \sum_{x \in W_X} p(x)(x - E(X))^2 = \sum_{x \in W_X} p(x)x^2 - E(X)^2 = E(X^2) - E(X)^2 \geq 0$$

und die *Standardabweichung* $\sigma(X)$ von Bedeutung. Eine größere Varianz/Standardabweichung deutet auf eine größere Variabilität der Werte einer Zufallsvariablen hin. Wird die Standardabweichung um den Erwartungswert bereinigt, so erhält man den *Variationskoeffizienten* $VK(X) = \sigma(X)/E(X)$. Eine Maßzahl für die Abhängigkeit von zwei Zufallsvariablen X und Y ist die Kovarianz

$$C(X, Y) = E((X - E(X)) \cdot (Y - E(Y))) = E(X \cdot Y) - E(X) \cdot E(Y) \tag{3.2}$$

die auch kleiner 0 sein kann. Die letzte Umformung lässt sich unter Anwendung der folgenden Rechenregeln für Erwartungswerte herleiten.

- $E(c \cdot X) = c \cdot E(X)$ für eine Konstante c,
- $E(h(X)) = \sum_{x \in W_X} p(x) \cdot h(x)$,
- $E(\sum_{i=1}^{n} c_i \cdot X_i) = \sum_{i=1}^{n} c_i \cdot E(X_i)$ für Konstanten c_i und Zufallsvariablen X_i

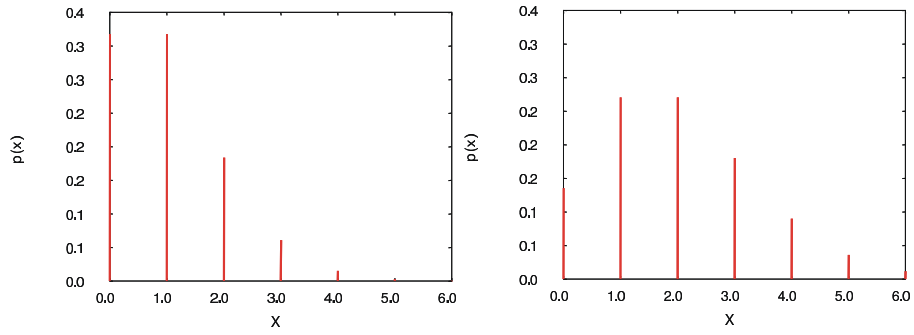


Abbildung 3.2: Poisson Prozess mit Parameter $\lambda = 1$ (linke Seite) und $\lambda = 2$ (rechte Seite).

Die letzte Gleichung gilt sogar, wenn die Zufallsvariablen X_i abhängig sind. Dagegen gilt $E(X \cdot Y) = E(X) \cdot E(Y)$ nur falls $C(X, Y) = 0$, also z.B. bei unabhängigen Zufallsvariablen.

Es werden nun einige typische diskrete Verteilungen vorgestellt. Die Bernoulli-Verteilung mit Parameter $p \in [0, 1]$ beschreibt binäre Entscheidungen. Mit Wahrscheinlichkeit p tritt ein Ereignis ein, mit Wahrscheinlichkeit $(1 - p)$ tritt es nicht ein. Damit gilt

$$p(x) = \begin{cases} p & \text{falls x=1} \\ 1-p & \text{falls x=0} \\ 0 & \text{sonst} \end{cases} \qquad F(x) = \begin{cases} 0 & \text{falls } x < 0 \\ 1-p & \text{falls } 0 \le x < 1 \\ 1 & \text{sonst} \end{cases} \qquad \begin{array}{l} E(X) = p \\ \sigma^2(X) = p \cdot (1-p) \end{array}$$

Die geometrische Verteilung hat ebenfalls einen Parameter $p \in (0, 1]$ und beschreibt die Anzahl der erfolglosen Versuche bis zu einem Erfolg, wenn der Erfolg in jedem Versuch mit Wahrscheinlichkeit p eintritt. Es gilt

$$p(x) = \begin{cases} p \cdot (1-p)^x & \text{falls } x \in \{0, 1, 2, \ldots\} \\ 0 & \text{sonst} \end{cases} \qquad F(x) = \begin{cases} 1 - (1-p)^{\lfloor x \rfloor + 1} & \text{falls } x \ge 0 \\ 0 & \text{sonst} \end{cases}$$

sowie $E(X) = (1-p)/p$ und $\sigma^2(X) = (1-p)/p^2$. Die geometrische Verteilung hat einige interessante Eigenschaften. Insbesondere besitzt die geometrische Verteilung als einzige diskrete Verteilung die so genannte *Gedächtnislosigkeitseigenschaft*. Dies bedeutet in diesem Fall, dass unabhängig vom Wert von x, die Wahrscheinlichkeit im nächsten Versuch einen Erfolg zu haben immer gleich bleibt. Die Gedächtnislosigkeitseigenschaft ist insbesondere wichtig zur Herleitung von analytischen Lösungsansätzen.

Der *Poisson-Prozess* wird zur Modellierung realer Abläufe breit eingesetzt. Er beschreibt die Anzahl auftretender Ereignisse, wenn ein einzelnes Ereignis mit konstanter Rate λ (> 0) auftritt. Für ein Intervall der Länge 1 ist dann $p(x)$ die Wahrscheinlichkeit, dass genau x Ereignisse im Intervall auftreten. Es gilt

$$p(x) = \begin{cases} \frac{e^{-\lambda} \cdot \lambda^x}{x!} & \text{falls } x \in \{0, 1, 2, \ldots\} \\ 0 & \text{sonst} \end{cases} \qquad F(x) = \begin{cases} e^{-\lambda} \cdot \sum_{i=0}^{\lfloor x \rfloor} \frac{\lambda^i}{i!} & \text{falls } x \ge 0 \\ 0 & \text{sonst} \end{cases} \qquad \begin{array}{l} E(X) = \lambda \\ \sigma^2(X) = \lambda \end{array}$$

Werden allgemeine Intervalle der Länge t betrachtet, so muss in den obigen Formeln λ durch λt ersetzt werden. Ein Poisson-Prozess modelliert reales Verhalten immer dann gut, wenn in der Realität Situationen betrachtet werden, bei denen Ereignisse aus vielen unabhängigen Quellen stammen können, die jede für sich mit geringer Rate Ereignisse generieren. Ein typisches Beispiel ist eine Vermittlungsstelle für Telefonanrufe. Viele potenzielle Anrufer können unabhängig voneinander Anrufe generieren und der einzelne Anrufer generiert selten einen Anruf. Es zeigt sich allerdings, dass das Modell des Poisson-Prozesses zwar die Realität mit menschlichen Anrufern gut wiedergibt, durch zusätzliche Dienste, die inzwischen über das Telefon abgewickelt werden, treten aber andere Ankunftsmuster auf, die sich mit Poisson-Prozessen nicht mehr genau genug modellieren lassen.

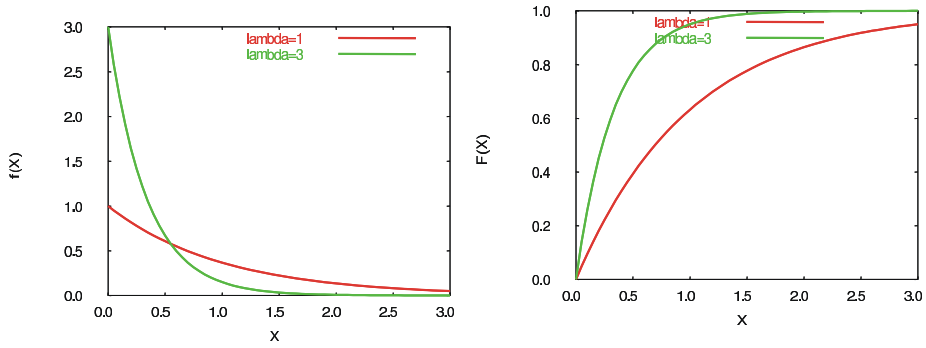


Abbildung 3.3: Dichtefunktion und Verteilungsfunktion der Exponentialverteilung.

3.1.2 Kontinuierliche ZV X mit Wertebereich W_X

Im Gegensatz zum diskreten Fall kann die Zufallsvariable nun überabzählbar viele Werte annehmen. Dies bedeutet, dass für die Wahrscheinlichkeit $p(x) = 0$ für alle $x \in W_X$ gelten kann und trotzdem eine Verteilung vorliegt. Deshalb wird statt der Wahrscheinlichkeit die Dichtefunktion (Dfkt) $f(x)$ für kontinuierliche Zufallsvariablen betrachtet. Für eine Dichtefunktion muss gelten

- $f(x) = 0$ für $x \notin W_X$
- $0 < f(x)$ für $x \in W_X$
- $\int_{x \in W_X} f(x)dx = 1.0$
- $\int_{x \in W_X \wedge x \leq y} f(x)dx = F(y)$
- $\int_y^z f(x)dx = F(z) - F(y) = P[x \in [y, z]]$ falls $y \leq z$
- $E(X^i) = \int_{x \in W_X} x^i \cdot f(x)dx$
- $\sigma^2(X) = \int_{x \in W_X} f(x)(x - E(X))^2 dx = \int_{x \in W_X} f(x) \cdot x^2 dx - E(X)^2$

Es sollen nun einige wichtige kontinuierliche Verteilungen vorgestellt werden. Die Exponentialverteilung mit Parameter $\lambda > 0$ ist durch die folgenden Funktionen/Maßzahlen charakterisiert.

$$f(x) = \begin{cases} \lambda \cdot e^{-\lambda x} & \text{falls } x \geq 0 \\ 0 & \text{sonst} \end{cases} \qquad F(x) = \begin{cases} 1 - e^{-\lambda x} & \text{falls } x \geq 0 \\ 0 & \text{sonst} \end{cases} \qquad \begin{array}{l} E(X) = 1/\lambda \\ \sigma^2 = 1/\lambda^2 \\ VK(X) = 1 \end{array}$$

Die Dichte- und Verteilungsfunktion der Exponentialverteilung wird in Abbildung 3.3 für $\lambda = 1$ und $\lambda = 3$ dargestellt. Exponentialverteilungen treten immer dann auf, wenn viele unabhängige Quellen für ein Ereignis existieren und jede Quelle selten ein Ereignis generiert. Ein typisches Beispiel sind die Zwischenankunftszeiten von Telefonanrufen an einer Vermittlungsstelle. Die Zeit zwischen dem Auftreten von Ereignissen in einem Poisson-Prozess ist exponentiell verteilt. Darüber hinaus ist die Exponentialverteilung die einzige kontinuierliche Verteilung, die gedächtnislos ist. Eine Zufallsvariable X ist gedächtnislos, wenn

$$P[X > t + s | X > t] = P[X > s]$$

für alle $s, t \geq 0$ gilt. Für die Exponentialverteilung ergibt sich

$$\begin{aligned} P[X > s + t | X > t] &= e^{-\lambda \cdot (s+t)} / e^{-\lambda t} \\ &= e^{-\lambda s} \qquad = P[X > s] \end{aligned}$$

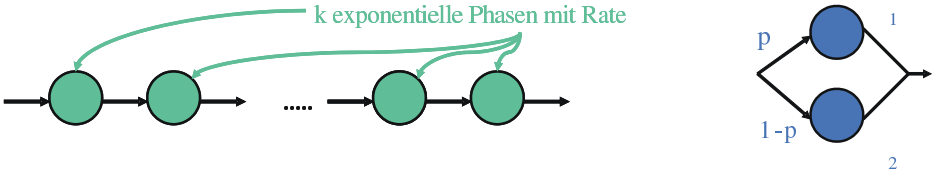


Abbildung 3.4: Struktur der Erlang- und Hyperexponential-Verteilung.

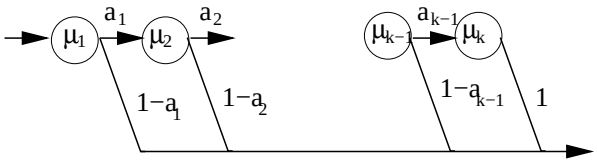


Abbildung 3.5: Cox-Verteilung mit k Phasen.

Die Gedächtnislosigkeit erweist sich als wichtig für analytische Berechnungen, da die Zeit seit dem letzten Ereignis nicht in die Zustandsbeschreibung mit einbezogen werden muss und trotzdem die Verteilung der Eintrittszeit des nächsten Ereignisses ermittelt werden kann.

Durch das Verbinden von mehreren exponentiellen Phasen lassen sich komplexere Verteilungen erzeugen. Zwei typische Beispiele werden in Abbildung 3.4 gezeigt und sollen kurz beschrieben werden. Bei der Erlang-k-Verteilung werden k exponentielle Phasen mit identischem Parameter λ durchlaufen. Die Gesamtzeit ergibt sich also aus der Summe der einzelnen Zeiten. Es gilt für Erlang-k-verteilte Zufallsvariablen $E(X) = k/\lambda$, $\sigma^2(X) = k/\lambda^2$ und $VK(X) = 1/\sqrt{k}$. Damit ist die Erlang-Verteilung weniger variabel als die Exponentialverteilung. Durch die Hinzunahme weiterer Phasen sinkt die Variabilität. Bei der Hyperexponentialverteilung wird alternativ eine von zwei Phasen ausgewählt. Die Verteilung hat 3 Parameter, nämlich p, die Wahrscheinlichkeit Phase 1 zu wählen, sowie λ_1 und λ_2, die Raten der exponentiellen Phasen. Es gilt $E(X) = p/\lambda_1 + (1-p)/\lambda_2$ und $\sigma^2(X) = p(2-p)/\lambda_1^2 + (1-p)^2/\lambda_2^2 - 2p(1-p)/(\lambda_1\lambda_2)$. Man kann nun einfach zeigen, dass für die Hyperexponentialverteilung immer $VK(X) \geq 1$ gilt. Die Hyperexponentialverteilung ist also mindestens so variabel wie die Exponentialverteilung. Man kann sogar zeigen, dass jeder beliebige Variationskoeffizient größer gleich 1 mit einer Hyperexponentialverteilung bei entsprechender Parametrisierung erreichbar ist. Hyperexponentialverteilungen können auch für mehr als zwei Phasen definiert werden.

Gewissermaßen eine Verallgemeinerung der Hyperexponential- und der Erlang-Verteilung bildet die Klasse der Cox-Verteilungen (siehe Abbildung 3.5). Eine Cox-Verteilung besteht aus k exponentiellen Phasen, die sequentiell durchlaufen werden können. Nach jeder Phase wird mit einer festen Wahrscheinlichkeit entschieden, ob die nächste Phase noch durchlaufen wird und damit noch eine exponentiell-verteilte Zufallszahl hinzugezählt wird, oder ob der aktuelle Wert als Wert der Verteilung verwendet wird. Cox-Verteilungen umfassen Erlang- und Hyperexponential-Verteilungen. Ersteres ist einfach nachzuvollziehen, letzteres erfordert eine Transformation. Mit Hilfe von Cox-Verteilungen kann fast jede kontinuierliche Verteilung beliebig genau approximiert werden.

Die Gleichverteilung (siehe Abbildung 3.6) hat zwei Parameter a, b ($a < b$) und ist durch einen festen Wert der Dichtefunktion auf dem endlichen Intervall $[a, b]$ gekennzeichnet. Sie ist charakterisiert durch

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{falls } a \leq x \leq b \\ 0 & \text{sonst} \end{cases} \qquad F(x) = \begin{cases} 0 & \text{falls } x < a \\ \frac{x-a}{b-a} & \text{falls } a \leq x \leq b \\ 1 & \text{sonst} \end{cases} \qquad \begin{array}{l} E(X) = (a+b)/2 \\ \sigma^2(X) = (b-a)^2/12 \\ VK(X) = \frac{b-a}{\sqrt{3}\cdot(a+b)} \end{array}$$

Gleichverteilungen sind von zentraler Bedeutung in der Simulation, da die $[0, 1]$-Gleichverteilung als Grundlage der Generierung von Zufallszahlen benutzt wird, wie im Laufe dieses Abschnitts gezeigt wird.

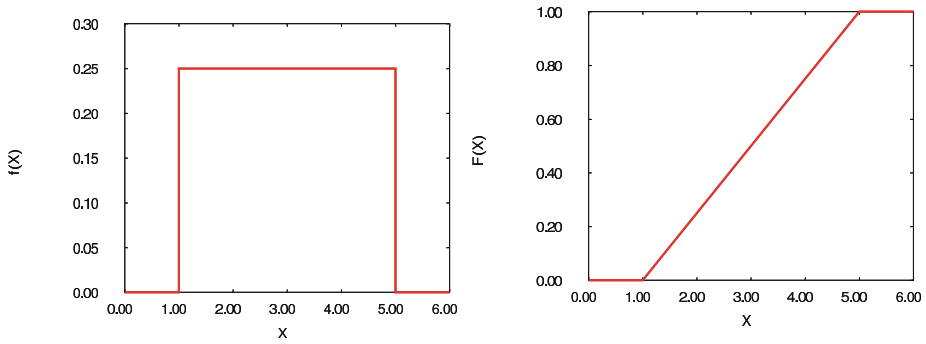


Abbildung 3.6: Dichtefunktion und Verteilungsfunktion der Gleichverteilung mit Parametern 1 und 5.

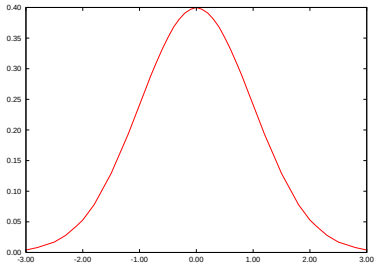


Abbildung 3.7: Dichtefunktion der Normalverteilung.

Die Normalverteilung hat zwei Parameter μ und σ, die ihren Mittelwert und ihre Standardabweichung beschreiben. Die Dichtefunktion der Normalverteilung lautet

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

und wird in Abbildung 3.7 gezeigt. Der Wert von μ kann benutzt werden, um die Dichtefunktion zu verschieben, während σ für die Form (bzw. Breite) der Dichtefunktion verantwortlich ist. Für die Verteilungsfunktion der Normalverteilung existiert keine geschlossene Form. Für normalverteilte Zufallsvariablen X mit Parametern μ und σ schreibt man oft $X \sim N(\mu, \sigma^2)$. Die Dichtefunktion der Normalverteilung ist symmetrisch um den Punkt μ und erreicht ihr Maximum in μ. Es gilt $f(x) > 0$ für alle x und $\lim_{x\to\infty} f(x) = \lim_{x\to-\infty} f(x) = 0$. Falls $X \sim N(0,1)$, dann ist $Y = \mu + \sigma \cdot X \sim N(\mu, \sigma^2)$ und es gilt

$$P[Y < y] = P[\mu + \sigma \cdot X < y] = P[X < (y - \mu)/\sigma]$$

Diese Wahrscheinlichkeit kann aus den Werten der $N(0,1)$-Verteilung ermittelt werden. Die kritischen Werte der $N(0,1)-$Verteilung, d.h. die Werte x mit $P[X < x] = \alpha$ für $X \sim N(0,1)$ und $\alpha = 0.9, 0.95, 0.99$, sind in Wahrscheinlichkeitstafeln in den meisten Statistikbüchern zu finden und können auch numerisch berechnet werden. Die $N(0,1)$-Verteilung bezeichnet man auch als Standardnormalverteilung. Der Normalverteilung kommt in der Statistik und damit auch in der Simulation eine zentrale Bedeutung zu. Dies liegt insbesondere im zentralen Grenzwertsatz begründet.

Satz 1 *Sei $X_1, \ldots, X_n$ eine Menge unabhängig identisch verteilter Zufallsvariablen mit Erwartungswert μ und Standardabweichung σ. Sei $\tilde{X}_n = 1/n \cdot \sum_{i=1}^{n} X_i$ der Mittelwertschätzer. Es gilt dann*

$$\lim_{n\to\infty} Z_n = \lim_{n\to\infty} \frac{\tilde{X}_n - \mu}{\sigma/\sqrt{n}} \sim N(0,1) \ .$$

Damit kann man im Prinzip durch mehrfaches Beobachten erreichen, dass die Summe der beobachtete Wert (fast) normalverteilt ist. Auf dieser Basis lassen sich statistische Aussagen über Zufallsvariablen mit unbekannter Verteilung machen, wie wir später sehen werden.

3.1.3 Schätzer für Verteilungsparameter

In der Statistik werden unbekannte Größen auf Grund von Beobachtungen geschätzt. Gleiches gilt für die Simulation. Wenn Parameter in einem Simulationsmodell gesetzt werden sollen, so werden sie auf Grund von Schätzungen festgelegt. Wenn in einem Simulator Zufallsvariablen zur Beschreibung von Parametern benutzt werden, so sind auch die Ausgaben Zufallsvariablen und Aussagen über das Verhalten des simulierten Modells können nur auf Basis von Schätzungen erfolgen.

Definition 3 *Sei X eine Zufallsvariable mit unbekannter Verteilung. Eine Menge von Beobachtungen $x_1, \ldots, x_n$ nennt man eine Stichprobe.*

Wir gehen in der Regel davon aus, dass die einzelnen Beobachtungen einer Stichprobe unabhängig sind, wenn dies nicht anders gesagt wird. Oftmals sollen Aussagen über einen Parameter Θ der Verteilung von X auf Basis einer Stichprobe $x_1, \ldots, x_n$ gewonnen werden. Typische Beispiele für Θ wären der Erwartungswert, die Varianz oder $P[X < x]$.

Definition 4 *Ein Schätzer $\tilde{\Theta}$ für einen Parameter Θ der Verteilung einer Zufallsvariablen X auf Basis einer Stichprobe $x_1, x_2, \ldots, x_n$ ist eine Funktion $g(x_1, \ldots, x_n) \rightarrow \tilde{\Theta}$. g nennt man die Schätzfunktion.*

Ein Schätzer kann, wie die Notation andeutet, als Zufallsvariable aufgefasst werden. D.h. je nach Realisierung der x_i ergibt sich ein anderer Wert für $\tilde{\Theta}$. Aussagen über alle möglichen Werte und deren Auftretenswahrscheinlichkeiten erhält man aus der Verteilung von $\tilde{\Theta}$.

Für Schätzer gibt es eine Reihe von Eigenschaften, von denen einige vorgestellt werden. $\tilde{\Theta}$ heißt

- erwartungstreu, wenn $E(\tilde{\Theta}) = \Theta$,
- asymptotisch erwartungstreu, wenn $\lim_{n\rightarrow\infty} E(\tilde{\Theta}) = \Theta$,
- konsistent, wenn $\lim_{n\rightarrow\infty} P[|\tilde{\Theta} - \Theta| > \epsilon] = 0$ für jedes $\epsilon > 0$.

Man ist natürlich bestrebt erwartungstreue und konsistente Schätzer zu verwenden. Die Schätzung des Parametern wird uns im Laufe dieses und der folgenden Abschnitte noch mehrfach beschäftigen, insbesondere werden wir unterschiedliche Schätzer kennen lernen.

3.2 Grundlagen der Generierung von Zufallszahlen

Nach der kurzen Einführung in die benötigten Grundlagen der Wahrscheinlichkeitsrechnung und Statistik wird nun die Generierung von Zufallszahlen behandelt. Dies ist ein zentraler Aspekt jeder stochastischen Simulation. Zufallszahlen dienen dazu, stochastisches Verhalten in den eigentlich deterministischen Programmablauf zu bringen. Die Qualität der Simulationsergebnisse hängt ganz entscheidend von der Qualität der verwendeten Zufallszahlen ab. Die Beantwortung der Frage *"Was sind gute Zufallszahlen?"* ist damit essentiell, leider ist die Antwort aber nicht einfach und es wird sich im Lauf dieses Abschnitts zeigen, dass wir keine Methoden haben, die zweifelsfrei die Güte für erzeugte Zufallszahlen festlegen. Die Erzeugung von Zufallszahlen wird in den meisten Simulationsbüchern ausführlich beschrieben, z.B. Law and Kelton [2000, Kap. 7,8] oder Banks et al. [2000, Kap. 7,8].

Ziel ist es, Realisierungen einer Zufallsvariablen X zu generieren. Es wird also eine Methode $zz(X) \rightarrow x$ gesucht, so dass für so erzeugte $x_1, x_2, \ldots$ gilt

- $P[X_j < y] = P[\mathrm{X<y}]$ für alle y (damit gilt auch $E(X_j^i) = E(X^i)$ für alle $i, j > 0$) und

- X_i und X_j für beliebige i, j $(i \neq j)$ seien unabhängig.

wobei X_i die Zufallsvariable ist, aus der x_i resultiert. Die Generierung von Realisierungen einer Zufallsvariablen X nennt man das *Ziehen von Zufallszahlen (ZZ)*. Das Ziehen von Zufallszahlen erfolgt in drei Schritten:

1. Erzeugung von gleichverteilten ganzzahligen Zufallszahlen im Intervall $[0, m)$.
2. Transformation in (approximativ) $[0, 1)$-verteilte Zufallszahlen.
3. Transformation der Zufallszahlen in die gewünschte Verteilung.

Zwei grundsätzlich unterschiedliche Ansätze zur Generierung von Zufallszahlen existieren, die Generierung von echten Zufallszahlen und die Nutzung von Pseudozufallszahlen. Echte Zufallszahlen resultieren aus der Beobachtung zufälliger Prozesse. So kann das Werfen einer Münze oder das Würfeln dazu benutzt werden, Zufallszahlen zu erzeugen. Wenn man davon ausgeht, dass die Münze oder der Würfel fair sind, d.h. unabhängig und identisch verteilte Resultate liefern, so kann man auf diese Weise Zufallszahlen erzeugen. Mit n Münzwürfen kann man zum Beispiel n Bits bestimmen und damit ganzzahlige Zufallszahlen aus dem Intervall $[0, 2^n)$ erzeugen. Da ein Simulationsprogramm keine Münzen oder Würfel werfen kann, böte sich in der Simulation eher die Beobachtung von zufälligen physikalischen Prozessen an. Beispiele für solche Prozesse sind der radioaktive Zerfall oder das weiße Rauschen. Der Nachteil echter Zufallszahlen ist ihre fehlende Reproduzierbarkeit. Es liegt natürlich in der Natur des Zufalls, dass wir das Verhalten eben nicht reproduzieren können. Für den Ablauf von Simulationsprogrammen hat dies aber einige entscheidende Nachteile. So läuft ein Programm, das echte Zufallszahlen nutzt, bei jedem Start unterschiedlich ab. Dies bedeutet, dass Debugging praktisch unmöglich ist. Die Erfahrungen aus dem Softwareentwurf zeigen, dass größere Programme ohne Debugging nicht realisierbar sind. Ein weiterer Nachteil fehlender Reproduzierbarkeit ist, dass Modelle nicht unter identischen Bedingungen (d.h. identischen Realisierungen des Zufalls) verglichen werden können.

Die Forderung nach Reproduzierbarkeit führt zu so genannten *Pseudozufallszahlen*. Wie der Name schon andeutet, handelt es sich dabei nicht um wirkliche Zufallszahlen, sondern um Zahlen, die nach einem Algorithmus erzeugt wurden. Für jemanden, der den Algorithmus kennt, sind die erzeugten Zahlen damit in keiner Weise zufällig, sondern rein deterministisch. Entscheidend ist, dass die erzeugten Zahlen ohne Kenntnis des Generierungsalgorithmus nicht von wahren Zufallszahlen zu unterscheiden sind. Die einfachste Form der Erzeugung von Pseudozufallszahlen ist das Auslesen aus einer Tabelle. Eine solche Tabelle könnte man zum Beispiel dadurch füllen, dass zufällig Prozesse beobachtet werden. Die Tabellenlösung wurde in der Vergangenheit oft angewendet, so erstellte ein Herr Tippett im Jahr 1927 eine Tabelle mit 41600 gleichverteilten Zahlen aus den Daten der Finanzverwaltung. Heute werden Tabellen nicht mehr benutzt, da sehr viele Zufallszahlen für Simulationen benötigt werden und die notwendigen Tabellen einen immensen Speicherplatz erfordern würden und auch das Auslesen der Zahlen zu viel Zeit verbrauchen würde.

3.2.1 Algorithmen zur Erzeugung von Zufallszahlen

Die Alternative zur Tabellenlösung ist ein Generierungsalgorithmus der Form $x_i = g(s_i)$ und $s_{i+1} = f(s_i)$. Der Algorithmus hat einen internen Zustand s_i aus dem durch eine Funktion g eine Zufallszahl erzeugt wird und durch eine Funktion f wird der nächste Zustand generiert. Heutige Generatoren setzen $s_i = x_i$ und benötigen damit nur noch die Funktion f. Durch Setzen eines festen Startwertes x_0 wird immer wieder dieselbe Sequenz von Zufallszahlen generiert. Damit ist die Reproduzierbarkeit offensichtlich gegeben. Durch die Wahl unterschiedlicher Startwerte können unterschiedliche Sequenzen von Zufallszahlen erzeugt werden. Offensichtlich erzeugt jeder solche Generierungsalgorithmus nur eine endliche Sequenz von Zufallszahlen, so dass $x_i = x_j$ dazu führen muss, dass $x_{i+k} = x_{j+k}$ für alle $k > 0$. Bei endlicher Zahlenlänge muss damit zwangsläufig irgendwann eine identische Zahl erreicht werden. Damit ein Generierungsalgorithmus "gute" Zufallszahlen erzeugt und in Simulationsmodellen einsetzbar ist, werden einige Anforderungen an ihn gestellt.

- Die generierten Zufallszahlen müssen gleichverteilt sein.
- Die generierten Zufallszahlen müssen unabhängig sein. Dies bedeutet, dass die Kenntnis der ersten n Zufallszahlen keinerlei Information über die $(n+1)$te Zufallszahl liefert. Dies gilt natürlich nur, wenn der Generierungsalgorithmus unbekannt ist.
- Die Sequenzlänge bis zur Wiederholung einer Zahl muss lang sein.
- Die Erzeugung muss effizient sein.
- Der Algorithmus muss portabel sein.

Es gibt zahlreiche unterschiedliche Generierungsalgorithmen. Wir betrachten hier kurz den ersten Algorithmus der zur Erzeugung von Pseudozufallszahlen publiziert wurde und stellen dann anschließend die am weitesten verbreitete Klasse von Generatoren vor, die in fast allen Simulationswerkzeugen zu finden sind.

Der erste Algorithmus ist die bekannte "Midsquare Method", die von Neumann und Metropolis 1940 publizierten. Ausgehend von einer achtstelligen Dezimalzahl Z_0 werden die mittleren vier Ziffern genommen. Vor diese vier Ziffern wird ein Dezimalpunkt gesetzt und es entsteht eine Zahl aus dem Intervall $[0, 1)$. Z_1 wird dadurch erzeugt, dass die mittleren vier Ziffern von Z_0 quadriert werden, wodurch wieder eine Zahl mit bis zu acht Ziffern entsteht. Bei weniger als acht Ziffern wird von links mit Nullen aufgefüllt. Anschließend wird mit Z_1 wie mit Z_0 verfahren. Ein Beispiel für eine Sequenz von Zufallszahlen wäre

$$\begin{aligned} Z_0 &= 43718244 \quad \rightarrow 0.7182 \\ Z_1 &= 51581124 \quad \rightarrow 0.5811 \\ Z_2 &= 33767721 \quad \rightarrow 0.7677 \\ Z_3 &= 58936329 \quad \rightarrow 0.9363 \\ Z_4 &= 87665769 \quad \rightarrow 0.6657 \\ Z_5 &= 44315649 \quad \rightarrow 0.3156 \end{aligned}$$

Auf den ersten Blick sehen die erzeugten die Zahlen recht zufällig aus. Trotzdem ist die Methode nicht gut, d.h. sie erzeugt keine wirkliche Sequenz von Pseudozufallszahlen, die die oben genannten Bedingungen erfüllen (überlegen warum!).

Die heute meistens verwendeten Generierungsalgorithmen basieren auf einer Arbeit von Lehmer (1951) und werden als *lineare Kongruenzgeneratoren* (LCGs) bezeichnet. Lineare Kongruenzgeneratoren sind durch die Funktion

$$x_i = \left(\sum_{j=1}^{r} a_j \cdot x_{i-j} + c \right) (\mod m)$$

mit $x_0, x_1, \ldots, x_{r-1}, a_1, \ldots, a_r, c \in \mathbb{N}$ gekennzeichnet. Man verwendet heute meistens die reduzierte Version

$$x_i = (a \cdot x_{i-1} + c)(\mod m)$$

Falls $c = 0$, so spricht man von einem multiplikativen Generator ansonsten von einem gemischten Generator. Den initialen Wert x_0 bezeichnet man auch als die Saat des Generators. Jeder LCG weist die folgenden Eigenschaften auf

- Falls $c > 0$ so gilt $x_i \in [0, m)$ und für $c = 0$ gilt $x_i \in (0, m)$
- $x_i = x_j \Rightarrow x_{i+k} = x_{j+k}$ für alle $k \geq 0$

Mit $\kappa = \min_{|i-j|}(x_i = x_j)$ bezeichnet man die Periode des Generators. Das folgende Beispiel wurde aus Page [1991] entnommen. Die Generierungsvorschrift lautet

$$x_{i+1} = (5 \cdot x_i)(\mod 17)$$

Die folgende Tabelle zeigt die Sequenz der generierten Zufallszahlen ausgehend von $x_0 = 5$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$5x_i$	25	40	30	65	70	10	50	80	60	45	55	20	15	75	35	5	25
x_{i+1}	8	6	13	14	2	10	16	12	9	11	4	3	15	7	1	5	8

Wie man sieht, erreicht der Generator die maximal mögliche Periode, alle Zahlen von 1 bis 16 werden erzeugt. In Simulationssystemen verwendete Generatoren haben natürlich eine deutlich größere Periode. Die folgenden Generatoren werden bzw. wurden in unterschiedlichen Systemen verwendet.

Name	m	a	c
Unix	2^{32}	1103515245	12345
RANDU	2^{31}	65539	0
Simula/Univac	2^{31}	5^{13}	0
SIMPL-I (IBM)	$2^{31}-1$	48271	0
SIMSCRIPT II.5	$2^{31}-1$	630360016	0
L'Ecuyer	$2^{63}-25$	4645906587823291368	0

Die ersten drei Generatoren wurden zwar breit eingesetzt, haben sich inzwischen allerdings als schlechte Generatoren herausgestellt. Insbesondere der RANDU wurde in der Vergangenheit in vielen Simulationswerkzeugen verwendet. Es fällt auf, dass die "schlechten" Generatoren alle als Modul eine Zweierpotenz haben. Damit stellen sich natürlich zwei Fragen. Warum wählte man eine Zweierpotenz als Modul? Warum sind dann die Generatoren schlecht? Bevor die Fragen beantwortet werden, muss erst einmal festgelegt werden, was unter einem guten Generator zu verstehen ist. Ein guter Generator sollte die folgenden drei Bedingungen erfüllen.

1. Er sollte eine große Periode haben.
2. Er sollte eine effiziente Berechnung der Zufallszahlen erlauben.
3. Er sollte statistische Test bestehen.

Während die ersten beiden Bedingungen den Bedingungen für gute Zufallszahlen entsprechen, unterscheidet sich die dritte Bedingung von den auf Seite 74 geforderten Eigenschaften für gute Zufallszahlen. Dies liegt darin begründet, dass man für einen Generator nicht formal zeigen kann, dass zwei generierte Zufallszahlen x_i, x_j unabhängig sind und jeweils aus einer Gleichverteilung gezogen wurden. Man kann nur anhand von statistischen Test überprüfen, ob diese Annahmen plausibel sind. Entsprechende Testverfahren werden später vorgestellt.

Die erste Bedingung für einen guten Generator fordert eine große Periode, damit viele unterschiedliche Zufallszahlen erzeugt werden können. Dies bedeutet, dass m möglichst groß sein sollte, also im Bereich der maximal darstellbaren ganzen Zahl. Weiterhin sollten in der Periode möglichst m bzw. $m-1$ unterschiedliche Werte erzeugt werden. Man kann Bedingungen an die Parameter a, c und m stellen, damit die volle Periodenlänge erreicht wird. Für einen gemischten Generator müssen die folgenden drei Bedingungen gelten Law and Kelton [2000, S. 408].

1. $ggT(m,c) = 1$ man sagt auch, dass m und c relativ prim sind
2. $a \mod p_i = 1$ für alle Primfaktoren p_i von m
3. $a \mod 4 = 1$ falls 4 ein Faktor von m ist

Die Bedingungen sind leicht zu prüfen, so dass ein gemischter Generator mit voller Periodenlänge konstruiert werden kann. Offensichtlich sind die Bedingungen nicht für multiplikative Generatoren anwendbar, da die erste Bedingung für $c = 0$ nicht erfüllbar ist. Für multiplikative Generatoren gelten die beiden folgenden Resultate Banks et al. [2000, S. 260].

- Falls $m = 2^b$, dann ist der maximale Wert für $\kappa = m/4 = 2^{b-2}$. Dieser Wert wird erreicht für ungerade Saaten x_0 und $a = 3 + 8k$ oder $a = 5 + 8k$ für $k = 0, 1, \ldots$

- Falls m eine Primzahl ist, dann wird $\kappa = m - 1$ erreicht, falls die kleinste ganze Zahl k, für die $a^k - 1$ durch m ganzzahlig dividiert werden kann, gleich $m - 1$ ist (a ist Primitivwurzel von m).

Die zweite Klasse von Generatoren bezeichnet man auch als *prime modulus multiplicative LCGs* (PMMLCGs). Sie sind natürlich besonders interessant, da sie im Gegensatz zu gemischten Generatoren keine Additionsoperation benötigen. Allerdings ist die Bedingung deutlich komplexer als die einfachen Bedingungen für gemischte Generatoren. Es ist tatsächlich so, dass PMMLCGs oft durch Ausprobieren und Prüfen erzeugt werden.

Die vorgestellten Generatoren hatten zumeist eine Periode im Bereich von $2^{32} \approx 4.2 \cdot 10^9$ der Wortlänge der üblichen Prozessoren zum Zeitpunkt ihrer Entwicklung. Heute ist die übliche Wortlänge zwar 64, aber es gibt erst sehr wenige Generatoren, die diese Wortlänge nutzen. Es bleibt natürlich die Frage, ob 2^{32} nicht ausreicht. Über lange Jahre hätte man diese Frage sicherlich mit ja beantwortet. Wenn man sich aber die Geschwindigkeit heutiger Prozessoren vor Augen führt, so benötigt ein PC mit 3 GHz Prozessor ca. 10 bis 15 Minuten um 2^{32} Zufallszahlen zu erzeugen. Simulatoren realistischer Systeme, wie etwa die Simulationen großer Rechnernetze oder Fertigungssysteme, laufen aber mehrere Stunden oder Tage. Auch wenn in der Simulation neben der Zufallszahlenerzeugung noch viele andere Aufgaben Zeit verbrauchen, ist es leicht nachvollziehbar, dass in einer langen Simulation der Zyklus der Zufallszahlen mehrfach durchlaufen wird. Dies widerspricht natürlich unserem Gefühl von Zufall. Deshalb besteht ein großes Interesse daran gute Generatoren mit größerer Periode zu entwickeln. Dies kann einmal dadurch geschehen, dass ganz neue Generatoren mit $m \approx 2^{64}$ oder noch größer entwickelt werden. Es können aber auch bekannte Generatoren kombiniert werden L'Ecuyer [1999]. Wir betrachten k Generatoren mit Periode m_j für den $j$ten Generator. Sei $x_{i,j}$ der $i$te Wert des $j$ten Generators, dann ist

$$x_i = \left(\sum_{j=1}^{k} (-1)^{j-1} x_{i,j} \right) (\mod m_1 - 1)$$

$[0, m_1 - 2]$-gleichverteilt und die maximal erreichbare Periode lautet

$$\frac{\left(\prod_{j=1}^{k} (m_j - 1) \right)}{2^{k-1}}$$

Ein populäres Beispiel für heutige aktuelle Generatoren ist der Mersenne Twister Generator Matsumoto and Nishimura [1998]. Dieser Generator wurde 1998 erstmal publiziert und seitdem ständig weiterentwickelt. Es basiert auf Matrixrekurrenzen. Die Generierungsvorschrift ist komplexer als bei den bisher betrachteten LCGs, basiert aber auf einfachen und effizient zu implementierenden *XOR*- und shift-Operationen. Die aktuelle Variante des Generators hat eine Periodenlänge von $2^{19937} - 1 \approx 10^{6000}$ und liefert gleichverteilte Zufallszahlen in 623 Dimensionen. Darüber hinaus passierte der Generator die meisten vorhandene Testverfahren zum der Test der Güte von Zufallszahlengeneratoren. Es wurde allerdings in L'Ecuyer and Panneton [2005] gezeigt, dass es Situationen gibt, in denen der Generator eine sehr kleine Zahl erzeugt und anschließend mehrere hundertausend Zufallszahlen erzeugt werden müssen, bis der Mittelwert der erzeugten Zufallszahlen wieder im Bereich von 0.5 liegt.

Die zweite Anforderung an Zufallszahlengeneratoren ist eine effiziente Generierung. Im Gegensatz zur Addition und Multiplikation ist Division sehr aufwändig. Deshalb sollte ein effizienter Generator möglichst keine oder wenige Divisionen benötigen. Falls $m = 2^b$ ist, so kann auf Divisionen verzichtet werden, da die modulo Operation durch das Weglassen von Stellen, d.h. shiften auf Bitebene, realisiert werden kann. So entspricht $10111011 \mod 2^6 = 00111011$ und in Dezimaldarstellung $187 \mod 64 = 59$. Leider zeigt sich aber, dass durch die modulo-Bildung mit einer Zweierpotenz Zyklen in den niederwertigen Bits auftreten. Dies kann am Beispiel von $x_{i+1} = (5x_i + 1) \mod 16$ erläutert werden. Ausgehend von $x_0 = 1$ ergibt sich die folgende Sequenz von Zahlen, die als Dezimal und als Binärzahlen dargestellt werden.

i	0	1	2	3	4	5	6	7
x_i	1	6	15	12	13	2	11	8
b_0	1	0	1	0	1	0	1	0
b_1b_0	01	10	11	00	01	10	11	00
$b_2b_1b_0$	001	110	111	100	101	010	011	000

Während die Dezimaldarstellung recht zufällig aussieht, erkennt man in der Binärdarstellung eine sehr regelmäßige Struktur, die offensichtlich alles andere als zufällig zu sein scheint. Aus diesem Grund weisen Zufallszahlen, die mit einem Generator mit $m = 2^e$ erzeugt wurden, auch schlechte statistische Eigenschaften auf.

Als Alternative bietet es sich an $m = 2^e - 1$ zu wählen. Es ist noch zu zeigen, dass $z = (a \cdot x)(\mod 2^e - 1)$ effizient, d.h. ohne Division berechnet werden kann. Sei dazu $y = (a \cdot x)(\mod 2^e)$. Dies ist offensichtlich durch einfaches Abschneiden der überzähligen Bits berechenbar. Nun muss noch z auf Basis von y berechnet werden. Dies erfordert die folgende einfache Berechnung.

$$z = \begin{cases} y + k & \text{falls } y + k < 2^e - 1 \\ y + k - (2^e - 1) & \text{sonst} \end{cases}$$

mit $k = \lfloor a \cdot x / 2^e \rfloor$. Man kann die Formel leicht herleiten, wenn man sich überlegt, dass bei Berechnung $\mod 2^e$ statt $\mod 2^e - 1$ bei jedem Überlauf über 2^e genau eins zu wenig berechnet wird. Die Summation ergibt genau k. Wenn durch die Addition von k ein weiterer Überlauf erzeugt wird, so muss $2^e - 1$ abgezogen werden. Man benötigt also zusätzliche Additionen, kann aber die Division vermeiden. Das Verfahren wird auch als simulierte Division bezeichnet und lässt sich auch für Modulen der Form $2^e - q$ verallgemeinern.

Bevor der dritte Aspekt, nämlich das Bestehen von statistischen Tests betrachtet wird, werden die erzeugten Zufallszahlen auf das Intervall $[0, 1)$ bzw. $(0, 1)$ normiert. Dies geschieht dadurch, dass durch m dividiert wird. Wenn die Zufallszahlen vorher gleichverteilte ganze Zahlen im Intervall $[0, m)$ bzw. $(0, m)$ waren, so kann man die resultierenden Werte als approximativ $[0, 1)$ bzw. $(0, 1)$ gleichverteilt ansehen. Wir betrachten im Folgenden den Fall $[0, 1)$, wenn die 0 ausgeschlossen ist, kann vollkommen analog vorgegangen werden. Streng genommen liegt natürlich keine Gleichverteilung vor, da nur endliche viele Werte erzeugt werden. Falls m aber dieselbe Länge wie die Mantissendarstellung hat, so liegen die erzeugten Werte optimal dicht und es werden alle darstellbaren Zahlen erreicht. Damit kann man die erzeugten Zufallszahlen als gleichverteilt ansehen, ähnlich wie double-Werte eine Approximation der reellen Zahlen sind.

3.2.2 Statistische Testverfahren für Zufallszahlen

Auf Basis der gleichverteilten Zufallszahlen können Testverfahren angewendet werden, um schlechte von guten Generatoren zu unterschieden. Gute Zufallszahlen müssen zwei Bedingungen erfüllen.

- Sie müssen gleichverteilt im Intervall $[0, 1)$ sein.
- Sie müssen unabhängig sein.

Da Zufallszahlen betrachtet werden, müssen zufällige Schwankungen in die Beobachtungen mit einbezogen werden und es ist nicht möglich Beweise zu führen. Stattdessen werden Testverfahren benutzt. Bevor konkrete Testverfahren vorgestellt werden, soll kurz allgemein auf Testverfahren in der Statistik eingegangen werden. Bei einem Test wird eine Hypothese H_0 aufgestellt. Im hier vorliegenden Fall könnte dies eine der beiden folgenden Alternativen sein.

Die mit Generator G erzeugten Zufallszahlen sind unabhängig, identisch $[0, 1)$-gleichverteilt oder

die mit Generator G erzeugten Zufallszahlen sind nicht unabhängig, identisch $[0, 1)$-gleichverteilt.

H_0 heißt *Nullhypothese* und entsprechend heißt $H_1 = \neg H_0$ *Alternativhypothese*. Testverfahren dienen dazu herauszufinden, ob H_0 gilt. Dies geschieht in der Regel auf Basis von Stichproben. Auf Grund der statistischen Schwankungen von Stichproben, können falsche Folgerungen gezogen werden. Tests sind keine Beweise! Man unterschiedet folgende Fehler.

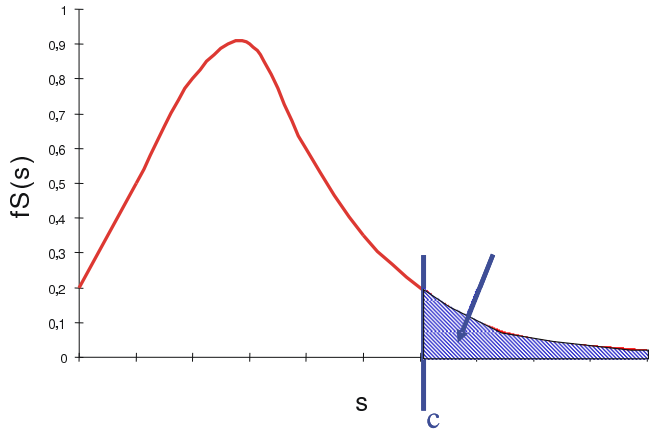


Abbildung 3.8: Prinzip des Vorgehens beim Testen.

- (Statistische) Fehler der 1. Art (α Fehler): H_1 wird angenommen, obwohl H_0 gilt. D.h. die Nullhypothese wird fälschlicherweise verworfen.
- (Statistische) Fehler der 2. Art (β Fehler): H_0 wird angenommen, obwohl H_1 gilt. D.h. die Nullhypothese wird fälschlicherweise angenommen.

Impliziert ein bestimmter Test mit Wahrscheinlichkeit $\leq \alpha$ Fehler der 1. Art, so heißt er "Test zum (Signifikanz-)Niveau α", unabhängig vom Fehler 2. Art. Natürlich sind aber Fehler 1. und 2. Art nicht unabhängig. Der triviale Test, der H_0 immer annimmt macht keinen Fehler der ersten Art, während der Test, der H_0 immer ablehnt, keinen Fehler der 2. Art macht. Beide Ansätze sind offensichtlich nicht sehr hilfreich. Die Bewertung von Testergebnissen beruht im Wesentlichen auf Tests zum Niveau α.

Beim Testen auf Basis einer Stichprobe wird die Stichprobe mit einer Teststatistik $S(Y_1, \ldots, Y_n)$ bewertet, wobei Y_i die Zufallsvariable ist, die den $i$ten Wert in der Stichprobe beschreibt. Je größer der Wert von $S(y_1, \ldots, y_n)$ für eine konkrete Stichprobe $y_1, \ldots, y_n$ ist, desto unwahrscheinlicher ist H_0 und damit steigt implizit die Wahrscheinlichkeit von H_1. Zur Anwendung sind die folgende Schritte notwendig:

- Bestimme die Verteilung $S(.)$ unter der Voraussetzung, dass H_0 gilt.
- Ermittle die kritischen Werte c_α (oder $c_{1-\alpha}$) ab denen H_0 zum Niveau α verworfen wird.

Eine Skizze des Prinzips ist in Abbildung 3.8 zu sehen. Der Wert von c_α wird so bestimmt, dass $P[S(Y_1, \ldots, Y_n) > c_\alpha | H_0] = \alpha$. Bei bekannter Verteilung von $S(.)$ ist die Bestimmung kein Problem. Übliche Werte für α sind 0.05 (signifikant) und 0.01 (hochsignifikant). Grundsätzlich kann ein Testverfahren einige schlechte Generatoren identifizieren, indem sie den Test nicht passieren. Es ist aber nicht möglich, die generelle Güte eines Generators nachzuweisen.

Der bisher beschriebene Testansatz basiert auf Stichproben und wird als empirischer Test bezeichnet. Alternativ gibt es einige theoretische Testverfahren, die auf Basis der Struktur und Parameter eines Generators Aussagen machen. Theoretische Testverfahren sind oft komplex und mathematisch anspruchsvoll, machen aber in der Regel Aussagen über alle generierten Zufallszahlen, während empirische Testverfahren nur die verwendete Stichprobe nutzen und damit für unterschiedliche Stichproben auch zu unterschiedlichen Resultaten kommen können.

Wir stellen im Folgenden erst einige empirische Testverfahren und anschließend theoretische Testverfahren vor. Für den Test, dass eine Gleichverteilung vorliegt können der Chi-Quadrat-Test und der Kolmogorov-Smirnov-Test verwendet werden, die in allgemeinerer Form als Anpassungstests in Abschnitt 4 vorgestellt werden.

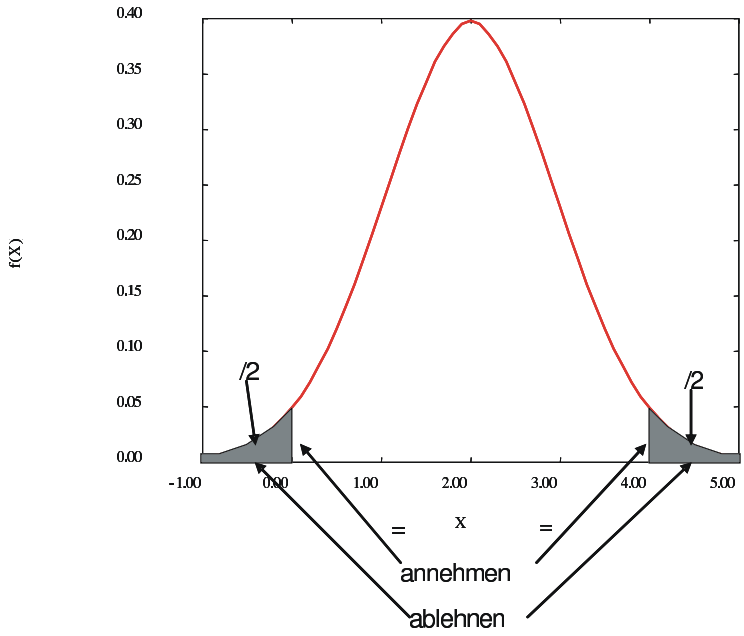


Abbildung 3.9: Kritische Werte der Normalverteilung und ihr Einsatz in Testverfahren.

Ein erstes Testverfahren zum Test der Unabhängigkeit von Zufallszahlen ist der Runs-Test, der in verschiedenen Varianten existiert. Wir betrachten hier die einfachste Variante, die jeweils Paare von Zufallszahlen in Klassen einteilt (siehe auch Banks et al. [2000, S. 270]). Ein Paar x_i, x_{i+1} gehört zur Klasse $+$, wenn $x_i < x_{i+1}$, ansonsten gehört es zur Klasse $-$. Für eine Stichprobe entsteht eine Sequenz von $+$ und $-$ Zeichen. Ein Run ist eine maximale Subsequenz identischer Zeichen. Sei n die Größe der Stichprobe und R_n die Zufallsvariable, die die Anzahl der Runs beschreibt. Für große n (laut Literatur reicht i.d.R. schon $n > 20$) ist R_n approximativ normalverteilt mit

$$E(R_n) = \frac{2n-1}{3} \quad \text{und} \quad \sigma^2(R_n) = \frac{16n-29}{90}.$$

Damit ist $Z = (R_n - (2n-1)/3)/\sqrt{(16n-29)/90} \sim N(0,1)$. Auf Basis der kritischen Werte der Normalverteilung (siehe Abbildung 3.9) kann dann die Hypothese der Unabhängigkeit angenommen oder verworfen werden. Sei r die Anzahl der Runs bei in einer Stichprobe der Größe n und $z = (r-(2n-1)/3)/\sqrt{(16n-29)/90}$, dann wird die Hypothese angenommen, wenn der Wert von z im mittleren, nicht schraffierten Bereich liegt. Werte im schraffierten Bereich führen zur Ablehnung. Die kritischen Wert der Normalverteilung werden auch mit $-z_{\alpha/2}$ und $z_{\alpha/2}$ bezeichnet. Also wird die Hypothese angenommen falls $-z_{\alpha/2} \leq z \leq z_{\alpha/2}$ gilt und andernfalls abgelehnt.

Der schon bekannte Generator $x_{i+1} = (5 \cdot x_i)(\mod 17)$ mit $x_0 = 5$ erzeugt die folgende Stichprobe der Länge 16:

8	6	13	14	2	10	16	12	9	11	4	3	15	7	1	5
	+	+		+	+			+			+			+	
-			-			-	-		-	-		-	-		

Dies ergibt 10 Runs. Laut Theorie gilt $E(R_{16}) = (2 \cdot 16 - 1)/3 = 10.333$. Damit liegt eine gute Übereinstimmung vor. Für $\alpha = 0.05$ würde die Hypothese H_0 (= Zufallszahlen sind unabhängig) für $R_{16} \in [8, 13]$ akzeptiert. In der vorgestellten Form macht der Runs-Test keine Aussagen darüber, ob die Werte $[0, 1)$-gleichverteilt sind, sondern testet nur die Unabhängigkeit.

Ein weiteres Verfahren zum Test auf Unabhängigkeit ist der Test auf Autokorrelation. Seien $X_1, \ldots, X_n$ identisch verteilte Zufallsvariablen mit Erwartungswert $E(X)$ und Varianz $\sigma^2(X)$. Der

Autokorrelationskoeffizient der Ordnung s ist definiert als

$$\rho(s) = \frac{\sum_{i=1}^{n-s} E((X_{i+s} - E(X))(X_i - E(X)))}{\sum_{i=1}^{n-s} E((X_i - E(X))^2)} = \frac{C(X_i, X_{i+s})}{\sigma^2(X)} = \frac{C_s}{C_0}.$$

Es gilt $-1 \leq \rho(s) \leq 1$ und für unabhängige X_i und X_{i+s} ist $\rho(s) = 0$. Die Umkehrung der letzten Aussage muss allerdings nicht gelten. Beim Test auf Autokorrelation wird überprüft inwieweit $\rho(s) \approx 0$ angenommen werden kann. Wenn statt der Zufallsvariablen eine konkrete Stichprobe eingesetzt wird, so benötigt man eine Schätzfunktion für den Wert von $\rho(s)$. Für $[0, 1)$-verteilte Zufallsvariablen X_i gilt $E(X) = 1/2$ und $\sigma^2(X) = C_0 = 1/12$. Da $C_s = E(X_i X_{i+s}) - E(X_i)E(X_{i+s})$ (siehe (3.2)), gilt auch

$$\rho(s) = \frac{C_s}{C_0} = \frac{E(X_i X_{i+s}) - E(X_i)E(X_{i+s})}{\sigma^2(X)} = \frac{E(X_i X_{i+s}) - 1/4}{1/12} = 12E(X_i X_{i+s}) - 3$$

Ein Schätzer für $\rho(s)$ lautet dann

$$\tilde{\rho}(s) = \frac{12}{h+1} \sum_{k=0}^{h} X_{1+ks} \cdot X_{1+(k+1)s} - 3$$

mit $h = \lfloor (n-1)/s \rfloor - 1$. $\tilde{\rho}(s)$ ist im Gegensatz zu $\rho(s)$ eine Zufallsvariable, deren konkrete Realisierung von der Stichprobe abhängt. Wie in Banks et al. [2000, S. 279] beschrieben ist $\tilde{\rho}(s)$ für unabhängig identisch $[0, 1)$-verteilte X_i normalverteilt mit Erwartungswert 0 und Standardabweichung $(\sqrt{13h+7})/(h+1)$. Damit kann für eine konkrete Stichprobe $x_1, \ldots, x_n$ der Wert

$$\hat{\rho}(s) = \frac{12}{h+1} \sum_{k=0}^{h} x_{1+ks} \cdot x_{1+(k+1)s} - 3$$

berechnet werden und $z = \hat{\rho}(s) \cdot (h+1)/(\sqrt{13h+7})$ gegen die kritischen Werte einer $N(0, 1)$-Verteilung getestet werden.

Theoretische Testverfahren machen Aussagen über alle erzeugten Zufallszahlen. Wenn der Generator alle Zahlen aus dem Intervall $[0, m)$ erzeugt, also volle Periode hat, dann ist der Mittelwert der erzeugten Zufallszahlen $1/2 - 1/(2m)$ und die Varianz $1/12 - 1/(12m^2)$. Beide Werte liegen für große m sehr nahe an den "wahren" Werten.

Der wichtigste theoretische Test leitet die Struktur der generierten Zufallszahlen im höher dimensionalen Raum her. Sei $x_1, x_2, \ldots$ die Sequenz der erzeugten Zufallszahlen. Überlappende d-Tupel $(x_1, \ldots, x_d), (x_2, \ldots, x_{d+1}), \ldots$ definieren jeweils Punkte im d-dimensionalen Hyperraum. Für LCGs fallen die Punkte auf relativ wenige Ebenen im Hyperraum der Dimension $d-1$. Im zweidimensionalen liegen die Punkte alle auf einem Gitter. Außerhalb dieses Gitters sind keine Punkte erreichbar. Die Qualität eines Generators hängt nun von der Anzahl der Hyperebenen ab, auf denen Punkte liegen können. Tests wie der Spektraltest oder Gittertest dienen zur Bewertung der Struktur in verschiedenen Dimensionen. Für niedrige Dimensionen kann man eine graphische Darstellung wählen. Höhere Dimensionen erfordern Transformationen.

Abbildung 3.10 zeigt für den Generator RANDU ($x_{i+1} = 65539 \cdot x_i \mod 2^{31}$)) die generierten Tupel (linke Seite der Abbildung) und die durch die Transformation $9x_i - 6x_{i+1} + x_{i+2}$ ins zweidimensional transformierte dreidimensionale Darstellung (rechte Seite der Abbildung). Während die zweidimensionale Darstellung recht zufällig aussieht, kann man an der transformierten dreidimensionalen Darstellung erkennen, dass die Werte auf sehr wenigen Hyperebenen im Dreidimensionalen liegen und der Generator daher schlechte statistische Eigenschaften hat.

Zusammenfassend kann man zum Testen von Zufallszahlengeneratoren die folgenden Aussagen machen:

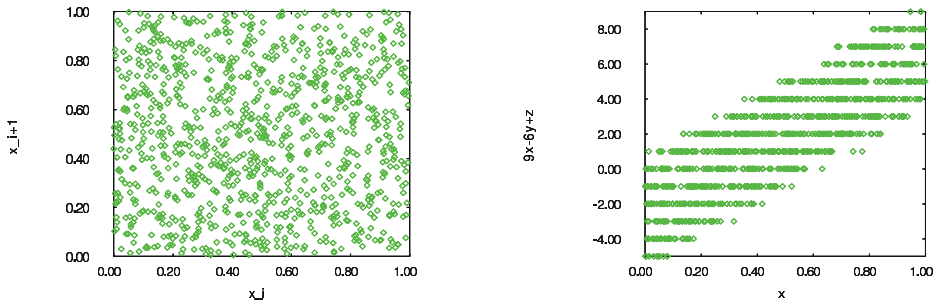


Abbildung 3.10: Darstellung von ZZs Tupeln und Tripeln für den RANDU-Generator.

- Es existiert eine Vielzahl unterschiedlicher Tests und es ist unklar, welche Tests die besten Ergebnisse liefern.
- Tests können keine beweisbar guten Zufallszahlengeneratoren liefern, sondern nur schlechte aussondern.
- Einige Generatoren wurden aufwändig getestet und haben sich bei allen Tests als "gut" bzgl. des jeweiligen Tests herausgestellt. Generatoren dieser Art sollten möglichst verwendet werden.
- Wenn nicht klar ist, ob ein Generator für ein gegebenes Problem "gut" ist, so kann durch die Nutzung unterschiedlicher Generatoren in Simulationsläufen die Verzerrung der Ergebnisse durch einzelne Generatoren entdeckt werden. Zur Umsetzung des Ansatzes muss das verwendete Simulationswerkzeug mehrere Generatoren beinhalten oder neue Generatoren müssen leicht integrierbar sein.

3.2.3 Transformation von $[0,1)$-verteilten Zufallszahlen

In den bisherigen Schritten wurden $[0,1)$-gleichverteilte Zufallszahlen erzeugt und bzgl. ihrer Güte bewertet. Für eine Simulation werden aber Zufallszahlen aus einer Verteilungsfunktion $FX(x)$ benötigt. Durch eine Transformation der Zufallszahlen u_i aus der $[0,1)$-Gleichverteilung in Zufallszahlen x_i aus der gewünschten Verteilung $FX(x)$ werden die gewünschten Zufallszahlen erzeugt. Unterschiedliche Transformationsmethoden werden zum Abschluss dieses Abschnitts vorgestellt. In der Regel liefern die Transformationen exakte Resultate, so dass gute $[0,1)$-gleichverteilte Zufallszahlen für gute Zufallszahlen aus einer beliebigen Verteilung sorgen.

Zur Erinnerung sei noch einmal erwähnt, dass für unsere $[0,1)$-gleichverteilten Zufallszahlen u gilt

$$FU(u) = \begin{cases} 0 & \text{falls } u < 0 \\ u & \text{falls } 0 \leq u < 1 \\ 1 & \text{sonst} \end{cases} \qquad fU(u) = \begin{cases} 1 & \text{falls } 0 \leq u < 1 \\ 0 & \text{sonst} \end{cases}$$

Für die erste Transformationsmethode nehmen wir an, dass die Verteilungsfunktion $FX(x)$ kontinuierlich ist, und dass für $x_1 < x_2$ mit $0 < F(x_1) \leq F(x_2) < 1$ auch $F(x_1) < F(x_2)$ gilt. Ist dies der Fall, so existiert die Umkehrfunktion F^{-1} von F. Das folgende, als *Inverse Transformation* bezeichnete Vorgehen ist naheliegend.

1. Generiere u aus einer $[0,1)$-Gleichverteilung
2. Transformiere $x = F^{-1}(u)$

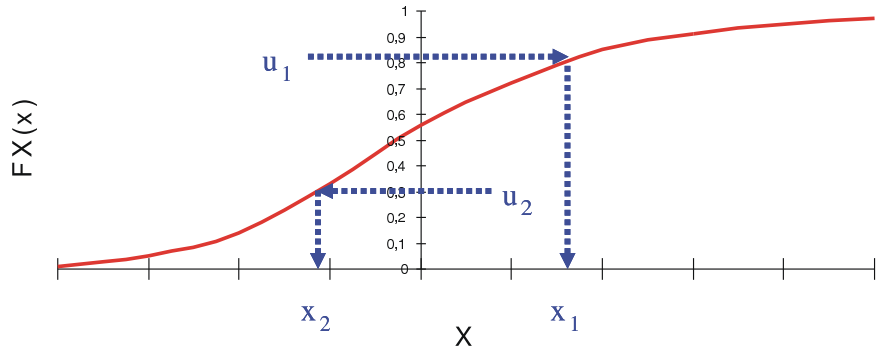


Abbildung 3.11: Skizze der inversen Transformation.

3.2.3.1 Inverse Transformation

Das Vorgehen wird noch einmal in Abbildung 3.11 verdeutlicht. Formal gilt $P[X \leq x] = P[F^{-1}(u) \leq x] = P[u \leq F(x)] = F(x)$.

Die inverse Transformation soll am Beispiel der Exponentialverteilung kurz vorgestellt werden. Für die Exponentialverteilung gilt $F(x) = 0$ falls $x \leq 0$ und $F(x) = 1 - e^{-\lambda x}$ für $x > 0$. Sei u aus einer $(0, 1)$-Gleichverteilung gezogen, dann werden die folgenden Schritte durchgeführt.

$$\begin{aligned} 1 - e^{-\lambda x} = u \Rightarrow \quad & e^{-\lambda x} = 1 - u \Rightarrow \\ -\lambda x = \ln(1-u) \Rightarrow \quad & x = \tfrac{\ln(1-u)}{-\lambda} \end{aligned}$$

Da $1 - u$ genauso wie u $(0, 1)$-gleichverteilt ist, kann $1 - u$ durch u ersetzt werden. Damit ergibt sich die folgende Transformation zur Erzeugung exponentiell verteilter Zufallszahlen.

1. Generiere u aus einer $(0, 1)$-Gleichverteilung
2. Transformiere $x = \ln(u)/(-\lambda)$

Es stellt sich natürlich die Frage, ob das Vorgehen auf alle kontinuierlichen Verteilungen anwendbar ist. Dies gilt leider nicht, da $F^{-1}(x)$ verfügbar sein muss, um die Transformation durchzuführen. Dies ist nicht bei allen Verteilungen der Fall. Es gibt allerdings für zahlreiche kontinuierliche Verteilungen inverse Transformationen (siehe Law and Kelton [2000, Kap. 8.3]). Bevor andere Ansätze zur Generierung von Zufallszahlen aus kontinuierlichen Verteilungen erläutert werden, soll die inverse Transformation für diskrete Verteilungen kurz vorgestellt werden.

Offensichtlich funktioniert das Vorgehen für kontinuierliche Verteilungen im diskreten Fall nicht, da $F(x)$ eine Treppenfunktion ist, für die keine Umkehrfunktion gebildet werden kann. Man kann allerdings im diskreten Fall eine sehr ähnliche Vorgehensweise anwenden, wie in Abbildung 3.12 gezeigt wird. Für diskrete Verteilungen gilt offensichtlich $F(x) = P[X \leq x] = \sum_{x_i \leq x} p(x_i)$. Damit ergibt sich folgendes Vorgehen der inversen Transformation bei diskreten Zufallsvariablen.

1. Generiere u aus eine $[0, 1)$-Gleichverteilung
2. Finde eine ganze Zahl I, so dass $\sum_{i<I} p(x_i) < u \leq \sum_{i \leq I} p(x_i)$ und liefere x_I als Wert zurück.

Für diskrete Verteilungen mit vielen Werten kann das Suchen nach dem geeigneten I aufwändig sein, insbesondere wenn linear gesucht wird und am Anfang viele Punkte mit kleinen Wahrscheinlichkeiten liegen. Deshalb sollten in diesen Fällen die Intervalle absteigend nach den Wahrscheinlichkeiten $p(x_i)$ geordnet werden. Das Vorgehen ist auch für diskrete Verteilungen mit unendlich vielen Werten, wie dem Poisson Prozess, geeignet.

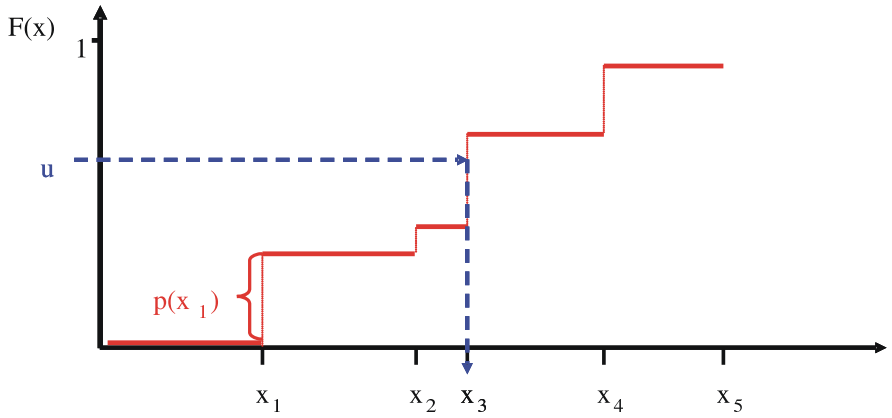


Abbildung 3.12: Inverse Transformation für eine diskrete Verteilung.

3.2.3.2 Weitere Generierungsmethoden

Die inverse Transformation sollte immer dann eingesetzt werden, wenn die Umkehrfunktion in geschlossener Form vorliegt und leicht ausgewertet werden kann. Falls $F^{-1}(x)$ nicht in geschlossener Form vorliegt, so kann prinzipiell der Wert numerisch berechnet werden. Dies ist aber für manche Verteilungsfunktionen recht aufwändig und unter Umständen auch numerisch instabil. Aus diesem Grund sollen noch einige andere Möglichkeiten der Transformation vorgestellt werden.

Falls sich eine Zufallsvariable X als Summe von Zufallsvariablen Y_i $(i = 1, \ldots, I)$ mit Verteilungsfunktionen $F_i(y)$ darstellen lässt und für die einzelnen $F_i(y)$ Generierungsmethoden vorhanden sind, so können mit dem folgenden Algorithmus Realisierung von X generiert werden.

$x = 0$;
for $i = 1$ to I do
  ziehe ZZ y aus $F_i(y)$;
  $x = x + y$;
end for
return(x) ;

Das Verfahren bezeichnet man als *Konvolutionsmethode*. Ein Beispiel für eine Verteilung, die sich mit der Konvolutionsmethode behandeln lässt, ist die Erlang-k-Verteilung (siehe Abbildung 3.4). In diesem Fall sind alle $F_i(y)$ identisch, es muss aber natürlich in jedem Schleifendurchlauf eine neue Zufallszahl gezogen werden.

Falls sich eine Zufallsvariable X als konvexe Linearkombination von Zufallsvariablen Y_i $(i = 1, \ldots, I)$ mit Gewichten p_i und Verteilungsfunktionen $F_i(y)$ darstellen lässt und für die einzelnen $F_i(y)$ Generierungsmethoden vorhanden sind, so können mit dem folgenden Algorithmus Realisierung von $X = \sum_{i=1}^{I} p_i \cdot Y_i$ generiert werden.

generiere i gemäß Verteilung p_i ;
ziehe ZZ x aus $F_i(x)$;
return(x) ;

Das Verfahren bezeichnet man als *Kompositionsverfahren*. Es kann zum Beispiel zur Generierung hyperexponentiell-verteilter Zufallsvariablen (siehe Abbildung 3.4) eingesetzt werden.

Die bisher vorgestellten Methoden generieren Zufallszahlen direkt. Dies bedeutet, dass zur Generierung einer Zufallszahl der gewünschten Verteilung eine vorgegebene Anzahl von $[0, 1)$-verteilten Zufallszahlen generiert werden muss. Bei der nun vorgestellten *Verwerfungsmethode* ist dies nicht der Fall. Es sollen Realisierungen einer Zufallsvariablen X mit bekannter Dichtefunktion $f_X(x)$ gezogen werden. Weiterhin existiere eine Zufallsvariable Y mit Dichtefunktion $f_Y(x)$ für die ein Generierungsverfahren bekannt ist und ein Parameter α, so dass $\alpha \cdot f_Y(x) \geq f_X(x)$ für alle x ist. Da $f_X(x)$ und $f_Y(x)$ Dichtefunktionen sind, gilt $\int_{-\infty}^{\infty} f_X(x)dx = \int_{-\infty}^{\infty} f_Y(x)dx = 1$, so dass

$\alpha > 1$ sein muss, wenn $f_X(x)$ und $f_Y(x)$ nicht identisch sind. Wenn die Voraussetzungen erfüllt sind, kann folgender Generierungsalgorithmus eingesetzt werden:

repeat
  ziehe ZZ y gemäß $f_Y(x)$;
  ziehe ZZ x aus einer $[0, \alpha \cdot f_Y(y))$-Gleichverteilung
until $x \leq f_X(y)$;
return(y) ;

Das Ziehen einer Zufallszahl aus einer $[a, b)$-Gleichverteilung ($a < b$) wird einfach dadurch realisiert, dass eine $[0, 1)$-verteilten Zufallszahl u mittels $a + u \cdot (b - a)$ transformiert wird.

Wir wollen uns kurz den Beweis anschauen, warum mit dem Algorithmus wirklich Zufallszahlen der gesuchten Verteilung generiert werden. Dazu muss gezeigt werden, dass $P[X \leq x] = \int_{-\infty}^{x} f_X(y)dy$ für jedes x gilt. Da die beiden Zufallszahlen y und x unabhängig voneinander sind, gilt $P[X \leq x] = P[Y \leq x|A]$ wobei A die Bedingung ausdrückt, dass y akzeptiert wird. Die bedingte Wahrscheinlichkeit kann wie folgt dargestellt werden (siehe Seite 66)

$$P[Y \leq x|A] = \frac{P[A \cap Y \leq x]}{P[A]} \tag{3.3}$$

Für die Wahrscheinlichkeit $P[A]$ gilt

$$P[A|Y = y] = P\left[u \leq \frac{f_X(y)}{\alpha \cdot f_Y(y)}\right] = \frac{f_X(y)}{\alpha \cdot f_Y(y)}$$

wobei u eine $[0, 1)$-gleichverteilte Zufallszahl ist und damit gilt

$$P[A] = \int_{-\infty}^{\infty} \frac{f_X(y)}{\alpha \cdot f_Y(y)} \cdot f_Y(y)dy = \frac{1}{\alpha}$$

Für den Term im Zähler von (3.3) gilt

$$\begin{aligned} P[A \cap Y \leq x] &= \int_{-\infty}^{x} P[A \cap Y \leq x|Y = y] \cdot f_Y(y)dy \\ &= \int_{-\infty}^{x} P[A|Y = y] \cdot f_Y(y)dy \\ &= \frac{1}{\alpha} \int_{-\infty}^{x} f_X(y)dy \end{aligned}$$

Damit gilt auch

$$P[X \leq x] = P[Y \leq x|A] = \frac{P[A \cap Y \leq x]}{P[A]} = \frac{\alpha^{-1} \cdot \int_{-\infty}^{x} f_X(y)dy}{\alpha^{-1}} = \int_{-\infty}^{x} f_X(y)dy$$

Die Anwendung der Verwerfungsmethode soll nun kurz am Beispiel einer $\beta(2, 4)$-Verteilung mit $f(x) = 20 \cdot x \cdot (1 - x)^3$ falls $0 \leq x \leq 1$ und 0 sonst vorgeführt werden. Die Dichtefunktion ist durch ein Rechteck der Höhe 2.11 beschränkt, wie in Abbildung 3.13 gezeigt. Damit ergibt sich folgender Generierungsalgorithmus

repeat
  ziehe y aus einer $[0, 1)$-Gleichverteilung ;
  ziehe x aus einer $[0, 2.11)$-Gleichverteilung ;
until $x \leq 20 \cdot y \cdot (1 - y)^3$;

Die Effizienz der Verwerfungsmethode hängt von den zwei folgenden Punkten ab.

1. Der Effizienz der Generierung von Zufallszahlen mit Dichtefunktion $f_Y(x)$
2. Der Wahrscheinlichkeit, dass eine Zufallszahl akzeptiert wird

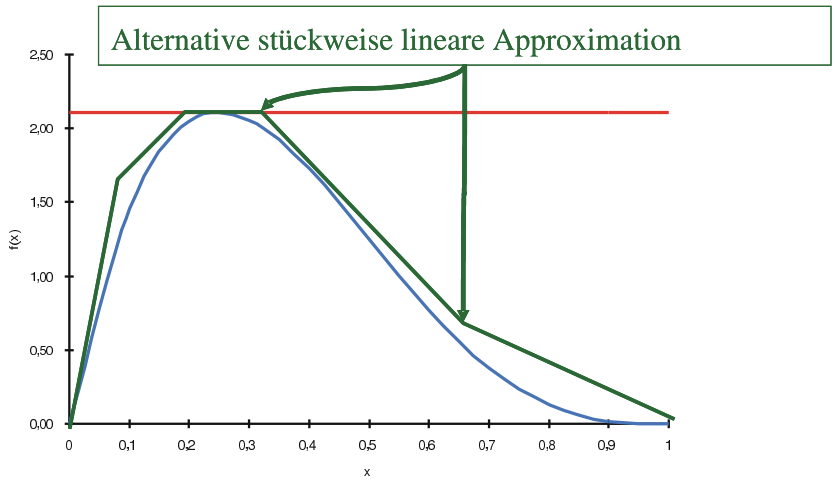


Abbildung 3.13: Anwendung der Verwerfungsmethode am Beispiel einer $\beta(2,4)$-Verteilung.

Die Wahrscheinlichkeit in Punkt 2 lautet $1 - \int_{-\infty}^{\infty}(\alpha \cdot f_Y(x) - f_X(x))dx/\alpha$ und entspricht der Differenz der Funktionen $f_X(x)$ und $\alpha \cdot f_Y(x)$. Durch eine bessere Anpassung der dominierenden Funktion $\alpha \cdot f_Y(x)$ an $f_X(x)$ kann die Wahrscheinlichkeit der Verwerfung reduziert werden. Wird dazu eine komplexere Funktion $f_Y(x)$ verwendet, so steigt dabei in der Regel der Aufwand in Schritt 1.

Für das Beispiel der $\beta(2,4)$-Verteilung könnte statt eines Rechtecks auch ein stückweise lineare Approximation der Dichtefunktion verwendet werden, wie auch in Abbildung 3.13 angedeutet. Dadurch lässt sich die Verwerfungswahrscheinlichkeit deutlich reduzieren, die Generierung von y erfordert aber mehr Aufwand.

Die Generierung von normalverteilten Zufallszahlen ist von großer praktischer Bedeutung, kann mit den bisher vorgestellten Methoden aber nicht effizient durchgeführt werden. Die Umkehrfunktion $F^{-1}(x)$ der Verteilungsfunktion existiert nicht in geschlossener Form, da für die Verteilungsfunktion nicht einmal eine geschlossene Darstellung existiert. Auch die Verwerfungsmethode ist nicht effizient anwendbar, da keine Dichtefunktion bekannt ist, aus der effizient Zufallszahlen generiert werden können und die gleichzeitig genügend nahe an der Dichtefunktion der Normalverteilung liegt. Es gibt aber mehrere Methoden um normalverteilte Zufallszahlen zu generieren. Zwei dieser Methoden sollen kurz vorgestellt werden.

Es sei noch einmal angemerkt, dass es ausreicht $N(0,1)$-verteilte Zufallszahlen zu generieren, da für $X \sim N(0,1)$ die Zufallsvariable $Y = \mu + \sigma \cdot X$ aus einer $N(\mu, \sigma^2)$-Verteilung stammt. Die erste Generierungsmethode basiert auf dem zentralen Grenzwertsatz. Nach dem zentralen Grenzwertsatz konvergiert die Summe unabhängig identisch verteilter Zufallszahlen gegen eine Normalverteilung. Diese Konvergenz ist besonders schnell für gleichverteilte Zufallszahlen. Ein einfacher Ansatz besteht nun darin, einfach n $[0,1)$-verteilte Zufallszahlen zu addieren, davon $n/2$ (den Erwartungswert) abzuziehen und durch $\sqrt{n/12}$ (die Standardabweichung) zu dividieren. Das Ergebnis konvergiert für wachsendes n gegen eine $N(0,1)$-Verteilung. Also eine $N(0,1)$-verteilte Zufallszahl x wird nach der folgenden Formel erzeugt.

$$x = \frac{\left(\sum_{i=1}^{n} u_i\right) - \frac{n}{2}}{\sqrt{n/12}}$$

wobei u_i aus eine $[0,1)$-Verteilung gezogen werden. Oft wird $n = 12$ gewählt, was ausreichend ist, um eine relativ gute Approximation der Normalverteilung zu erreichen. Offensichtlich können in diesem Fall keine Werte größer 6 oder kleiner -6 erzeugt werden. Die Wahrscheinlichkeit, dass eine $N(0,1)$-Verteilung einen Wert außerhalb des Intervalls $[-6,6]$ annimmt liegt aber bei ungefähr

$5.7 \cdot 10^{-7}$ und ist damit für die meisten Anwendungen zu vernachlässigen. Der Nachteil dieser Art der Generierung von normalverteilten Zufallszahlen ist, dass für eine Zufallszahl aus $N(0,1)$ 12 Zufallszahlen aus der $[0,1)$-Gleichverteilung erzeugt werden müssen. Dies ist natürlich aufwändig.

Eine andere Methode zur Generierung von $N(0,1)$-verteilten Zufallszahlen geht auf Box und Muller (1958) zurück (siehe auch Law and Kelton [2000, S. 465]). Seien u_1 und u_2 zwei $(0,1)$-gleichverteilte Zufallszahlen, dann sind

$$x_1 = \cos(2 \cdot \pi \cdot u_1) \cdot \sqrt{-2 \cdot \ln(u_2)} \quad \text{und} \quad x_2 = \sin(2 \cdot \pi \cdot u_1) \cdot \sqrt{-2 \cdot \ln(u_2)}$$

zwei unabhängig identisch $N(0,1)$-verteilte Zufallszahlen. Man benötigt damit zwei gleichverteilte Zufallsvariablen, um zwei normalverteilte Zufallszahlen zu erzeugen. Man kann allerdings zeigen, dass die Unabhängigkeit von x_1 und x_2 nicht gegeben ist, wenn u_1 und u_2 zwei konsekutive Werte aus einem LCG sind. Es sollten also entweder unabhängige Ströme eines LCGs zu Generierung von u_1 und u_2 verwendet werden oder eine andere Methode zur Generierung. Unabhängige Ströme entstehen dadurch, dass mit unterschiedlichen Saaten $x_0^{(1)}$ und $x_0^{(2)}$ begonnen wird und u_1, u_2 aus unterschiedlichen Sequenzen gezogen werden. Die Saaten sollten dabei so gewählt werden, dass die resultierenden Sequenzen sich möglichst spät überlappen. In den meisten Generatoren sind Mechanismen vorgesehen, um adäquate Saaten zu generieren, wie wir später noch sehen werden.

3.2.3.3 Generierung abhängiger Zufallszahlen

Bisher haben wir die Generierung von unabhängigen Zufallszahlen betrachtet und in den meisten Simulationsmodellen werden auch unabhängige Zufallszahlen eingesetzt. Viele zufällige Prozesse in der Realität sind aber nicht unabhängig. So weist die Größe und das Gewicht eines Menschen sicherlich eine Korrelation auf, genauso wie die Temperatur und die Regenmenge oder die Zahl Ein-/Ausgabeoperationen und der CPU-Zeitbedarf eines Jobs auf einem Rechner. Die Annahme von unabhängigen Zufallsvariablen in der Simulation, für Prozesse, die in der Realität korreliert sind, kann zu starken Verfälschungen führen. So sind zum Beispiel Zwischenankunftszeiten von Nachrichten an einem Router in einem Rechnernetz stark positiv korreliert. Wird im Simulationsmodell des Routers diese Korrelation nicht beachtet und die Pufferbelegung analysiert, so liefert das Modell eine deutlich geringere Wahrscheinlichkeit eines Pufferüberlaufs als in der Realität beobachtet werden kann und führt damit zu einer Unterdimensionierung des Routers. Die auftretenden Unterschiede können dabei mehrere Größenordnungen betragen, so dass die Ergebnisse der Simulation praktisch wertlos sind.

Wir wollen die Generierung abhängiger Zufallsvariablen nur sehr kurz betrachten. Es soll allerdings darauf hingewiesen werden, dass vor der Generierung zwei andere Probleme auftreten. Zum einen muss die Abhängigkeit geschätzt werden, was nicht einfach ist und gerade bei starken Abhängigkeiten sehr viele Beobachtungen erfordert. Zum anderen muss die resultierende Verteilung kompakt dargestellt werden, was in vielen Fällen ebenfalls problematisch ist.

Formal liegt ein Zufallsvektor $\mathbf{X} = (X_1, \ldots, X_d)$ mit Verteilungsfunktion $F_{X_1,\ldots,X_d}(x_1, \ldots, x_d)$ vor. Wenn sich die Verteilungsfunktion als Menge von d bedingten Verteilung der Form $F_i(x_i|x_1, \ldots, x_{i-1})$ darstellen lässt, so ist ist die Generierung relativ einfach. Es muss für jedes $F_i(x_i|x_1, \ldots, x_{i-1})$ ein geeigneter Generator gefunden werden und anschließend kann x_1 gemäß $F_1(x)$ generiert werden, anschließend x_2 gemäß $F_2(x|x_1)$ usw. Bis schließlich alle x_i vorhanden sind und der resultierende Vektor die gesuchten Zufallszahlen enthält.

Als einfaches Beispiel soll die Generierung von Zufallszahlen aus einer bivariaten Normalverteilung betrachtet werden. X_1 und X_2 seien zwei korrelierte normalverteilte Zufallsvariablen mit Erwartungswerten μ_i, Standardabweichungen σ_i und Korrelationskoeffizient $\rho = C(X_1, X_2)/(\sigma_1 \cdot \sigma_2)$. Dann können mit dem folgenden Algorithmus Realisierungen generiert werden.

1. Erzeuge z_1 und z_2 als unabhängige $N(0,1)$-verteilte Zufallszahlen.
2. $x_1 = \mu_1 + \sigma_1 \cdot z_1$
3. $x_2 = \mu_2 + \sigma_2 \cdot (\rho \cdot z_1 + \sqrt{(1-\rho^2)} \cdot z_2)$

x_1 und x_2 sind dann Realisierungen aus einer bivariaten Normalverteilung. Der beschriebene Ansatz kann auf multivariate Normalverteilungen erweitert werden.

In der Praxis werden oft stochastische Prozesse zur Modellierung korrelierter Vorgänge eingesetzt. Die Behandlung dieser Thematik geht allerdings über den Stoff der Vorlesung hinaus (siehe Law and Kelton [2000, Kap. 8.5], Rubinstein and Melamed [1998, Kap. 2.3, 2.7, 2.8]).

Kapitel 4

Modellierung von Eingabedaten

Nachdem wir die Methoden zur Beschreibung und Realisierung von Zufallszahlen in Simulationsmodellen kennen gelernt haben, stellt sich natürlich die Frage, wie man an die Verteilungen zur Beschreibung der Parameter eines Simulationsmodells kommt. Dieser Frage wird in diesem Abschnitt nachgegangen. Es hat sich herausgestellt, dass viele reale Vorgänge durch Zufallsvariablen oder stochastische Prozesse in einem Modell adäquat abgebildet werden können. Gleichzeitig sollte klar geworden sein, dass eine genügend genaue Modellierung realer Abläufe notwendig ist, um eine ausreichende Abbildungsgüte des Modells zu erreichen.

Es gibt zwei wesentliche Quellen zur Datengewinnung. Die erste Quelle ist das a priori Wissen, welches vorhanden ist. So können unter Umständen Resultate aus vorherigen oder ähnlichen Modellierungen wiederverwendet werden oder es kann auf die vorhandene Erfahrung oder die Theorie zurückgegriffen werden. So ist aus der Theorie bekannt, dass gewisse Prozesse sich mit bestimmten Verteilungen gut modellieren lassen. Beispiele sind Weibull-Verteilungen für das Ausfallverhalten von Komponenten, Normalverteilungen für viele Größen aus einer großen Grundgesamtheit. Man muss allerdings darauf achten, ob die Bedingungen, die bisher galten, auch für die neue Modellierung gelten. Ein Beispiel wäre der Ankunftsprozess von Verbindungswünschen an einer Vermittlungsstelle. In der Vergangenheit konnte dieser Prozess sehr gut durch einen Poisson-Prozess beschrieben werden. Durch die Einführung der automatischen Wahlwiederholung und die Verwendung der Telefonleitungen für vielschichtige Dienste, änderte sich der Ankunftsprozess, so dass die Modellierung als Poisson-Prozess nicht länger adäquat ist.

Die zweite Quelle bei der Modellierung sind natürlich Daten aus Messungen, also Stichproben der zu modellierenden Größen. Im Idealfall werden die Messungen genau an dem System vorgenommen, welches durch das Modell abgebildet wird. Oftmals sollen aber gerade Systeme modelliert werden, die so nicht in der Realität vorhanden sind. Deshalb müssen Daten an ähnlichen Systemen oder mit Hilfe vorhandener und als gut befundener Modelle gewonnen werden. Wenn feststeht wo und wie Daten erhoben werden sollen, müssen die folgenden Schritte durchgeführt werden, um zu einer Repräsentation im Modell zu gelangen:

1. Die Daten müssen erhoben und zur Auswertung aufbereitet werden.
2. Es ist eine Entscheidung zu fällen, wie die gemessenen Daten im Modell dargestellt werden. Dazu existieren die folgenden Möglichkeiten:
   (a) Es wird eine Konstante zur Darstellung im Modell verwendet, der zugehörige Parameter ist damit deterministisch.
   (b) Man benutzt eine diskrete empirische Verteilung zur Repräsentation der Daten im Modell.
   (c) Man benutzt eine kontinuierliche empirische Verteilung zur Repräsentation der Daten im Modell.
   (d) Man benutzt eine stochastische Verteilung zur Repräsentation der Daten im Modell. Falls 2.d gewählt wurde, müssen noch folgende Schritte ausgeführt werden.

3. Auswahl des zu verwendenden Verteilungstyps.
4. Schätzung der Verteilungsparameter aus der Stichprobe.
5. Überprüfung der Passgüte der gewählten Verteilung durch einen Anpassungstest.

Die einzelnen Schritte sollen nun etwas näher beschrieben werden.

4.1 Methoden zur Datenerhebung

Die *Sammlung und Messung von Daten* ist ein aufwändiges und oft frustrierendes Unterfangen, dessen Zeitbedarf bei den meisten Modellierungsprojekten deutlich unterschätzt wird. Da die erhobenen Daten aber die Grundlage für die weiteren Schritte sind, können Nachlässigkeiten an dieser Stelle schnell dazu führen, dass die Realität nur sehr ungenau im Modell abgebildet wird. Insgesamt gilt das GIGO-Prinzip (Garbage in Garbage out), aus schlechten Daten können keine vernünftigen Modelle entstehen.

Die zentrale Frage lautet, was sind gute Daten oder was ist eine gute Stichprobe. Dies lässt sich nicht allgemein beantworten und hängt sehr stark von der Anwendung ab. Insgesamt ist das Ziel, auf Basis einer endlichen Stichprobe das "übliche Verhalten" zu beschreiben. Die Stichprobe muss also repräsentativ sein. Da viele Abläufe stark schwanken, ist vorab zu definieren, für welche Situationen das Verhalten repräsentativ sein soll. Wird zum Beispiel das Verkehrsaufkommen auf einer Straße untersucht, so kann man starke Unterschiede beim Verkehrsaufkommen in Abhängigkeit von der Uhrzeit und dem Wochentag beobachten. Je nach Ziel der Modellierung müssen Daten immer an bestimmten Tagen zu festgelegten Uhrzeiten erhoben werden, wenn zum Beispiel die Phase hohen Verkehrsaufkommens analysiert werden soll, oder es muss kontinuierlich gemessen werden, wenn der gesamte Ablauf nachgebildet wird. Auch an dieser Stelle ist also die Festlegung des Ziels der Modellierung von zentraler Bedeutung.

Bei der Datenerhebung können eine Reihe von Problemen auftreten. Dies gilt insbesondere, wenn auf Basis vorhandener Daten modelliert werden soll. Gerade bei Simulationsprojekten, die von externen Dienstleistern durchgeführt werden, was in der Praxis oft der Fall ist, muss dieser Weg gegangen werden, da eine unbeschränkte Datenerhebung nicht möglich oder nicht gewünscht ist. So wird und darf ein Telekommunikationsunternehmen keine detaillierte Erhebung der Verbindungsdaten erlauben. Dem stehen Datenschutz- und Wettbewerbsgründe entgegen. Ähnliche Probleme treten inzwischen bei der Datenerhebung im Internet auf. So geben die meisten Betreiber viel frequentierter Server keine detaillierten Zugriffsstatistiken mehr heraus, da befürchtet wird, dass aus diesen Daten Wettbewerber zu viele Informationen bekommen.

Bei vorhandenen Daten kann es vorkommen, dass zu wenige Daten vorliegen. Der Stichprobenumfang ist zu gering oder die Daten sind nur als summarische Statistiken vorhanden. Im Extremfall liefern die Daten nur qualitative Informationen, aus denen sich keine quantitativen Größen ableiten lassen. Es gibt durchaus auch die Situation, dass zu viele Daten vorhandenen sind. Dies ist gerade dann der Fall, wenn automatische Messungen ablaufen oder Traces geschrieben werden. Beispiele findet man in der Informatik im Rechnernetzbereich, wo die Messung des Verkehrsaufkommens zu immensen Datenmengen führt, aber auch in den Naturwissenschaften, insbesondere der Physik, existieren zahlreiche Experimente, bei denen das Datenaufkommen selbst mit heutigen Rechnern kaum analysierbar und speicherbar ist. Das Problem bei zu großen Datenmengen liegt darin, wesentliche Informationen zu extrahieren, was unter Umständen einen sehr hohen Aufwand erfordert oder mit verfügbaren Techniken nicht in vertretbarer Zeit möglich ist. Ein weiteres Problem ist die Verfügbarkeit falscher Daten. So müssen Daten zu der zu modellierenden Situation passen. Gerade bei Systemen, die sich in der Planungsphase befinden, muss auf Daten zurückgegriffen werden, die aus anderen Systemen stammen und deshalb oft nicht zur eigentlichen Zielstellung passen.

Die folgenden Hinweise dienen dazu die Datenerhebung zu unterstützen, können aber auch verwendet werden, um die Qualität vorhandener Daten zu bewerten.

1. Vor der eigentlichen Datenerhebung sollten einige Vorläufe stattfinden, um das Erhebungsintervall, die Auflösung der Messung und den Stichprobenumfang festzulegen. So ist bei stark

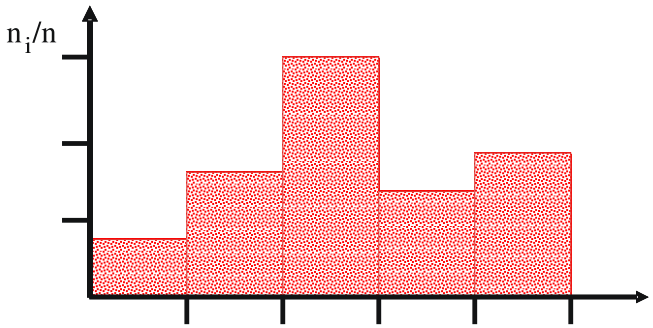


Abbildung 4.1: Beispiel eines Histogramms.

schwankenden oder korrelierten Daten ein größerer Stichprobenumfang notwendig als bei unabhängigen und wenig variierenden Daten.

2. Falls möglich, sollten die Daten während der Erhebung analysiert werden, um so relevante von irrelevanten Informationen zu unterschieden.

3. Wenn mehrere Stichproben kombiniert werden, so ist sicherzustellen, dass die Stichproben homogen sind, d.h. aus identischen Verteilungen stammen. Dies kann mit entsprechenden Testverfahren überprüft werden.

4. Es ist darauf zu achten, inwieweit Daten zensiert (censored data) sind. Man spricht dann von zensierten Daten, wenn auf Grund der Datenerhebung/Messumgebung gewisse Effekte nicht beobachtbar sind. Ein Beispiel ist die Beobachtung der Ausfallzeit von Komponenten in einem Intervall $[0, T)$. In diesem Fall können offensichtlich keine Ausfallzeiten größer T beobachtet werden. Eine andere Art der Zensur tritt auf, wenn nur zu diskreten Zeitpunkten gemessen wird und dadurch Werte aufgerundet werden.

5. Die erhobenen Daten sollten auf Korreliertheit bzw. Unkorreliertheit untersucht werden. Die dazu vorhandenen Methoden werden später vorgestellt.

6. Bei der Messung sollte zwischen Eingabe- und Ausgabedaten unterschieden werden. Zur Modellparametrisierung werden Eingabedaten benötigt, während Ausgabedaten zur Validierung eingesetzt werden. In einem Warteschlangennetz ist zum Beispiel die Zwischenankunftszeit eine Eingabegröße, während die Verzögerungszeit eine Ausgabegröße ist.

4.2 Repräsentation und Bewertung von Daten

Nachdem Daten vorhanden sind, ist zu entscheiden, wie diese im Modell repräsentiert werden. Im Wesentlichen geht es darum, die Struktur und die Abhängigkeiten der Daten zu erkennen. Dazu existieren zwei Ansätze, nämlich Maßzahlen und graphische Darstellungen. Maßzahlen sind im Wesentlichen Schätzungen von Momenten, Quantilen oder Korrelationskoeffizienten. Die zugehörigen Schätzer werden später beschrieben. An dieser Stelle sollen graphische Darstellungen vorgestellt werden. Sei dazu $(x_1, \ldots, x_n)$ eine Stichprobe und $(y_1, \ldots, y_n)$ sei die nach Größe geordnete Stichprobe (d.h. $y_i \leq y_{i+1}$). Eine graphische Darstellung gibt einen visuellen Eindruck der Stichprobe und muss vom Modellierer bewertet werden. Damit hat die Bewertung graphischer Darstellungen auch immer einen subjektiven Aspekt.

Die naheliegenste Form der Visualisierung sind Histogramme als Approximation der Dichtefunktion. Ein Histogramm wird dadurch erstellt, dass der Datenbereich, unter Umständen nach Elimination von Ausreißern, in gleich große Abschnitte/Zellen unterteilt wird. Die Daten werden dann den Zellen zugeordnet. Anschließend wird das Histogramm graphisch dargestellt, wobei die Höhe einer Zelle proportional zur Anzahl der Werte in der Zelle ist. Als freier Parameter der

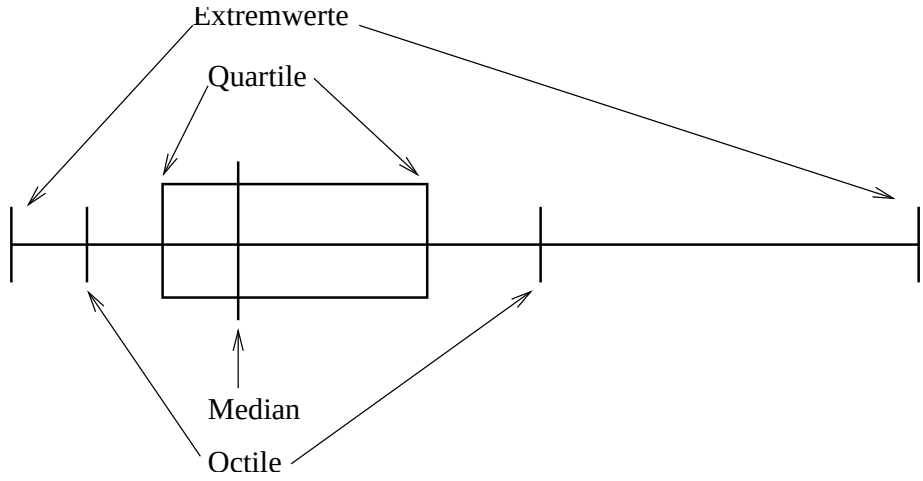


Abbildung 4.2: Beispiel für einen Box-Plot.

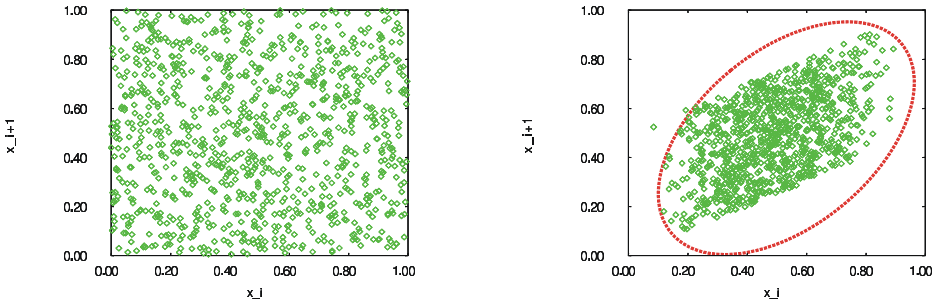


Abbildung 4.3: Beispiele für Tupel konsekutiver Werte aus einer Stichprobe.

Histogrammdarstellung bleibt die Intervallbreite. Leider hängt der visuelle Eindruck manchmal von der Intervallbreite ab. Deshalb sollten immer mehrere Intervallbreiten zur Bewertung der Daten ausprobiert werden. Ist die Intervallbreite zu groß, so werden viele Effekte heraus gemittelt, während eine zu kleine Intervallbreite, mit nur wenigen Werten pro Zelle, zu starken statistischen Schwankungen führt.

Eine Alternative zum Histogramm ist die graphische Darstellung von Maßzahlen. Insbesondere Schätzer für die Quantile der Verteilung eignen sich zur Visualisierung. Die Maßzahlen werden mit Hilfe der geordneten Stichprobe $(y_1, \ldots, y_n)$ definiert.

Die folgenden Quantialschätzer sind von besonderer Bedeutung:

- Median y_i mit $i = (n+1)/2$
- Quartile y_j und y_{n-j+1} mit $j = (\lfloor i \rfloor + 1)/2$
- Octile y_k und y_{n-k+1} mit $k = (\lfloor j \rfloor + 1)/2$
- Extremwerte y_1 und y_n

Dabei wird $y_{m+0.5}$ für $m \in \mathbb{N}$ als $0.5 \cdot (y_m + y_{m+1})$ definiert.

Die genannten Maßzahlen können graphisch in einem sogenannten Box-Plot dargestellt werden, wie in Abbildung 4.2 gezeigt. Mit Hilfe von Box-Plots können wichtige Eigenschaften von Stichproben und Verteilungen visualisiert werden. So kann das Verhältnis des Median zu den Extremwerten oder die Symmetrie der Verteilung abgelesen werden. Insbesondere sind Box-Plots unabhängig von frei zu wählenden Parametern.

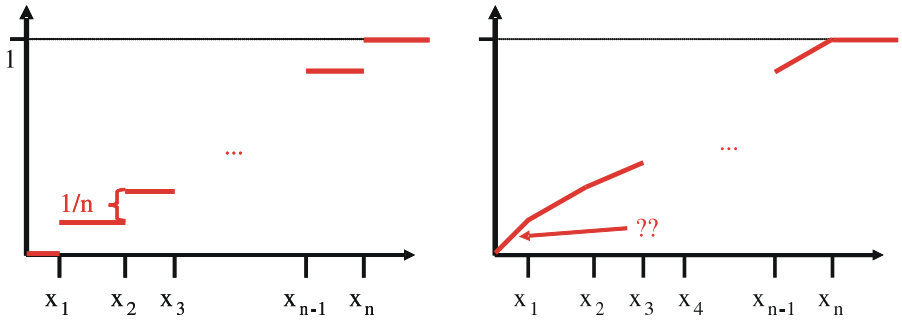


Abbildung 4.4: Diskrete und kontinuierliche empirische Verteilung.

Zur Darstellung der Abhängigkeiten zwischen Daten eignen sich Plots von Datentupeln, wie in Abbildung 4.3 gezeigt. Dabei werden aufeinanderfolgende Werte jeweils als ein Punkt im zweidimensionalen Raum dargestellt. Während in der linken Abbildung keine Abhängigkeiten erkennbar sind, zeigt die rechte Abbildung eine deutlich positive Korrelation. Man kann die Darstellung auch benutzen, um Daten über eine Distanz k (≥ 1) in Beziehung zu setzen.

Nachdem die vorliegenden Daten analysiert wurden, sind sie im Simulationsmodell geeignet zu repräsentieren. Falls keine oder nur sehr geringe Schwankungen in den Daten beobachtet werden, so können sie durch eine Konstante im Modell beschrieben werden. Ansonsten gibt es drei unterschiedliche Möglichkeiten:

- Repräsentation durch eine theoretische Verteilung aus der Menge der verfügbaren Verteilungen. Die Verfügbarkeit von Verteilungen richtet sich u.a. nach der Menge der Verteilungen, die in der verwendeten Simulationssoftware verfügbar ist.
- Repräsentation der Daten durch eine diskrete empirische Verteilung.
- Repräsentation der Daten durch eine kontinuierliche empirische Verteilung.

Bevor wir die Auswahl-Anpassung theoretischer Verteilungen einführen und die Vor- und Nachteile theoretischer/empirischer Verteilungen diskutieren, soll kurz auf empirische Verteilungen zur Datenrepräsentation eingegangen werden.

4.3 Empirische Verteilungen zur Datenrepräsentation

Die Idee der empirischen Verteilung besteht darin, die Daten möglichst direkt als Repräsentanten im Modell zu verwenden und damit kein mathematisches Modell zur Datenbeschreibung zu generieren. Auf extreme Weise wird dies bei der Trace-getriebenen Simulation durchgeführt, bei der genau die Abläufe aus der Realität kopiert werden. Dies bedeutet, dass die verfügbare Datenmenge die maximale Länge des Simulationslaufs bestimmt und stochastische Einflüsse nur bzgl. der nicht durch die Trace-Daten beschriebenen Teile integriert werden können. Empirische Verteilungen benutzen dagegen die Daten zur Generierung von Zufallszahlen. Damit können beliebig lange Simulationsläufe durchgeführt werden und durch unterschiedliche Saaten des Zufallszahlengenerators werden auch unterschiedliche Sequenzen von Zufallszahlen aus den Daten erzeugt. Die beiden grundsätzlichen Varianten empirischer Verteilungen, nämlich die diskrete und die kontinuierliche Version, werden in Abbildung 4.4 dargestellt.

Bei der diskreten empirischen Verteilung werden die Daten als direkte Repräsentanten gewählt. Wenn n Werte in der Stichprobe vorliegen, so wird mit Wahrscheinlichkeit $1/n$ einer der Werte ausgewählt und als Zufallszahl verwendet. Die zugehörige Verteilungsfunktion ist durch eine homogene Treppenfunktion mit Sprungstellen der Höhe $1/n$ gekennzeichnet, sofern alle Werte unterschiedlich sind (d.h. $y_i < y_{i+1}$). Falls $y_i = y_{i+1}$ gilt, so entstehen Sprünge größerer Höhe und entsprechend weniger Stufen. Das Ziehen einer neuen Zufallszahl erfolgt jeweils gedächtnislos.

Alle Charakteristika der verwendeten Verteilung (z.B. Momente, Quantile etc.) entsprechen damit genau den Charakteristika der Stichprobe.

Die kontinuierliche empirische Verteilung benutzt die ermittelten Daten als Stützstellen zur Konstruktion einer Verteilungsfunktion. Die einfachste Form ist eine lineare Interpolation. Man unterscheidet zwischen einer Interpolation zwischen den gemessenen Daten und einer zusätzlichen Extrapolation am Ende und/oder am Anfang. Im ersten Fall wird zwischen den Werten y_{i-1} und y_i eine Gerade mit den Endpunkten $(y_{i-1}, (i-1)/(n-1))$ und $(y_i, i/(n-1))$ gezogen. Im zweiten Fall wird $n-1$ im Nenner durch n oder $n+1$ ersetzt. Ausgehend vom kleinsten und/oder größten Wert kann dann extrapoliert werden, wobei der Endpunkt der Extrapolation zusätzlich festzulegen ist. Wenn man zum Beispiel Zeiten betrachtet, so ist bekannt, dass die generierten Zufallszahlen nur nicht-negativ sein können. Damit bietet es sich an, zwischen 0 und y_1 eine weitere Gerade zu ziehen (siehe rechte Seite von Abbildung 4.4). Kontinuierliche empirische Verteilungen weisen in der Regel andere Charakteristika als die Stichprobe auf. So können sich z.B. die Momente der Stichprobe und der empirischen Verteilung, die aus der Stichprobe generiert wurde, unterscheiden. Zufallszahlen aus kontinuierlichen empirischen Verteilungen werden in zwei Schritten gezogen. Im ersten Schritt wird eine diskrete Zufallszahl $[1, n-1]$ (bzw. $[1, n]$ oder $[1, n+1]$, falls extrapoliert wurde) gezogen. Diese bestimmt das Intervall. Anschließend wird der Wert durch Ziehen einer gleichverteilten Zufallszahl aus $[y_{i-1}, y_i)$ erzeugt.

4.4 Vergleich empirischer und theoretischer Verteilungen

Ob empirische oder theoretische Verteilungen in der Simulation verwendet werden sollen, wird in der Literatur kontrovers diskutiert und hängt letztendlich primär von der Problemstellung und Datenlage ab. Die folgenden Argumente sprechen für die Verwendung empirischer Verteilungen bzw. gegen die Verwendung theoretischer Verteilungen:

- Die Anpassung theoretischer Verteilungen ist immer mit Informationsverlust verbunden. Die Stichprobe beinhaltet die gesamte gemessene Information.
- Die Wahl einer Verteilungsfamilie zur Repräsentation der Daten mit einer theoretischen Verteilung ist unklar.
- Die Schätzung der Verteilungsparameter einer theoretischen Verteilung ist in vielen Fällen nicht robust, so dass kleinere Schwankungen in den Daten zu deutlichen Unterschieden bei den Parametern führen.
- Es gibt theoretische Verteilungen, für die keine effiziente Methode zur Generierung von Zufallszahlen existiert.

Für die Verwendung von theoretischen Verteilungen und damit gegen die Verwendung von empirischen Verteilungen spricht:

- Empirische Verteilungen hängen stark von der Stichprobe ab. Unterschiedliche Stichproben eines Systems können zu unterschiedlichen Verteilungen und damit auch zu deutlichen Unterschieden im Modellverhalten führen. Insbesondere Ausreißer in den Messungen können das Verhalten stark beeinflussen.
- Empirische Verteilungen erlauben die Realisierung von Zufallszahlen in der gemessenen Bandbreite, die Extrapolation von Werten ist schwierig und kann zu Verfälschungen führen.
- Die Darstellung theoretischer Verteilungen ist in den meisten Fällen deutlich kompakter. Für empirische Verteilungen müssen vollständige Stichproben gespeichert werden. Kleine Stichprobenumfänge stellen das Verhalten nicht genau genug dar, während große Stichprobenumfänge den Speicherbedarf erhöhen.

- Die Generierung aus theoretischen Verteilungen ist dann effizienter, wenn die empirische Verteilung sehr viele Stützstellen aufweist.
- Es gibt oftmals theoretisch und empirisch fundierte Gründe, für bestimmte Abläufe einen ganz bestimmten Verteilungstyp zu wählen.
- Die Verwendung theoretischer Verteilungen erlaubt unter Umständen die Anwendung alternativer Analyseansätze, während empirische Verteilungen nur in Kombination mit Simulation einsetzbar sind.

In der Praxis werden in großen Modellen meistens theoretische Verteilungen eingesetzt, da die Verwendung empirischer Verteilungen zu aufwändig bzgl. Datenerhebung und Darstellung der Verteilungen im Simulationsprogramm ist.

4.5 Anpassung theoretischer Verteilungen

Die Anpassung theoretischer Verteilungen erfolgt in den folgenden drei Schritten, die nachfolgend näher erläutert werden.

1. Bestimmung des Verteilungstyps.
2. Schätzung der Parameter.
3. Bestimmung der Anpassungsgüte.

Zur Wahl des Verteilungstyps kann man folgende Szenarien unterscheiden:

a) Der Verteilungstyp ist aus der Theorie bekannt, die Parameter werden aus einer Stichprobe geschätzt.

b) Der Verteilungstyp und die Parameter werden aus einer Stichprobe geschätzt.

c) Es gibt keine theoretischen Erkenntnisse und auch keine Stichprobe.

In den Fällen a) und b) werden jeweils die Schritte 2. und 3. ausgeführt. Fall b) erfordert zusätzlich Schritt 1. Fall c) ist eher unangenehm, da kaum verwertbare Information vorhanden ist. Dieser Fall soll kurz am Ende der Beschreibung der Anpassung von theoretischen Verteilungen behandelt werden.

Begonnen wird mit der Bestimmung des Verteilungstyps. Im Fall a) lässt sich dieser aus der Theorie ableiten. Die Theorie beruht natürlich auf empirischen Erkenntnissen, die man im Laufe der Zeit gewonnen hat. So können gewisse Abläufe in der Realität mit bestimmten Verteilungen in Verbindung gebracht werden. Einige Beispiele dazu:

- Falls eine Zufallsvariable aus einer größeren Anzahl unabhängiger zufälliger Ereignisse resultiert, so könnte sie normalverteilt sein (siehe zentraler Grenzwertsatz).
- Eine Zufallsvariable, die das Minimum einer größeren Anzahl zufälliger Ereignisse ist, könnte Weibull-verteilt sein.
- Eine Zufallsvariable, die zeitlich aufeinander folgende Ereignisse beschreibt, die einzeln mit konstanter Rate auftreten, könnte exponentiell-verteilt sein.
- Eine Zufallsvariable, die das Produkt einer größeren Anzahl zufälliger Einflüsse ist, könnte log-normal-verteilt sein.
- Die Ausfallzeiten technischer Systeme können oft durch Mischungen aus Weibull-Verteilungen repräsentiert werden.

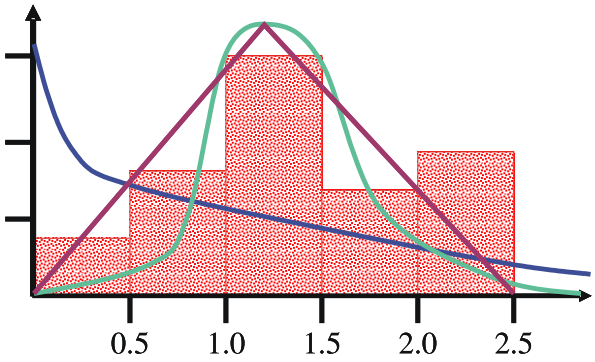


Abbildung 4.5: Histogramm und Dichtefunktionen zur Bewertung der Anpassung.

- Die Zugriffsdauern auf Web-Server können oftmals durch Pareto-Verteilungen repräsentiert werden.
- ...

Theoretische Erkenntnisse erlauben die Identifikation von Verteilungstypen, sollten aber in jedem konkreten Fall hinterfragt und getestet werden.

Im Fall b) muss der Verteilungstyp aus der Stichprobe geschätzt werden. Auch hier ist Vorabinformation über die Struktur und Herkunft der Daten oft sinnvoll, um die Klasse der möglichen Verteilungen einzugrenzen. Weitere Eingrenzungen können dadurch erfolgen, dass Verteilungscharakteristika aus der Stichprobe geschätzt werden (mehr zu Schätzungen später). So lassen sich Mittelwert und Varianz, das Verhältnis von Median zu Erwartungswert oder höhere Momente aus der Stichprobe schätzen und für theoretische Verteilungen berechnen. Durch Vergleich der Werte aus der Stichprobe mit den für eine bestimmte Verteilung erreichbaren Werte, lassen sich einzelne Verteilungen ausschließen. So ist zum Beispiel der Variationskoeffizient $VK(Y) = \sigma(Y)/E(Y)$ einer exponential-verteilten Zufallsvariablen Y immer 1. Falls die Schätzung aus der Stichprobe einen von 1 verschiedenen Wert ergibt, so kann die Exponentialverteilung zur Repräsentation der Daten ausgeschlossen werden. Auf diese Weise lässt sich die Menge der möglichen Verteilungen in der Regel reduzieren, es ist aber meistens keine Festlegung auf einen Verteilungstyp möglich.

Dichtefunktionen lassen sich am besten durch visuellen Vergleich der Histogrammdarstellung der Stichprobe mit dem Verlauf der Dichtefunktion vergleichen. In einem ersten Schritt wird auf Grund von Erfahrung und Wissen dem Histogramm ein Verteilungstyp zugeordnet. Der konkrete Vergleich zwischen Histogramm und Dichtefunktion ist allerdings erst nach der Parameterschätzung möglich. Ausgehend von der Wahl eines Verteilungstyps werden die Parameter geschätzt und dadurch der "beste" Repräsentant aus der Familie von Verteilungen ermittelt. Die zugehörige Dichtefunktion kann anschließend in Kombination mit dem Histogramm dargestellt werden (siehe Abbildung 4.5). Heute wird der Anpassungsprozess durch Software unterstützt. Die Software führt natürlich keinen visuellen Vergleich durch, sondern schätzt die Parameter für alle Verteilungen aus einer Liste von Verteilungen und berechnet anschließend Maßzahlen, die die Qualität der Anpassung widerspiegeln (siehe auch Bewertung der Anpassung in Schritt 3.). Die Verteilung mit der besten Anpassung wird anschließend ausgewählt.

Auch wenn das Vorgehen automatisierbar ist, so ist das Ergebnis keineswegs eindeutig, sondern von der Histogrammdarstellung abhängig. Diese beinhaltet aber eine Reihe von Freiheitsgraden, durch die die Darstellung und damit auch die Verteilungsanpassung beeinflusst wird. Die beiden folgenden Entscheidungen müssen getroffen werden.

- Die Anzahl und Breite der Zellen ist festzulegen.
  - I.d.R. sollten Zellen gleicher Breite gewählt werden, ansonsten muss die Höhe angepasst werden.

  - Die Zahl der Zellen ist so zu wählen, dass
    - die Form der Dichte erkennbar wird und
    - jede Zelle genügend Werte (≥ 10) enthält. Bei weniger Werten ändert sich das Aussehen des Histogramms zu schnell, wenn die Stichprobe nur leicht variiert.

- Der überdeckte Bereich ist zu definieren. Dies bedeutet, dass
  - der Start der ersten Zelle und das Ende der letzten Zelle festgelegt werden müssen und
  - entschieden werden muss, welche Werte als Ausreißer vernachlässigt werden.

Es ist anzuraten, die Anpassung für mehrere Histogrammparameter durchzuführen, um bewerten zu können, inwieweit die erzeugte Verteilung von den Histogrammparametern abhängt.

4.6 Schätzung der Verteilungsparameter

Der nun folgende Schritt ist die Parameterschätzung. Jede Verteilungsfamilie weist Parameter auf. Bei der Exponentialverteilung ist dies die Rate λ, bei der Normalverteilung der Mittelwert μ und die Standardabweichung σ, bei der Dreiecksverteilung, die linke Grenze a, die rechte Grenze b und der Modalwert c. Ziel der Parameterschätzung ist es, die Parameter so zu wählen, dass die Stichprobe durch die Verteilung möglichst gut repräsentiert wird. Zur formalen Darstellung sei $\Theta = (\Theta_1, \ldots, \Theta_p)$ der Parametervektor und $\mathbf{x} = (\mathbf{x_1}, \ldots, \mathbf{x_n})$ die Stichprobe. Unter Verwendung dieser beiden Vektoren sei $f(\mathbf{x}, \Theta)$ eine allgemeine Darstellung der Dichtefunktion mit Parametervektor Θ für Stichprobe $\mathbf{x}$. Θ ist so zu wählen, dass Verteilung und Stichprobe möglichst gut korrespondieren. Es gibt mehrere Methoden zur *Parameterschätzung* aus der Stichprobe. Wir betrachten hier die *Momentenmethode* und die *Maximum-Likelihood-Methode*.

4.6.1 Momentenmethode:

Bei der Momentenmethode werden die Parameter der Verteilung als Funktion der Momente dargestellt. Dies ist für die meisten Verteilungen möglich. Sei $(x_1, \ldots, x_n)$ die konkrete Stichprobe, jeder Wert x_i ist die Realisierung einer Zufallsvariablen X_i, alle X_i seien unabhängig und identisch verteilt. Wir definieren $\tilde{X}^i$ als einen Schätzer für $E(X^i)$ und $\hat{X}^i$ als den konkreten Schätzwert. Für den Schätzer des ersten Moments $\tilde{X}^1$ benutzen wir oft die Schreibweise $\tilde{X}$. Der folgende Schätzer ist erwartungstreu

$$\tilde{X}^i = \frac{1}{n}\sum_{j=1}^{n} (X_j)^i \quad \text{und} \quad \hat{X}^i = \frac{1}{n}\sum_{j=1}^{n} (x_j)^i$$

ist der resultierende Schätzwert. $\tilde{S}^2$ und $\hat{S}^2$ bezeichnen den Schätzer und Schätzwert für die Varianz. Ein erwartungstreuer Schätzer für die Varianz lautet

$$\tilde{S}^2 = \frac{1}{n-1}\sum_{j=1}^{n} \left(X_j - \tilde{X}^1\right)^2$$

Der Schätzer ist erwartungstreu, da

$$\begin{aligned}
E(\tilde{S}^2(n)) &= E\left(\frac{\sum_{i=1}^{n}\left(X_i - \tilde{X}\right)^2}{n-1}\right) &&= \frac{E\left(\sum_{i=1}^{n} X_i^2 - 2n\tilde{X}^2 + n\tilde{X}^2\right)}{n-1} \\
&= \frac{\left(\sum_{i=1}^{n} E(X_i^2) - nE(\tilde{X}^2)\right)}{n-1} &&= \frac{\left(nE(X_i^2) - n\left(\sigma^2(\tilde{X}) + E(\tilde{X})^2\right)\right)}{n-1} \\
&= \frac{n(\sigma^2 + E(X)^2) - n(\sigma^2/n + E(X)^2)}{n-1} &&= \sigma^2
\end{aligned}$$

Die Umformung in der ersten Zeile nutzt die Beziehung $\sum_{i=1}^{n} X_i = n\tilde{X}$. Von der ersten zur zweiten Zeile gelangt man durch Hereinziehen der Erwartungswerte in die Summe (siehe Seite 67). Die

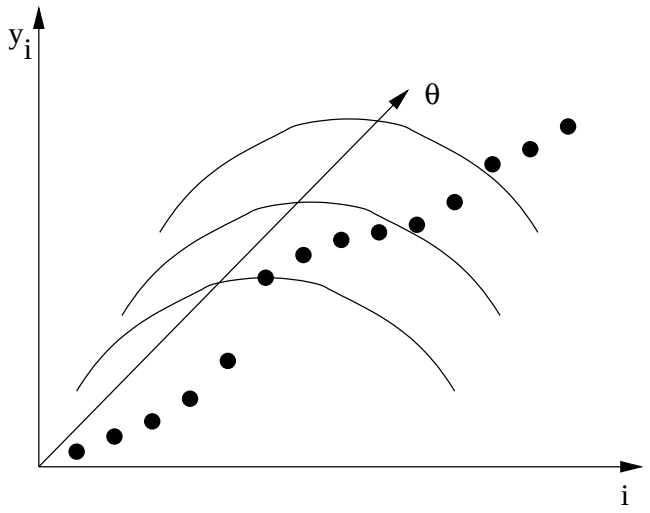


Abbildung 4.6: `Prinzip der ML-Methode.`

Umformung in der zweiten Zeile folgt aus $\sigma^2(X) = E(X^2) - E(X)^2$, die auch für den Übergang von der zweiten in die dritte Zeile verwendet wird. Zusätzlich wird noch $\sigma^2 = n\sigma^2(\tilde{X})$ benutzt. Dies gilt, da

$$\begin{aligned} \sigma^2(\tilde{X}) &= E\left(\left(\tfrac{1}{n}\sum_{i=1}^{n} X_i - E(\tilde{X})\right)^2\right) &&= E\left(\left(\tfrac{1}{n}\sum_{i=1}^{n} X_i - E(X)\right)^2\right) \\ &= \tfrac{1}{n^2}\left(\sum_{i=1}^{n} E((X_i - E(X))^2)\right) &&= \tfrac{1}{n}\sigma^2 \end{aligned}$$

Es soll nun eine Darstellung $\Theta_j = \phi_j(E(X^1), \ldots, E(X^p))$ gefunden werden, so dass Parameter Θ_j als Funktion $\phi_j(.)$ der ersten p Momente dargestellt wird. Ein Momentschätzer $\tilde{\Theta}_j$ entsteht dadurch, dass die Momente durch die Schätzer für die Momente ersetzt werden, also $\tilde{\Theta}_j = \phi_j(\tilde{X}^1, \ldots, \tilde{X}^p)$. Wir betrachten als einfache Beispiele die Exponential- und die Normalverteilung. Für den Parameter λ der Exponentialverteilung gilt $\lambda = (E(X))^{-1}$. Damit ergibt sich der Schätzer $\tilde{\lambda} = 1/\tilde{X}^1 = n/\left(\sum_{j=1}^{n} X_j\right)$. Für die Normalverteilung gilt $\mu = E(X^1)$ und $\sigma = \sqrt{(E(X^2) - E(X^1)^2)}$, mit dem resultierenden Schätzer $\mu = \tilde{X}^1$ und $\sigma = \tilde{S}$.

Die aus der Momentenmethode resultierenden Schätzer sind meistens asymptotisch erwartungstreu und konsistent. Trotzdem sind es oft keine guten Schätzer, da nur die Momente, nicht aber die Form der Verteilungs-/Dichtefunktion berücksichtigt werden. Deshalb sind, wenn vorhanden, die im Folgenden vorgestellten maximum-likelihood-Schätzer in der Regel vorzuziehen.

4.6.2 Maximum-likelihood-Schätzer

Die Basis der maximum-likelihood-Schätzung ist die Suche nach der plausibelsten Lösung. Dies soll zuerst an einem einfachen Beispiel erläutert werden. Wir betrachten dazu eine diskrete Verteilung mit Parameter θ, so dass $p_\theta(x)$ der Wahrscheinlichkeit von x bei Parameter θ entspricht. Wir nehmen nun an, dass die Stichprobe $(x_1, \ldots, x_n)$ festliegt und der Parameter θ variabel ist. Die Likelihoodfunktion ist dann definiert als

$$L(\theta) = p_\theta(x_1) \cdot \ldots \cdot p_\theta(x_n)$$

Ziel der maximum-likelihood (ML)-Methode ist es, Parameter θ so zu wählen, dass $L(\theta)$ maximal wird. Man sucht also $\max_\theta(L(\theta))$. Der Punkt der maximalen Beobachtungswahrscheinlichkeit wird

gesucht. Das Vorgehen kann auch für den Fall von Parametervektoren Θ benutzt werden. In diesem Fall ist ein mehrdimensionales Optimierungsproblem zu lösen.

Das Prinzip der ML-Methode für den kontinuierlichen Fall wird in Abbildung 4.6 dargestellt. Durch Variation von θ verschiebt sich die Dichtefunktion, so dass der Wert der Likelihoodfunktion sich ändert. Im kontinuierlichen Fall ist der Ansatz allerdings weniger intuitiv, da die Wahrscheinlichkeit in jedem Punkt 0 sein kann. Deshalb wird der Wert der Dichtefunktion $f_\theta(x)$ benutzt. Die Likelihoodfunktion lautet dann

$$L(\theta) = \prod_{j=1}^{n} f_\theta(x_j)$$

Gesucht wird $\theta_{\max}$ mit $L(\theta_{\max)} \geq L(\theta)$ für alle θ.

Das Vorgehen soll kurz am Beispiel der Exponentialverteilung erläutert werden. Es gilt

$$L(\lambda) = \prod_{j=1}^{n} \lambda \cdot e^{-\lambda \cdot x_j} = \lambda^n \cdot e^{-\lambda \cdot \sum_{j=1}^{n} x_j}$$

Das Maximum dieses Produktes ist aufwändig zu ermitteln, deshalb macht man sich zu Nutze, dass eine Funktion und deren Logarithmus, sofern er definiert ist, die Extremwerte an denselben Stellen haben. Da die Likelihoodfunktion nicht negativ ist, ist ihre Logarithmus definiert, wenn der Wert größer 0 ist. Im speziellen Fall der Exponentialverteilung ist der Wert der Likelihoodfunktion größer 0 für $0 < \lambda < \infty$. Da dies der Bereich der relevanten Parameterwerte ist, gilt

$$l(\lambda) = \ln(L(\lambda)) = n \cdot \ln \lambda - \lambda \cdot \sum_{j=1}^{n} x_j$$

Die Ableitung von $l(\lambda)$ lautet $l'(\lambda) = n/\lambda - \sum_{j=1}^{n} x_j$. Die Nullstelle der Ableitung ist gerade der ML-Schätzer für λ. Also $\hat{\lambda} = n/\left(\sum_{j=1}^{n} x_j\right)$. In diesem Fall stimmen Momenten- und ML-Schätzer überein. Dies ist natürlich nicht immer so. Für die Normalverteilung liefert die ML-Methode die Schätzer $\tilde{\mu} = \tilde{X}^1$ und $\tilde{\sigma} = \sqrt{(n-1)/n \cdot \tilde{S}^2}$ im Gegensatz zu $\tilde{\sigma} = \tilde{S}$ als Momentenschätzer.

Die ML-Methode kann auch für Parametervektoren Θ angewendet werden. Allgemein wird dann

$$l(\Theta) = \ln(L(\Theta)) = \sum_{j=1}^{n} \ln\left(f_\Theta(x_j)\right)$$

bzgl. des Parametervektors Θ maximiert. Oft hat das resultierende Optimierungsproblem keine geschlossene Lösung und muss deshalb mit einem Verfahren zur nichtlinearen Optimierung (siehe auch Kapitel **??**) gelöst werden. Für viele bekannte Verteilungen sind ML-Schätzer für die Parameter bekannt (siehe Law and Kelton [2000]).

ML-Schätzer haben in der Regel die folgenden Eigenschaften:

- Sie sind asymptotisch erwartungstreu.
- Sie sind konsistent.
- Sie sind asymptotisch normalverteilt. Dies bedeutet, dass mit den Methoden aus Kapitel 5 Konfidenzintervalle für die Parameter bestimmbar sind und damit auch quantifiziert werden kann, wie robust die Schätzung ist.
- Sie sind in der Regel nicht schlechter bzw. besser als die Momentenschätzer.

Aus den genannten Gründen werden in den meisten Fällen ML-Schätzer verwendet, falls solche bekannt sind oder ermittelt werden können.

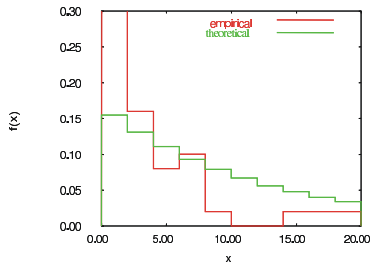


Abbildung 4.7: Histogramm der empirischen Verteilung und der theoretischen Verteilung.

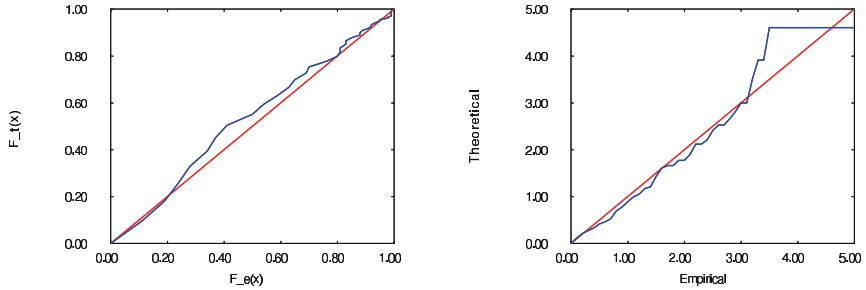


Abbildung 4.8: P-P-Plot und Q-Q-Plot.

4.7 Bestimmung der Anpassungsgüte

Nachdem eine theoretische Verteilung angepasst wurde, ist die Anpassung zu bewerten. Dieser Schritt kann auch verwendet werden, um unterschiedliche Verteilungsanpassungen zu vergleichen. Man unterscheidet zwischen heuristischen Prozeduren und statistischen Testverfahren.

Heuristische Prozeduren vergleichen die Struktur der theoretischen und der empirischen Verteilungs- oder Dichtefunktion. Der Vergleich der Dichtefunktionen erfolgt wieder auf Basis von Histogrammen. Sei dazu k die Anzahl der Intervalle und Δ_i das $i$te Intervall. Wenn n_i die Anzahl der Werte der Stichprobe im $i$ten Intervall ist, so ist $h_i = n_i/n$ die relative Häufigkeit der Stichprobenwerte im Intervall Δ_i. Die Wahrscheinlichkeit einen Wert der theoretischen Verteilung im Intervall Δ_i zu finden lautet $p_i = \int_{x\in\Delta_i} f_t(x)dx$, wobei $f_t(x)$ die Dichtefunktion der theoretischen Verteilung ist. Die Abstandmaße $D = \sum_{i=1}^{k} |h_i - p_i|$ und $D' = \sum_{i=1}^{k}(h_i - p_i)^2$ beschreiben die Passgüte zwischen empirischer und theoretischer Verteilung. Der Wert der Abstandsmaße hängt von der Lage und Größe der Histogrammzellen ab. Es ist auch nicht a priori klar, welcher Wert von D bzw. D' eine gute oder zumindest ausreichende Anpassung beschreibt. Auf Grund des Vergleichs der Werte von D kann aber die Anpassungsgüte unterschiedlicher Verteilungen verglichen werden. Über die Qualität der Anpassung sagt oftmals ein visueller Vergleich der Histogramme, wie in Abbildung 4.7 gezeigt, mehr aus als die Maßzahl.

Neben den Dichtefunktionen können auch die Verteilungsfunktionen verglichen werden. Auf Grund der Struktur von Verteilungsfunktionen ist der rein visuelle Vergleich von empirischer und theoretischer Verteilungsfunktion nicht sehr hilfreich. Stattdessen haben sich $Q-Q$-Plots und $P-P$-Plots als alternative Darstellungsformen eingebürgert. Sei $F_t(x)$ die theoretische Verteilungsfunktion und $F_e(x)$ die empirische Verteilungsfunktion. Für die Werte der empirischen Verteilungsfunktion gilt $F_e(x) = \max_{y_i \le x}(i/n)$. Beim $Q-Q$-Plot werden die Quantile der beiden Verteilungsfunktionen zueinander in Beziehung gesetzt. Sei q_j $(0 < q_j < 1)$ das $j$te dargestellte Quantil, dann wird auf der x-Achse y_i mit $i = \arg\max_k(k/n \le q_i)$ und auf der y-Achse $F_t^{-1}(q_j)$ abgetragen (siehe die rechte Graphik in Abbildung 4.8). Wenn F_t und F_e vollständig korrespondieren und die Kurve für alle Werte von q dargestellt wird, so entsteht eine Strecke durch die Punkte $(0,0)$ und $(1,1)$. Abweichungen zeigen die Unterschiede zwischen beiden Verteilungsfunktionen. Beim $P-P$-Plot werden die Wahrscheinlichkeiten miteinander in Beziehung gesetzt. Für x wird

$F_e(x)$ auf der x-Achse abgetragen und $F_t(x)$ auf der y-Achse. Bei vollständiger Korrespondenz entsteht wieder eine Strecke mit Steigung 1, Abweichungen davon zeigen Abweichungen der Verteilungsfunktionen an. Während $Q-Q$-Plots Unterschiede im hinteren Teil der Verteilungsfunktion (im tail) betonen, zeigen $P-P$-Plots mehr die Unterschiede in der Mitte.

4.7.1 Anpassungstests

Die bisher vorgestellten Vergleichsverfahren geben kein Maß an, um zu entscheiden, wann die theoretische Verteilung die empirischen Daten genau genug modelliert. Auf Grund stochastischer Schwankungen ist zu erwarten, dass zwangsläufig Unterschiede zwischen Stichprobe und theoretischer Verteilung existieren müssen. Eine Möglichkeit Hinweise zu bekommen, wann eine Modellierung ausreichend genau ist, sind Anpassungstests. Wie bei allgemeinen Testverfahren (siehe auch Seite 78ff) wird eine Hypothese H_0 aufgestellt:
H_0: Die Stichprobe $(x_1, \ldots, x_n)$ wurde aus der Verteilung $F_t(x)$ gezogen

Ein Anpassungstest liefert die Antwort, ob H_0 verworfen oder angenommen werden soll. Wie bei allen statistischen Testverfahren ist das Ergebnis mit Fehlern behaftet. Wie oft üblich, wird der Fehler der ersten Art vorgegeben, d.h. die Hypothese wird mit Wahrscheinlichkeit α abgelehnt, obwohl die Stichprobe aus der Verteilung gezogen wurde. Über den Fehler der zweiten Art werden in der Regel keine Angaben gemacht.

Neben diesen grundsätzlichen Grenzen der Testverfahren beobachtet man bei Anpassungstests noch ein zweites Phänomen. So wird für kleine Stichprobenumfänge H_0 meistens angenommen, da zu wenig Information für eine Ablehnung vorhanden ist. Für große Stichprobenumfänge wird dagegen H_0 in fast allen Fällen abgelehnt, da zwangsläufig Unterschiede zwischen empirischen Daten und theoretischer Verteilung existieren. Man sollte sich diesen Schwächen von Testverfahren bewusst sein. Trotzdem bieten Testverfahren in Kombination mit den anderen vorgestellten Verfahren zum Vergleich von empirischer und theoretischer Verteilung ein mächtiges Instrumentarium zur Modellierung von Daten mittels theoretischer Verteilungen. Es sollen im Folgenden kurz die zwei bekanntesten Anpassungtests vorgestellt werden.

4.7.1.1 Chi-Quadrat-Test

Der Chi-Quadrat-Test ist ein sehr altes Testverfahren, welches schon um 1900 entwickelt wurde. Er basiert auf dem formalen Vergleich der Histogramme der empirischen und theoretischen Verteilung. Seien $b_0 < b_1 < \ldots < b_k$ die Intervallgrenzen, so dass $\Delta_i = [b_{i-1}, b_i)$. Wie bereits weiter oben definiert ist $n_i = |\{y_j | y_j \in \Delta_i\}|$ die Anzahl der Element im Intervall Δ_i und $p_i = \int_{x \in \Delta_i} f_t(x)dx$ die Wahrscheinlichkeit der theoretischen Verteilung einen Wert im Intervall Δ_i zu generieren. Als Abstandsmaß zwischen empirischer und theoretischer Verteilung wird $\sum_{i=1}^{k} (n_i - p_i \cdot n)^2 / (n \cdot p_i)$ verwendet. Wie bereits erläutert, steigt die Wahrscheinlichkeit, dass die Stichprobe aus der theoretischen Verteilung generiert wurde, wenn der Wert der Summe kleiner wird. Damit ein Testverfahren definiert werden kann, muss allerdings eine Teststatistik vorhanden sein, d.h. es muss bekannt sein, welche Verteilung der zu testende Wert hat, wenn H_0 gilt.

Bevor eine solche Teststatistik definiert wird, wird kurz die χ^2-Verteilung eingeführt. Seien dazu $Y_1, \ldots, Y_k$ unabhängige $N(0,1)$ verteilte Zufallsvariablen, dann ist $Y = \sum_{i=1}^{k} Y_i^2$ χ^2-verteilt mit k Freiheitsgraden. Die χ^2-Verteilung ist eigentlich eine Familie von Verteilungen (jeder Freiheitsgrad definiert eine Verteilung), die keine explizite funktionale Form hat. Die kritischen Werte liegen aber in vertafelter Form vor und sind in den meisten Statistikbüchern zu finden. $\chi^2_{k,1-\alpha}$ bezeichnet den Wert, der mit Wahrscheinlichkeit α von einer χ^2-Verteilung mit k Freiheitsgraden überschritten wird (siehe auch Abbildung 4.9). Als Teststatistik des Chi-Quadrat-Tests wird

$$d = \sum_{i=1}^{k} \frac{(n_i - n \cdot p_i)^2}{n \cdot p_i}$$

verwendet. d ist die Realisierung einer Zufallsvariablen D. Falls H_0 gilt, dann

- ist D asymptotisch χ^2-verteilt (d.h. für große n ist D approximativ χ^2-verteilt), wobei

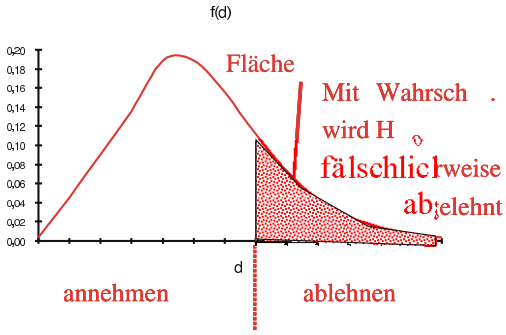


Abbildung 4.9: Skizze des Vorgehens beim Chi-Quadrat-Test.

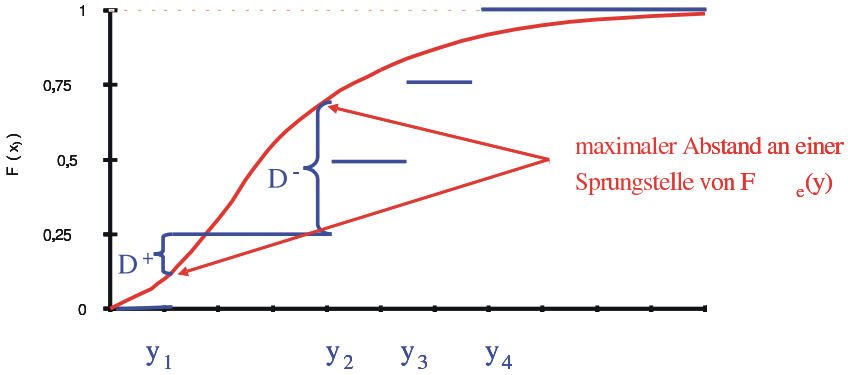


Abbildung 4.10: Skizze zum Vorgehen beim Kolmogorov-Smirnov-Test.

- falls keine Verteilungsparameter aus der Stichprobe geschätzt wurden, die χ^2-Verteilung $k-1$ Freiheitsgrade hat,
- während bei der Schätzung von p Verteilungsparametern aus der Stichprobe, die Zahl der Freiheitsgrade sich auf $k-p-1$ reduziert.

Damit ergibt sich folgendes Vorgehen:

- Festlegung der Intervalle,
- Berechnung von d,
- Vergleich des Wertes von d mit dem kritischen Wert zum gewählten Signifikanzniveau,
  - falls d kleiner als der kritische Wert ist, Hypothese annehmen,
  - ansonsten Hypothese ablehnen.

Der Test beinhaltet als subjektive Komponente die Lage und Breite der Intervalle. Bei kontinuierlichen Verteilungen wird empfohlen, die Intervalle so zu wählen, dass p_i für alle Intervalle identisch ist. Dies bedeutet, dass die Intervallbreiten variieren. Weiterhin sollte p_i so gewählt werden, dass $n \cdot p_i \geq 5$. Bei diskreten Verteilungen sollten sich die Intervalle nach der Lage der diskreten Werte richten. Für große Datenmengen wird als Intervallzahl ein Wert zwischen $\sqrt{n}$ und $n/5$ vorgeschlagen.

4.7.1.2 Kolmogorov-Smirnov-Test

Der Kolmogorov-Smirnov-Test vergleicht die theoretische Verteilungfunktion $F_t(x)$ mit der empirischen Verteilungsfunktion $F_e(x)$. $F_e(x) = \max_{y_i \leq x}(i/n)$ ist eine Treppenfunktion, als Teststatistik

wird der maximale Abstand zwischen $F_t(x)$ und $F_e(x)$ verwendet. Also

$$D_n = \max_x(|F_e(x) - F_t(x)|)$$

Falls nötig, kann das Maximum durch das Supremum ersetzt werden. Da $F_t(x)$ eine monoton nicht fallende Funktion ist und $F_e(x)$ eine Treppenfunktion ist, kann man sich leicht überlegen, dass der maximale Abstand zwischen beiden Funktionen an einer Sprungstelle von $F_e(x)$ angenommen werden muss (siehe auch Abbildung 4.10). Der maximale Abstand kann deshalb wie folgt definiert und ermittelt werden

$$D_n^+ = \max_{1\le i\le n}\{i/n - F_t(y_i)\} \quad D_n^- = \max_{1\le i\le n}\{F_t(y_i) - (i-1)/n\} \quad D_n = \max\left\{D_n^-, D_n^+\right\}$$

Ein großer Wert von D_n deutet auf eine schlechte Anpassung hin. Zur Anwendung des Tests müssen zwei Fälle unterschieden werden:

- Falls keine Verteilungsparameter aus der Stichprobe geschätzt wurden, so ist H_0 zu verwerfen, falls $\left(\sqrt{n} + 0.12 + \frac{0.11}{\sqrt{n}}\right) \cdot D_n \ge c_{1-\alpha}$. Die Werte von $c_{1-\alpha}$ sind vertafelt und in den meisten Statistikbüchern zu finden.
- Falls die Verteilungsparameter aus der Stichprobe geschätzt wurden, so ist die Teststatistik verfälscht und die vertafelten Werte sind zu optimistisch (d.h. H_0 würde zu oft angenommen). In diesem Fall existieren Teststatistiken nur für spezielle Verteilungen, wie die Normalverteilung, Exponentialverteilung oder Weibull-Verteilung.

Chi-Quadrat- und Kolmogorov-Smirnov-Test können beide auch für das Testen von Zufallszahlen verwendet werden.

4.8 Datenmodellierung ohne Stichprobe und theoretische Information

Dies ist der Fall c, der bisher ausgespart wurde. In der Praxis tritt dieser Fall aber leider häufiger bei der Planung von Systemen auf. Eine genaue Modellierung ist in einem solchen Fall natürlich nicht möglich, trotzdem können in der Regel zumindest eingeschränkte Informationen über die Daten gewonnen werden. So kann oftmals zwischen diskreten und kontinuierlichen Werten unterschieden werden und es können minimale und maximale Werte festgelegt werden. Manchmal kann auch ein Mittelwert geschätzt werden. In diesen Fällen, können die Daten dann durch eine Gleichverteilung, Dreiecksverteilung oder Beta-Verteilung im Modell repräsentiert werden.

4.9 Korrelierte Daten und stochastische Prozesse

Bisher wurde immer von unabhängig, identisch verteilten Daten ausgegangen. Die meisten Methoden der Statistik und auch die vorhandene Software zur Datenmodellierung (z.B. Arena Input Analyzer, ExpertFit, etc.) ist auch auf diesen Fall beschränkt. In der Realität zeigt sich aber oftmals, dass die Annahmen nicht gelten. So sind Daten

- korreliert bzgl. der Zeitintervalle
  - Ankunftsprozesse in einem Rechnernetz, die Korrelationen über mehrere Zeitskalen aufweisen,
  - Fehler in einem technischen System, die durch gegenseitige Beeinflussung von Komponenten entstehen,
- korreliert bzgl. verschiedener Messgrößen

  - Größe und Gewicht von Menschen,
  - CPU-Zeitbedarf und Speicherplatzbedarf von Jobs in einem Rechner,

- über einen längeren Zeitraum nicht identisch verteilt sind
  - Ankunftsprozess in einem Restaurant,
  - Zugriffe auf einen Web-Server.

Während der dritte Fall, nämlich zeitlich variierende Prozesse, oft dadurch abgebildet werden kann, dass für unterschiedliche Zeiträume unterschiedliche Verteilungen angepasst werden, muss die Korrelation von Daten direkt bei der Modellierung berücksichtigt werden und hat damit Einfluss auf die stochastischen Modelle. Es reicht nicht mehr aus, Verteilungen anzupassen, sondern es müssen komplexere Modelle, wie Zufallsvektoren, Markovsche Ankunftsprozesse, nichtstationäre Poisson-Prozesse, autoregressive Modelle oder multivariate Normalverteilungen verwendet werden. Während die Generierung von Zufallszahlen aus diesen Verteilungen in der Regel mehr oder weniger problemlos realisiert werden kann, ist die Anpassung der Modelle auf Basis von Stichproben ein komplexes und erst in Ansätzen gelöstes Problem.

An dieser Stelle soll als ein einfaches Beispiel die Schätzung der Parameter einer bivariaten Normalverteilung vorgestellt werden (siehe Seite 86). Seien $(X_{11}, \ldots, X_{1n})$ und $(X_{21}, \ldots, X_{2n})$ die Stichproben der beiden Zufallsvariablen X_1 und X_2. Wir nehmen an, dass aus den beiden Stichproben ableitbar ist, dass die beiden Zufallsvariablen normalverteilt sind. Ferner seinen

$$\tilde{X}_j = \frac{1}{n} \cdot \sum_{i=1}^{n} X_{ji} \quad \text{and} \quad \tilde{S}_j^2 = \frac{1}{n-1} \cdot \sum_{i=1}^{n} \left(X_{ji} - \tilde{X}_j \right)^2$$

die Schätzer für den Erwartungswert und die Varianz ($j = 1, 2$). Der Schätzer für die Kovarianz lautet (siehe Banks et al. [2000, S. 355]

$$\widetilde{C}(X_1, X_2) = \frac{1}{n-1} \cdot \sum_{i=1}^{n} \left(\left(X_{1i} - \tilde{X}_1 \right) \cdot \left(X_{2i} - \tilde{X}_2 \right) \right) = \frac{1}{n-1} \cdot \left(\left(\sum_{i=1}^{n} X_{1i} \cdot X_{2i} \right) - n \cdot \tilde{X}_1 \cdot \tilde{X}_2 \right)$$

und

$$\tilde{\rho} = \frac{\widetilde{C}(X_1, X_2)}{\tilde{S}_1 \cdot \tilde{S}_2}$$

ist der Schätzer für den Korrelationskoeffizienten. Mit diesen Parametern ist die bivariate Normalverteilung vollständig spezifiziert und es können Realisierungen mit den schon vorgestellten Ansätzen zur Generierung erzeugt werden.

Im Prinzip können ähnliche Verfahren auch zur Schätzung von Korrelationen höherer Ordnung eingesetzt werden. Allerdings sind die dabei auftretenden praktischen Probleme nicht zu vernachlässigen, da die Schätzer oft wenig robust und selbst wieder korreliert sind.

Kapitel 5

Auswertung von Simulationsläufen

An dieser Stelle sei noch einmal an das Ziel einer Simulationsstudie erinnert. Es sollen Aussagen über ein System S gewonnen werden, indem ein Modell analysiert wird. Wie herausgearbeitet wurde, soll der Einfluss der kontrollierbaren Eingabegrößen C und teilweise auch der unkontrollierbaren Eingabegrößen U auf die Ausgabegrößen P ermittelt werden. Wir haben gesehen, dass wir Zufallseinflüsse durch deterministische Algorithmen in das Modell integrieren können und die Realisierung des Zufalls damit nur von der Saat des Zufallszahlengenerators abhängt. Wenn wir die Saat des Zufallszahlengenerators als Teil von U betrachten, so soll der Einfluss der Größen C über alle möglichen Saaten des Zufallszahlengenerators ermittelt werden.

Zuerst einmal soll das Vorgehen etwas eingeschränkt werden, indem wir annehmen, dass der Wert eines Leistungsmaßes Y ermittelt werden soll. Ohne dass an dieser Stelle schon festgelegt wird, was Y genau ist bzw. wie es beobachtet wird, können wir davon ausgehen, dass in stochastischen Modellen Y schwankende Verläufe zeigt, d.h. mehrfaches Beobachten wird zu unterschiedlichen Resultaten führen. In der Simulation bedeutet dies, dass durch unterschiedliche Saaten des Zufallszahlengenerators variierende Resultate entstehen. Auch in der Realität können viele Beispiele für schwankende Verläufe von dynamischen Abläufen beobachtet werden. So ist das Verkehrsaufkommen an einer Straßenkreuzung nicht deterministisch, die Ausfallzeiten von technischen Komponenten variieren, das Wachstum von Pflanzen ist auch bei identischen Umgebungsbedingungen für eine Menge von Pflanzen nicht gleich. Um also Aussagen über Y zu treffen, müssen im Prinzip Aussagen über alle möglichen Abläufe gewichtet mit der Wahrscheinlichkeit eines Ablaufs getroffen werden. Aus praktischer Sicht ist diese Anforderung in zweifacher Hinsicht problematisch. Die Zahl der möglichen Abläufe ist oft unendlich und die Wahrscheinlichkeit des Auftretens eines bestimmten Ablaufs ist in der Regel nicht bestimmbar. Die nahe liegende Möglichkeit, alle möglichen Abläufe zu beobachten und die resultierenden Beobachtungen von Y gewichtet aufzusummieren entfällt damit und es müssen Aussagen über alle Abläufe auf der Basis einer endlichen Beobachtung getroffen werden. Solche Aussagen können nur mit gewisser, noch näher zu spezifizierender Wahrscheinlichkeit korrekt sein.

Zentral für die Auswertung von Simulationsläufen ist der Begriff des stochastischen Prozesses.

Definition 5 *Ein stochastischer Prozess $Y(t)$ ist eine Funktion über dem Parameterraum T, deren Resultate Zufallsvariablen sind.*

Der Parameterraum T wird in der Regel als Zeit interpretiert, wobei oft zwischen zeitdiskreten Prozessen ($T = \mathbb{N}$) und zeitkontinuierlichen Prozessen ($T = \mathbb{R}_{\geq 0}$) unterschieden wird. Für einen stochastischen Prozess kann man für jedes $t \in T$ die Momente, Dichtefunktion und Verteilungsfunktion der Zufallsvariablen $Y(t)$ definieren. Eine Realisierung von $Y(t)$ über T bezeichnet man als eine Trajektorie. Einige Beispiele für stochastische Prozesse wären:

- Das Würfeln mit zwei unterscheidbaren Würfeln, so dass $Y(t)$= Y(t-1) + Wert des 1. Würfels - Wert des 2. Würfels, mit $Y(0) = 0$. Der Parameterraum ist diskret und $Y(t)$ kann diskrete Werte im Intervall $[-5t, 5t]$ annehmen.

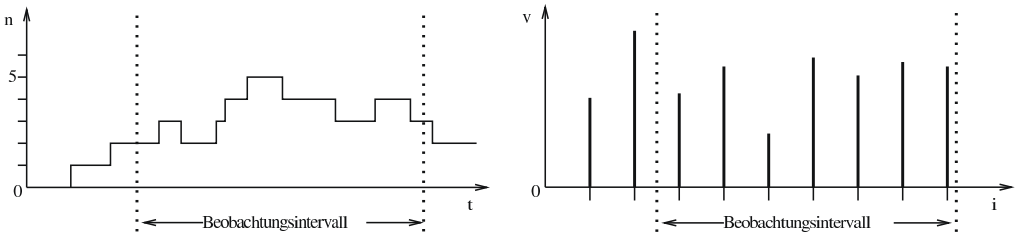


Abbildung 5.1: Typische Beobachtungsszenarien von Simulationsläufen.

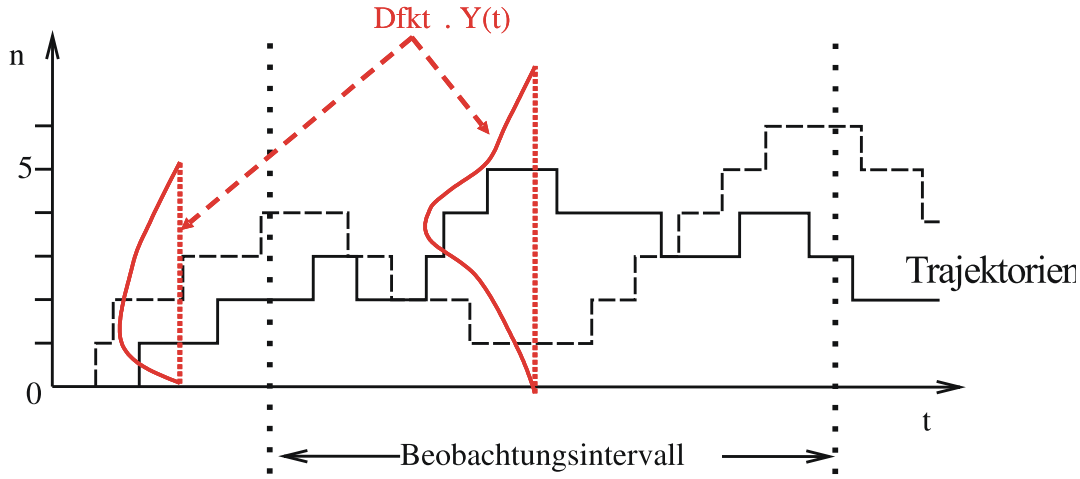


Abbildung 5.2: Beobachtung unterschiedlicher Trajektorien.

- Die Kundenzahl an einem Bankschalter, so dass sich der Wert von $Y(t)$ zu den Ankunfts- und Abgangszeiten von Kunden ändert. In diesem Fall ist der Parameterraum kontinuierlich. Wenn man davon ausgeht, dass in jedem Intervall potenziell eine beliebige Kundenzahl ankommen kann und der Schalterraum unendlich viele Kunden aufnehmen kann, so kann $Y(t)$ Werte aus $\mathbb{N}$ annehmen.

Typische Beobachtungsszenarien der Simulation haben wir bereits vorher kennen gelernt. Abbildung 5.1 zeigt die zwei generellen Beobachtungsszenarien, nämlich die Beobachtung eines Zustands über einen Zeitraum, bei der der Zustand sich zu Ereigniszeitpunkten sprunghaft ändert und die Beobachtung des Schicksals diskreter, temporärer Einheiten der Simulation, bei der jede Einheit einen Wert annimmt.

5.1 Grundlegendes Beobachtungsszenarium

Die Darstellung in Abbildung 5.1 gibt jeweils einen Ablauf wieder, bei stochastischen Simulationen verläuft aber jede Trajektorie etwas anders, so dass eher ein Beobachtungsszenarium, wie in Abbildung 5.2 gezeigt, vorliegt. Bei wiederholten Beobachtungen werden unterschiedliche Abläufe beobachtet. Zu jedem Zeitpunkt $t \in T$ nimmt $Y(t)$ einen Wert gemäß der Dichtefunktion der zugehörigen Zufallsvariablen ein. Der Verlauf der Dichtefunktionen ist in der Abbildung quer zum Ablauf der Trajektorien dargestellt. Wie wir später sehen werden, reicht selbst die Kenntnis der Dichtefunktionen allein nicht aus, um Trajektorien zu charakterisieren, da in den meisten Fällen die Werte einer Trajektorie korreliert sind. Zum Beispiel hängt die Kundenzahl an einem Schalter zum Zeitpunkt t stark von der Kundenzahl zum Zeitpunkt $t - \Delta$ ab, wenn Δ relativ klein ist. Die daraus resultierende Komplexität der Beobachtungslage führt dazu, dass nicht der gesamte stochastische Prozess, sondern nur einzelne Charakteristika per Simulation analysiert werden. Ein

Beispiel wäre die Ermittlung von $E(Y(t))$ für ein $t \in T$.

Als Hilfsmittel für die Analyse können Realisierungen von Y während der Simulation beobachtet werden. Wir nehmen an, dass insgesamt m Beobachtungen während eines Simulationslaufs durchgeführt werden und Y_j die Zufallsvariable ist, die Beobachtung j beschreibt. Der Parameter j kann unterschiedliche Bedeutungen haben.

- Es können Kunden gezählt werden und Y_j beschreibt die Verweilzeit des $j$ten Kunden im System.
- Man kann den Durchsatz eines Modells einer Fertigungsstraße nach j Stunden als $j$te Beobachtung interpretieren.
- Die Zeit des Ausfalls der $j$ten Komponente kann ebenfalls als $j$te Beobachtung dienen.

Der Wert von j ergibt sich damit aus der Simulationszeit oder dem Auftreten eines bestimmten Zustands oder Ereignisses. Durch Variation der Saat des Zufallszahlengenerators können unterschiedliche Trajektorien erzeugt werden. Wir nehmen im Folgenden an, dass n unterschiedliche Trajektorien beobachtet werden und bezeichnen mit y_{ij} die Realisierung von Y_j in der $i$ten Beobachtung (Trajektorie). Die Beobachtungen können in einer Matrix angeordnet werden, wobei Zeilen Beobachtungen einer Trajektorie und Spalten Beobachtungen einer Zufallsvariablen beinhalten.

$$\begin{array}{ccccc} y_{11}, & \cdots & y_{1j}, & \cdots & y_{1m} \\ \vdots & & \vdots & & \vdots \\ y_{i1}, & \cdots & y_{ij}, & \cdots & y_{im} \\ \vdots & & \vdots & & \vdots \\ y_{n1}, & \cdots & y_{nj}, & \cdots & y_{nm} \end{array}$$

Spalte j beschreibt Beobachtungen Y_j, die gemäß $F_{Y_j}(y)$ verteilt sind. Betrachten wir nun zwei Beobachtungen y_{ij}und y_{kl} und deren Relation. Falls $j = l$ und $i \neq k$, so beschreiben beide Realisierungen von Y_j. Wenn die Sequenzen von Zufallszahlen, die zur Beobachtung von y_{ij} und y_{kj} führen sich nicht überlappen, so sind die generierten Zufallszahlen als unabhängig anzusehen und damit sind auch y_{ij} und y_{kj} unabhängige Beobachtungen von Y_j. Falls $j \neq l$ und $i \neq k$, so beschreiben y_{ij} und y_{kl} Beobachtungen Y_j und Y_l. Wie der Zusammenhang zwischen diesen Zufallsvariablen ist, muss problemspezifisch unterschieden werden, wie später gezeigt wird.

Als Beispiel soll ein so genanntes $M/M/1$-System dienen. Die Schreibweise bezeichnet eine Station an der Kunden mit unabhängigen exponentiell verteilten Zwischenankunftszeiten eintreffen, in einem Warteraum potenziell unendlicher Kapazität nach FCFS bis zur Bedienung warten und anschließend von einem einzelnen Bediener eine exponentiell verteilte Zeit bedient werden. Sei λ die Ankunftsrate (= mittlere Anzahl Aufträge, die pro Zeiteinheit eintreffen) und μ die Bedienrate (= mittlere Anzahl Aufträge, die pro Zeiteinheit bedient werden, wenn der Bediener permanent arbeitet). Seien $\lambda = 0.5$ und $\mu = 1$ die konkreten Werte. Ferner sei Y_j die Kundenzahl im System nach j Zeiteinheiten und n_0 sei die Anzahl Kunden im System zum Zeitpunkt 0. Abbildung 5.3 zeigt den Verlauf von $E(Y(t))$ für unterschiedliche Werte von n_0. Es ist zu beachten, dass der Verlauf des Erwartungswerts dargestellt wird, also Aussagen über alle Trajektorien gemacht werden und nicht eine konkrete Trajektorie. In diesem speziellen Fall ist der Verlauf von $E(Y(t))$ analytisch berechenbar. Eine einzelne Trajektorie würde eine Treppenfunktion bilden, wie in Abbildung 5.2 gezeigt wird.

Bei der Betrachtung der Trajektorien fällt auf, dass für jede Saat des ZZ-Generators ein anderer Verlauf von $Y(t)$ zu beobachten ist. Die Betrachtung der Erwartungswerte $E(Y(t))$ zeigt für unterschiedliche Anfangspopulationen einen unterschiedlichen Verlauf, scheint aber gegen einen festen Werte, unabhängig von der Anfangspopulation, zu konvergieren.

Das kleine Beispiel macht zwei Probleme deutlich, nämlich die Abhängigkeit der beobachteten Größe vom initialen Zustand des Modells und die Abhängigkeit von aufeinander folgenden Beobachtungen.

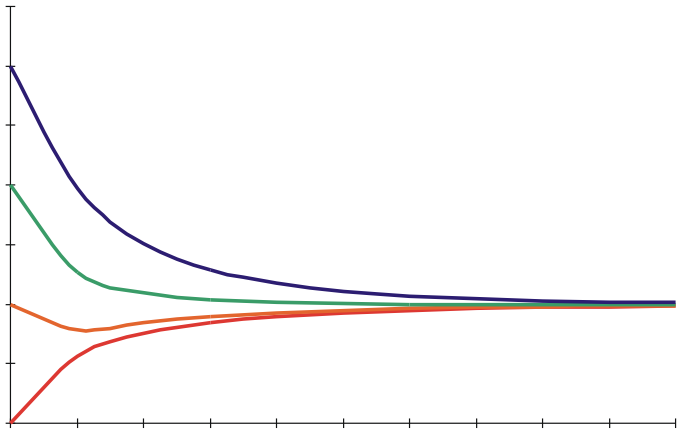


Abbildung 5.3: Verlauf von $E(Y(t))$ für unterschiedliche Startwerte n_0.

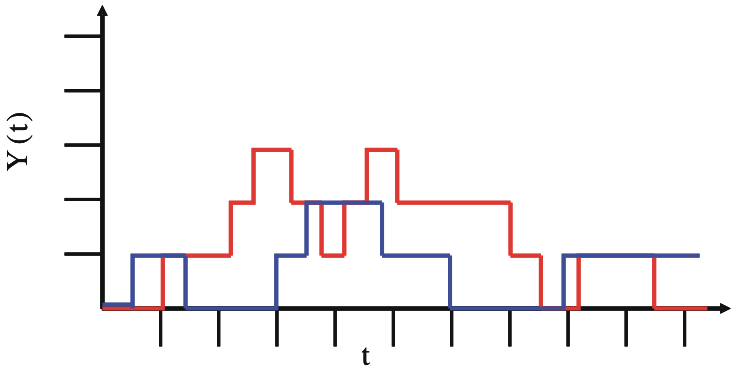


Abbildung 5.4: Einzelne Trajektorien von $Y(t)$.

5.1.1 Schätzung von $E(Y)$

In der Simulation soll Y mit Hilfe von Beobachtungen charakterisiert werden. In der Regel soll $E(Y)$ ermittelt werden, wobei durch entsprechende Interpretation des Erwartungswerts zahlreiche Größen ermittelt werden können, wie später noch gezeigt wird. $E(Y)$ soll geschätzt werden, deshalb benutzen wir wieder als Schreibweise $\tilde{Y}$ für den Schätzer und $\hat{Y}$ für den konkreten Schätzwert. Um Aussagen über die Qualität des Schätzer zu machen, werden Informationen über $\sigma^2(Y)$, die Varianz der beobachteten Größe benötigt, die natürlich ebenfalls nur geschätzt werden kann.

Wenn ausgehend von der bereits beschriebenen Beobachtungslage $E(Y_j)$ aus n Beobachtungen ermittelt werden soll, so ist

$$\hat{Y}_j = \frac{1}{n} \cdot \sum_{i=1}^{n} y_{ij}$$

die nahe liegende Methode zur Bestimmung des Schätzwertes. Man nennt $\hat{Y}_j$ einen *Punktschätzer* für $E(Y_j)$. Offensichtlich variiert der Wert von $\hat{Y}_j$ für unterschiedliche Saaten des Zufallszahlengenerators für die einzelnen Beobachtungsläufe. Um Aussagen über $E(Y_j)$ machen zu können, müssen wir Aussagen über mögliche Werte von $\hat{Y}_j$ machen und letztendlich beantworten, wie weit $\hat{Y}_j$ von $E(Y_j)$ entfernt ist. Da $\hat{Y}_j$ eine Realisierung von $\tilde{Y}_j$ ist und $\tilde{Y}_j$ eine Zufallsvariable ist, können Aussagen nur auf Grund der Verteilungscharakteristika gemacht werden.

Wir haben bereits gesehen, dass $\tilde{Y}_j$ ein erwartungstreuer Schätzer von $E(Y_j)$ ist womit $E(\tilde{Y}_j) = E(Y_j)$ gilt. Dies gilt sogar, wenn die einzelnen Beobachtungen nicht unabhängig sind. Weiterhin ist der Schätzer konsistent und damit gilt $\lim_{n\to\infty} P[|\tilde{Y}_j - E(Y_j)| > \epsilon] = 0$ für alle $\epsilon > 0$. Die Erwartungstreue und Konsistenz eines Schätzer macht keinerlei Aussagen über die Qualität des konkreten Schätzwertes nach n Beobachtungen. Ein erster Schritt, um zu solchen Aussagen zu gelangen ist die Ermittlung der Varianz des Schätzers $\tilde{Y}_j$. Die Varianz ist ein Maß für die mögliche Schwankung der Realisierungen $\hat{Y}_j$ und gibt damit Hinweise über die Genauigkeit der Schätzung. Die folgende Formel zeigt die Herleitung der Varianz des Schätzer bei n Beobachtungen.

$$\begin{array}{lll} \sigma^2(\tilde{Y}_j) & = E\left(\left(\tilde{Y}_j - E(\tilde{Y}_j)\right)^2\right) & = \\ E\left(\left(\frac{1}{n} \cdot \sum_{i=1}^{n} Y_{ij} - E(Y_j)\right)^2\right) & = E\left(\left(\frac{1}{n} \cdot \sum_{i=1}^{n} (Y_{ij} - E(Y_j))\right)^2\right) & = \\ \frac{1}{n^2} \cdot \left(\sum_{i=1}^{n} E\left((Y_{ij} - E(Y_j))^2\right) \right. & \left. + \sum_{i=1}^{n} \sum_{k=1,k\neq i}^{n} E\left((Y_{ij} - E(Y_j)) \cdot (Y_{kj} - E(Y_j))\right)\right) & \end{array} \tag{5.1}$$

Man sieht, dass in die Varianz die Abhängigkeiten zwischen den Beobachtungen durch die Doppelsummen einfließen. Im Gegensatz zum Erwartungswert des Schätzers wird die Varianz durch korrelierte Beobachtungen beeinflusst. Wenn angenommen wird, dass die beobachteten Werte y_{ij} $(i = 1, \ldots n)$ unabhängig und damit unkorreliert sind, so ist der Wert der Doppelsumme 0 und die Berechnung der Varianz reduziert sich auf die folgende einfache Form (siehe auch Seite 97).

$$\sigma^2(\tilde{Y}_j) = \frac{1}{n^2} \sum_{i=1}^{n} E\left((Y_{ij} - E(Y_j))^2\right) = \frac{1}{n^2} \cdot n \cdot \sigma^2(Y_j) = \frac{\sigma^2(Y_j)}{n}$$

Aus dieser Darstellung lässt sich ablesen, dass die Schwankungsbreite des Schätzers

- mit der Schwankungsbreite der Beobachtungsgröße steigt und
- mit der Anzahl Beobachtungen fällt.

Damit kann für eine vorgegebene Beobachtungsgröße die Schwankung der Schätzung dadurch reduziert werden, dass die Zahl der Beobachtungen erhöht wird. Dies ist ein nachvollziehbares Verhalten. Zur konkreten Bestimmung der Varianz des Schätzers wird die unbekannte Varianz der

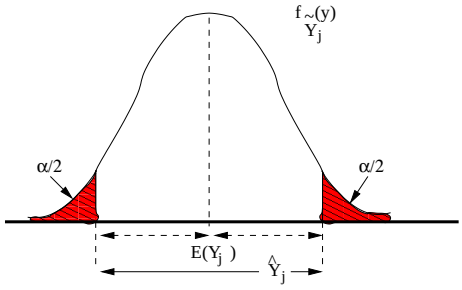


Abbildung 5.5: Skizze zur Verteilung des Schätzers $\tilde{Y}_j$.

Beobachtungsgröße benötigt, die damit ebenfalls geschätzt werden muss. Ein erwartungstreuer Schätzer für $\sigma^2(Y_j)$ lautet

$$\tilde{S}_j^2 = \frac{1}{n-1} \sum_{i=1}^{n} \left(Y_{ij} - \tilde{Y}_j \right)^2 \tag{5.2}$$

Damit ist $\tilde{S}_j^2/n$ ein erwartungstreuer Schätzer für $\sigma^2(\tilde{Y}_j)$ und $\hat{S}_j^2/n$ ist der konkrete Schätzwert. Die Varianz des Schätzers allein macht noch keine konkrete Aussagen über die potenzielle Abweichung eines Schätzwertes vom wahren Wert. Dazu sind weitere Verfahren und Annahmen notwendig. Bevor die zugehörigen Methoden vorgestellt werden, soll auf die Berechnung der Schätzwerte für Erwartungswert und Varianz eingegangen werden.

5.1.2 Praktische Berechnung der Schätzer

Beginnen wir mit der Berechnung von $\hat{Y}_j$. Es ist unpraktisch erst alle Werte y_{ij} zu speichern und abschließend $\hat{Y}_j$ zu berechnen, da damit bei vielen Beobachtungen ein hoher Speicherplatzbedarf einhergeht und Ergebnisse auch erst zum Ende der Simulation vorliegen. Besser ist die Verwendung der Rekursion

$$\hat{Y}_j(k) = \frac{k-1}{k} \hat{Y}_j(k-1) + \frac{y_{kj}}{k}$$

mit $\hat{Y}_j(0) = 0$ und $k = 1, \ldots, n$. Die Berechnung ist für relativ große Werte von n numerisch stabil. Falls n sehr groß wird, so kann mehrstufig vorgegangen werden, indem nicht einzelne Werte y_{kj}, sondern Mittelwerte aus mehreren Beobachtungen aufsummiert werden und damit die Zahl der zu summierenden Werte reduziert wird.

Problematischer ist die Berechnung der Stichprobenvarianz. Die direkte Auswertung der Formel für den Schätzwert der Varianz

$$\hat{S}_j^2 = \frac{1}{n-1} \sum_{i=1}^{n} \left(y_{ij} - \hat{Y}_j \right)^2$$

führt dazu, dass die Varianz erst berechnet werden kann, wenn $\hat{Y}_j$ vorliegt. Da dies erst der Fall ist, wenn alle Werte ermittelt wurden, müssen alle y_{ij} gespeichert werden. Alternativ können die Summen der y_{ij} und y_{ij}^2 gespeichert werden und

$$\hat{S}_j^2 = \frac{1}{n-1} \left(\sum_{i=1}^{n} (y_{ij})^2 - n \cdot \left(\hat{Y}_j \right)^2 \right)$$

berechnet werden. Dieses Vorgehen erfordert zwar wenig Speicherplatz, ist aber für große n numerisch instabil. Trotzdem wird in den meisten Implementierungen mit dieser Variante gearbeitet. Die Größe von n wird dadurch beschränkt, dass nicht einzelne Werte sondern Mittelwerte aus mehreren Beobachtungswerten in die Formel einfließen, wie später noch gezeigt wird.

5.1.3 Interpretation und Bestimmung von Konfidenzintervallen

Auch nach Schätzung der Varianz bleibt noch die Frage, wie weit der ermittelte Schätzwert $\hat{Y}_j$ vom wahren Wert $E(Y_j)$ entfernt ist. Da wir es mit Zufallsvariablen zu tun haben, kann der wahre Abstand nur ermittelt werden, wenn $E(Y_j)$ bekannt ist, dann ist aber eine Bestimmung per Simulation unnötig. Die korrekte Frage müsste eigentlich lauten:

Wie groß ist die Wahrscheinlichkeit, dass $\tilde{Y}_j$ um mehr als ϵ von $E(Y_j)$ entfernt ist?

Also die Wahrscheinlichkeit α ist zu ermitteln, so dass $P[|\tilde{Y}_j - E(Y_j)| \geq \epsilon] = \alpha$. Angenommen die Dichtefunktion $f_{\tilde{Y}_j}(y)$ sei bekannt, dann liegt eine wie in Abbildung 5.5 gezeigte Situation vor. Um den wahren Wert $E(Y_j)$ wird ein Intervall der Breite 2ϵ gezogen, so dass die Wahrscheinlichkeit, dass der Wert von $\tilde{Y}_j$ innerhalb des Intervalls liegt $1-\alpha$ entspricht (d.h. $\int_{E(Y_j)-\epsilon}^{E(Y_j)+\epsilon} f_{\tilde{Y}_j}(x)dx = 1-\alpha$). Mit Wahrscheinlichkeit α liegt der Wert von $\tilde{Y}_j$ außerhalb des definierten Intervalls. Bei unbekannter Verteilung liefert die Kenntnis der Varianz des Schätzers oder eines Schätzwertes $\hat{S}_j^2$ nur ein Indiz für die Breite des Intervalls, aber kein konkretes Intervall.

Diese Beobachtungslage legt zwei Interpretationen nahe:

1. Wie groß ist die Wahrscheinlichkeit, dass $\hat{Y}_j$ aus dem Intervall $[E(Y_j)-\epsilon, E(Y_j)+\epsilon]$ herausfällt?

2. Wie groß ist die Wahrscheinlichkeit, dass das Intervall $[\hat{Y}_j-\epsilon, \hat{Y}_j+\epsilon]$ den wahren Wert $E(Y_j)$ nicht enthält?

Die erste Fragestellung ist weniger relevant, da $E(Y_j)$ nicht bekannt ist. Hilfreicher ist der zweite Fall, da dort Aussagen über $E(Y_j)$ ausgehend vom bekannten $\hat{Y}_j$ getroffen werden. Damit kann man prinzipiell, d.h. falls die Verteilung $\tilde{Y}_j$ bekannt wäre, Aussagen über ein Intervall treffen, in dem $E(Y_j)$ mit vorgegebener Wahrscheinlichkeit liegt.

Offensichtlich existiert eine Abhängigkeit zwischen α und ϵ. Ausgehend von Abbildung 5.5 wird α kleiner, wenn ϵ größer und umgekehrt. Bisher wurde dabei so argumentiert, dass zu einem ϵ ein passendes α gesucht wurde. Es wurde also die Breite des Intervalls vorgegeben und dann die Wahrscheinlichkeit ermittelt, dass der wahre Wert im Intervall liegt. In der Praxis ist allerdings eine andere Fragestellung üblicher. Es wird die Wahrscheinlichkeit α vorgegeben und dazu die passende Intervallbreite ϵ ermittelt. Die übliche Fragestellung lautet:

Wenn man mit Wahrscheinlichkeit $1-\alpha$ sicher sein will, dass $E(Y_j)$ nicht mehr als ϵ von $\hat{Y}_j$ abweicht, welches ϵ ist dann zu wählen?

Man nennt $\hat{Y}_j \pm \epsilon$ ein $(1-\alpha)\cdot 100\%$ Konfidenzintervall für $E(Y_j)$.
$[\hat{Y}_j - \epsilon, \hat{Y}_j + \epsilon]$ ist ein Intervallschätzer.

Offensichtlich sind Intervallschätzer ungleich wertvoller als Punktschätzer, da sie Wahrscheinlichkeitsaussagen machen, während Punktschätzer keinerlei Angaben darüber enthalten, wie groß die Variabilität des Schätzer ist. Bisher haben wir nur Schätzer für den Erwartungswert und die Varianz von $\tilde{Y}_j$ kennen gelernt, ohne Kenntnis der Verteilung des Schätzers kann daraus kein Konfidenzintervall gewonnen werden. Wir werden im Folgenden zwei Methoden zur Bestimmung von Konfidenzintervallen vorstellen. Im ersten Fall werden Schranken berechnet, die für alle möglichen Verteilungen gelten und im zweiten Fall wird eine bestimmte Verteilung vorausgesetzt, die unter sehr allgemeinen Annahmen beobachtet werden kann.

5.1.3.1 Konfidenzintervalle nach Tschebyscheff

Die bekannte Ungleichung von Tschebyscheff lautet: Für eine Zufallsvariable X mit Erwartungswert $E(X)$ und Varianz $\sigma^2(X)$ gilt für beliebiges $c > 0$:

$$P[|X - E(X)| \geq c] \leq \sigma^2(X)/c^2$$

Angewendet auf die Berechnung von Konfidenzintervallen wie beschrieben, wird c durch ϵ und X durch $\tilde{Y}_j$ ersetzt. Die Varianz von $\tilde{Y}_j$ lautet $\sigma^2(Y_j)/n$, so dass

$$P[|\tilde{Y}_j - E(Y_j)| \geq \epsilon] \leq \frac{\sigma^2(Y_j)}{n \cdot \epsilon^2}$$

Wir können dies an einem einfachen Beispiel, nämlich der Berechnung des 90% Konfidenzintervalls für $\hat{Y}_j$ vorführen. Dann ist $\alpha = \sigma^2(Y_j)/(n \cdot \epsilon^2) = 0.1$. Bei einem Stichprobenumfang von 10 gilt dann

$$\frac{\sigma^2(Y_j)}{10 \cdot \epsilon^2} = 0.1 \;\Leftrightarrow\; \epsilon^2 = \sigma^2(Y_j) \;\Leftrightarrow\; \epsilon = \sigma(Y_j)$$

Bei einem errechneten Punktschätzer $\hat{Y}_j$ ist $[\hat{Y}_j - \epsilon, \hat{Y}_j + \epsilon]$ in diesem Fall das gesuchte Konfidenzintervall. Wenn man dieses Konfidenzintervall bestimmt, so wird man mit einer Wahrscheinlichkeit von mindestens 90% ein Intervall erhalten, in dem der gesuchte Wert liegt.

Bei einer festen Varianz des beobachteten Wertes Y_j, was vorausgesetzt werden kann, erfordert eine Halbierung der Breite des Konfidenzintervalls eine Vervierfachung der Zahl der notwendigen Beobachtungen, da

$$\frac{\sigma^2(Y)}{n \cdot \epsilon^2} = \frac{\sigma^2(Y)}{m \cdot (\epsilon/2)^2} \;\Leftrightarrow\; m = 4 \cdot n$$

Dies impliziert auch, dass der Aufwand zur Ermittlung sehr genauer Simulationsresultate (d.h. schmaler Konfidenzintervalle) sehr groß werden kann.

Die Konfidenzintervallbestimmung nach Tschebyscheff hat zwei wesentliche Nachteile:

1. Die Berechnung des Konfidenzintervalls verwendet die Varianz $\sigma^2(Y_j)$ der beobachteten Größe. Dieser Wert ist aber in praktisch allen relevanten Fällen unbekannt und muss durch den geschätzten Wert $\hat{S}_j^2$ ersetzt werden. Damit sind die Voraussetzungen von Tschebyscheff verletzt und die ermittelten Schranken gelten nicht mehr.

2. Tschebyscheff liefert sehr breite Konfidenzintervalle, da der Satz allgemein für beliebige Verteilungen gilt und damit Extremfälle einbezieht, die für die gegebene Beobachtungslage selten oder nie auftreten.

Üblicherweise wird der "pessimistische Charakter" der Konfidenzintervalle nach Tschebyscheff genutzt, um die Verwendung $\hat{S}_j^2$ statt $\sigma^2(Y_j)$ als Intervallschätzer zu rechtfertigen.

Wir betrachten als einfaches Beispiel das Werfen einer fairen Münze und interpretieren Zahl=1, Kopf=0. Dann gilt $E(Y) = 0.5$ und $\sigma^2(Y) = 0.25$. Es werden zwei Experimentserien mit jeweils 10 Würfen durchgeführt und einzeln und als konkatenierte Serie mit 20 Experimenten ausgewertet. Im Einzelnen liegen folgende Ergebnisse vor.

1. Experimentserie 1: 0000101001 liefert $\hat{Y} = 0.3$ und $\hat{S}^2 = 0.23333$. Daraus ergibt sich für $\alpha = 0.1$ das Konfidenzintervall $[-0.183, 0.783]$. Aus der Kenntnis des Modells folgt, dass der Wert nicht kleiner als 0 sein kann, so dass die untere Grenze durch 0 ersetzt werden kann.

2. Experimentserie 2: 0110111001 liefert $\hat{Y} = 0.6$ und $\hat{S}^2 = 0.26667$. Für $\alpha = 0.1$ erhält man das Konfidenzintervall $[0.084, 1.116]$. Auch hier hilft die Kenntnis des Modells, so dass die obere Grenze durch 1.0 ersetzt werden kann.

3. Durch Konkatenation der beiden Serien entsteht eine Serie der Länge 20 mit $\hat{Y} = 0.45$, $\hat{S}^2 = 0.2605$ und einem Konfidenzintervall $[0.089, 0.811]$ für $\alpha = 0.1$.

Es sollen noch einmal die Aussagen eines Intervallschätzer vor Augen geführt werden. Im obigen Fall liegt der wahre Wert in 90% der Fälle im ermittelten Intervall und liegt mit 10% Wahrscheinlichkeit außerhalb des Intervalls. Also von 10 Experimenten der obigen Art, sollte im Mittel ein Experiment ein Intervall liefern, das 0.5 nicht enthält. Da Tschebyscheff aber sehr pessimistische Schranken liefert, kann man in der Regel sagen, dass der wahre Wert mit mindestens Wahrscheinlichkeit $1 - \alpha$ im ermittelten Intervall liegt.

5.1.3.2 Konfidenzintervalle nach dem zentralen Grenzwertsatz

Um bessere, also schmalere Konfidenzintervalle zu gewinnen, muss Information über die Verteilung des Schätzers $\tilde{Y}_j$ in die Konfidenzintervallbestimmung einbezogen werden. Offensichtlich hängt die Verteilung $\tilde{Y}_j$ aber von der in der Regel unbekannten Verteilung von Y_j ab. Die Frage bleibt damit, ob Aussagen über die Verteilung von $\tilde{Y}_j$ ohne Einbeziehung der Verteilung von Y_j gemacht werden können. Zur Hilfe kommt uns dabei der zentrale Grenzwertsatz (siehe 3.2.1 auf Seite 74).

Der zentrale Grenzwertsatz gilt auch, wenn σ^2 durch $\tilde{S}^2$ ersetzt wird. Damit können, falls n groß genug ist, Konfidenzintervalle aus den "kritischen" Werten der Normalverteilung gewonnen werden. Ausgehend von den vertafelten Werten von $N(0,1)$ kann damit wie folgt ein Konfidenzintervall für $\tilde{Y}$ gewonnen werden.

$$P[|Z| \geq \epsilon] = P\left[\left|\frac{\tilde{Y} - E(Y)}{\tilde{S}/\sqrt{n}}\right| \geq \epsilon\right] = \alpha \;\Rightarrow\; P\left[\left|\tilde{Y} - E(Y)\right| \geq \epsilon \cdot \tilde{S}/\sqrt{n}\right] = \alpha$$

Damit lautet das Konfidenzintervall

$$\tilde{Y} \pm z_\alpha \cdot \tilde{S}/\sqrt{n}$$

wobei z_α der Wert des $\alpha/2$ Quantils von $N(0,1)$ ist (siehe Abbildung 5.5).

Übertragen auf unser Würfelbeispiel liefert die Normalverteilung ein Konfidenzintervall von $[0.262, 0.638]$ für die konkatenierte Sequenz. Dies ist eine deutliche Reduktion des Intervalls [0.089, 0.811], das mittels Tschebyscheff berechnet wurde. Allgemein liefert die Normalverteilung deutlich schmalere Konfidenzintervalle als Tschebyscheff.

Es bleibt aber die Frage, wie groß n sein muss, damit die Annahme der Normalverteilung für den Schätzer gerechtfertigt ist und damit die nach dem zentralen Grenzwertsatz ermittelten Konfidenzintervalle den wahren Wert wirklich mit der vorgegebenen Wahrscheinlichkeit enthalten.

Die notwendige Größe von n hängt offensichtlich von der unbekannten Verteilung von Y ab. Es ist deshalb wichtig, sich vor Augen zu führen, welche Auswirkungen eine irrtümliche Normalverteilungsannahme hat. Auch dies hängt natürlich von der wahren Verteilung von Y ab. Im ungünstigen Fall, kann der wahre Wert $E(Y)$ mit einer Wahrscheinlichkeit, die deutlich größer als α ist, außerhalb des berechneten Konfidenzintervalls liegen. Man spricht dann davon, dass die Abdeckung des Konfidenzintervalls kleiner als α ist. Dies bedeutet, dass die Simulationsresultate eine Präzision vortäuschen, die nicht gerechtfertigt ist und auf dieser Basis unter Umständen falsche Entscheidungen getroffen werden. Eine solche Situation sollte möglichst vermieden werden.

Leider sind nur sehr wenige Resultate über die Verteilung von $\tilde{Y}$ bekannt, selbst unter der unrealistischen Annahme, dass die Verteilung von Y bekannt ist. Eines der wenigen bekannten Resultate betrachtet den Fall, dass Y selbst schon normalverteilt ist. In diesem Fall ist

$$\frac{\tilde{Y} - E(Y)}{\tilde{S}/\sqrt{n}}$$

t-verteilt mit $n-1$ Freiheitsgraden. Die kritischen Werte der t-Verteilung sind, ähnlich wie die Werte der Normalverteilung, vertafelt (siehe Banks et al. [2000, Tabelle A.5]). Damit lautet das Konfidenzintervall

$$\hat{Y} \pm t_{n-1,1-\alpha/2} \cdot \frac{\hat{S}}{\sqrt{n}} \tag{5.3}$$

wobei $t_{n-1,1-\alpha/2}$ das $1-\alpha/2$ Quantil einer t-Verteilung mit $n-1$ Freiheitsgraden ist (siehe Law and Kelton [2000, S. 254]). Für $n \to \infty$ konvergiert die t-Verteilung gegen eine Normalverteilung. Für kleine n sind die Quantile der t-Verteilung allerdings größer als die zugehörigen Quantile der Normalverteilung, so dass die Konfidenzintervalle pessimistischer sind. Auf unser Beispiel angewendet liefert die t-Verteilung ein Konfidenzintervall von $[0.252, 0.647]$ im Gegensatz zu $[0.262, 0.638]$ mittels der Normalverteilung. Auch wenn in der Praxis Y nicht unbedingt normalverteilt ist, werden oftmals Konfidenzintervalle mit Hilfe der t-Verteilung und nicht mit der Normalverteilung ermittelt, da die resultierenden Konfidenzintervalle etwas pessimistischer sind, ohne gleich so breit wie bei Verwendung von Tschebyscheff zu sein.

Verteilung	n=5	n=10	n=20	n=40
Normal	0.910	0.902	0.898	0.900
Exponential	0.854	0.878	0.870	0.890
Lognormal	0.758	0.768	0.842	0.852
Hyperexp.	0.584	0.586	0.682	0.774

Tabelle 5.1: Überdeckung der mit der t-Verteilung ermittelten Konfidenzintervalle (aus Law and Kelton [2000, S. 257]).

Mangels theoretischer Resultate, kann eine Untersuchung der Abdeckung von Konfidenzintervallen nur experimentell und damit empirisch erfolgen. Dazu werden Zufallszahlen aus einer bekannten Verteilung gezogen und es wird ein Konfidenzintervall für den Erwartungswert ermittelt. Dieses Experiment wird mehrfach wiederholt und der Anteil der Konfindenzintervalle bestimmt, der den wahren Wert enthalten. Bei k Wiederholungen und Wahrscheinlichkeit α sollten $\approx (1-\alpha) \cdot k$ Experimente Konfidenzintervalle liefern, die den wahren Wert beinhalten. Tabelle 5.1 gibt die Resultate einiger Experimente wieder. Es wurden jeweils $k = 500$ Wiederholungen der Experimente durchgeführt und 90% Konfidenzintervalle für den Erwartungswert aus $n = 5, 10, 20, 40$ Zufallszahlen einer gegebenen Verteilung bestimmt. In der Tabelle wird der Anteil der Experimente angegeben, bei denen der wahre Wert im Konfidenzintervall lag. Da es sich um Experimente handelt, ist nicht zu erwarten, dass genau 90% der Experimente Werte im Konfidenzintervall liefern und 10% Werte außerhalb liefern. Trotzdem sollte bei 500 Wiederholungen der ermittelte Wert in diesem Bereich liegen. Wie die Experimente zeigen, ist die Überdeckung für die Normalverteilung sehr gut. Dies war zu erwarten, da genau eine t-Verteilung vorliegt. In den anderen Fällen scheinen die Konfidenzintervalle zu optimistisch zu sein. So wird die Überdeckung mit steigendem n zwar besser, für die Verteilungen die sich deutlich von einer Normalverteilung unterscheiden, wie die Lognormalverteilung und insbesondere die Hyperexponentialverteilung, ist die Überdeckung aber auch für $n = 40$ noch nicht zufriedenstellend und die Simulationsergebnisse würden eine nicht gegebene Sicherheit vortäuschen. In diesen Fällen sind also deutlich höhere Beobachtungszahlen notwendig.

5.1.4 Bestimmung weiterer Größen neben $E(Y)$

Bisher wurde die Auswertung der Simulation auf die Schätzung von Erwartungswerten eingeschränkt. Auch wenn Erwartungswerte von Größen wie Verweilzeiten oder Pufferfüllungen erste Aussagen über das Systemverhalten erlauben, sind sie oft nicht ausreichend für die Systemanalyse. So gibt eine mittlere Pufferfüllung von 10 keine Hinweise darauf, wie oft ein Puffer der Größe 20 überlaufen wird. Eine mittlere Bearbeitungszeit von 0.1 Sekunden in einem System mit Realzeitanforderungen sagt nicht darüber aus, wie viele Bearbeitungen länger als die Zeitschranke von 0.5 Sekunden dauern. Für solche und ähnliche Fragestellungen werden üblicherweise Schätzungen der folgenden Größen benötigt.

- Die Schätzung höherer Momente $E(Y^k)$ für $k = 2, 3, 4, \ldots$
- Die Schätzung von Werten der Verteilungsfunktion an einer Stelle y, also $P[Y \leq y] = F(y)$.
- Die Schätzung von p-Quantilen ($p \in (0, 1)$), also des Wertes y_p, so dass $P[Y \leq y_p] = F(y_p) = p$.

Die ersten beiden Fälle lassen sich sehr einfach auf das bisherige Vorgehen abbilden. Dazu wird eine neue Resultatvariable X definiert, deren Erwartungswert gerade die gesuchte Größe ist. Für die Schätzung des $k$ten Moments von Y_i wäre $X = (Y_i)^k$, $\bar{X} = (1/n) \cdot \sum_{j=1}^{n} (Y_{ij})^k$ wäre der Schätzer und $\hat{X} = (1/n) \cdot \sum_{j=1}^{n} (y_{ij})^k$ der konkrete Schätzwert. Für die Schätzung eines Wertes der Verteilungsfunktion definieren wir eine binäre Zufallsvariable X mit

$$X = \begin{cases} 1 & \text{falls } Y \leq y \\ 0 & \text{sonst} \end{cases}$$

Dann ist $E(X) = P[Y \leq y]$ und $\tilde{X}$ kann wie üblich ermittelt werden. Es wird dabei der Parameter einer Binomialverteilung geschätzt. Dabei ist zu beachten, dass für diese Verteilung Konfidenzintervalle, die mit Hilfe des zentralen Grenzwertsatzes geschätzt wurden, für kleinere Werte von n oder p eine sehr schlechte Abdeckung aufweisen. In der Statistik wurden deshalb bessere Verfahren zur Konfidenzintervallberechnung für Binomialverteilungen entwickelt Brown et al. [2001].

Auch wenn die einzelnen Größen sich als Schätzer von Erwartungswerten darstellen lassen, so ist ihre praktische Ermittlung oftmals problematisch. Gerade bei höheren Momenten und großen/kleinen Werten der Verteilungsfunktion, die von wenigen Werten über-/unterschritten werden, sind oftmals sehr große Stichproben notwendig, um akzeptable Konfidenzintervalle zu erreichen.

Komplexer ist die Ermittlung von Quantilen. Da der Wert von y_p nicht vorab bekannt ist, werden Quantile oftmals erst nach Ermittlung aller Werte der Stichprobe aus der geordneten Stichprobe ermittelt. Dies hat natürlich den Nachteil, dass alle Werte gespeichert werden müssen und Schätzer erst nach Ablauf der Simulation zur Verfügung stehen. In Raatikainen [1990] wird eine Methode vorgestellt, mit deren Hilfe Quantile auch während der Simulation geschätzt werden können. Der zugehörige Algorithmus ist allerdings relativ komplex und würde den Rahmen der Vorlesung sprengen.

5.2 Klassifizierung von Simulationen bzgl. der Auswertung

Bisher wurde die Auswertung einer Resultatgröße relativ abstrakt untersucht. Es soll nun konkret unterschieden werden, welche Arten von Resultaten in einer Simulation ermittelt werden können. Dazu betrachten wir zuerst einige Beispiele.

- In einer Fertigungsstraße, die im Dreischichtbetrieb 7 Tage in der Woche läuft, soll der Durchsatz ermittelt werden. Es kann davon ausgegangen werden, dass die Belastung des Systems unabhängig von der Zeit ist und auch die einzelnen Arbeitsschritte zeitunabhängig ausgeführt werden. Weiterhin kann man davon ausgehen, dass die Fertigungsstraße über einen längeren Zeitraum störungsfrei arbeitet.

- Die Kapazität eines Kommunikationsnetzes soll geplant werden. Dazu wird die zu erwartende Ankunftsrate von Nachrichten und die Belegungszeit in Hochlastphasen geschätzt und es wird eine Leitungskapazität gesucht, so dass eine vorgegebene Dienstqualität eingehalten wird. Zur Ermittlung dieser Kapazität wird das System für die zur Verfügung stehenden Kapazitäten analysiert.

- Eine optimale Belieferungsstrategie eines Einzelhändlers mit Speiseeis soll ermittelt werden. Die Belieferungsmenge muss natürlich abhängig vom Verbrauch sein. Der Verbrauch richtet sich nach dem Wetter und der Temperatur, die wiederum von der Jahreszeit beeinflusst werden.

- Die Zeit bis zum Ausfall eines Computersystems soll bestimmt werden. Dazu sind die Ausfallzeiten der einzelnen Komponenten und deren Interaktion bekannt.

- Die Kassenauslastung eines Supermarktes zwischen 9^{00} und 12^{00} Uhr ist zu ermitteln. Der zu erwartende Kundenstrom richtet sich natürlich nach der Uhrzeit. Weiterhin ist davon auszugehen, dass der Supermarkt um 9^{00} Uhr öffnet.

Die verschiedenen Aufgabenstellungen stellen unterschiedliche Anforderungen an die Auswertung der Simulationsmodelle. In den ersten beiden Fällen wird ein System betrachtet, bei dem von einer sehr langen Laufzeit ausgegangen wird, während der sich das System gleich verhält. Es ist zu beachten, dass gleiches Verhalten nicht bedeutet, dass das Verhalten deterministisch ist. Es kann sehr wohl stochastische Schwankungen geben, die beteiligten Zufallsvariablen und stochastischen Prozesse in der Modellbeschreibung sind aber unabhängig von der Modellzeit. Das dritte Beispiel beschreibt ebenfalls eine Situation, in der das System über einen langen Zeitraum untersucht wird

(mehrere Jahre). Im Gegensatz zu den ersten beiden Beispielen ändern sich aber die Zufallseinflüsse im Modell mit der Modellzeit (d.h. Jahreszeit). Das vierte Beispiel beschreibt einen Ablauf, der mit dem Ausfall des Systems endet. Die Simulation terminiert also, wenn ein bestimmter Systemzustand erreicht wird. Auch im fünften Beispiel haben wir es mit einer terminierenden Simulation zu tun. Im Gegensatz zum vorherigen Beispiel ist das Ende der Simulation aber nun durch die Modellzeit 12^{00} vorgegeben. Im Gegensatz zu den ersten drei Beispielen, läuft das System nicht längere Zeit, da der Supermarkt abends schließt und am nächsten Morgen praktisch neu begonnen wird.

Die Beispiele liefern eine Basis, auf der man Simulationsexperimente bzgl. ihrer Auswertung unterscheiden kann. Zum einen betrachtet man *terminierende Simulation*. Die Simulation startet zum Zeitpunkt $t = 0$ und endet zu einem Zeitpunkt T_E. T_E kann entweder zu Beginn der Simulation vorgegeben werden oder ergibt sich im Laufe der Simulation durch das Eintreten eines Zustandes oder Ereignisses. Ziel ist die Ermittlung von $Y(t)$ zu einem Zeitpunkt $t = T_E$ oder in einem Intervall $[T_0, T_E]$[1] mit $0 \leq T_0 < T_E$. Zum anderen werden *nicht-terminierende Simulationen* betrachtet. Dieser Fall wird später etwas weiter unterteilt, da bei nicht-terminierenden Simulation unterschiedliche Verhalten auftreten können.

5.2.1 Terminierende Simulation

Wie bereits dargestellt wird bei der terminierenden Simulation das gesuchte Resultat entweder zum Zeitpunkt T_E oder im Intervall $[T_0, T_E]$ analysiert. Ziel ist die Schätzung von $E(Y)$, mit $Y = Y(T_E)$ oder $Y = \int_{T_0}^{T_E} Y(t)dt$. Zur Bestimmung des Schätzers $\tilde{Y}$ werden Beobachtungen der Zufallsvariable Y benötigt. Im Gegensatz zur auf Seite 106 dargestellten Situation kann aus einem Simulationslauf genau eine Beobachtung ermittelt werden. Die Beobachtungen aus n Simulationsläufen werden deshalb als $Y_1, \ldots, Y_n$ nummeriert. Jedes Y_i ist eine Zufallsvariable mit einer Verteilung, die der von Y entspricht. Der Simulationslauf wird n mal wiederholt, man spricht in diesem Fall von n *Replikationen*. Wir erweitern die Notation etwas und benutzen $\tilde{Y}(n)$, $\hat{Y}(n)$, $\tilde{S}^2(n)$ und $\hat{S}^2(n)$ für die entsprechenden Größen nach n Replikationen. Ferner ist y_i der Wert der $i$ten Replikation. Das Konfidenzintervall wird dann wie üblich berechnet.

$$\hat{Y}(n) \pm t_{n-1,1-\alpha/2} \cdot \sqrt{\frac{\hat{S}^2(n)}{n}} \tag{5.4}$$

Die Abdeckung dieses Konfidenzintervalls hängt ab, von der Nähe der Verteilung von Y zu einer Normalverteilung, der Größe von n und der Unabhängigkeit der Y_i.

Die Unabhängigkeit der Y_i hängt von der Unabhängigkeit der verwendeten Zufallszahlen ab. Ausgehend von einer Saat s_i wird Y_i erzeugt. Da üblicherweise LCGs (siehe Abschnitt 3) zur Generierung verwendet werden, entspricht s_i einem Wert in der Sequenz der generierten Zufallszahlen. Die ausgehend von s_i und s_j generierten Zufallszahlen können als unabhängig gelten, wenn sich die Sequenzen nicht überlappen. Dies bedeutet, dass die mit s_j startende Sequenz nicht s_i enthalten darf und umgekehrt. Um dies praktisch zu erreichen gibt es verschiedene Möglichkeiten. Viele Generatoren erlauben die Definition von Zufallszahlenströmen, die an weit auseinander liegenden Punkten im Zyklus beginnen. Es kann für jede Replikation ein neuer Strom definiert werden, so dass sich die Zufallszahlen nicht überlappen, solange nicht mehr Zahlen erzeugt werden, als der Abstand zwischen den Strömen beträgt. Wenn man davon ausgeht, dass die Replikationen nacheinander ausgeführt werden, was wir tun werden, so kann die Saat am Ende einer Replikation (= der aktuelle Wert von x_i im Generator) ausgelesen werden und als neue Saat für die nächste Replikation verwendet werden. Eine Funktion zum Auslesen der Saat ist bei den meisten Zufallszahlengeneratoren vorhanden. Mit diesem Vorgehen erreicht man Folgen unabhängiger Zufallszahlen in den Replikationen, solange der Generator an sich eine Folge unabhängiger Zufallszahlen erzeugt und insgesamt weniger als κ Zufallszahlen erzeugt werden.

Auch wenn die verwendeten Zufallszahlen unabhängig sind, bedeutet dies nicht zwangsläufig, dass die Abdeckung der nach (5.4) bestimmten Konfidenzintervalle wirklich α ist, da die Berech-

[1] Es können natürlich auch offene oder halboffene Intervalle betrachtet werden.

M/M/1-System			Sys. mit Ausfällen		
n	Abdeckung	Breite rel. zu $\hat{Y}(n)$	n	Abdeckung	Breite rel. zu $\hat{Y}(n)$
5	0.880 ± 0.024	0.67	5	0.708 ± 0.033	1.16
10	0.864 ± 0.025	0.44	10	0.750 ± 0.032	0.82
20	0.886 ± 0.023	0.30	20	0.800 ± 0.029	0.60
40	0.914 ± 0.021	0.21	40	0.840 ± 0.027	0.44

Tabelle 5.2: Beispiele für die Abdeckung der Konfidenzintervalle nach (5.4) für ein M/M/1-System und ein System mit ausfallenden Komponenten Law and Kelton [2000, S. 508-509]

nung von normalverteilten Beobachtungen ausgeht und sonst nur asymptotisch für $n \to \infty$ gilt. Auch hier kann die Abdeckung nur empirisch an Hand von Beispielen untersucht werden, bei denen der exakte Wert bekannt ist. In diesem Fall werden dann n Experimente durchgeführt, wobei für jedes Experiment eine Anzahl von Replikationen ausgeführt wird und ein Konfidenzintervall ermittelt wird. Wenn s dieser Experimente den wahren Wert enthalten, so ist $\hat{p} = s/n$ ein Schätzer für die Abdeckung des Konfidenzintervalls. Auch für diesen Schätzer kann man ein Konfidenzintervall zum Signifikanzniveau α' bestimmen, welches dann

$$\hat{p} \pm z_{\alpha'/2} \cdot \sqrt{\frac{\hat{p} \cdot (1-\hat{p})}{n}}$$

lautet, wobei $z_{\alpha'}$ das α' Quantil von $N(0,1)$ ist.

Die Ergebnisse von zwei Beispielen sind in Tabelle 5.2 zu finden. Das erste Beispiel ist ein $M/M/1$-System mit Bedienrate $\mu = 1$ und Ankunftsrate $\lambda = 0.9$. Ausgehend von einem leeren System wird die mittlere Verweilzeit der ersten 25 Aufträge ermittelt. Der Mittelwert und das zugehörige Konfidenzintervall wird für $n = 5, 10, 20, 40$ Replikationen bestimmt und das Experiment 500mal wiederholt, um die Abdeckung des 90% Konfidenzintervall zu schätzen. Die Abdeckung wird mit dem zugehörigen $\alpha' = 95\%$ Konfidenzintervall geschätzt. Ein solches Vorgehen ist hier möglich, da der exakte Wert der Verweilzeit analytisch berechenbar ist, so dass für jedes per Simulation bestimmte Konfidenzintervall entschieden werden kann, ob der exakte Wert im Intervall liegt oder nicht. Wie in der Tabelle deutlich wird, ist die Abdeckung des Konfidenzintervalls in allen Fällen, also auch für kleine n, gut und die Breite des Konfidenzintervalls nimmt mit steigendem n ab, wie es zu erwarten war.

In einem zweiten Beispiel wird ein System betrachtet, das aus drei Komponenten besteht. Jede Komponente hat eine Weibull-verteilte Ausfallzeit. Das System arbeitet, wenn die erste und die zweite oder die dritte Komponente arbeiten. Auch für dieses Beispiel können die exakten Ausfallzeiten numerisch bestimmt werden. Da die Weibull-Verteilung stark von einer Normalverteilung abweicht, ist zu erwarten, dass auch die Ausfallzeit insgesamt stark von der Normalverteilung abweicht. Darüber hinaus war im ersten Beispiel der Resultatwert aus den 25 Einzelresultaten zusammengesetzt, während nun der erste oder zweite Komponentenausfall zum Systemausfall führt. Wie man aus der Tabelle entnehmen kann, ist die Abdeckung der Konfidenzintervalle relativ schlecht, in keinem Fall ist die vorgegebene Abdeckung von 0.9 im Konfidenzintervall für die Abdeckung enthalten. In diesem Beispiel ist also n deutlich größer zu wählen, um eine vernünftige Abdeckung zu bekommen.

Bisher wurde die Anzahl Replikationen und das Signifikanzniveau α vorgegeben und es entstand ein Konfidenzintervall unbekannter Breite. Eigentlich soll aber ein Resultat mit vorgegebener Genauigkeit ϵ^* und vorgegebenem Signifikanzniveau α ermittelt werden. Es soll also gelten

$$1 - \alpha \approx P[\tilde{Y} - \epsilon \leq E(Y) \leq \tilde{Y} + \epsilon] = P[|\tilde{Y} - E(Y)| \leq \epsilon] \leq P[|\hat{Y} - E(Y)| \leq \epsilon^*]$$

Sei $n(\epsilon^*)$ die Anzahl Replikationen, die notwendig ist, damit die halbe Breite des Konfidenzintervalls kleiner gleich ϵ^* ist.

$$n(\epsilon^*) = \min \left\{ n : t_{n-1,1-\alpha/2} \sqrt{\frac{\hat{S}^2(n)}{n}} \leq \epsilon^* \right\}$$

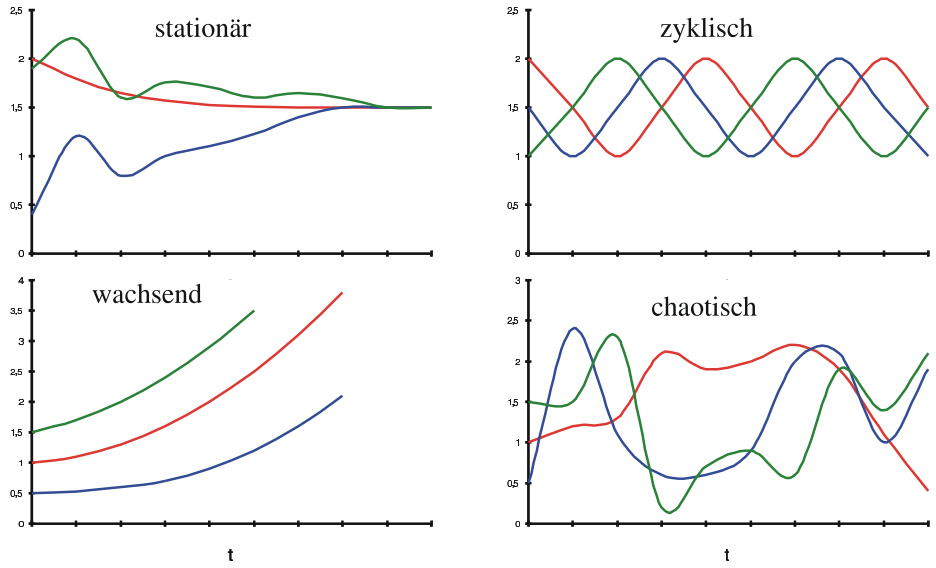


Abbildung 5.6: Mögliche Verläufe von $Y(t)$ für nicht-terminierende Simulationen.

Der Wert von $n(\epsilon^*)$ lässt sich nicht vorab bestimmen. Deshalb wird das Konfidenzintervall während der Simulation bestimmt und mit dem gewünschten Wert verglichen. Falls das Konfidenzintervall die gewünschte Breite hat, wird die Simulation abgebrochen.

In der Regel ist eine Genauigkeit γ relativ zum zu ermittelnden Wert $E(Y)$ zu bestimmen. Also $t_{n-1,1-\alpha/2} \cdot \left(\sqrt{\hat{S}^2(n)/n}\right) / E(Y) \leq \gamma$ sollte gelten, falls $E(Y) \neq 0$. Für $E(Y)$ gleich 0 oder nahe bei 0 sollte mit absoluten Werten gearbeitet werden. Da $E(Y)$ unbekannt ist, wird es durch $\hat{Y}(n)$ ersetzt und wir erhalten $\epsilon^* = \gamma \cdot \hat{Y}(n)$. Während der Simulation kann der Wert der notwendigen Beobachtungen $n(\gamma)$ aus den Ergebnissen nach n $(< n(\gamma))$ Beobachtungen geschätzt werden. Es gilt dann

$$t_{n-1,1-\alpha/2} \cdot \sqrt{\frac{\hat{S}^2(n)}{n(\gamma)}} \leq \max(|\hat{Y}(n)| \cdot \gamma, \gamma_{abs}) \;\Rightarrow\; n(\gamma) \approx \frac{(t_{n-1,1-\alpha/2})^2 \cdot \hat{S}^2(n)}{\max((|\hat{Y}(n)| \cdot \gamma), \gamma_{abs})^2}$$

Ein weiterer Aspekt, der bisher nicht explizit behandelt wurde, ist die Bestimmung des initialen Zustands zu Beginn der Simulation. Bei der terminierenden Simulation ist der initiale Zustand oft vorgegeben. Übliche Beispiele sind, dass alle Komponenten lauffähig sind, das System leer ist, etc. Wenn der initiale Zustand nicht vorgegeben ist, so soll oft in einem typischen Zustand gestartet werden. Ein solcher kann aus Messungen resultierend oder in einer Vorabsimulation vor dem Start der eigentlichen Simulation erzeugt werden. Einige weitere Details werden bei der nicht-terminierenden Simulation betrachtet.

5.2.2 Nicht-terminierende Simulation

Im Gegensatz zur terminierenden Simulation mit einem endlichen Beobachtungszeitraum geht es bei der nicht-terminierenden Simulation darum, ein System über einen potenziell unendlichen Zeitraum zu beobachten und zu analysieren. In der Realität läuft natürlich kein System und auch kein Modell unendlich lange. Bei nicht terminierenden Simulationen wird aber davon ausgegangen, dass das System lange genug läuft, um typisches Verhalten zu zeigen. Bei unendlicher/sehr langer Laufzeit kann ein Modell/System unterschiedliche Verhaltensweisen zeigen. Abbildung 5.6 stellt den Verlauf von $E(Y(t))$ für vier verschiedene Systeme und jeweils drei unterschiedliche initiale Zustände s_0 dar. Der initiale Zustand wird auch als Startzustand bezeichnet. Es ist zu beachten, dass hier tatsächlich der Verlauf von $E(Y(t))$ dargestellt wird, der so nicht beobachtet werden kann.

Eine einzelne Simulation würde stochastische Schwankungen aufweisen und ließe sich deshalb nicht so eindeutig einordnen. Es lassen sich auf Basis der Beobachtung von $E(Y(t))$ vier Ablaufmuster unterscheiden.

Im ersten Fall, dem stationären Verhalten, konvergiert $E(Y(t))$ gegen einen Wert für $t \to \infty$ unabhängig vom initialen Zustand. Im Allgemeinen gilt dann sogar $\lim_{t\to\infty} Y(t) = Y$ für eine Zufallsvariable Y, d.h. nicht nur der Erwartungswert, sondern die gesamte Verteilung konvergiert unabhängig vom Startzustand. Bei zyklischem Verhalten ändert sich $E(Y(t))$, es existiert aber ein $\Delta > 0$, so dass $E(Y(t)) = E(Y(t + \Delta))$. Je nach Startzustand verschiebt sich der Verlauf. Bei wachsendem Verhalten wächst $Y(t)$ mit t über alle Grenzen, d.h. $\lim_{t\to\infty} Y(t) = \infty$. Der letzte Fall ist das chaotische Verhalten, bei dem $E(Y(t))$ in Abhängigkeit von s_0 ganz unterschiedliche Verhaltensmuster aufweist. Kleine Änderungen im Startzustand können zu größeren Änderungen beim beobachteten Verhalten führen.

Wir werden uns nur mit den ersten beiden Abläufen beschäftigen, da die beiden letzten Abläufe vom Standpunkt der Systemanalyse nur durch terminierende Simulationen untersucht werden können. Primär wird uns die stationäre Simulation interessieren, da diese sehr breit eingesetzt wird, wenn Systeme lange laufen und einen *eingeschwungenen Zustand* erreichen, d.h. $E(Y(t))$ ist konstant über einen großen Teil des Beobachtungsintervalls $[0, T]$.

5.2.3 Stationäre Simulation

Sei $E(Y(t)|s_0)$ der Erwartungswert von Y zum Zeitpunkt t unter der Bedingung, dass das Modell zum Zeitpunkt 0 im Zustand s_0 war. Bei den im Folgenden behandelten Systemen wird davon ausgegangen, dass das Modell den stationären Zustand unabhängig vom Startzustand erreicht. D.h. $\lim_{t\to\infty} E(Y(t)|s_0) = E(Y)$ für alle s_0. Ziel der Simulation ist die Bestimmung von $E(Y)$. Da die Anforderung als Grenzwert definiert ist, werden keine Aussagen über endliche Beobachtungsintervalle gefordert. In den meisten Fällen wird deshalb gelten

$$E(Y(t)|s_0) \neq E(Y(t)|s_0') \quad \text{und} \quad E(Y(t')|s_0) \neq E(Y(t)|s_0)$$

für $t \neq t'$ und $s_0 \neq s_0'$. Um sinnvolle Ergebnisse per Simulation ermitteln zu können, wird deshalb weiterhin vorausgesetzt, dass ein Zeitpunkt t_s existiert, so dass für alle $t \geq t_s$

$$E(Y(t)|s_0) \approx E(Y(t)|s_0') \approx E(Y)$$

für alle s_0, s_0'. Auch diese Anforderung ist natürlich formal nicht nachweisbar und muss vorausgesetzt und mittels Beobachtungen der Simulation validiert werden. In der Regel wird der Zeitpunkt t_s ab dem $E(Y(t)|s_0) \approx E(Y)$ gilt von s_0 abhängig sein. Die Wahl von s_0 wird später untersucht und es wird davon ausgegangen, dass s_0 fest vorgegeben ist. Zur Vereinfachung der Schreibweise benutzen wir in diesem Fall $E(Y(t))$ statt $E(Y(t)|s_0)$.

Der erste Schritt der Auswertung ist die Bestimmung von t_s, so dass $E(Y(t)) \approx E(Y)$ für $t \geq t_s$. Da vor dem Zeitpunkt t_s offensichtlich $E(Y(t)) \neq E(Y)$ gilt, sollte wie folgt bei der Simulation vorgegangen werden:

- Starte den Simulator im Zustand s_0 zum Zeitpunkt $t = 0$.
- Simuliere bis zum Zeitpunkt t_s, ohne Daten zu erheben.
- Starte die Beobachtung der Simulation zum Zeitpunkt t_s und simuliere bis zum Simulationsende.

Analog zum allgemeinen Fall, können einer oder mehrere Simulationsläufe durchgeführt werden. Bei einer Beobachtungslage wie auf Seite 106 wurden n Simulationsläufe durchgeführt und in jedem Lauf m Beobachtungen getätigt. Alle Beobachtungen wurden nach dem Zeitpunkt t_s erhoben. Zur Wahl von t_s gibt es zwei widersprüchliche Argumente:

- Das Sicherheitsargument impliziert, dass t_s möglichst groß gewählt wird, damit man sicher ist, dass $E(Y(t)) \approx E(Y)$ für $t \geq t_s$ wirklich gilt.

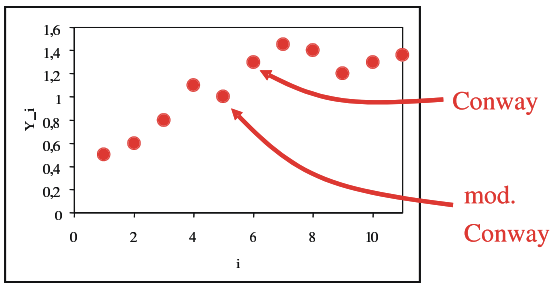


Abbildung 5.7: Beispiel für die Anwendung der ersten und modifizierten Methode von Conway.

- Das Aufwandsargument impliziert, dass t_s möglichst klein gewählt wird, damit möglichst viele Beobachtungen ermittelt werden können.

Nahe liegender Weise ist t_s natürlich vom Modell abhängig und sollte so klein wie möglich und so groß wie nötig gewählt werden. Das Intervall $[0, t_s)$ bezeichnet man auch als *transiente Phase* der Simulation. Es gibt zahlreiche Heuristiken und statistische Tests zur Ermittlung von t_s. Kein Ansatz kann als wirklich befriedigend angesehen werden, wie auch in der Literatur bestätigt wird. Die meisten Simulationswerkzeuge beinhalten deshalb auch keine automatischen Methoden zur Ermittlung von t_s, es wir vielmehr vorausgesetzt, dass t_s vom Modellierer gesetzt wird. Dies ist natürlich kritisch und kann zu Verfälschungen der Simulationsresultate führen. Deshalb sollen einige Methoden zur Ermittlung von t_s aus der Beobachtung der Simulation vorgestellt werden.

t_s muss auf Grund der Beobachtung der zu messenden Leistungsgröße ermittelt werden. Sei $(y_1, \ldots, y_n)$ die zugehörige Stichprobe. Gesucht wird ein Index i, so dass die Werte $y_1, \ldots, y_{i-1}$ bei der Auswertung unberücksichtigt bleiben, da sie vor t_s erhoben wurden. Die folgenden beiden Regeln zur Bestimmung von i wurden von Conway 1963 und 1978 publiziert (siehe auch Asmussen et al. [1992]).

- Sei $y_k^+ = \max(y_k, \ldots, y_n)$ und $y_k^- = \min(y_k, \ldots, y_n)$, wähle $i = \min_k(y_k^- < y_k < y_k^+)$.
- Sei $y_k^+ = \max(y_1, \ldots, y_k)$ und $y_k^- = \min(y_1, \ldots, y_k)$, wähle $i = \min_k(y_k^- < y_k < y_k^+)$.

Der Vorteil der zweiten Variante ist, dass zur Laufzeit entschieden werden kann, ab wann Daten erhoben werden, während in der ersten Variante erst die gesamte Stichprobe vorliegen muss. Abbildung 5.7 zeigt ein Beispiel für die Anwendung der Methode. Beide Regeln funktionieren allerdings nicht, wenn sie direkt auf einzelne Messdaten angewendet werden. Die einzelnen Werte der Stichprobe müssen vielmehr Mittelwerte mehrerer Messungen sein, ansonsten wird die Länge der transienten Phase gravierend unterschätzt. Die Zahl der Einzelwerte, aus denen Mittelwerte gebildet werden, hängt vom Verlauf der Beobachtungen ab, weshalb der Ansatz kaum automatisierbar ist.

Etwas einfacher einzusetzen ist die so genannte "crossing the mean" Regel. Der Mittelwert der ersten k Werte der Stichprobe ist gegeben durch

$$\bar{Y}_k = \frac{1}{k} \cdot \sum_{j=1}^{k} y_j$$

Der Wert von i wird so gewählt, dass $\bar{Y}_k$ $(k = 1, \ldots, i)$ von den Werten y_{k-1} und y_k genau K mal gekreuzt wird. Eine Kreuzung liegt dann vor, wenn $y_{k-1} < \bar{Y}_k < y_k$ oder $y_{k-1} > \bar{Y}_k > y_k$. K wird in der Regel zwischen 3 und 5 gewählt. Abbildung 5.8 zeigt ein Beispiel für die Anwendung der Regel.

Der letzte Ansatz benutzt die so genannte "batch means" Methode, die auch zur Ermittlung der Konfidenzintervalle für Resultate stationärer Simulationen eine große Rolle spielt. Die Stichprobe

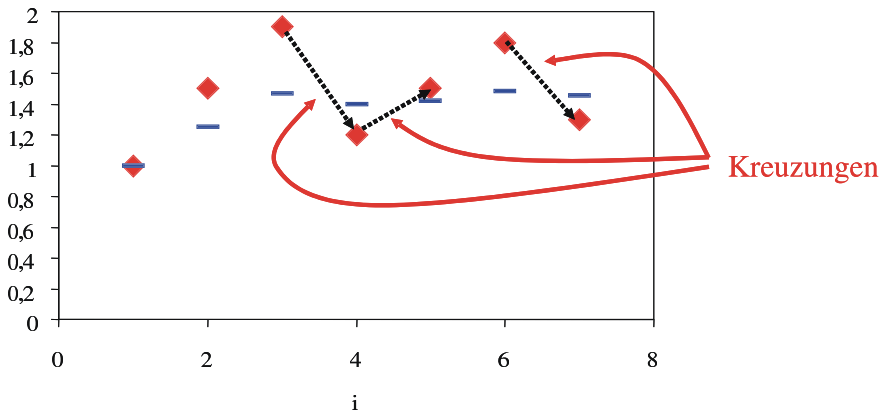


Abbildung 5.8: Beispiel für die Anwendung der "crossing the mean" Regel.

wird dazu in Gruppen (batches) der Größe i ($i = 5, 10, \ldots, n$) unterteilt. Der Gruppenmittelwert lautet dann

$$\bar{Y}_j(i) = \frac{1}{i} \cdot \sum_{k=(j-1)\cdot i+1}^{i\cdot j} y_k$$

Wenn m Gruppen entstehen, so ergibt sich der Gesamtmittelwert aus

$$\hat{Y} = \bar{Y}(i) = \frac{1}{m} \cdot \sum_{k=1}^{m} \bar{Y}_k(i)$$

Mit wachsendem i (steigender batch-Größe), schwindet der Einfluss der ersten Beobachtungen. Falls die Sequenz $\bar{Y}_2(i), \ldots, \bar{Y}_m(i)$ keinen Trend mehr erkennen lässt, werden die ersten i Beobachtungen gelöscht und das Resultat aus den verbleibenden Werten berechnet. Auf Seite 123 wird ein Verfahren zur Auswertung stationärer Simulationen vorgestellt, dass diese Art der Bestimmung von i und damit t_s einbezieht.

Weitere Ansätze und theoretische Resultate zur Ermittlung von t_s kann man in Asmussen et al. [1992] finden. Neuer Ergebnisse in Grassmann [2008] zeigen für einfache Systeme, dass der Start in einem "*nicht zu unwahrscheinlichen Zustand*" ausreicht, um die Auswirkungen der transienten Phase zu beseitigen. Inwieweit sich diese Ergebnisse auf komplexere Modelle übertragen lassen, bleibt abzuwarten.

Da t_s von s_0 abhängt, sollte s_0 natürlich so gewählt werden, dass t_s möglichst klein ist. So wird es bei einem hoch ausgelasteten System länger dauern bis der stationäre Zustand erreicht wird, wenn mit leeren Puffern gestartet wird, als wenn die Puffer direkt gefüllt werden. Bei einem System mit niedriger Auslastung ist es gerade umgekehrt. In der Literatur gibt es viele Heuristiken zur Wahl des Startzustandes. Üblicherweise reicht es aus, einen einfach implementierbaren und erreichbaren Zustand zu wählen. Dies wird oft ein System mit leeren Puffern und funktionsfähigen Maschinen sein.

Die Simulation wird durch das beschriebene Vorgehen in zwei Phasen unterteilt. In Phase 1, die das Intervall $[0, t_s)$ umfasst, werden keine Daten erhoben. In der zweiten Phase, die zum Zeitpunkt t_s beginnt und mit dem Ende der Simulation endet, werden Daten erhoben. Auf Grund der Annahmen kann davon ausgegangen werden, dass alle in der zweiten Phase erhobenen Werte aus der gleichen Verteilung stammen. Dies bedeutet natürlich nicht, dass die Daten unabhängig sind. Zur Erinnerung sei noch einmal erwähnt, dass zwei Zufallsvariablen X und Y unabhängig sind, wenn $P[X \leq x, Y \leq y] = P[X \leq x] \cdot P[Y \leq y]$ gilt. In diesem Fall ist die Kovarianz $C(X, Y)$ gleich 0. Wie in (5.1) gezeigt ergibt sich die Varianz des Schätzer $\tilde{Y}$ zu

$$\sigma^2(\tilde{Y}) = \frac{1}{n^2} \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} C(Y_i, Y_j) \quad \text{mit} \quad C(Y_i, Y_i) = \sigma^2(Y_i) \tag{5.5}$$

Die Varianz des Schätzers geht direkt in die Breite der Konfidenzintervalle ein (siehe (5.3)). Die bisherige Schätzung $\hat{S}^2$ (siehe (5.2)) berücksichtigt nur die Terme in der obigen Formel bei denen $i = j$ ist und führt daher zu erheblichen Verfälschungen, falls die restlichen Terme nicht fast 0 sind. Wenn die Kovarianz positiv ist, wäre die tatsächliche Varianz größer als der ermittelte Wert und die so berechneten Konfidenzintervalle wären zu optimistisch, d.h. die Abdeckung wäre u.U. deutlich kleiner als $1 - \alpha$. Dies bedeutet, dass die Simulation eine nicht gerechtfertigte Sicherheit vortäuscht.

Da nicht terminierendes Verhalten untersucht wird, gibt es zwei Möglichkeiten der Datenerhebung

1. n Simulationsläufe mit je m Beobachtungen und
2. ein Simulationslauf mit $n \cdot m$ Beobachtungen

In beiden Varianten werden gleich viele Beobachtungen erzeugt. In der ersten Version treten allerdings n transiente Phasen auf, so dass n-mal Werte vor t_s verworfen werden. Bei der zweiten Variante werden nur einmal Werte verworfen. Damit liefert die zweite Variante mehr verwertbare Beobachtungen bei gleichem Aufwand. Betrachten wir nun die Unabhängigkeit der Beobachtungen.

Im ersten Fall sind Y_{ij} und Y_{kj} unabhängig, wenn $i \neq k$ und die Saaten des Zufallszahlengenerators entsprechend gewählt werden. Damit ergibt sich ein nahe liegendes Vorgehen:

- Führe n Replikationen durch, wobei die Länge m so gewählt wird, dass y_{im} nach t_s erhoben wird.
- Verwende y_{im} $(i = 1, \ldots, n)$ zur Berechnung von $\hat{Y}$.

Der Wert von t_s kann mit einem der vorgestellten Ansätze oder mit einer Kombination der Ansätze ermittelt werden. Falls die batch means Methode verwendet wird, so kann y_{ij} durch den Mittelwert der batches $j = 2, \ldots, m$ ersetzt werden. Das Verfahren lässt sich sehr gut parallelisieren, da die einzelnen Simulationsläufe unabhängig voneinander ausgeführt werden können.

Der Nachteil des Vorgehens, das auch als *unabhängige Replikationen* bezeichnet wird, ist die mehrfache Simulation der transienten Phase. Dieser Punkt spricht für die Anwendung der zweiten Variante. Seien Y_j und Y_k die Zufallsvariablen, die die $j$te und $k$te Beobachtung beschreiben $(1 \leq j, k, \leq n \cdot m)$. Falls beide Beobachtungen nach t_s erhoben wurden, so können sie als identisch verteilt angesehen werden. Die Unabhängigkeit ist zumindest in manchen Fällen fragwürdig, wie man an den folgenden Beispielen sehen kann. Sei dazu $j < k$.

- Wenn Y_j die Wartezeit des $j$ten Kunden und Y_k die Wartezeit des $k$ten Kunden ist, so haben beide Kunden u.U. gleichzeitig gewartet, wenn die Differenz zwischen k und j nicht zu groß ist oder die Warteschlange sehr lang war. Offensichtlich sind beide Größen dann positiv korreliert.
- Wenn Y_j die Pufferbelegung zum Zeitpunkt $j \cdot \Delta$ ist und k nicht viel größer als j ist, so umfasst die Pufferbelegung Y_k zum Zeitpunkt $k \cdot \Delta$ zum Teil dieselben Kunden. Auch in diesem Beispiel sind die Zufallsvariablen positiv korreliert.

Die Beispiele zeigen, dass oft positive Korrelationen auftreten werden und, wie oben erläutert, die Konfidenzintervalle zu schmal sein werden bzw. nicht die gewünschte Abdeckung haben. Eine Berücksichtigung der Kovarianz oder besser Autokorrelation, da die Werte Realisierungen eines Prozesses sind, erfordert zusätzliche Annahmen, die in der Praxis aber meistens erfüllt sind. So geht man davon aus, dass der Prozess schwach stationär ist. Dies bedeutet, dass

$$\sigma^2(Y_i) = \sigma^2(Y) = \sigma^2 \text{ und } C(Y_i, Y_j) = C_F(Y, |i - j|)$$

Die Varianz ist in diesem Fall konstant und identisch für alle Werte und die Kovarianz hängt vom Abstand der Werte ab. Wir haben bisher nicht definiert, was Abstand bedeutet. Den unterschiedlichen Interpretationen der Resultate entsprechend unterscheidet man zwischen individuenbezogenen Werten, wie der Verweilzeit eines Kunden im System, und zeitbezogenen Werten, wie der

Pufferfüllung zu den Zeitpunkten t_i und t_j. Bei zeitbezogenen muss man in der Regel voraussetzen, dass der Abstand $|t_i - t_j| = |t_k - t_l|$ ist, falls $|i - j| = |k - l|$ ist. Auf dieser Basis kann man den Autokorrelationskoeffizienten der Ordnung s definieren. Es gilt

$$\rho(s) = \rho(|i-j|) = \frac{C_F(Y, |i-j|)}{\sigma^2(Y)} \tag{5.6}$$

wobei $\rho(0) = 1$ ist. Bei bekannten Autokorrelationskoeffizienten kann man die folgende Darstellung der Varianz des Schätzers $\tilde{Y}$ aus (5.5) und (5.6) herleiten.

$$\begin{aligned}
\sigma^2(\tilde{Y}) &= \tfrac{1}{n^2} \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} C(Y_i, Y_j) \\
&= \tfrac{1}{n^2} \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} \rho(|i-j|) \cdot \sigma^2(Y) \\
&= \tfrac{\sigma^2(Y)}{n} \cdot \left(1 + \tfrac{1}{n} \cdot \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \rho(|i-j|) \right) \\
&= \tfrac{\sigma^2(Y)}{n} \cdot \left(1 + \tfrac{2}{n} \cdot \sum_{i=1}^{n} \sum_{j=1, j>i}^{n} \rho(|i-j|) \right) \\
&= \tfrac{\sigma^2(Y)}{n} \cdot \left(1 + \tfrac{2}{n} \cdot \sum_{s=1}^{n-1} (n-s) \cdot \rho(s) \right)
\end{aligned}$$

Wie bereits ausgeführt, kann man in den meisten Fällen davon ausgehen, dass $\rho(s) > 0$ gilt und damit die Summe positiv ist und die Varianz des Schätzers vergrößert wird. Es ist natürlich nicht realistisch, $\rho(s)$ und $\sigma^2(Y)$ als bekannt vorauszusetzen. Deshalb müssten die Werte von $\rho(s)$ auch aus der Stichprobe geschätzt werden. Dies ist zwar im Prinzip möglich, führt aber in der Praxis zu Problemen, da die bekannten Schätzer wenig robust sind und insbesondere gegenseitig abhängig sind. Dies bedeutet, dass die Schätzungen für mehrere $\rho(s)$ voneinander abhängigen, wenn also der Wert von $\rho(1)$ unterschätzt wird, so wird mit großer Wahrscheinlichkeit auch der Wert von $\rho(2)$ unterschätzt, wenn dieselben Daten verwendet werden. Man müsste also entweder für jedes s neue Daten verwenden, was den Aufwand zu stark erhöhen würde oder es müssten weitere Annahmen über die Struktur von $\rho(s)$ gemacht werden. Letzteres ist die Basis der Spektralmethode und auch der autoregressiven Modelle. Da beide Ansätze einige, zum Teil nicht einfach zu prüfenden, Voraussetzungen benötigen und nicht blind anwendbar sind, werden sie hier nicht weiterverfolgt.

Wir wollen hier einen Ansatz betrachten, der mit relativ wenigen Voraussetzungen auskommt und weitgehend automatisierbar ist. Wir nehmen dazu an, dass $|\rho(s)|$ eine monoton fallende Funktion mit $\lim_{s\to\infty} \rho(s) = 0$. Also bei steigendem Abstand gibt es eine fallende Abhängigkeit von Beobachtungen. Diese Annahme ist plausibel, wie man an den folgenden Beispielen ablesen kann.

- Die Wartezeit des $k$ten Kunden wird nicht von der Wartezeit des $j$ten Kunden abhängen, wenn j lange vor k im System war.
- Die Population zum Zeitpunkt $k \cdot \Delta$ ist unabhängig von der Population zum Zeitpunkt $j \cdot \Delta$, wenn der Abstand $|j \cdot \Delta - k \cdot \Delta|$ genügend groß ist.

Wenn die Annahme stimmt, so gibt es einen Wert s_c, so dass $\rho(s) \approx 0$ für $s \geq s_c$. Ein nahe liegender Ansatz ist die batch means-Methode mit einer batch-Größe, die groß genug ist, um anzunehmen, dass die batch-Mittelwerte unkorreliert sind. Aus den batch-Mittelwerten kann dann ein Konfidenzintervall für den Mittelwert berechnet werden, wobei die Ansätze für unabhängige Daten angewendet werden.

Es wird nun ein Verfahren vorgestellt, um Konfidenzintervalle für $E(Y)$ aus den Beobachtungen eines einzelnen Simulationslaufs zu schätzen. Seien $(y_1, \ldots, y_m)$ die beobachteten Werte. Die ersten i_0 Werte werden verworfen, da sie zur transienten Phase gehören. Der Wert von i_0 kann mit einer

der beschriebenen Methoden geschätzt werden. Die restlichen Daten werden in batches der Größe i_1 unterteilt. Die Schätzung von i_1 ist ebenfalls schwierig, da die Größe so gewählt werden muss, dass die Mittelwerte der batches unkorreliert sind. Für gegebene m, i_0 und i_1 entstehen $(m-i_0)/i_1$ batches[2]. Batch k umfasst die Werte $i_0 + (k-1) \cdot i_i + 1$ bis $i_0 + k \cdot i_1$ und es gilt

$$\bar{Y}_k(i_1) = \frac{1}{i_1} \cdot \sum_{j=i_0+(k-1)\cdot i_1+1}^{i_0+k\cdot i_1} y_j$$

Ausgehend von einer gegebenen Beobachtungszahl m kann man die Anzahl der batches K wählen. Ein größeres K sorgt dafür, dass mehr batch-Mittelwerte zur endgültigen Auswertung vorliegen. Bei einem kleineren K werden mehr Werte in einem batch zusammengefasst, dadurch sinkt die Varianz der batch-Mittelwerte und die Verteilung der batch-Mittelwerte wird ähnlicher zu einer Normalverteilung. Letztere Argumente überwiegen in der Regel, deshalb wird K meist zwischen 10 und 30 gewählt.

Es bleibt damit noch die Frage, ob die batch Mittelwerte unkorelliert sind. Dies kann natürlich nur auf Basis der Stichprobenwerte untersucht werden. Üblicherweise wird dazu $\rho(1)$ geschätzt und falls dabei $\rho(1) \approx 0$ gilt, wird Unabhängigkeit angenommen. Der Schätzwert für den Autokorrelationskoeffizienten der Ordnung 1 wird wie folgt berechnet

$$\hat{\rho}(1) = \frac{\sum_{k=1}^{K-1} \left(\left(\bar{Y}_k - \hat{Y} \right) \cdot \left(\bar{Y}_{k+1} - \hat{Y} \right) \right)}{\sum_{k=1}^{K-1} \left(\bar{Y}_k - \hat{Y} \right)^2}$$

Da der Schätzer verzerrt ist, sollte $\hat{\rho}(1)$ für ein nicht zu kleines K bestimmt werden. Die angegeben Wert von 10 bis 30, die für die Schätzung der Mittelwerte und Varianz ausreichen, werden als zu klein angesehen, vorgeschlagen wird $K \in [100, 400]$ (siehe Banks et al. [2000, S: 437]). Falls $\hat{\rho}(1)$ *klein genug* ist, können anschließend batches zusammengefasst werden, so dass K verkleinert wird.

Damit kann die folgende heuristische batch means-Methode zur Bestimmung der Konfidenzintervalle für stationäre Größen aus einem Simulationslauf verwendet werden.

1. Ermittle während der Simulation den Wert von i_0.
2. Wähle ein initiales $m \geq 10 \cdot i_0$
3. Unterteile die Daten in K $(100 \leq K \leq 400)$ batches und bestimme während des Simulationslaufs deren Mittelwerte $\bar{Y}_k$ $(k = 1, \ldots, K)$
4. Bestimme den Schätzwert $\hat{\rho}(1)$ für den Autokorrelationskoeffizienten der Ordnung 1.
5. Teste Autokorrelation
   (a) falls $\hat{\rho}(1) \leq 0.2$ reduziere die Anzahl batches auf L $(20 \leq L \leq 30)$ durch Zusammenfassen der bisherigen batch Mittelwerte und bestimme das Konfidenzintervall aus einer t-Verteilung mit $L-1$ Freiheitsgraden.
   (b) ansonsten verdopple m und fahre bei 3. fort.

Der Ansatz lässt sich relativ einfach implementieren. Insbesondere wird nur ein Array der Größe K zur Datenspeicherung benötigt, wenn die Mittelwerte jeweils zur Laufzeit berechnet werden.

[2]Wir nehmen zur Vereinfachung an, dass die Werte so gewählt werden, dass eine ganzzahlige Anzahl von batches entsteht.

5.3 Schätzung mehrerer Leistungsmaße aus einem Simulationslauf

In den meisten Simulationsläufen wird mehr als ein Leistungsmaß ermittelt. Beispiele sind die Ermittlung der Verweilzeit an mehreren Stationen oder die Bestimmung von Durchsätzen und mittleren Pufferbelegungen. Wenn für alle Leistungsmaße Konfidenzintervalle ermittelt wurden, stellt sich die Frage, welche Aussage man über die Qualität aller Resultate machen kann.

Wir nehmen an, dass k Leistungsmaße ermittelt werden sollen und α_i die Signifikanzwahrscheinlichkeit des $i$ten Leistungsmaßes ist. Wenn die Schätzer für die einzelnen Leistungsmaße unabhängig sind, dann liegen mit Wahrscheinlichkeit

$$1 - \alpha = \prod_{i=1}^{k} (1 - \alpha_i)$$

alle Leistungsmaße innerhalb der ermittelten Konfidenzintervalle. Wenn alle α_i identisch sind, so kann man über die Wahrscheinlichkeiten einer Binomialverteilung Wahrscheinlichkeiten dafür ermitteln, dass l von k Resultatwerte innerhalb ihrer Konfidenzintervalle liegen.

Die Annahme unabhängiger Schätzer für mehrere Leistungsmaße ist aber in den meisten Fällen unrealistisch. So sind z.B. Verweilzeit und Warteschlangenlänge an einer Station stark miteinander korreliert, auch die Verweilzeit an aufeinander folgenden Warteschlangen ist oftmals korreliert. Für korrelierte Schätzer gibt es sehr wenige Resultate. Das allgemeinste Ergebnis ist die so genannte Bonferroni-Ungleichung, die eine untere Schranke für die Wahrscheinlichkeit liefert, dass alle Konfidenzintervalle den wahren Wert beinhalten.

$$1 - \alpha \geq 1 - \sum_{i=1}^{k} \alpha_i$$

Man sieht, dass für größere k schnell der triviale Fall $1-\alpha \geq 0$ erreicht wird. Für kleinere Werte von k, vorgegebene Breiten der Konfidenzintervalle ϵ_i und vorgegebene Signifikanzwahrscheinlichkeit α kann das folgende heuristische Vorgehen angewendet werden.

- Bestimme während der Simulation α_i, so dass alle Konfidenzintervallbreiten $\leq \epsilon_i$ sind.
- Falls $(1 - \sum_{i=1}^{k} \alpha_i) \geq 1 - \alpha$ beende die Simulation, ansonsten fahre mit der Simulation fort.

Die während der Simulation ermittelten α_i hängen von den vorgegebenen ϵ_i ab und unterscheiden sich für die verschiedenen Leistungsmaße. Der Ansatz funktioniert praktisch nur für kleine Werte von k, da der Aufwand auf Grund der konservativen Schätzung von $1 - \alpha$ sonst sehr groß wird.

5.4 Schätzung stationärer zyklischer Resultate

Das bisherige Vorgehen eignete sich für stationäre Prozesse, bei denen der Erwartungswert gegen einen festen Wert konvergiert. Wie in Abbildung 5.6 gezeigt, existiert aber auch stationär zyklisches Verhalten, welches mittels nicht-terminierender Simulation analysierbar ist. In diesem Fall gibt es eine Zykluszeit Δ_C, so dass $F_{Y_i}(y) = F_{Y_{i+\Delta_C}}(y)$ bzw. $F_{Y_t}(y) = F_{Y_{t+\Delta_C}}(y)$ ($i \in \mathbb{N}$, $t \in \mathbb{R}_{\geq 0}$). In manchen Fällen ist Δ_C aus der Modellbeschreibung bekannt. So ist in allen Modellen in denen ein Zeitablauf über Tage, Wochen etc. beschrieben wird, der Zeitablauf in den Eingabeparametern definiert. In anderen Fällen muss Δ_C erst geschätzt werden. Dies kann dadurch geschehen, dass batches unterschiedlicher Größe gebildet werden und die batch Mittelwerte auf Gleichheit getestet werden. Falls die batch Größe Δ_C entspricht, so sollen die batch Mittelwerte identisch sein.

Nachdem Δ_C bekannt ist, können Werte $E(Y_i)$ bzw. $E(Y_t)$ mit $i, t \in [0, \Delta_C)$ geschätzt werden. Dazu werden die Beobachtungen $y_{i+k\cdot\Delta_C}$ bzw. $y_{t+k\cdot\Delta_C}$ ($k = 0, 1, 2 \ldots$) verwendet und die Methoden der stationären Analyse benutzt. Falls mehrere Werte aus einer Simulation zu ermitteln sind, ergeben sich die schon angesprochenen Probleme der Ermittlung simultaner Konfidenzintervalle.

Kapitel 6

Simulationssoftware

Der Abschnitt über Simulationssoftware wird in dieser Version der Notizen nicht weiter ausgeführt, da es sich im Wesentlichen um eine informelle Vorstellung unterschiedlicher Ansätze handelt, die am Besten in der Originalliteratur über die entsprechenden Softwaresysteme nachgelesen werden kann.

Es werden kurz die relevanten Literaturstellen für die in der Vorlesung vorgestellten Ansätze vorgestellt. Eine Übersicht über die Simulationssprache GPSS findet man in Karian and Dudewicz [1998]. Die Sprache DEMOS wurde ursprünglich von Birwistle als Erweiterung von Simula entwickelt Birwistle [1985]. Inzwischen gibt es verschiedene Versionen in Java, u.a. JavaDemos von der Universität Essen-Duisburg Müller-Clostermann [2005] und DESMO-J der Universität Hamburg Page et al. [2000]. Page and Kreutzer [2005] gibt einen Überblick über die Simulation unter Nutzung von UML und DESMO-J. Arena wird in Kelton et al. [2007] detailliert vorgestellt Für das Netzwerktool OmNeT++ existiert eine hervorragende Web-Seite mit ausführlicher Dokumentation OMN. Der Prozesskettenansatz ProC/B wird schließlich in Bause et al. [2002] vorgestellt.

Für eine Übersicht über Simulationssoftware sei auf die Folienkopien zur Vorlesung verwiesen. Weitere allgemeine Ausführungen findet man auch in Banks et al. [2000, Kap. 4] und Law and Kelton [2000, Kap. 3]. Übersichten über spezielle Werkzeuge sind auch unter den Papieren der Winter Simulation Konferenzen zu finden WSC.

Kapitel 7

Möglichkeiten und Grenzen der Simulation

Wir haben Simulation als eine Methode kennen gelernt, um Experimente an realen Systemen durch Experimente an einem Modell zu ersetzen. Es wurde deutlich, dass die Statistik eine zentrale Rolle bei der Modellbildung und Modellauswertung spielt und dass dadurch viele Aspekte nicht beweisbar sondern nur testbar sind. Damit ist natürlich die Möglichkeit von Modellierungsfehlern und Fehlinterpretationen gegeben. Man sollte sich deshalb der Möglichkeiten und Grenzen der Simulation bewusst sein, wenn man diese einsetzt. In diesem Abschnitt werden einige generelle Aspekte untersucht, während der folgende Abschnitt sich Methoden zur Bewertung der Realitätsnähe von Simulationsmodellen widmet.

Die Vorteile der Simulation als Instrument der Systemanalyse liegen klar auf der Hand.

- Viele reale Systeme können mit analytischen Modellen nicht genau genug beschrieben werden, während in einem Simulationsmodell die realen Abläufe sehr gut nachgebildet werden können.
- Simulation bietet die Möglichkeit, Systeme in ganz unterschiedlichen Umgebungen und unter unterschiedlichen Bedingungen zu analysieren. Diese Bedingungen können aus der realen Umwelt stammen, können aber auch vollkommen irreal sein.
- Modifikationen am System sind oft durch einfache Änderungen am Simulator modellierbar.
- Experimente können prinzipiell relativ einfach und kostengünstig durchgeführt werden. Da in der Simulation praktisch alle Faktoren beeinflussbar sind, lassen sich Experimente unter quasi beliebigen Bedingungen durchführen.
- Simulation erlaubt es Systeme auch in sehr kurzen oder sehr langen Zeitintervallen zu beobachten, wenn der Simulator die Zeit entsprechend streckt oder staucht.
- Mit heutigen Rechnerkapazitäten lassen sich viele und aufwändige Simulationsexperimente mit den vorhandenen Ressourcen durchführen.

Diese Vorteile der Simulation verdecken oft den Blick auf die leider auch vorhandenen Nachteile des Vorgehens.

- Die meisten Simulationsmodelle sind stochastisch, so dass Resultate nur geschätzt werden können. Die dazu verwendeten Methoden basieren fast immer auf zwar plausiblen aber nicht beweisbaren Annahmen, so dass für einzelne Modelle falsche Resultate ermittelt werden.
- Simulationsmodelle haben einen hohen Datenbedarf, der bei vielen realen Experimenten nicht befriedigt wird. So kann ein Modell nicht besser als der schwächste Punkt sein. In der Simulation bedeutet dies oft, dass zwar detaillierte Modelle erstellt werden, die benötigten

Parameter aber nur auf Grund von sehr wenigen Beobachtungen geschätzt werden und in manchen Fällen nur auf Grund von nicht validierten Annahmen gesetzt werden.

- Simulationsmodelle sind aufwändig und teuer in der Entwicklung,
  - neben den üblichen Problemen bei der Erstellung großer Programme
  - treten simulationsspezifische Probleme, wie die Datenmodellierung, auf.
- Simulationsmodelle liefern in manchen Fällen extreme Datenmengen, die oft nicht detailliert genug analysiert werden, so dass Resultate oft überinterpretiert oder falsch interpretiert werden.

Grundsätzlich lassen sich Fehlinterpretationen von Simulationsresultaten nicht ausschließen. Es gibt allerdings zahlreiche Fehler, die sich bei einem systematischen Vorgehen der Modellerstellung und Modellauswertung vermeiden lassen. Einige der üblichen Fehler sind im Folgenden aufgezählt.

- Die Modellerstellung wird ohne konkrete Zielsetzung durchgeführt, oft verbunden mit einer späteren Nutzung des Modells für unterschiedliche und heterogene Ziele.
- Ein falscher Detaillierungsgrad des Modells wird gewählt. Oft werden unnötige Details abgebildet und relevante Aspekte vergessen.
- Der Aufwand für die Datenerhebung und Validierung wird in Modellierungsprojekten unterschätzt. Oft werden diese Schritte erst am Ende der Simulationsstudie angegangen und dann aus Zeitmangel sehr kurz abgehandelt. So werden dann Mittelwerte statt Verteilungen für Parameter eingesetzt und die Resultate nicht validiert. Zum Teil wird nicht einmal eine Plausibilitätsprüfung durchgeführt.
- Die Animation, die Teil vieler moderner Simulatoren ist, wird oft überbewertet. So zeigt eine Animation zwar die Abläufe, sie ist aber keine Garantie für eine korrekte Modellierung.
- Statistische Methoden werden oftmals nicht oder nur in sehr geringem Umfang eingesetzt. Dadurch werden Eingabedaten falsch modelliert und Simulationsresultate nicht ausgewertet. In vielen Fällen werden Simulationsresultate nur bzgl. der ermittelten Mittelwerte interpretiert, ohne dass die Qualität der Mittelwertschätzer untersucht wird.
- Es wird keine Sensitivitätsanalyse durchgeführt. D.h. es wird nicht untersucht, wie sich die Ergebnisse bei Änderung einzelner Parameter ändern. Dieser Schritt ist aber immens wichtig, um das Systemverhalten zu verstehen und um die Plausibiltät der ermittelten Resultate nachzuweisen.
- Der Gültigkeitsbereich der Modelle wird oft nicht abgeschätzt. So mag ein Modell für den Istzustand korrekte Ergebnisse liefern, es ist aber nicht klar, dass bei Änderungen der Systemparameter (z.B. der Ankunftsraten, Puffergrößen etc.) das Modell das System immer noch genau genug abbildet.

Man sieht, dass Simulationen und Simulationsergebnisse mit einer gewissen Vorsicht zu interpretieren sind. Simulation ist nicht das Allheilmittel für alle Probleme der Systemanalyse, kann aber, wenn sie richtig eingesetzt wird, wichtige Ergebnisse liefern. Eine zentrale Aussage ist, dass ***Simulation nur dann eingesetzt werden sollte, wenn keine analytische Analysemethode verfügbar ist***. Dies bedeutet, dass immer zuerst untersucht werden sollte, ob das Problem mit Hilfe analytischer Ansätze lösbar ist.

Kapitel 8

Validierung von Modellen

Einer der wichtigsten Schritte jeder simulativen Analyse eines Systems ist die Validierung des genutzten Modells. Zur Erinnerung sei noch einmal erwähnt, dass Modelle erstellt werden, um Experimente am realen System durch Experimente am Modell zu ersetzen. Auf Basis der Resultate des Modells sollen Entscheidungen getroffen werden, die das System betreffen. Dies bedeutet, dass die Folgerungen aus der Modellanalyse, im hier betrachteten Fall aus den Simulationsexperimenten, weitestgehend den Folgerungen entsprechen sollen, die aus entsprechenden Objekt-Analysen des Systems gewonnen würden. In der Praxis sind die dazu notwendigen Objekt-Experimente unter Umständen gar nicht möglich, da das zu untersuchende System nicht existiert. Dies bedeutet, dass in irgendeiner Weise plausibel gemacht werden muss, dass Modell- und System-Experiment zu gleichen oder zumindest ähnlichen Folgerungen führen. Dieser Abschnitt beschäftigt sich damit, welche Anforderungen an Modellexperimente gestellt werden müssen, wie Modelle so verändert werden können, dass sie die benötigten Resultate liefern und welche mathematischen Methoden existieren, um Verhaltensunterschiede zwischen Modell und Realsystem zu bewerten.

Experimente werden unter bestimmten Bedingungen durchgeführt, d.h. die kontrollierbaren Größen C werden auf bestimmte Werte eingestellt und die unkontrollierbaren Größen U nehmen bestimmte Werte an. Wir sprechen in diesem Zusammenhang von einer Experimentsituation oder einer Situation. Wenn das Modell und das reale System in einer Situation beobachtet werden so führen sie zu gleichen Folgerungen, wenn die beobachteten Resultate identisch sind. Ist eine solche Identität der Resultate zu erwarten oder überhaupt erreichbar? Es gibt zwei Gründe, die dagegen sprechen. Zum einen sind das reale System und das Modell nicht identisch, das Modell beschreibt nur eine Abbildung des Systems unter einem vorgegebenen Analyseziel. Zum anderen zeigen Modell und System in den meisten Fällen stochastisches Verhalten. Bei mehrfacher Beobachtung des Systems oder des Modells unter gleichen Bedingungen, beim Modell aber mit variierender Saat des Zufallszahlengenerators, beobachtet man unterschiedliche Ergebnisse. Damit ist die zentrale Frage, welche Verhaltensunterschiede tolerierbar sind.

8.1 Grundlagen des Vergleichs von System und Modell

Um System und Modell zu vergleichen ist eine vernünftige Definition von *Realitätstreue* oder *Gültigkeit* (engl. validity) notwendig. Sei dazu V_R ein Verhaltensmaß für das Realsystem und V_S das Verhalten für das simulierte Modell. $D(V_R, V_S)$ ist das benötigte Maß für die auftretenden Verhaltensunterschiede. Realitätstreue soll bejaht werden, falls $D(V_R, V_S)$ unter einem problemabhängig und zielabhängig festzulegenden Grenzwert liegt. Dieser Grenzwert ist idealerweise so zu wählen, dass unvermeidlich existierende Verhaltensunterschiede, die die zu treffende Entscheidung nicht beeinflussen, toleriert werden.

Bevor wir ein Maß für Verhaltensunterschiede festlegen, sollen die Ursachen für Verhaltensunterschiede betrachtet werden. Zwischen Modell und System gibt es strukturelle Unterschiede, da es sich bei dem Einen um ein Computerprogramm und beim Anderen um einen Ausschnitt aus der

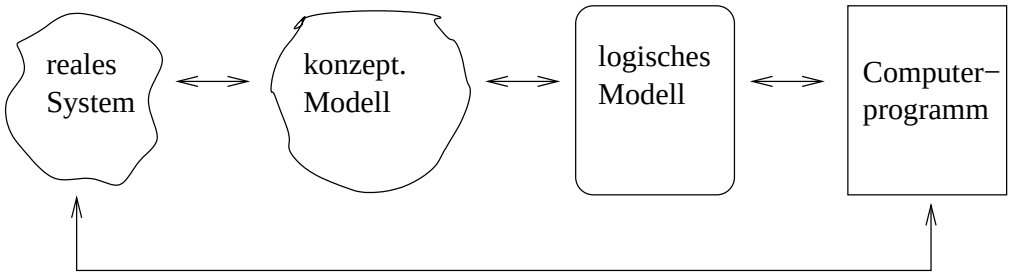


Abbildung 8.1: Transformationsschritte im Rahmen der Modellbildung.

realen Umwelt handelt. Der Prozess der Modellierung führt zu beabsichtigten Abstraktionen und Aggregierungen der Realität. Dazu kommen unbeabsichtigte Ungenauigkeiten und sogar Fehler. Eine weitere Quelle für Abweichungen sind die Parameterwerte, die oftmals durch Zufallsvariablen angeglichen, nicht aber exakt wiedergegeben werden. Schließlich sind noch die stochastischen Schwankungen des Verhaltens zu beachten, die schon bei einem System/Modell zu variierenden Beobachtungen führen.

Eigentlich soll V_R zur Entscheidungsfindung hernagezogen werden. Da V_R nicht vorliegt, nutzen wir ein Modell und damit V_S. Wenn das Modell stochastische Komponenten beinhaltet, was die Regel ist, so können wir nur $\tilde{V}_S$ ermitteln, so dass eigentlich die folgende Dreiecksungleichung gilt

$$D(V_{R,}V_S) \leq D(V_R, \tilde{V}_S) + D(V_S, \tilde{V}_S) \ .$$

Wenn wir zusätzlich noch V_R beobachten, so kommt noch $D(V_R, \tilde{V}_R)$ hinzu. So dass eigentlich

$$D(V_{R,}V_S) \leq D(\tilde{V}_R, \tilde{V}_S) + D(V_S, \tilde{V}_S) + D(V_R, \tilde{V}_R)$$

analysiert werden müsste. Die Abweichungen $D(V_R, \tilde{V}_R)$ und $D(V_S, \tilde{V}_S)$ können mit den Methoden aus Kapitel 5 behandelt werden.

Das Vorgehen bei der Modellbildung soll noch einmal rekapituliert werden. Wie in Abbildung 8.1 gezeigt, erfolgt die Modellbildung in mehreren Schritten. Ausgehend von einem schlecht strukturierten Realsystem und einer in den meisten Fällen vage definierten Problemstellung soll ein wohl definiertes Computerprogramm entstehen. Dieser Übergang erfolgt in mehreren Schritten, bei jedem dieser Schritte kann es zu Fehlern, Irrtümern oder Verzerrungen kommen, die dazu führen, dass Realsystem und Modell bzgl. der zu messenden Resultatgrößen abweichen. Wenn man sich die in Abbildung 8.1 gezeigten Schritte genauer anschaut, so werden beim Übergang vom realen System zum konzeptuellen Modell die Systemgrenzen und die relevanten Systemelemente festgelegt. Bleiben dabei wichtige Aspekte unberücksichtigt, so wird das Simulationsmodell keine Aussagen über das System erlauben. Beim Übergang vom konzeptuellen Modell zum logischen Modell werden die relevanten Zusammenhänge zwischen Systembestandteilen festgelegt und die Umwelteinflüsse auf das System definiert. Auch in diesem Schritt existiert eine Vielzahl von Möglichkeiten, relevante Zusammenhänge ungenau oder falsch darzustellen oder wegzulassen. Der nachfolgende Schritt des Übergangs vom logischen Modell zum Computerprogramm wird heute durch eine Vielzahl von Methoden und Werkzeugen unterstützt, birgt aber trotzdem eine Reihe von Fehlermöglichkeiten. So ist selbst bei einer vollständigen formalen Spezifikation der Nachweis, dass das resultierende Programm die Spezifikation implementiert alles andere als trivial und durchaus Gegenstand aktueller Forschungsarbeiten. Aus den Resultaten des Simulationsprogramms müssen anschließend Rückschlüsse auf das Systemverhalten gezogen werden und es müssen mögliche Änderungen am System hergeleitet werden. Auch an dieser Stelle können vielfältige Fehler auftreten, indem Resultate falsch interpretiert werden und zu falschen Schlussfolgerungen genutzt werden.

Die Terminologie im Kontext des Vergleichs und der Anpassung von System- und Modellverhalten ist oft uneinheitlich. Wir wollen die folgende Begriffsbildung verwenden.

- **Verifikation**: Unter Verifikation soll die Bestätigung aller Modell-Eingabegrößen und Modell-Annahmen inklusive struktureller Annahmen, der Programmverifikation und der Bestätigung

der Eingabegrößen und Parameter verstanden werden. Teilweise, wie bei der Programmverifikation, können dazu formale Methoden eingesetzt werden, die zu beweisbaren Resultaten führen. Bei der Bestätigung von Eingabegrößen und Parametern handelt es sich formal um ein Problem der Statistik, so dass keine Beweise sondern nur Wahrscheinlichkeitsaussagen möglich sind. Auch für die Nutzung stochastischer Methoden zum Nachweis der Gültigkeit von Annahmen soll der Begriff Verifikation verwendet werden, auch wenn dies in der Literatur nicht unumstritten ist.

- **Validierung**: Der Begriff Validierung soll für die Bestätigung der Modellresultate verwendet werden. Es ist nachzuweisen oder besser plausibel zu machen, dass die Resultate des Modells nicht zu stark von den Resultaten des Systems abweichen.
- **Kalibrierung**: Ziel der Kalibrierung ist die Reduktion der Verhaltensunterschiede $D(V_R, V_S)$. Falls diese Unterschiede als zu groß bewertet werden, umfasst die Kalibrierung Ansätze zur Anpassung des Modells an das System. Dies geschieht durch gezielte Änderungen am Modell.

Auch wenn davon ausgegangen wird, dass große Mühe auf eine gute Wahl der Eingabegrößen gelegt wurde, steigt zwar die Hoffnung auf ein gültiges Modell, es gibt jedoch keine Garantie für ein gültiges Modell. Ein erster Schritt ist nahe liegender Weise der Vergleich von System und Modell für eine Situation, falls dies möglich ist. Stimmen die ermittelten Ergebnisse überein, so könnte bei einer positivistischen Sichtweise davon ausgegangen werden, dass das realisierte Modell das System genügend genau abbildet. Dies bedeutet andererseits, dass man durch gezielte Änderungen das Modell so anpassen muss, dass Modellresultate und verfügbare Resultate aus dem System übereinstimmen. Ein solches Vorgehen nutzt die Kalibrierung ohne Verifikation und Validierung einzusetzen und ist potenziell sehr gefährlich. Man muss sich vor Augen führen, dass Modelle genutzt werden, um Aussagen über Systeme in solchen Situationen zu machen, in denen das System nicht analysiert werden kann oder soll. Die Situationen zu denen das Systemverhalten bekannt ist, sind eigentlich uninteressant. Identische oder ähnliche Resultate über bekannte Situationen sind natürlich kein Beweis für ähnliche Resultate in unbekannten Situationen.

Der Ansatz der Kalibrierung ohne Validierung wird leider oft in der induktiven Modellierung verwendet. So wird ein mathematisches Modell an Daten aus der Vergangenheit angepasst und damit die Zukunft vorhergesagt. Dies führt oft zu abstrusen Aussagen, insbesondere wenn über einen kurzen Zeitraum in der Vergangenheit exponentielles Wachstum beobachtet wurde. Beispiele für solche Modelle sind insbesondere aber nicht nur in der Ökonomie zu finden. Man erinnere sich dazu an manche Prognosen über die Entwicklung des Aktienmarktes zu Zeiten des Internet-Booms.

Es soll in den nachfolgenden Abschnitten ein Vorgehen beschrieben werden, welches Kalibrierung mit Validierung und Verifikation vereint und so zu belastbaren Aussagen kommt. Auf Grund der Problemstellung wird sich aber zeigen, dass die Qualität eines Modells nur sehr eingeschränkt beweisbar ist und deshalb in den meisten Fällen nur plausibel gemacht werden kann. Grundsätzlich gibt es einen Kosten-Nutzen Effekt. Man kann durch zusätzlichen Aufwand die Plausibilität eines Modell und das Vertrauen in ein Modell steigern, muss dazu aber zusätzliche Kosten in Kauf nehmen. Da dieser Prozess grundsätzlich nicht endet, muss ein Kompromiss zwischen Aufwand und Qualität der Abbildung gefunden werden. Wo dieser Kompromiss liegt hängt vom vertretbaren Aufwand und natürlich von der Zielstellung der Modellierung ab. Es stellt sich dabei primär die Frage, welche Folgen falsche Entscheidungen auf Grund der Modellresultate in der Praxis haben.

8.2 Verifikation in der Modellbildung von Simulationsmodellen

Für die Verifikation werden oft formale und automatisierbare Techniken eingesetzt. Man unterscheidet zwischen simulationsspezifischen und allgemeinen Schritten. Die allgemeinen Schritte umfassen primär Ansätze der Programmverifikation, wie sie bei jedem größeren Software-Projekt notwendig sind. Ein wichtiger Aspekt ist die Verifikation des Simulationsprogramms am logischen oder konzeptuellen Modell. Dies ist eine Frage der Programmverifikation, die wir hier nicht behandeln

wollen, da sie Inhalt anderer Vorlesungen ist. Weitere Aspekte der Verifikation des Simulationsprogramms sind

- eine strukturierte Programmierung mit dem Test/Debugging von Modulen und Subprogrammen,
- ein Code-Review durch andere Mitarbeiter,
- die Verwendung von (semi-)automatischen Spezifikationstechniken und
- ein inkrementeller Entwurf und Test.

Alle diese Punkte gehören zu einem strukturierten Vorgehen im Software Engineering und sind nicht Inhalt dieser Vorlesung.

Es gibt darüber hinaus ein Reihe simulationsspezifischer Ansätze zur Verifikation von Simulationsprogrammen. Der Simulator sollte unter unterschiedlichen aber realistischen Umgebungsparametern (Situationen) getestet werden. Mögliche Umgebungsparameter ergeben sich aus der Beobachtung der Realität und der Zielstellung der Simulationsexperimente. Ein wichtiger Aspekt ist das Erstellen und Analysieren von Traces. Die meisten Simulatoren bieten die Möglichkeit detaillierte Traces zu erzeugen. Diese können auf unterschiedliche Weise analysiert und visualisiert werden. Eine Möglichkeit der Visualisierung ist die Animation im Modell. Wenn das Modell graphisch spezifiziert wurde, so können in der Graphik die Abläufe angezeigt und vom Benutzer bewertet werden. Auf diese Weise lässt sich manchmal Fehlverhalten erkennen. Gleichzeitig stellen Traces eine Verbindung zwischen Simulation und Realität dar. Im Prinzip können Traces auch in der Realität gemessen und dann mit Traces der Simulation verglichen oder sogar als Eingabe der Simulation verwendet werden (trace-getriebene Simulation). Mit Hilfe von Beobachtungen und auch formalen Testverfahren lassen sich so simulierte und reale Abläufe vergleichen. Die Bewertung des Simulationsverhaltens wird ebenfalls einfacher, wenn man bestimmte Abläufe vorgibt oder Modelle vereinfacht, indem z.B. die Zufallszahlen durch deterministische Größen ersetzt werden.

Neben diesen Untersuchungen am Modell, reduziert die Verwendung von etablierten und zuverlässigen Simulationswerkzeugen die Fehlermöglichkeiten. Für viele Simulationswerkzeuge existieren bereits vordefinierte Modelle von Standardsystemen, die als Teilmodelle komplexer Modell genutzt werden können. Da diese Modelle oft schon vielfach verwendet und getestet wurden, sind sie weniger fehlerbehaftet als neu geschriebene Modelle. Beispiele für solche Standardmodelle findet man im Bereich der Rechnernetzsimulation, wo für viele Simulatoren bereits Simulationsmodelle für die üblichen Protokolle existieren. Als Beispiel sei auf die frei verfügbaren Simulatoren ns-2 und OMNet++ OMN verwiesen. Ähnliche Standardmodelle egibt es auch in anderen Bereichen.

Als letzter Punkt im Kontext der Verifikation des Simulationsmodells soll kurz die Verifikation von Eingabegrößen mit statistischen Testverfahren erwähnt werden. Das dazu notwendige Vorgehen wurde in Kapitel 4 detailliert vorgestellt.

Die meisten der vorgestellten Ansätze zur Verifikation führen nicht zu Beweisen. Deshalb muss der Begriff Verifikation, der manchmal auf den Beweis von Eigenschaften beschränkt wird, weiter gefasst werden, indem vom Beweis zum Nachweis übergegangen wird.

8.3 Allgemeine Überlegungen zur Validierung

Ein Simulationsmodell soll nützlich sein, d.h. mit sinnvoller Genauigkeit Aussagen über das modellierte System erlauben. Eine wichtige Überlegung ist, dass der Begriff *Validität* eines Modells nicht absolut definierbar ist. Validierung ist immer Modell-individuell. Je nach Anforderung und Zielsetzung der Modellierung ist Validität sehr unterschiedlich festzulegen. So sind an ein Modell, mit dem sicherheitskritische Aspekte eines technischen Systems untersucht werden deutlich höhere Ansprüche zu stellen als an ein Modell, das die Konfiguration eines Rechnernetzes unter Kapazitätsplanungsgesichtspunkten bewerten soll. Da das Modell für die Systemanalyse verwendet wird oder nicht verwendbar ist, muss eine binäre Entscheidung im Validierungsprozess getroffen

werden. Trotzdem ist Validierung graduell, Modelle sind mehr oder weniger valide und es ist zu entscheiden, ab welcher Grenze das Modell akzeptiert wird. Da Validität nicht beweisbar ist, ist sie oft das Ergebnis eines Verhandlungsprozesses verschiedener Partner. So sind in umfangreicheren Simulationsprojekten neben den eigentlichen Modellierern auch die Entwerfer und Betreiber des modellierten Systems und das Management involviert. Zwischen allen beteiligten Personen muss Einvernehmen hergestellt werden, dass das Modell hinreichend genaue Aussagen über das System erlaubt. Dies setzt voraus, dass Validierung als projektbegleitender Prozess betrieben wird.

Man unterscheidet die folgenden Ansätze der Validierung:

- Die *funktionsbezogene Validierung* untersucht die Plausibiltät des Systems. Es werden dabei oft qualitative Aussagen verglichen und untersucht, wie sich das Modell in Extremsituationen und für einfach nachvollziehbare Abläufe verhält.
- Die *theoriebezogene Validierung* vergleicht die Übereinstimmung des Simulationsmodells mit einem analytischen Modell. In der Regel wird das analytische Modell das System nur für eingeschränkte Situationen nachbilden. Es erlaubt aber für diese Situationen einen einfachen Resultatvergleich und kann auch eingesetzt werden, wenn das System nicht existiert und deshalb auch keine Daten über das Systemverhalten erhoben werden können.
- Die *ergebnisbezogene Validierung* vergleicht die Ergebnisse des Simulationsmodells mit denen des realen Systems und stellt fest, welche Abweichungen tolerierbar sind.

Im Folgenden wird primär die ergebnisbezogene Kalibrierung und Validierung untersucht. Die Kalibrierung und Validierung setzt dann voraus, dass Daten über das Realsystem vorhanden sind oder erhoben werden können, damit V_R bestimmt werden kann. Man unterscheidet den Fall, bei dem Daten im Realsystem erhoben werden können und den Fall, bei dem das Realsystem nicht existiert und Daten anderweitig beschafft werden müssen.

Falls das Realsystem verfügbar ist und Daten erhoben werden können oder auf erhobene Daten zurückgegriffen werden kann, so treten immer noch eine Reihe von Problemen auf. Ähnlich wie bei der Modellierung von Eingabedaten können die gemessenen Daten gestört sein, es können Daten in falscher Form vorliegen etc. Da die Probleme ähnlich zur Modellierung von Eingangsdaten sind, kann auf die in Kapitel 4 vorgestellten Methoden zurückgegriffen werden. Falls das zu modellierende System nicht existiert, so müssen Daten aus ähnlichen Systemen, Schätzungen oder sogar aus anderen Modellen verwendet werden. Die dabei auftretenden Probleme unterscheiden sich nicht wesentlich von den Problemen bei der Datenerhebung im realen System. Die verfügbaren Daten sind aber oftmals ungenauer und damit weniger repräsentativ für das zu modellierende System. Damit sind natürlich auch die Möglichkeiten der Validierung und Kalibrierung eingeschränkt.

8.4 Schritte zur Kalibrierung von Modellen

Ziel der Kalibrierung ist die Identifikation und Beseitigung von Verhaltensunterschieden zwischen realem System und Modell. Zur Beseitigung von Verhaltensunterschieden muss das Modell angepasst d.h. geändert werden. Es gibt zwei Klassen möglicher Änderungen.

- *Strukturänderungen* modifizieren die Struktur des Modells, indem der Programm-Code geändert wird und neue Aspekte hinzugenommen oder vorhandene Abläufe modifiziert werden.
- *Parameteränderungen* modifizieren Werte der einzelnen Modellparameter. Dazu sind keine Änderungen an der Struktur des Simulationsprogramms notwendig.

Parameteränderungen sind natürlich in den meisten Fällen viel einfacher auszuführen als Strukturänderungen. Bei der Modellierung komplexer Systeme erfordert die Kalibrierung aber in fast allen Fällen auch Strukturänderungen. Bevor Änderungen am Modell durchgeführt werden können, müssen erst die Ursachen für Verhaltensunterschiede abgeleitet werden. Strukturelle Ungenauigkeiten und Fehler sind primär aus dem Vergleich qualitativen Verhaltens etwa in Form von Traces oder Animationen ableitbar. Parameter Ungenauigkeiten können durch den Vergleich

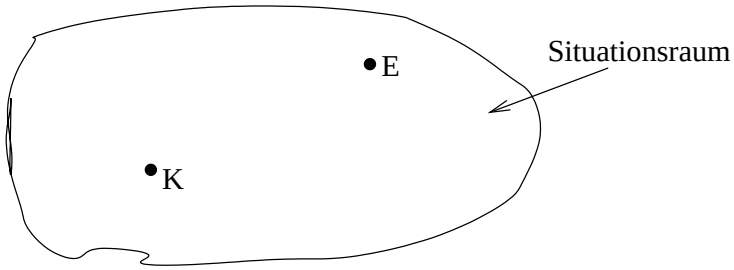


Abbildung 8.2: Situation nach der Kalibrierung.

quantitativer Größen erkannt werden. In beiden Fällen ist es hilfreich den Bereich, in dem Probleme auftreten, möglichst frühzeitig einzugrenzen. Dazu sind detaillierte Informationen über das Verhalten notwendig. So gibt die Beobachtung der Abweichung der Antwortzeit eines komplexen Systems/Modells, zum Beispiel eines Rechnernetzes, wenig Aufschluss über die Gründe der Abweichung. Wenn aber festgestellt wird, dass die Verweilzeiten an bestimmten Komponenten, zum Beispiel an einzelnen Routern, stark abweichen, so sind notwendige Änderungen viel einfacher zu identifizieren. In diesen Fällen muss die Komponente, an der die Abweichungen auftreten, genauer analysiert und modelliert werden.

Bei stochastischen Modellen, wie wir sie fast ausschließlich betrachten, ergeben sich zwei spezifische Testprobleme. Bei Strukturänderungen entstehen mehrere Modelle $S_1, \ldots, S_K$ und es ist auf Basis der zugehörigen Werte $D(V_R, V_{S_k})$ zu entscheiden, ob ein Modell besser als ein anderes ist. Da die zugehörigen D's Zufallsvariablen sind, ist festzustellen, ob die Unterschiede zwischen den Konfigurationen/Modellen das normale Schwankungsmaß überschreiten, d.h. ob die Unterschiede signifikant sind.

Bei Parameteränderungen steht das Tuning von Parametern im Mittelpunkt. Wenn $S(p)$ das Modell mit Parametervektor p ist, so wird $p_{opt} = \arg\min_p D(V_R, V_{S(p)})$ gesucht. Dies beschreibt ein typisches stochastisches Optimierungsproblem.

Methodisch gesehen ist Kalibrieren eines stochastischen Simulators damit identisch zum Experimentieren mit einem stochastischen Simulator. Für ein gegebenes Realsystem R mit Verhalten V_R wird mit Hilfe von Experimenten ein Modell S mit Verhalten V_S gesucht, so dass $D(V_R, V_S)$ kleiner als eine vorgegebene Schranke ist. Um zu S zu gelangen wird ein vorhandenes Modell durch Ausprobieren struktureller Änderungen und das Tuning von Parametern verbessert. Methoden des Experimentierens, die wir hier nicht detailliert behandeln werden, sind hilfreich für das Finden einer guten Lösung. Für die detaillierte Methodik sei auf die Literatur verwiesen Law and Kelton [2000, Kap. 12], Montgomery [2005].

Angenommen wir hätten unser primäres Ziel erreicht und ein Modell E gefunden, so dass $D(V_R, V_E) < D_{ertr}$ für die vordefinierte obere Schranke der akzeptablen Verhaltensunterschiede D_{ertr} gilt. Ist damit auch das Ziel erreicht, dass Folgerungen aus Objektexperimenten identisch sind zu Folgerungen aus Modell-Experimenten? Dies ist zumindest fraglich, wie aus folgenden Überlegungen deutlich wird. Wir haben über Änderungen am Modell Simulator-Verhalten und Modell-Verhalten angepasst, und das jeweils nur für einen Zustand der Umwelt und einen Zustand des Systems, also für eine Situation. Ziel der Simulation sind aber Aussagen über andere Situationen, für die das Systemverhalten unbekannt ist. Das prinzipielle Problem wird in Abbildung 8.2 verdeutlicht. Der Situationenraum wird in der Abbildung zweidimensional dargestellt und für einen Punkt K in diesem Situationenraum wurde die Nähe des Modellverhaltens nachgewiesen. Als Ersatz für das System soll das Modell aber an einem Punkt E eingesetzt werden. Ist der Unterschied zwischen Modell- und Systemverhalten an der Stelle E genauso oder zumindest ähnlich wie an der Stelle K? Dies ist zumindest fraglich. Anzunehmen ist eher, dass es einen Bereich G gibt in dem $D(V_r, V_S) < D_{ertr}$ gilt und einen Bereich $\neg G$ in dem $D(V_R, V_S) \geq D_{ertr}$ gilt.

Die Frage reduziert sich damit zu der Frage, ob E innerhalb oder außerhalb von G liegt. Diese Frage ist prinzipiell nur beantwortbar, wenn vom Verhalten für gewisse Situationen auf Verhalten in anderen Situationen geschlossen werden kann. Dies würde gleichzeitig ermöglichen Aussagen

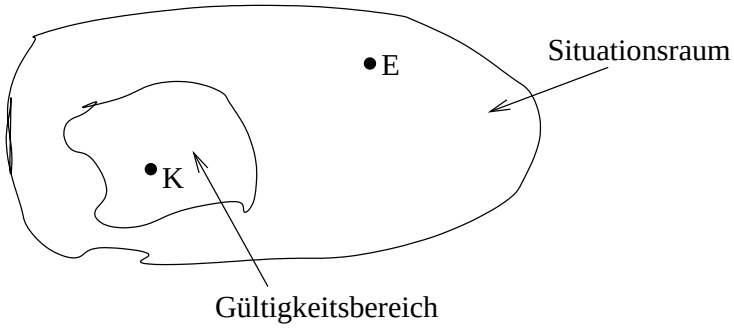


Abbildung 8.3: Kalibriersituation mit Gültigkeitsbereich.

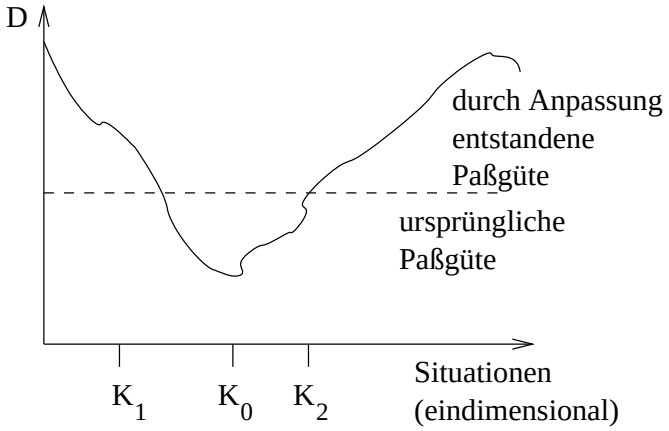


Abbildung 8.4: Das Problem der Überanpassung durch zusätzliche Kalibrierung.

über Systemverhalten ohne Experimente am Realsystem zu erlangen. Bei praktisch allen realen Systemen ist dies aus prinzipiellen Gründen unmöglich. Statt eines Beweises für die Gültigkeit eines Modells kann nur eine gewisse Zuversicht in die Gültigkeit erreicht werden. Dies ist natürlich ein deutlich schwächeres Resultat, aber letztendlich ein Problem jedes vorausschauenden Modellierens.

Trotzdem kann aus der Gültigkeit für eine Situation K durchaus etwas mehr an Information gewonnen werden. So ist anzunehmen, dass Situationen nahe bei K eher zu G gehören als Situationen, die sich deutlich unterscheiden, da wir oft stetiges Verhalten in den Parametern beobachten. Dieses Phänomen ist in Abbildung 8.3 dargestellt. Es muss aber in der Praxis nicht so sein, dass der Gültigkeitsbereich sich als ein zusammenhängender Bereich darstellt.

Das bisherige Vorgehen ging davon aus, dass ein Modell S gefunden wird, welches Verhaltensabweichungen liefert, die unterhalb eines tolerierbaren Wertes liegen. Man könnte nun weiter den begonnenen Weg voranschreiten und S durch zusätzliche Modifikationen so verbessern, dass $D(V_R, V_S)$ weiter reduziert wird, um so zu einem besseren Modell zu gelangen. In der Praxis zeigt sich allerdings oft, dass dieses Vorgehen nicht zu einer Vergrößerung von G führt. Im Gegenteil, man beobachtet bei diesem Vorgehen oft das Problem der Überanpassung. Verhaltensunterschiede werden für die Kalibriersituation reduziert, steigen aber gleichzeitig für andere Situationen. Gerade bei sehr detaillierten Modellen mit vielen Parametern lässt sich im Prinzip jedes Verhalten anpassen, dies sagt aber nichts über die Validität des Modells über einen größeren Bereich aus. Die Gefahr der Überanpassung kann dadurch reduziert werden, dass nicht nur an einer Stelle, sondern an mehreren Stellen kalibriert wird. Dies setzt natürlich voraus, dass entsprechende Daten vorhanden sind.

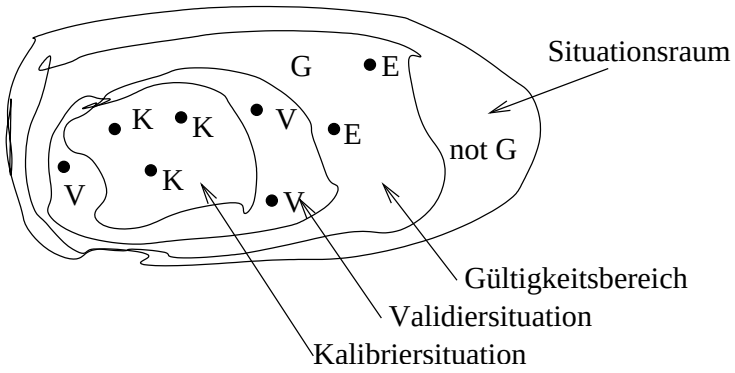


Abbildung 8.5: Kombinierte Kalibrierung und Validierung.

Dennoch bleibt zu zeigen, dass keine Überanpassung erfolgte. Das Vertrauen in das Modell kann nur erhöht werden, wenn die Gültigkeit für Situationen nachgewiesen wird, die nicht bereits zur Kalibrierung verwendet wurden. Diese Überprüfung bezeichnet man als Validierung. Es wird überprüft, ob die Verhaltensunterschiede für Validiersituationen so ähnlich sind, wie für Kalibriersituationen. Im Gegensatz zu einer Kalibriersituation wird das Ergebnis einer Validiersituation nur bewertet, nicht aber zur Modellanpassung genutzt. Abbildung 8.5 skizziert das Vorgehen. Die verfügbaren Daten über das Systemverhalten werden unterteilt. Ein Teil wird zur Kalibrierung und der Rest zur Validierung verwendet.

Trotzdem bleibt ungewiss, ob eine unbekannte Experimentsituation E innerhalb oder außerhalb des Gültigkeitsbereichs G liegt. Dies kann nur retrospektivisch nachgewiesen werden, ist dann aber für Vorhersagen uninteressant. Trotzdem erhöht die Validierung, auch die retrospektivische Validierung, das Vertrauen in ein Modell. Es bleibt aber immer fraglich, ob Experimentsituationen E, die weit weg von Kalibrier- und Valdiersituationen liegen durch das Modell genügend genau beschrieben werden. Gerade diese Situationen sind aber beim vorausschauenden Modellieren von besonderem Interesse.

8.5 Messung von Verhaltensunterschieden

Die Messung von Verhaltensunterschieden ist wesentlich bei der Kalibrierung zur Reduktion der Verhaltensunterschiede und bei der Validierung zur Bewertung der Verhaltensunterschiede. Kalibrierung und Validierung sollen das Vertrauen in das Modell erhöhen. Deshalb ist ein mathematisch fundierter Vergleich der Verhalten notwendig. Es stellen sich die folgenden Fragen:

1. Mit dem Verhalten welchen Systems soll das Verhalten des Simulators verglichen werden?
2. Welche Verhaltens-Aspekte sollen für den Vergleich gewählt werden?
3. Welche Vergleichs-Methoden und -Techniken sollen eingesetzt werden?

Frage 1 zielt darauf, das erwünschte Simulatorverhalten festzulegen. Ideal ist die Beobachtung des realen Objekt-Systems. Dies setzt voraus, dass das System in den gewünschten Situationen und bzgl. der gewünschten Größen beobachtbar ist oder Beobachtungen bereits existieren. Falls keine vorhandenen Aufzeichnungen genutzt werden können, muss das System in den gewünschten Situationen betrieben und das gewünschte Verhalten gemessen werden. In diesem Kontext ist die trace-getriebene Simulation zur Verifikation des Modells aber auch zur Validierung des Verhaltens hilfreich. Ein Trace wird im realen System gemessen und der Simulator mit den Trace-Daten betrieben. Das beobachtete Simulatorverhalten kann dann qualitativ bzgl. der auftretenden Abläufe und quantitativ bzgl. der ermittelten Resultate mit dem Realsystem verglichen werden. Die Einbringung von Trace-Daten in den Simulationsabläufen ist in vielen Simulatoren prinzipiell möglich, kann aber sehr aufwändig sein, da die verfügbaren Daten normalerweise erst interpretiert

und gefiltert und anschließend an geeigneter Stelle in die Simulation eingebunden werden müssen. Dies erfordert meistens zusätzlichen Programmieraufwand im Simulator und passende Filter zur Filterung relevanter Information aus einem Trace.

Falls Daten über das Real-System nicht vorhanden sind und auch nicht erhoben werden können, kann versucht werden auf Daten aus ähnlichen Systemen zurückzugreifen. Dieses Vorgehen erfordert eine gewisse Sorgfalt, da sicherzustellen ist, dass das ähnliche System sich auch ähnlich bzgl. relevanter Größen verhält.

Wenn keine Systemdaten erhältlich sind, so kann auf Daten aus anderen Modellen zurückgegriffen werden. Es wurde bereits erwähnt, dass für Marginalsituationen durchaus analytische Modelle zum Vergleich benutzt werden können. Ansonsten können Daten aus anderen Simulationsmodellen dieses oder ähnlicher Systeme verwendet werden. Es ist aber offensichtlich, dass die Gefahr von unzureichenden oder falschen Daten wächst und damit V_R nur sehr ungenau bestimmbar ist.

Alternative Methoden der Gewinnung von Daten über reale Systeme basieren auf der Nutzung von Expertenwissen. Es geht im Wesentlichen darum, ob ein Experte das Verhalten des Simulators vom Verhalten des (potenziellen) Realsystems unterscheiden kann. Dieser Ansatz ist in der Praxis aber schwer umsetzbar und wird kaum eingesetzt.

Frage 2 beschäftigt sich mit der Festsetzung des zu vergleichenden Verhaltens. Da es das Ziel ist, das System auf Basis mehrerer ausgewählter Leistungskriterien zu bewerten, sollten diese auch zum Vergleich herangezogen werden. Bei mehreren Maßen treten dabei oft Zielkonflikte auf, d.h. die Verbesserung des Simulatorverhaltens bzgl. eines Maßes führt zur Verschlechterung bzgl. eines anderen Maßes. Das gewählte Vorgehen muss deshalb ähnlich zum Vorgehen bei einer multikriteriellen Optimierung sein. Entweder werden die Verhaltensunterschiede in den einzelnen Maßen auf einen skalaren Wert abgebildet oder das Ergebnis von $D(V_R, V_S)$ ist ein Vektor mit einem Element pro Leistungsmaß. Auch D_{ertr} wird dann als Vektor definiert und $D(V_R, V_S) < D_{ertr}$ muss komponentenweise gelten. Wir gehen im folgenden allerdings davon aus, dass Verhalten sich durch eine skalare Größe beschreiben lässt. Die verwendeten Verfahren lassen sich zwar prinzipiell auf mehrdimensionale Probleme erweitern, dies ist aber meist alles andere als trivial, da im mehrdimensionalen Fall die beobachteten Größen meistens stark korreliert sind.

Die Beantwortung der 3. Frage hängt davon ab, womit das Simulatorverhalten verglichen wird. Beim Vergleich mit einem analytischen Modell wird eine Stichprobe mit einer analytischen Verteilung verglichen und es ist zu entscheiden, ob die Stichprobe hinreichend ähnlich oder unähnlich ist. Dieses Problem trat aber bereits bei der Modellierung von Eingabedaten mit theoretischen Verteilungen in Kapitel 4 auf. Wie dort beschrieben können der χ^2- oder K.-S.-Test zum Vergleich verwendet werden. Beim Vergleich des Simulatorverhaltens mit dem Verhalten des Realsystems oder eines anderen Simulators sind zwei Stichproben zu vergleichen. Die dazu notwendigen Methoden werden nun vorgestellt.

8.5.1 Vergleich von Stichproben

Das Verhalten sei durch Zufallsvariablen V_R und V_S beschrieben. Die Zufallsvariable kann z.B. die Verweilzeit eines Kunden an einem Bankschalter oder die mittlere Pufferbelegung eines Routers in einem Rechnernetz beschreiben. Es liegen zwei Stichproben $v_R = (v_{R_1}, \ldots, , v_{R_n})$ und $v_S = (v_{S_1}, \ldots, v_{S_m})$ vor, die Realisierungen von V_R und V_S beschreiben. Zu entscheiden ist, ob beide Stichproben hinreichend ähnlich sind. Man unterscheidet Bewertungsverfahren in

- subjektive Verfahren, die auf dem Vergleich graphischer Repräsentationen der Stichproben basieren und durch einen Menschen bewertet werden
- objektive Methoden, die auf statistischen Test- oder Bewertungsverfahren basieren und ein Resultat aus den Stichproben ableiten.

Beim subjektiven Vorgehen, welches auch als Inspektionsansatz bezeichnet wird, werden die Werte der Stichproben visualisiert. Wie wir bereits kennen gelernt haben, gibt es unterschiedliche Visualisierungsmöglichkeiten. So kann die Dichtefunktion durch ein Histogramm dargestellt, die Verteilungsfunktion kann dargestellt, Q-Q- oder P-P-Plots können benutzt oder Box-Plots (siehe

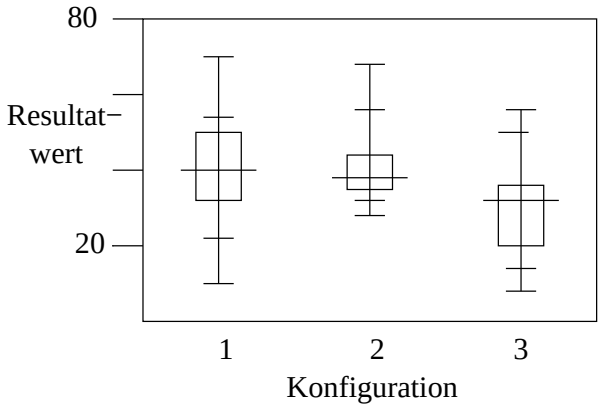


Abbildung 8.6: Vergleich von Box-Plots für drei Konfigurationen.

Abbildung 8.6) verglichen werden. Einige graphische Darstellungen haben freie Parameter, deren Festlegung zu unterschiedlichen Repräsentationen führen kann. Des Weiteren muss entschieden werden, ob die ganze Stichprobe repräsentiert wird oder ob Ausreißer erst eliminiert werden. Die abschließende Entscheidung, ob die Stichproben ähnlich oder unähnlich sind hat immer eine subjektive Komponente und sollte, wenn möglich, von mehreren Personen unabhängig getroffen werden.

Trotz der subjektiven Einflüsse ist die Inspektionsmethode ein wichtiges Entscheidungskriterium, da die gesamte Verteilung betrachtet werden kann. Sie sollte allerdings mit objektiven Verfahren ergänzt oder komplettiert werden.

8.5.2 Vergleich von Konfidenzintervallen

Es gibt zwei unterschiedliche objektive Ansätze, den Vergleich von Konfidenzintervallen und statistische Testverfahren. Der Vergleich von Konfidenzintervallen wird hier detailliert beschrieben, während auf Testverfahren nur kurz am Ende des Abschnitts eingegangen wird. Zum Vergleich der Konfidenzintervalle setzen wir voraus, dass die Beobachtungen in beiden Stichproben unabhängig identisch verteilt sind. Auf Basis von Konfidenzintervallen sollen die Mittelwerte beider Stichproben verglichen werden. Es wird ein Konfidenzintervall für die Differenz der Stichprobenmittelwerte bestimmt. Enthält dieses die 0 nicht, so können die Mittelwerte als unterschiedlich angesehen werden. Gleichzeitig gibt die Lage und Breite des Konfidenzintervalls Auskunft über die Unterschiede und es kann entschieden werden, ob diese tolerabel sind. Wir betrachten zwei Verfahren

- ungepaarte Stichproben (Welch-Verfahren Law and Kelton [2000, Kap. 10.2.2]),
- gepaarte Stichproben (paired t-Konfidenzintervalle Law and Kelton [2000, Kap. 10.2.1]).

Für das Welch-Verfahren müssen die Werte zwischen den beiden Stichproben unabhängig sein. Es werden aber keine Annahmen bzgl. der Varianz gemacht. Die Schätzer für Mittelwert und Varianz der beiden Stichproben werden wie üblich definiert.

$$\tilde{\mu}_R = \frac{1}{n}\sum_{i=1}^{n} V_{R_i} \quad \tilde{S}_R^2 = \frac{1}{n-1}\left(\sum_{i=1}^{n}\left(V_{R_i} - \tilde{\mu}_R\right)^2\right)$$
$$\tilde{\mu}_S = \frac{1}{m}\sum_{i=1}^{m} V_{S_i} \quad \tilde{S}_S^2 = \frac{1}{m-1}\left(\sum_{i=1}^{m}\left(V_{S_i} - \tilde{\mu}_S\right)^2\right)$$

Die konkreten Schätzwerte $\hat{\mu}_s$, $\hat{\mu}_R$, $\hat{S}_R^2$ und $\hat{S}_S^2$ resultieren aus den obigen Formeln, wenn die Zufallsvariablen durch die konkreten Werte ersetzt werden. Uns interessiert die Differenz $\mu_{RS} = \mu_R - \mu_S$. Wenn wir ein Konfidenzintervalle für μ_{RS} ermittelt haben, so kann entschieden werden,

ob die Stichproben als ähnlich oder unähnlich klassifiziert werden. Falls $m = n$ ist, so kann die Stichprobe der Differenz $v_{RS} = (v_{RS_1}, \ldots, v_{RS_n})$ mit $v_{RS_i} = v_{R_i} - v_{S_i}$ gebildet werden. Aus dieser Stichprobe kann der Mittelwert $\hat{\mu}_{RS}$ und die Varianz $\hat{S}^2_{RS}$ geschätzt werden, so dass

$$\hat{\mu}_{RS} \pm t_{n-1,1-\alpha/2} \cdot \sqrt{\frac{\hat{S}^2_{RS}}{n}}$$

ein Konfidenzintervall für den Mittelwert ist. Dieses Konfidenzintervall ist formal aber nur korrekt, wenn

- beide Stichproben gleiche Beobachtungszahlen beinhalten $n = m$ und
- die Varianzen von V_R und V_S identisch sind.

Gleiche Beobachtungszahlen sind zwar immer zu erreichen, indem Werte aus einer Stichprobe weggelassen werden. Oft ist es aber so, dass für das Realsystem Stichproben fester Länge vorliegen und die Erhebung zusätzlicher Daten aufwändig ist, während in der Simulation sehr einfach zusätzliche Daten generiert werden können. Die Beschränkung auf Stichproben gleicher Länge führt dann zu einem Informationsverlust. Die Forderung nach identischer Varianz ist natürlich nicht realistisch. Es zeigt sich allerdings, dass bei gleicher Stichprobenlänge und nicht zu stark differierenden Varianzen das berechnete Konfidenzintervall eine recht gute Abdeckung hat.

Für den allgemeinen Fall ungleicher Stichprobenumfänge und ungleicher Varianzen ist die Varianz des Schätzers für μ_{RS} nicht exakt bestimmbar. Die folgende Approximation von Welch Law and Kelton [2000] hat sich allerdings in vielen Experimenten als sehr gut erwiesen.

$$\hat{\mu}_{RS} = \hat{\mu}_R - \hat{\mu}_S \ , \qquad \hat{S}^2_{RS} = \frac{\hat{S}^2_R}{n} + \frac{\hat{S}^2_S}{m} \ , \qquad \hat{f} = \frac{\left(\hat{S}^2_{RS}\right)^2}{\frac{1}{n-1} \cdot \left(\frac{\hat{S}^2_R}{n}\right)^2 + \frac{1}{m-1} \cdot \left(\frac{\hat{S}^2_S}{m}\right)^2}$$

Damit wird das folgende Konfidenzintervall berechnet.

$$\hat{\mu}_{RS} \pm t_{\hat{f},1-\alpha/2} \cdot \hat{S}_{RS}$$

Da $\hat{f}$ in der Regel keine ganze Zahl sein wird, kann der Wert gerundet werden oder der Wert wird interpoliert als

$$t_{\hat{f},1-\alpha/2} = \left(\left\lceil \hat{f} \right\rceil - \hat{f}\right) \cdot t_{\lfloor \hat{f} \rfloor,1-\alpha/2} + \left(\hat{f} - \left\lfloor \hat{f} \right\rfloor\right) \cdot t_{\lceil \hat{f} \rceil,1-\alpha/2}$$

Das folgende Beispiel stammt aus Law and Kelton [2000, Kap. 10.2].

i	v_{R_i}	v_{S_i}	$v_{R_i} - v_{S_i}$
1	126.97	118.21	8.76
2	124.31	120.22	4.09
3	126.68	122.45	4.23
4	122.66	122.68	0.02
5	127.23	119.40	7.83

Die Werte für V_R sind in der Stichprobe größer als die von V_S. Die Frage ist, ob dieser Unterschied signifikant ist. Die folgenden Schätzwerte resultieren aus den Daten. $\hat{\mu}_R = 125.57$, $\hat{\mu}_S = 120.59$, $\hat{S}^2_R = 4.00$ und $\hat{S}^2_S = 3.76$, sowie $\hat{\mu}_{RS} = 4.98$, $\hat{S}^2_{RS} = 1.55$ und $\hat{f} = 7.99$. Dies liefert ein Konfidenzintervall von $[2.67, 7.29]$ zum Signifikanzniveau $\alpha = 0.1$ und von $[0.81, 9.15]$ zum Signifikanzniveau $\alpha = 0.01$. In beiden Fällen ist 0 nicht im Konfidenzintervall enthalten, so dass der Erwartungswert der beobachteten Größe im Realsystem als größer angenommen werden kann. Inwieweit der beobachtete Unterschied ausreicht, um das Modell als nicht ausreichend valide anzusehen hängt von den Anforderungen an das Ergebnis ab. Weiterhin muss man sich vor Augen führen, dass eine Schätzung auf Basis von 5 Werten eher unzuverlässig ist, da die Annahmen des zentralen Grenzwertsatzes bei so kleinen Beobachtungszahlen nicht gelten müssen.

Der zweite betrachtete Ansatz zur Konfidenzintervallberechnung basiert auf so genannten gepaarten Stichproben. Als zusätzliche Voraussetzung wird $n = m$ benötigt. Es werden aber keine Annahmen bzgl. identischer Varianz von V_R und V_S oder Unabhängigkeit zwischen den Stichproben vorausgesetzt. Aus diesem Grund eignet sich der Ansatz auch für die trace-getriebene Simulation, wobei v_R Beobachtungen aus einem Trace beschreibt und v_S Resultate einer Simulation beinhaltet, bei der einige Parameter des Traces mit verwendet wurden, so dass die Werte der Stichproben positiv korreliert sind. $\hat{\mu}_{RS}$ wird wie in der vorherigen Methoden berechnet und für die Varianz wird folgende Formel benutzt.

$$\hat{S}^2_{RS} = \frac{\sum_{i=1}^{n} \left((v_{R_i} - v_{S_i}) - \hat{\mu}_{RS} \right)^2}{n \cdot (n-1)}$$

Damit kann das folgende Konfidenzintervall berechnet werden.

$$\hat{\mu}_{RS} \pm t_{n-1,1-\alpha/2} \cdot \hat{S}_{RS}$$

Aus dem vorherigen Beispiel resultiert $\hat{S}_{RS} = 1.56$ und Konfidenzintervalle $[1.66, 8.30]$ zum Signifikanzniveau $\alpha = 0.1$ und $[-2.20, 12.16]$ zum Signifikanzniveau $\alpha = 0.01$. Im letzteren Fall ist die 0 im Konfidenzintervall enthalten. Was bedeutet nun diese Aussage? Nach Definition der Konfidenzintervalle, beschreibt ein Konfidenzintervall den Bereich, in dem der wahre Wert mit Wahrscheinlichkeit $1-\alpha$ liegt. Wenn wir zu 99% sicher sein wollen, so können wir nicht ausschließen, dass die Erwartungswerte gleich sind, da die 0 im Konfidenzintervall enthalten ist. Reichen 90% Signifikanzwahrscheinlichkeit aus, so würden wir die Erwartungswerte als ungleich klassifizieren.

Wenn beide vorgestellten Ansätze anwendbar sind, so ist nicht klar, welche Methode schmalere Konfidenzintervalle liefert. Es sollte deshalb immer nach Datenlage über die Anwendung entschieden werden.

Neben dem Vergleich von Konfidenzintervallen für Mittelwertschätzer besteht noch die Möglichkeit Testverfahren anzuwenden. Im Gegensatz zu Konfidenzintervallen quantifizieren Tests den Abstand der Stichproben aber nicht. Da Realsystem und Simulator fast nie vollständig identische Verhalten haben, neigen Testverfahren dazu bei größeren Stichprobenumfängen Gleichheit der Stichproben abzulehnen. Insofern ist der Vergleich der Konfidenzintervalle in den meisten Fällen vorzuziehen.

Teil II

Optimierung

Die bisherigen Kapitel behandelten die Modellbildung und Analyse dynamischer ereignisdiskreter Systeme. Der dabei auftretende Aufwand für die Modellbildung und anschließende simulative Analyse war recht hoch. Zentraler Aspekt der Modellierung war die Einbeziehung von Stochastik zur Abbildung komplexer Vorgänge in der Realität. Im nun folgenden Teil der Vorlesung steht nicht mehr die Analyse sondern die Optimierung im Mittelpunkt des Interesses. Ziel ist es, für eine Funktion $f(x)$ Eingaben x zu finden, so dass die Lösung in einem zu definierenden Sinne optimal ist. Diese allgemeine Sichtweise umfasst zahlreiche Spezialfälle, die jeweils unterschiedliche Herangehensweisen erforderlich machen. Das Feld der Optimierung ist sehr breit und kann und soll nicht vollständig oder auch nur annähernd umfassend hier behandelt werden.

Insofern bleibt die Frage, welche Schwerpunkte gesetzt werden sollen. Aus dem bisher Behandelten würde folgen, dass $f(x)$ ein Simulationsmodell ist und wir die Optimierung von Simulationsmodellen untersuchen. Dies ist allerdings ein eher komplexes Problem, für das es auch nur relativ wenige etablierte Ansätze und entsprechend viele offene Fragen gibt. Viele Optimierungsansätze für Simulationsmodelle basieren auf Heuristiken, die wir in der Vorlesung nicht behandeln werden. Es werden deshalb einfachere Funktionen $f(x)$ untersucht, deren Auswertung keine Probleme bereitet, deren Optimierung aber durchaus sehr anspruchsvoll sein kann. Des Weiteren werden wir uns darauf beschränken, skalare Funktionen $f(x)$ zu optimieren, so dass Minimum oder Maximum eindeutig definierbar sind. Auch unter diesen Festlegungen ergeben sich noch zahlreiche Varianten.

Eine zentrale Festlegung ist, aus welcher Grundmenge x zu wählen ist. In der Informatik sind unter algorithmischen Gesichtspunkten gerade sogenannte kombinatorische Probleme, bei denen x aus einer diskreten Menge von Alternativen zu wählen ist, von großem Interesse. Zahlreiche bekannte Beispielprobleme, wie das Handlungsreisendenproblem, das Rucksackproblem oder die Reihenfolgeplanung von Aufträgen fallen in diese Klasse. Es zeigt sich dabei, dass kombinatorische Probleme oft deutlich schwerer zu optimieren sind als kontinuierliche Probleme, bei denen zumindest lokale Optimalitätsbedingungen in vielen Fällen bekannt sind.

Wir beginnen den Teil über Optimierung mit einer allgemeinen Übersicht und einer Klassifizierung von Optimierungsproblemen. Daran anschließend behandeln wir die einfachste Form der Optimierung unter Nebenbedingungen, nämlich die lineare Optimierung. Die dabei verwandten Methoden finden in vielen Optimierungsmethoden für komplexere Problem als Basisfunktionen Anwendung. Kapitel 11 beschäftigt sich mit kombinatorischen Problemen mit linearen Zielfunktionen. Schließlich werden schrittweise Optimierungsprobleme, wie sie in der dynamischen Programmierung auftreten, vorgestellt und zugehörige Optimierungsmethoden eingeführt. Nicht behandelt wird das weite Feld der kontinuierlichen Optimierung, das im Wesentlichen in der Mathematik beheimatet ist.

Über Optimierung gibt es zahlreiche Lehrbücher, die sehr unterschiedliche Herangehensweisen beschreiben. Das Spektrum reicht von sehr anwendungsnahen Ausrichtungen bis zu sehr theoretischen Betrachtungen. Die folgenden Kapitel basieren auf anwendungsnäheren Arbeiten. Optimierung ist ein zentraler Aspekt des Operations Research als Querschnittsdisziplin zwischen Mathematik, Informatik und insbesondere betriebswirtschaftlicher Anwendung. Mathematisch orientierte Bücher des Operations Research bieten deshalb eine gute Grundlage. Als Basis für die meisten vorgestellten Verfahren eignen sich Neumann and Morlock [1993] oder Hillier and Lieberman [1995], eine kompaktere aber auch weniger umfassende Darstellung liefert Domschke and Drexl [1993]. Mehr mathematisch orientiert ist Jarre and Stoer [2004]. Eine implementierungsnahe Beschreibung verschiedener Algorithmen findet man in Press et al. [2002].

Kapitel 9

Einführung, Klassifizierung und Grundlagen

Es gibt verschiedene Möglichkeiten Optimierungsverfahren zu klassifizieren. In diesem Kapitel sollen einige dieser Möglichkeiten und die daraus resultierenden Optimierungsprobleme beschrieben werden. Dazu wird im ersten Abschnitt eine allgemeine Formulierung für Optimierungsprobleme vorgestellt. In Abschnitt 9.2 werden einige grundlegende Ideen von Optimierungsalgorithmen vorgestellt und auf dieser Basis Algorithmen klassifiziert. Zum Abschluss wird ein kurzer Überblick über den Inhalt und die Ziele der weiteren Kapitel gegeben.

9.1 Formalisierung von Optimierungsproblemen

Bei der simulativen Analyse von Modellen haben wir deren Verhalten über die Zeit untersucht. Auch bei der Optimierung geht man davon aus, dass in Abhängigkeit von Eingabeparametern eine Ausgabe erzeugt wird (siehe auch Abb. 1.15). Dies schließt natürlich nicht aus, dass auch zeitliches Verhalten analysiert wird. Die Resultate über einen Zeitraum oder zu einem Zeitpunkt müssen dann nur auf einen oder mehrere Resultatwerte abgebildet werden. Dies ist das übliche Vorgehen bei der Optimierung. Es gibt allerdings auch Problemstellungen, bei denen zeitliche Aspekte eine Rolle spielen. Zu den zugehörigen Optimierungsproblemen werden wir später kommen und an dieser Stelle erst einmal den Standardfall einer zeitinvarianten Zielfunktion untersuchen.

Formal betrachten wir eine Funktion $f(x) : W \to \mathbb{R}$. Die Erweiterung der multikriteriellen Optimierung, bei der eine Menge von Funktionen $f_i(x) : W \to \mathbb{R}$ $(i = 1, \ldots, r)$ optimiert werden soll, werden wir nur kurz ansprechen. W ist der zulässige Bereich, aus dem die Parameter gewählt werden können. Wir nehmen an, dass die Eingaben n-dimensional sind, wir also n Parameter haben. Die naheliegendste Annahme wäre somit $W \subseteq \mathbb{R}^n$. Wenn $W = \mathbb{R}^n$ ist, spricht man von einem unrestringierten oder unbeschränkten kontinuierlichen Optimierungsproblem, ansonsten von einem restringierten oder beschränkten Problem. Der zulässige Bereich restringierter Probleme wird üblicherweise durch Gleichungen und Ungleichungen definiert. Es gilt dann

$$x \in W \;\Leftrightarrow\; g_i(x) = 0 \ (i = 1, \ldots, p) \text{ und } h_j(x) \leq 0 \ (j = 1, \ldots, q).$$

Ziel ist dann die Bestimmung von

$$\min_{x \in W}(f(x)).$$

Da $\max_{x \in W}(f(x)) = \min_{x \in W}(-f(x))$, reicht es aus, Minimierungsprobleme zu untersuchen.

Minimierung über den gesamten Bereich W bezeichnet man auch als globale Minimierung (Optimierung). Also x^* ist ein globales Minimum, falls $f(x^*) \leq f(y)$ (globales striktes Minimum falls $f(x^*) < f(y)$) für alle $y \in W$. Idealerweise ist die Ermittlung des globalen Optimums natürlich das Ziel einer Optimierung. Bei der praktischen Anwendung zeigt sich allerdings oft, dass globale

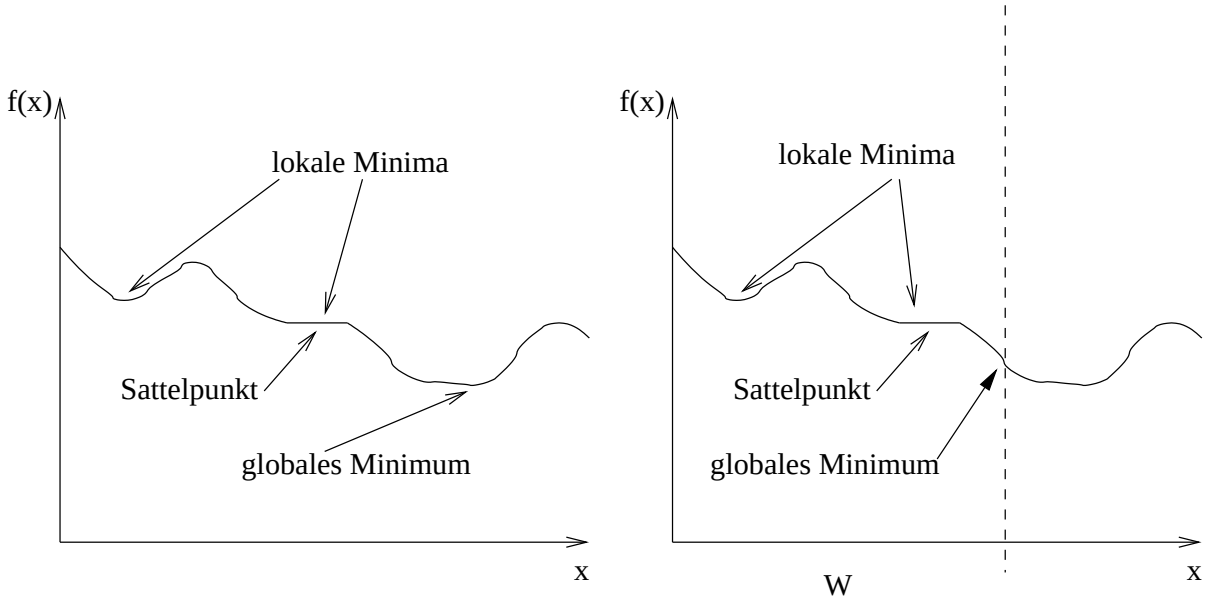


Abbildung 9.1: Lokale und globale Minima einer Funktion.

Optima für allgemeine Funktionen oft nicht bestimmbar sind. Oftmals muss man sich mit sogenannten lokalen Optima begnügen. Um diese zu definieren, sei $N(x)$ die Nachbarschaft von x. Die Nachbarschaft kann man definieren als

$$N(x) = \{y | y \in W \wedge \|y - x\| \leq \epsilon\} \text{ für } \epsilon > 0 \text{ und } x \in W.$$

Ein Punkt x^* ist ein lokales Minimum, falls $f(x^*) \leq f(y)$ (striktes lokales Minimum falls $f(x^*) < f(y)$) für alle $y \in N(x^*)$. Abbildung 9.1 zeigt den Verlauf einer Funktion mit skalarem Parameter und markiert dort lokale und globale Minima. Es wird dabei auch schon das Problem beschränkter Funktionen deutlich, nämlich die Lage von Extrempunkten auf dem Rand des zulässigen Bereichs, ohne dass der zugehörige Punkt ein Extrempunkt der unrestringierten Zielfunktion ist.

Im Wesentlichen haben wir damit die Klasse der Probleme schon definiert, die wir im Folgenden betrachten wollen. Unterschiedliche Unterklassen ergeben sich nun für unterschiedliche Arten der Zielfunktion und der Nebenbedingungen. Beide Aspekte beeinflussen die verwendbaren Lösungsalgorithmen und auch die überhaupt erreichbaren Resultate. Typische Kombinationen sind:

- Lineare Zielfunktion und lineare Nebenbedingungen führen zur linearen Optimierung oder Programmierung die in Kapitel 10 behandelt wird.
- Quadratische Zielfunktionen mit linearen Nebenbedingungen führen zur quadratischen Programmierung.
- Wenn Zielfunktion und Nebenbedingungen konvexe Funktionen sind, so spricht man von konvexer Programmierung
- Wenn neben der Konvexität die Funktionen auch nur als Summe eindimensionaler Funktionen darstellbar sind (d.h. $f(x) = \sum_{i=1}^{n} f_i(x_i)$), so liegt ein Problem der separablen Programmierung vor.
- In den übrigen Fällen spricht man in der Regel von nichtkonvexen oder auch nichtlinearen Problemen.

Wir beschränken uns im Wesentlichen darauf, Methoden zur Optimierung der ersten Klasse von Problemen zu behandeln.

Die obige Klassifikation umfasst bei weitem noch nicht alle kontinuierlichen Probleme. Bisher sind wir implizit davon ausgegangen, dass $f(x)$ deterministisch ist, ansonsten wäre das Minimierungsproblem ohne weitere Annahmen nicht definiert. Wir haben aber bei der Modellierung gesehen, dass viele realen Vorgänge sich nur mittels stochastischer Modelle adäquat abbilden lassen. Dies bedeutet, dass $f(x)$ eine Zufallsvariable ist oder anders aufgeschrieben, dass wir eigentlich $f(x, u)$ analysieren, wobei u die Saat des Zufallszahlengenerators ist. Dies ist genau das Szenario, dass bei der Optimierung von Simulationsmodellen auftritt. Üblicherweise wird dann $\min_{x \in W}(E(f(x)))$ gesucht. Man kann aber auch andere Ziele, wie zum Beispiel die Minimierung der Varianz der Lösung oder die Minimierung der Wahrscheinlichkeit, dass die Lösung einen bestimmten Wert überschreitet, betrachten. Im stochastischen Fall ist eine Optimierung deutlich komplexer, oft hilft man sich damit, dass der Erwartungswert einfach durch den Mittelwertschätzer ersetzt wird und dieser als konstanter Wert interpretiert wird. Dann lassen sich Optimierungsalgorithmen für deterministische Funktionen einsetzen, die Resultate können aber unbefriedigend sein.

Wenn man die Festlegung auf zeitinvariante Funktionen fallen lässt, so kommt man zu neuen Klassen von Optimierungsproblemen. In der Praxis tritt oft der Fall auf, dass sich die Zielfunktion mit der Zeit ändert, da sich die Einflussfaktoren ändern. Das Optimum muss immer wieder neu berechnet werden. Im Prinzip kann man in regelmäßigen Abständen ein neues Optimierungsproblem lösen und so die optimale Lösung nachführen. In der Praxis ist man in einem solchen Fall natürlich bestrebt, aus einer bekannten optimalen Lösung eine neue zu bestimmen. Dies erfordert spezielle Algorithmen, wie zum Beispiel stochastische Suchverfahren, die wir allerdings nicht im Detail betrachten wollen.

Eine spezielle Form der Zeitabhängigkeit tritt bei den Problemen der dynamischen Optimierung oder Programmierung auf. In diesem Fall werden stufenweise Entscheidungen betrachtet. Dies bedeutet, dass Probleme untersucht werden, bei denen jeweils am Ende einer Periode eine Entscheidung getroffen werden muss. Der Zeithorizont kann dabei endlich oder auch unendlich sein. In Abhängigkeit von den getroffenen Entscheidungen entwickelt sich das System. Ein typisches Beispiel sind Lagerhaltungsprobleme. Es liegt eine Liste von Auslieferungsaufträgen vor, die jeweils eine Menge von Gütern und die Auslieferungsperiode umfasst. Güter müssen vor der Auslieferung eingekauft werden. Die Preise der Güter hängen von der Bestellperiode ab, zusätzliche Kosten treten durch die Lagerung auf. Darüber hinaus kann es noch weitere Nebenbedingungen geben, wie zum Beispiel ein beschränktes Lager oder eine maximale Lagerdauer. Ziel ist es, eine kostenoptimale Einkaufsstrategie zu finden. Dieses Problem, welches für eine ganze Klasse von Problemen steht, kann man mit Methoden der dynamischen Optimierung lösen, die in Kapitel 12 vorgestellt werden.

Wenn die zulässige Menge W keine Teilmenge des $\mathbb{R}^n$ ist, so müssen andere Optimierungsalgorithmen verwendet werden. Oft werden die resultierenden Probleme prinzipiell schwieriger lösbar. Falls $W \subseteq \mathbb{N}^n$, spricht man von ganzzahliger Optimierung. Ist der zulässige Bereich endlich, so spricht man auch von einem kombinatorischen Optimierungsproblem. Ein typisches Beispiel ist ein zulässiger Bereich, der durch alle Permutationen der Zahlen 1 bis n definiert ist. Probleme, die mit dem Finden eines optimalen Weges zu tun haben, betrachten solche zulässigen Bereiche. Auch wenn der zulässige Bereich endlich ist und damit theoretisch ein Optimum dadurch gefunden werden kann, dass alle Punkte ausgewertet werden, gibt es viele Probleme, die zur Komplexitätsklasse NP gehören und damit für größere n nicht effizient lösbar sind. Es folgt damit, ähnlich dem kontinuierlichen Fall, dass ein globales Optimum nicht zwangsläufig in akzeptabler Zeit gefunden wird. Für viele Probleme gibt es aber Algorithmen, die eine gute, teilweise sogar beweisbar gute, Lösung in kurzer Zeit finden.

Neben rein ganzzahligen oder rein reellwertigen Problemen existieren gemischte Probleme, bei denen einige Parameter ganzzahlig, andere reellwertig sind. Neben diesen beiden Parametertypen existieren noch Probleme mit allgemeinen diskreten Parametern aus endlichen Mengen, auf denen keine Ordnung definiert ist. Ein Beispiel wäre die Scheduling-Strategie an einem Bediener, bei der aus der Menge {FCFS, PS, Random} gewählt werden kann. Im Prinzip lassen sich diese Werte natürlich auf die Zahlen $1, 2, 3$ abbilden, problematisch ist dann nur, dass die Reihenfolge keine Aussage über das Verhalten von $f()$ erlaubt. Alle Optimierungsalgorithmen basieren aber letzt-

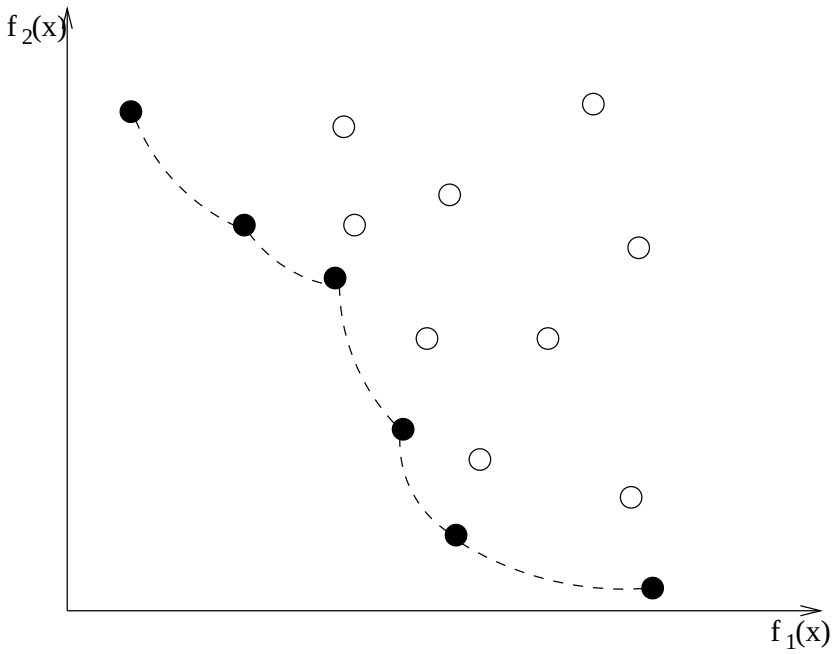


Abbildung 9.2: Pareto-Front eines multikriteriellen Optimierungsproblems.

endlich darauf, dass sich die Zielfunktion bei der Vergrößerung eines Parameters in eine bestimmte Richtung ändert und daraus eine Suchrichtung abgeleitet werden kann. Wenn eine solche Suchrichtung nicht existiert, so bleibt nur eine systematische erschöpfende Suche oder eine Zufallssuche.

Auch wenn in Michalewicz and Fogel [2004] ausgeführt wird, dass auf einem Digitalrechner eigentlich jedes Problem ein kombinatorisches Problem ist, da nur endlich viele Parameterwerte betrachtet werden können, ergeben sich doch deutliche Unterschiede zwischen Methoden der kontinuierlichen Optimierung und der ganzzahligen oder kombinatorischen Optimierung. Nur die Methoden aus Kapitel 11 lassen sich mit kleineren Modifikationen für ganzzahlige Probleme einsetzen. Bzgl. allgemeiner Methoden für diese Problemklassen sei auf diverse andere Vorlesungen und die Literatur verwiesen.

Um die Situation bei mehreren Zielfunktionen $f_i(x)$ $(i = 1, \ldots, r)$ zu verdeutlichen, kann man sich die in Abbildung 9.2 dargestellte Funktion für $r = 2$ ansehen. Die Punkte im Graphen geben jeweils die Funktionswerte $f_1(x)$ und $f_2(x)$ an. Die zugehörigen x Werte werden nicht dargestellt. Ziel ist es, $f_1(x)$ und $f_2(x)$ zu minimieren. Man kann zwei Gruppen von Punkten erkennen, die in der Graphik durch schwarze und weiße Punkte dargestellt werden. Um den Unterschied zu definieren, betrachten wir den allgemeinen Fall und zwei Parametervektoren x_1 und x_2 sowie die zugehörigen Funktionswerte $y_1 = (f_1(x_1), \ldots, f_r(x_1))$ und $y_2 = (f_1(x_1), \ldots, f_r(x_2))$. y_1 dominiert y_2 (Notation $y_1 \preceq y_2$), falls $y_1(i) \leq y_2(i)$ für $i = 1, \ldots, r$ und $j \in \{1, \ldots, r\}$ existiert, so dass $y_1(j) < y_2(j)$. Wenn $\mathcal{R} = \{y_1, \ldots, y_k\}$ eine Menge von Resultatwerten ist, so definieren wir eine Menge $\mathcal{P}(\mathcal{R}) = \{y_i | y_i \in \mathcal{R} \wedge \nexists z \in \mathcal{R} : y_z \preceq y_i\}$. $\mathcal{P}(\mathcal{R})$ bezeichnet man als die Pareto-Front. Sie enthält alle Lösungen, die nicht von einer anderen bisher gefundenen Lösung dominiert werden. In der Abbildung gehören die schwarzen Punkte zur Pareto-Front. Die weißen Punkte werden von mindestens einem schwarzen Punkt dominiert, d.h. man kennt eine Lösung, die in allen Komponenten besser ist. Die gestrichelten Linien in der Abbildung beschreiben natürlich nur eine Approximation des realen Verlaufs, da nur die Punkte bekannt sind.

Bei einer mehrdimensionalen Zielfunktion sind im Prinzip alle Punkte der Pareto-Front optimal, da eine Verbesserung eines Zielfunktionswertes immer mit einer Verschlechterung mindestens eines anderen Wertes einhergeht. Damit ist die Auswahl eines Optimums ohne weitere Informationen nicht möglich. Ziel einer Optimierung kann damit die Approximation der Pareto-Front sein, so dass ein menschlicher Nutzer aus dieser Menge eine Lösung auswählen kann. Die zur Approximati-

on der Pareto-Front notwendigen Methoden sollen hier nicht betrachtet werden. Eine Alternative ist die gewichtete Summation der Lösungvektoren, um zu einem skalaren Wert zu gelangen. Also $\sum_{i=1}^{k} \alpha_i y_i$ ($\alpha_i \in \mathbb{R}_+$) wird gebildet und der kleinste Wert als Optimum verwendet. Dieses Vorgehen ist äquivalent zur Optimierung der eindimensionalen Funktion $f(x) = \sum_{i=1}^{k} \alpha_i f_i(x)$ und kann damit mit einem Optimierungsverfahren für eindimensionale Funktionen behandelt werden.

9.2 Lösungsansätze und Algorithmen für kontinuierliche Probleme

Da wir den Bereich der kontinuierlichen Optimierung, von der linearen Optimierung abgesehen, nicht weiter behandeln werden, soll an dieser Stelle ein sehr kurzer Überblick über in diesem Bereich genutzte Optimierungsverfahren gegeben werden. Zur Optimierung von $f(x)$ werden sehr unterschiedliche Ansätze verwendet. Diese Ansätze hängen von der Struktur der Zielfunktion und den Nebenbedingungen ab. Da der unbeschränkte Fall in der Regel einfacher zu handhaben ist, betrachten wir ihn zuerst. Gesucht wird $\min_{x \in \mathbb{R}^n}(f(x))$. Die meisten Optimierungsmethoden sind Suchmethoden, die ausgehend von einer gegebenen Lösung, einen besseren Funktionswert suchen, so lange bis das Optimum oder ein (lokales) Optimum gefunden wurde oder die Methode anderweitig abgebrochen wird. Optimierungsmethoden unterscheiden sich primär dadurch, welche Informationen über $f(x)$ verwendet werden.

- In einer *white box*-Methode wird die Struktur von $f(x)$ zur Optimierung genutzt. In der Regel ist $f(x)$ analytisch auswertbar und die (partiellen) Ableitungen bzw. der Gradient

$$\text{grad}(f(x)) = g(x) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)^T$$

  sind bestimmbar und werden benutzt, um die Suchrichtung festzulegen. Unter Umständen sind auch höhere Ableitungen bestimmbar. Weiterhin kann es vorkommen, dass auf Grund der Struktur Aussagen über Anzahl und Lage möglicher Optima gemacht werden können und damit auch ein globales Optimum gefunden werden kann.

- Dem gegenüber stehen *black box*-Methoden, die keine Information über die Zielfunktion verwenden, sondern diese nur an einigen Punkten auswerten und in Abhängigkeit der dabei erzielten Resultate neue Auswertungen durchführen. Diese Methoden können damit im Prinzip mit beliebigen Funktionen verwendet werden, sind aber in der Regel ineffizienter als Methoden, die spezielle Funktionseigenschaften nutzen. Die Optimierung von komplexen Modellen, bei denen $f(x)$ nicht in geschlossener Form vorliegt (z.B. Simulationsmodelle), erfordert in der Regel die Verwendung von black box-Methoden.

Die meisten white box-Methoden nutzen die Gradienteninformation zur Definition einer Suchrichtung, entlang der sich der Funktionswert verkleinert. Wenn ein Punkt erreicht wurde, an dem keine Verbesserung mehr möglich ist, d.h. der Gradient gleich 0 ist, so ist ein Optimum erreicht. Dies ist aber in der Regel ein lokales Optimum, nur für spezielle Funktionsklassen entspricht dies einem globalen Optimum oder ist feststellbar, ob es sich um ein globales Optimum handelt. Für Funktionen mit vielen lokalen Optima findet eine solche Optimierungsmethode ausgehend von einem Startpunkt x^0 ein nahe gelegenes lokales Optimum. Dieses kann beliebig weit vom globalen Optimum entfernt sein.

Für die globale Optimierung werden oftmals Heuristiken eingesetzt. Ein einfaches Vorgehen besteht darin, ein lokales Optimierungsverfahren ausgehend von unterschiedlichen Startpunkten mehrfach zu starten und jeweils lokale Optima zu suchen. Das Minimum der gefundenen Lösungen wird dann als Lösung des Optimierungsproblems genutzt. Die Startpunkte können bei diesem Ansatz entweder zufällig generiert werden oder gitterförmig im Suchraum verteilt werden. Klarerweise kann mit dem beschriebenen Vorgehen nicht garantiert werden, dass das globale Optimum gefunden wird, auch wenn die verwendeten lokalen Optimierungsverfahren konvergieren, d.h. immer ein lokales Optimum "in der Nähe" gefunden wird.

Im Gegensatz zur white box-Optimierung nutzen black box-Verfahren keine Informationen über die zu optimierende Funktion. Es werden lediglich Funktionsauswertungen benötigt, um damit den Optimierungsalgorithmus zu steuern. Es gibt unterschiedliche Möglichkeiten, diese Verfahren zu klassifizieren. Wir definieren die folgenden drei Klassen.

- Deterministische Suchverfahren, die ausgehend von einem oder einer Menge von Startpunkten Lösungen berechnen und aus diesen neue Punkte bestimmen, an denen die Funktion ausgewertet wird. Dieses Vorgehen wird iterativ solange wiederholt, bis eine Abbruchbedingung erfüllt ist. Es gibt zahlreiche Methoden, die man in diese Klasse von Optimierungsalgorithmen einordnen kann, eine recht umfassende Übersicht findet man in Kolda et al. [2003].

- Stochastische Verfahren und Metaheuristiken nutzen im Gegensatz zu deterministischen Verfahren Zufallszahlen zur Festlegung neuer Suchpunkte. Klassische Verfahren bestimmen ausgehend von einem Startpunkt einen neuen Suchpunkt, indem Komponenten des bisherigen Punktes zufallsgesteuert verändert werden und entscheiden dann deterministisch oder stochastisch ob der neue Punkt als Startpunkt akzeptiert wird oder nicht. Populationsbasierte Verfahren arbeiten mit einer ganzen Population von Suchpunkten. Neue Suchpunkte entstehen durch zufällige Modifikationen vorhandener Punkte oder durch deren Kombination. Der Evolution nachempfunden werden Generationen definiert, wobei jeweils gute Punkte aus der vorherigen Generation übernommen werden oder als Eltern für Individuen in der neuen Generation genutzt werden. In Michalewicz and Fogel [2004], Weicker [2002] findet man weitere Informationen zu stochastischen Verfahren.

- Die letzte Klasse von Methoden sind metamodellbasierte Ansätze. Die Idee dieser Verfahren ist es die Funktion $f(x)$ an einigen Punkten $x_1, \ldots, x_k$ auszuwerten und auf Basis der Auswertungen ein neues Modell g anzupassen, so dass $f(x_i) \approx g(x_i)$ für $i = 1, \ldots, k$. g bezeichnet man als Metamodell. Es sollte natürlich einfacher als das ursprüngliche Modell zu handhaben sein. Typische Metamodelle sind lineare Regressionsmodelle und Polynome niedrigen Grades und Korrelationsmodelle. Mit Hilfe des Metamodells wird anschließend eine Suchrichtung ermittelt, und entlang dieser Suchrichtung werden weitere Auswertungen vorgenommen. Je nachdem, ob das Metamodell global (d.h. $f(x)$ im gesamten Definitionsbereich approximiert) oder lokal (d.h. $f(x)$ nur im Bereich der untersuchten Punkte approximiert) ist, werden neue Punkte hinzugenommen oder ein neues Metamodell bestimmt. Metamodellbasierte Verfahren werden in dieser Vorlesung nicht behandelt, auch wenn sie sehr oft zur Optimierung von Simulationsmodellen eingesetzt werden. Für Details über diese Methoden sei auf die Literatur verwiesen (Myers and Montgomery [2002], Santner et al. [2003]).

Neben diesen Verfahren kann in machen Fällen eine Approximation der Ableitung auf Basis des Differenzenquotienten berechnet werden. Man nutzt dann

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x + \delta e_i) - f(x)}{\delta}$$

wobei e_i ein n-dimensionaler Vektor ist, der an der Stelle i eine 1 und sonst überall eine 0 enthält. $\delta > 0$ ist eine Konstante. Die Approximation der partiellen Ableitung kann anschließend in Verfahren genutzt werden, die auf Basis der Ableitung Funktionen optimieren. Es ist allerdings bekannt, dass die Approximation in vielen praktischen Anwendungen zu schlechten Ergebnissen führt, da Funktionen nicht exakt auswertbar sind und damit für kleine δ schon das Vorzeichen der Ableitung falsch sein kann und für große δ die Approximation zu ungenau wird.

Die bisherigen Ansätze gehen im Prinzip alle von deterministischen Funktionen aus, d.h. der Funktionswert kann in einem Punkt x exakt berechnet werden. Wir wissen aber bereits, dass dies oftmals nicht der Fall ist und Resultate aus stochastischen Modellen stammen und damit nicht exakt, sondern nur auf Basis von Intervallschätzungen bestimmbar sind. Der am weitesten verbreitete Ansatz zur Behandlung stochastischer Funktionen besteht darin, die Funktion mehrfach

auszuwerten und den so ermittelten Mittelwertschätzer als deterministischen Wert in einem Optimierungsverfahren zu nutzen. Darüber hinaus gibt es spezifische Methoden für spezielle Probleme aber keine allgemeine Theorie, wie im deterministischen Fall. Gut etabliert ist die Theorie der sogenannten Markovschen Entscheidungsprozesse, die wir im Rahmen der dynamischen Programmierung in Kapitel 12 kurz untersuchen werden.

Bisher wurden Nebenbedingungen nicht in die Betrachtung mit einbezogen, wir sind also von einer Minimierung einer Funktion $f(x)$ über dem $\mathbb{R}^n$ oder $\mathbb{R}^n_+$ ausgegangen. Durch die Einführung von Nebenbedingungen wird das Problem grundsätzlich schwieriger, wie man aus dem Beispiel in Abbildung 9.1 sehen kann. Das Minimum der Funktion auf der rechten Seite liegt am Rand des zulässigen Bereichs und ist kein lokales Optimum der Zielfunktion ohne Nebenbedingungen. Diese Eigenschaft erschwert natürlich die Suche nach einem Optimum. Es ist deshalb naheliegend, dass ein regulärer zulässiger Bereich einfacher zu handhaben ist, als ein zerklüfteter und irregulärer Bereich. Damit sind Optimierungsprobleme mit linearen Nebenbedingungen in der Regel deutlich einfacher zu handhaben, als Optimierungsprobleme mit nichtlinearen Nebenbedingungen.

Es gibt unterschiedliche Ansätze, mit Nebenbedingungen umzugehen. Für spezielle Problemstellungen, wie etwa die lineare Optimierung, können Lösungen direkt berechnet werden. In den verschiedenen Suchalgorithmen werden Lösungen, die die Nebenbedingungen nicht erfüllen, oftmals direkt verworfen. Ein allgemeiner Ansatz besteht darin, die Nebenbedingungen in die Zielfunktion einzubeziehen und dadurch den Zielfunktionswert von Lösungen, die Nebenbedingungen verletzen, zu verschlechtern. Es gibt unterschiedliche Ansätze, eine solche Modifikation der Zielfunktion zu realisieren.

9.3 Übersicht und Ziele

Aus der bisherigen Zusammenfassung sollte deutlich geworden sein, dass wir den Bereich der Optimierung nur in Ansätzen behandeln können. Die Auswahl der behandelten Themen ist damit auch von subjektiven Kriterien geprägt. Gerade in der Optimierung gibt es auf der einen Seite mathematisch fundierte Verfahren, deren Verhalten detailliert analysiert werden kann und eine Menge von Heuristiken, die zum Teil sehr erfolgreich in der Praxis eingesetzt werden, sich aber formaleren Analysen schon aus prinzipiellen Gründen widersetzen. Wir werden uns im Wesentlichen auf relativ einfache Probleme beschränken, für die Optimerungsalgorithmen zur Verfügung stehen, die die Problemstruktur zur Optimierung nutzen. Zum Thema Heuristiken und wie man Heuristiken fundiert nutzen kann, empfiehlt sich ein Blick in Michalewicz and Fogel [2004].

In den nachfolgenden Kapiteln sollen die folgenden Ziele erreicht werden:

- Optimierungsprobleme sollen klassifiziert werden können.
- Lineare Optimierung und der Simplexalgorithmus werden eingeführt.
- Ansätze zur Optimierung kombinatorischer und ganzzahliger Probleme werden vorgestellt.
- Die Reihenfolgeplanung wird als spezielles kombinatorisches Optimierungsproblem etwas detaillierter analysiert.
- Die Klasse der mittels dynamischer Programmierung lösbaren Probleme soll eingeführt werden.
- Lösungsmethoden für dynamische Programmierungsprobleme werden für den deterministischen und stochastischen Fall vorgestellt.

Kapitel 10

Lineare Optimierung

Die einfachste Form der Optimierung mit Nebenbedingungen tritt auf, wenn sowohl Zielfunktion als auch Nebenbedingungen lineare Funktionen sind. Man spricht dann von der linearen Optimierung oder linearen Programmierung. Auch wenn viele reale Probleme sich nicht exakt durch lineare Funktionen modellieren lassen, lassen sie sich doch relativ gut durch solche Funktionen approximieren, so dass das Minimum des linearen Modells eine gute Approximation des Minimums des realen Problems ist. Historisch erlangte die lineare Programmierung ihre Bedeutung durch die Entwicklung des Simplexalgorithmus Mitte der 40er Jahre des vorherigen Jahrhunderts. Inzwischen gehört die lineare Programmierung zu den mathematischen Methoden, die in der Praxis sehr breit eingesetzt werden. So haben zum Beispiel 85% der 500 größten amerikanischen Unternehmen angegeben, dass sie lineare Programmierung in der Unternehmensplanung nutzen.

Bei der linearen Programmierung soll das Minimum der Zielfunktion

$$f(x) = \sum_{j=1}^{n} c_j \cdot x_j$$

unter den Nebenbedingungen (udN)

$$\sum_{j=1}^{m} a_{ij} \cdot x_j \leq b_i \text{ und } x_j \geq 0 \ (i = 1, \ldots, n;\ j = 1, \ldots, m)$$

gefunden werden. Dies kann man etwas kompakter in Vektor-/Matrixschreibweise formulieren.

$$\min \left(c^T x\right) \text{ udN } Ax \leq b \text{ und } x \geq 0 \tag{10.1}$$

Diese Darstellung wird als Normalform bezeichnet. Andere Formen, bei denen zum Beispiel für die Variablen untere und obere Schranken angegeben werden, können in die Normalform transformiert werden, wie wir später sehen werden. Ziel ist es, eine Lösung im $\mathbb{R}^n$ zu finden, die (10.1) erfüllt.

Die lineare Programmierung wird exemplarisch dazu dienen, eine Optimierungsmethode detailliert herzuleiten und zu beweisen. Für die Methoden in den nachfolgenden Kapiteln werden wir diesen Detaillierungsgrad nicht beibehalten können. Im ersten Abschnitt wird ein einfaches Beispiel vorgestellt und mittels einer grafischen Methode gelöst. Daran anschließend werden die formalen Grundlagen einer Lösungsmethode erläutert, indem wir die geometrische Struktur der zulässigen Menge und die Lage der Minima herleiten. Auf dieser Basis kann das Prinzip des Simplexalgorithmus und der eigentliche Algorithmus in den Abschnitten 10.3 und 10.4 leicht entwickelt werden. Allgemeinere Problemformulierungen, bei denen das Problem nicht in Normalform vorliegt, werden im allgemeinen Simplexverfahren behandelt, welches anschließend eingeführt wird. Typische Anwendungsbeispiele, das Prinzip der Dualität, bei dem Nebenbedingungen und Zielfunktion quasi vertauscht werden, und die Sensitivitätsanalyse, bei der die Auswirkungen von Parameteränderungen analysiert werden, sind Inhalt der weiteren Abschnitte. Den Abschluss des Kapitels bildet eine kurze Übersicht über weitere in der Vorlesung nicht behandelte Aspekte der linearen Optimierung.

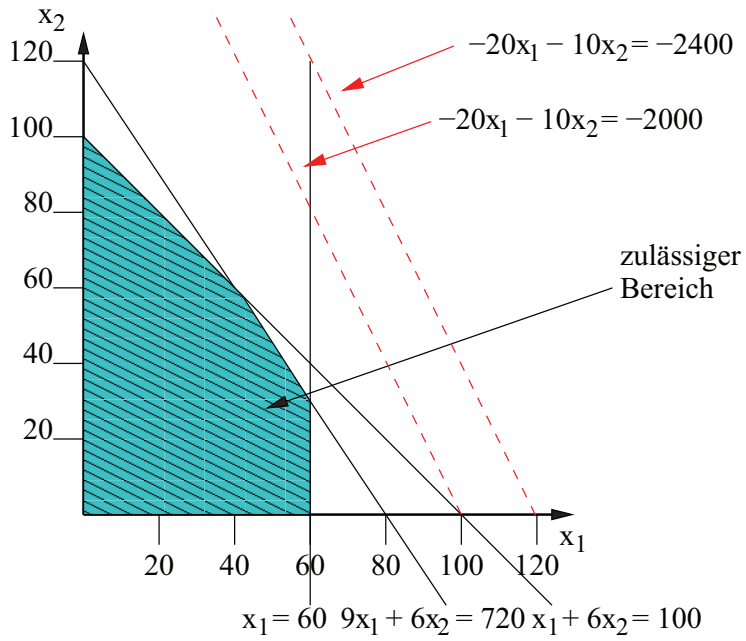


Abbildung 10.1: Zulässiger Bereich und Isogewinn-Geraden des Beispielproblems.

10.1 Beispiele und Lösungsprinzip

Das nun folgende Beispiel wurde sinngemäß aus Domschke and Drexl [1993] entnommen und dient dazu, die grundsätzliche Problematik linearer Programmierungsprobleme und die Idee des Simplexalgorithmus zu erläutern. Als Beispiel betrachten wir die Optimierung des Anbaus von Blumen und Gemüse, die ein Hobbygärtner vornimmt. Es stehen dem Hobbygärtner $100m^2$ Fläche zur Verfügung, von denen $60m^2$ zum Anbau von Blumen geeignet sind, Gemüse kann auf der gesamten Fläche angebaut werden. Die Kosten der Setzlinge betragen $9Euro/m^2$ für Blumen und $6Euro/m^2$ für Gemüse. Dem Gärtner stehen insgesamt 720 Euro Kapital zur Verfügung. An Erlösen kann er später $20Euro/m^2$ für die Blumen und $10Euro/m^2$ für das Gemüse erwarten. Ziel ist es, die Anbaufläche so aufzuteilen, dass der Gewinn am Ende maximiert wird. Sei dazu x_1 die Anbaufläche für Blumen und x_2 die Anbaufläche für Gemüse, jeweils in m^2. Damit lautet die Zielfunktion

$$\min(-20x_1 - 10x_2)$$

unter den Nebenbedingungen

$$\begin{aligned} &x_1 + x_2 \leq 100 \\ &x_1 \leq 60 \\ &9x_1 + 6x_2 \leq 720 \\ &x_1 \geq 0 \text{ und } x_2 \geq 0 \end{aligned}$$

Die Nebenbedingungen definieren eine Menge W, die alle Punkte enthält, die alle Nebenbedingungen einschließlich der Nichtnegativitätsbedingungen erfüllen. W bezeichnet man auch als zulässigen Bereich.

Da es sich um ein zweidimensionales Problem handelt, lassen sich Zielfunktion und Nebenbedingungen gut visualisieren. Wenn die Nebenbedingungen erfüllt sind und damit die Ungleichungen zu Gleichungen werden, definieren sie jeweils Geraden, die den zulässigen Bereich begrenzen. Die Eckpunkte des zulässigen Bereichs werden durch den Schnittpunkt von normalerweise 2 Geraden definiert. Abbildung 10.1 zeigt den zulässigen Bereich des Problems, der durch die $n + m = 5$ Nebenbedingungen definiert ist. Da die Zielfunktion linear ist, liegen die Wertepaare (x_1, x_2), die zu konstanten Werten der Zielfunktion führen, auf einer Geraden. Diese Gerade bezeichnet man auch als Isogewinn-Gerade. Sie ist in der Abbildung als gestrichelte Linie dargestellt.

Ein einfacher graphischer Lösungsansatz besteht nun darin, durch Parallelverschiebung der Isogewinn-Geraden die optimale Lösung zu ermitteln. Offensichtlich vergrößert sich der Zielfunktionswert bei einer Parallelverschiebung in eine Richtung und verkleinert sich in die andere Richtung. Die Isogewinn-Gerade wird also in Richtung der Verkleinerung so lange verschoben, bis eine weitere Verschiebung dafür sorgen würde, dass keine zulässigen Punkte mehr auf der Isogewinn-Geraden liegen. Damit liegt der optimale Punkt am Rand des zulässigen Bereichs. Falls eine Nebenbedingung parallel zur Isogewinn-Geraden liegt, kann das Optimum erreicht werden, wenn die Isogewinn-Gerade genau auf der entsprechende Geraden der Nebenbedingung liegt, ansonsten wird der optimale Punkt der Schnittpunkt von Nebenbedingungsgeraden sein. In unserem Beispiel erhalten wir den Punkt $x_1^* = 60$ und $x_2^* = 30$, der den Schnittpunkt der Nebenbedingungen $x_1 = 60$ und $9x_1 + 6x_2 = 720$ definiert.

Bevor wir die Vorgehensweise verallgemeinern, sollen kurz die in allgemeinen Problemen auftretenden Strukturen vorgestellt werden.

Im Beispiel	**Allgemein**
2 Variablen	nicht selten $n > 1000$ Variablen
$n + m$ Restriktionen definieren je	
eine (Halb-)Fläche des $\mathbb{R}^2$	einen Halbraum des $\mathbb{R}^n$
unter Einschluss der trennenden	
Geraden	Hyperebenen
zulässiger Bereich ist Schnitt von	
5 Halbflächen des $\mathbb{R}^2$	Halbräumen des $\mathbb{R}^n$
$f(x) = \text{const} = Z$ definiert Isogewinn-	
Gerade	Hyperebene der Dimension $n - 1$

Nach unseren Beobachtungen am Beispiel sollte die optimale Lösung in einem Eckpunkt des zulässigen Bereichs liegen. Ein solcher Eckpunkt ist der Schnittpunkt von mindestens n Hyperebenen, in unserem Beispiel 2 Geraden. Da $n + m$ Restriktionen existieren, gibt es bis zu $\binom{n+m}{n}$ Eckpunkte. In unserem Beispiel sind dies $\binom{5}{2} = 10$ Eckpunkte. Es sind natürlich nicht unbedingt alle Schnittpunkte auch Eckpunkte des zulässigen Bereichs, wie man auch am Beispiel sehen kann, bei dem 5 der 10 Schnittpunkte auch Eckpunkte sind.

Das informell beschriebene Vorgehen legt einen naiven Algorithmus nahe:

1. Erzeuge nacheinander alle Schnittpunkte der begrenzenden Hyperebenen.
2. Prüfe bei jedem Schnittpunkt, ob er Eckpunkt des zulässigen Bereichs ist.
3. Der minimale Funktionswert, der an einem Eckpunkt des zulässigen Bereichs gefunden wurde, ist das gesuchte Minimum.

Unter Umständen müssen wir mit diesem Verfahren sehr viele Punkte untersuchen, wenn unsere Ahnung, dass das Optimum immer in einem Eckpunkt zu finden ist, aber zutrifft, würden wir mit dem Vorgehen das Optimum finden. Bevor wir uns mit dem formalen Beweis beschäftigen, wollen wir uns noch kurz Gedanken machen, ob man nicht effizienter vorgehen könnte. Dies führt schon zur eigentlichen Idee das Simplexalgorithmus.

Offensichtlich kann man das Verfahren verbessern, indem die Eckpunkte systematisch untersucht werden. Dazu kann folgendes Vorgehen genutzt werden:

1. Starte in einem initialen Eckpunkt des zulässigen Bereichs. In Normalform ist 0 ein solcher zulässiger Eckpunkt.
2. Wähle einen besseren Nachbar-Eckpunkt, d.h. einen Eckpunkt mit kleinerem Zielfunktionswert, der durch Austausch einer definierenden Hyperebene entsteht. Auf der verbindenden Kante sollte der Zielfunktionswert linear fallen.
3. Fahre fort, bis kein besserer Nachbareckpunkt mehr zu finden ist.

Der terminale Eckpunkt sollte die optimale Lösung repräsentieren. Neben der Voraussetzung, dass die optimale Lösung in einem Eckpunkt zu finden ist, wird bei diesem Vorgehen noch vorausgesetzt,

- dass ein lokal optimaler Eckpunkt, ohne bessere Nachbarn, auch global optimal ist und
- dass überhaupt ein Optimum existiert.

Wir wollen uns erst der Frage widmen, ob immer eine Lösung (d.h. ein Optimum) existieren muss. Dazu reicht es aus, den eindimensionalen Fall zu untersuchen. Also cx soll unter den Nebenbedingungen $ax \leq b$ und $x \geq 0$ minimiert werden. Dabei können die folgenden vier Fälle unterschieden werden.

$a > 0,\ b > 0$	scheint ok zu sein
$a > 0,\ b < 0$	der zulässige Bereich ist leer
$a < 0,\ c > 0$	scheint ok zu sein
$a < 0,\ c < 0$	Zielfunktion ist unbeschränkt

Damit existiert im zweiten und vierten Fall kein Optimum. Es lässt sich erahnen, dass bei mehrdimensionalen Zielfunktionen und mehreren Nebenbedingungen zusätzliche Probleme auftreten können. Damit muss auch ein Verfahren gefunden werden, mit dem festgestellt werden kann, ob überhaupt eine Lösung existiert.

In Normalform (10.1) werden die Nebenbedingungen des linearen Problems als Ungleichungen beschrieben. Da sich mit Gleichungen leichter als mit Ungleichungen rechnen lässt, wollen wir die sogenannte Standardform einführen indem aus den Ungleichungen Gleichungen gemacht werden. Die folgende Darstellung bezeichnen wir im Folgenden als Standardform.

$$\min \left(c^T x\right) \text{ udN } Ax = b \text{ und } x \geq 0 \tag{10.2}$$

Ein lineares Programmierungsproblem in Normalform kann durch Einführung von Schlupfvariablen in ein lineares Programmierungsproblem in Standardform überführt werden. Statt

$$Ax \leq b \text{ schreiben wir } Ax = b - u \ \Rightarrow\ Ax + u = b \ \Rightarrow\ (A, I) \begin{pmatrix} x \\ u \end{pmatrix} = b$$

mit der Zielfunktion $c^T x + 0^T u$. Ferner muss gelten $x \geq 0$ und $u \geq 0$.

Damit kann jedes lineare Programmierungsproblem von der Normalform in die Standardform transformiert werden. Falls Nebenbedingungen der Form $Ax \geq b$ vorliegen, so wird statt (A, I) die Matrix $(A, -I)$ verwendet.

Im Folgenden setzen wir voraus, dass die Schlupfvariablen direkt integriert werden, so dass Vektor u in den Vektor x integriert wird. Wir benutzen die Schreibweise $Z = \min\ f(x) = c^T x$ unter den Nebenbedingungen $Ax = b$ und $x \geq 0$. A ist eine $m \times n$ Matrix $(m < n)$ und wir setzen voraus, dass $rg(A) = rg(A, b) = m$, so dass das Gleichungssystem redundanten Gleichungen enthält, die sich als Linearkombination anderer Gleichungen darstellen lassen. Da $m < n$ gilt, ist das Gleichungssystem unterbestimmt, und es existiert ein Lösungsraum. Damit eine Lösung zulässig ist, müssen auf Grund der Nichtnegativitätsbedingungen alle Komponenten nichtnegativ sein. Es kann deshalb auch vorkommen, dass das Gleichungssystem keine zulässige Lösung hat und damit kein Punkt die Nebenbedingungen erfüllt und der zulässige Bereich leer ist.

10.2 Formale Grundlagen

Das Simplexverfahren basiert insbesondere auf der geometrischen Struktur des zulässigen Bereichs W. Für die Herleitung werden die folgenden einfachen Definitionen und Eigenschaften benötigt:

- Die Euklidische Norm $\|x\|$ für $x \in \mathbb{R}^n$ ist definiert als $\|x\| = \sqrt{(x_1^2 + x_2^2 + \ldots + x_n^2)}$.
- Der Abstand zwischen zwei Punkten $x, y \in \mathbb{R}^n$ ist definiert als $\|x - y\|$.
- Eine Kugel K mit Radius ϵ und Mittelpunkt $m \in \mathbb{R}^n$ (ϵ-Kugel) ist die Punktmenge $K = \{x |\, \|x - m\| \leq \epsilon\}$

- Eine Menge $M \subseteq \mathbb{R}^n$ heißt beschränkt, wenn es eine Konstante $c \in \mathbb{R}$ gibt, so dass $\|x\| \leq c$ für alle $x \in M$.
- Eine Menge $M \subseteq \mathbb{R}^n$heißt abgeschlossen, falls für jede konvergente Folge $\{x_i \in M\}_{i=1,\ldots,\infty}$ der Grenzwert in M liegt. x_i ist eine konvergente Folge, falls $\|x - x_i\| < \epsilon$ für $i \geq i_0$, $\epsilon > 0$ und x der Grenzwert der Folge ist.
- Eine Gerade g im $\mathbb{R}^n$ ist definiert durch die Menge aller Punkte $\{x + \lambda y | -\infty \leq \lambda \leq +\infty\}$ für ein $y \neq 0$.
- Seien $x^1, \ldots, x^r \in \mathbb{R}^n$ und $\lambda_1, \ldots, \lambda_r \in \mathbb{R}_{\geq 0}$, dann heißt $x = \sum_{j=1}^n \lambda_j x^j$ eine nichtnegative Linearkombination. Falls zusätzlich $\sum_{j=1}^n \lambda_j = 1$ gilt, so spricht man von einer konvexen Linearkombination und falls $\lambda_j > 0$ von einer echten konvexen Linearkombination.

Auf dieser Basis kann nun die Struktur des zulässigen Bereichs hergeleitet werden.

Definition 6 *Für zwei Punkte $x, y \in \mathbb{R}^n$ heißt die Punktmenge $\lambda x + (1-\lambda)y$ $(0 \leq \lambda \leq 1)$ die Verbindungsstrecke zwischen x und y.*

Eine Menge $M \subseteq \mathbb{R}^n$ heißt konvex, wenn für alle Punktepaare $x, y \in M$ auch die Verbindungsstrecke in M liegt.

Eine Funktion $f : M \to \mathbb{R}$ heißt streng konvex, falls M eine konvexe Menge ist und für alle $x, y \in M$ $(x \neq y)$ und $\lambda \in (0,1)$ stets folgt $f(\lambda x + (1-\lambda)y) < \lambda f(x) + (1-\lambda)f(y)$. Falls nur $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$ gilt, so heißt die Funktion konvex.

Konvexe Funktionen und Mengen sind von großer Bedeutung, da bei ihnen lokale und globale Optima zusammenfallen, was die globale Optimierung oft erst ermöglicht.

Satz 2 *Der zulässige Bereich W eines linearen Optimierungsproblems ist konvex und abgeschlossen.*

Beweis: Seien $x, y \in W$, dann gilt $xA = yA = b$. Sei $z = \lambda x + (1-\lambda)y$, dann gilt

$$zA = \lambda xA + (1-\lambda)yA = \lambda b + (1-\lambda)b = b \ ,$$

so dass die Menge konvex ist. Die Abgeschlossenheit folgt, da der zulässige Bereich die Schnittmenge abgeschlossener Mengen ist.

□

Die Zielfunktion eines linearen Optimierungsproblems $c^T x$ ist konvex aber nicht streng konvex. Sei Z der minimale Wert des linearen Optimierungsproblems und $L(Z) = \left\{x | c^T x = Z\right\}$ die Menge der Punkte, die einen minimalen Funktionswert liefern.

Satz 3 *$L(Z)$ ist eine konvexe Menge.*

Beweis: Seien $x, y \in L(Z)$, dann gilt $z = \lambda x + (1-\lambda)y$

$$c^T z = \lambda c^T x + (1-\lambda)c^T y = \lambda Z + (1-\lambda)Z = Z$$

womit $z \in L(Z)$ gilt. □

Die Struktur der zulässigen Menge soll nun noch etwas weiter eingegrenzt werden. Die Menge aller konvexen Linearkombination endlicher vieler Punkte des $\mathbb{R}^n$ heißt konvexes Polytop. Ein konvexes Polytop, das von $n+1$ nicht auf einer Hyperebene liegenden Punkte des $\mathbb{R}^n$ aufgespannt wird heißt Simplex. Offensichtlich ist ein konvexes Polytop beschränkt und auch abgeschlossen (d.h. kompakt). Dies gilt für die zulässige Menge eines linearen Optimierungsproblems nicht, da wir schon bei dem einfachen eindimensionalen Beispiel gesehen haben, dass der zulässige Bereich durchaus unbeschränkt sein kann. Ein konvexes Polyeder erhält man dadurch, dass man bei einem konvexen Polytop einzelne beschränkende Hyperebenen weglässt. Formal kann man ein konvexes

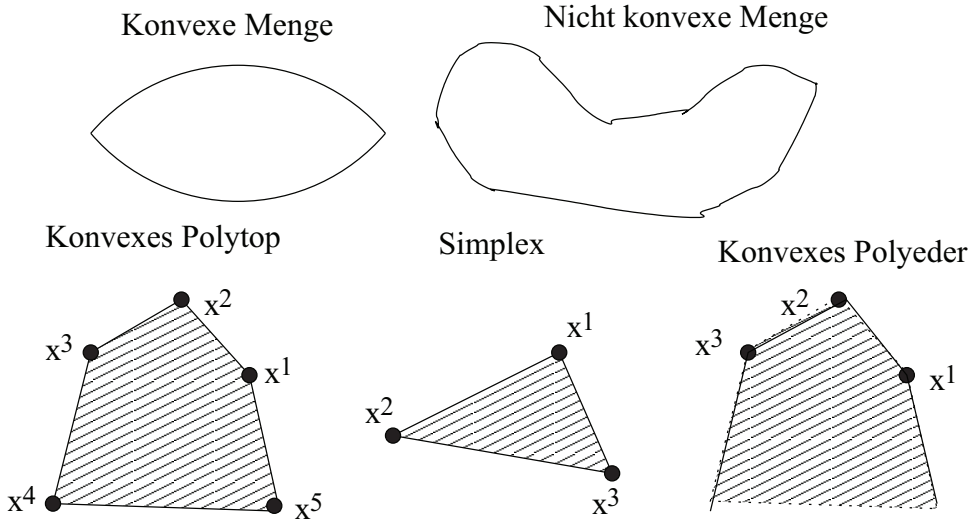


Abbildung 10.2: Verschiedene konvexe Strukturen.

Polyeder als Summe eines konvexen Polytops und eines konvexen polyedrischen Kegel beschreiben. Ein konvexer polyedrischer Kegel ist die Menge aller nichtnegativen Linearkombinationen endlich vieler Punkte im $\mathbb{R}^n$. Aus einem konvexen Polytop P und einem konvexen polyedrischen Kegel C entsteht ein konvexes Polyeder.

$$C + P = \left\{x = x^P + x^C | x^P \in P, x^c \in C\right\}$$

Man kann ein konvexes Polyeder auch als Durchschnitt endlich vieler Halbräume $\left\{x \in \mathbb{R}^n | a^j x \leq b_j\right\}$ mit festen Zeilenvektoren a^j und reellen Zahlen b_j definieren, also als Lösung linearer Ungleichungssysteme. Offensichtlich müssen diese Mengen nicht beschränkt sein.

Abbildung 10.2 zeigt Beispiele für die verschiedenen Strukturen, die wir kennengelernt haben. Ein Punkt x aus einer konvexen Menge M heißt

- Randpunkt von M, wenn keine ϵ-Kugel ($\epsilon > 0$) um x existiert, die nur Punkte aus M enthält.
- Extrempunkt, wenn er sich nicht als echte konvexe Linearkombination zweier verschiedener Punkte aus M darstellen lässt.

Offensichtlich ist jeder Extrempunkt auch Randpunkt. Die Extrempunkte eines konvexen Polyeders werden auch Ecken genannt. Ein Punkt $x = (x_1, \ldots, x_n)$ ist genau dann ein Extrempunkt, wenn diejenigen Spaltenvektoren $A_{\bullet j}$ der Matrix A, deren Koeffizienten in x ungleich 0 sind, linear unabhängig sind und $\sum_{j=1}^{n} A_{\bullet j} x_j = b$ gilt.

Die folgende Herleitung ist relativ detailliert und basiert im Wesentlichen auf den Beweisen in Neumann and Morlock [1993, Kap. 1] und Wegener and Hofmeister [2000, Kap. 2]. Wir weisen zuerst formal nach, dass das Minimum eines linearen Optimierungsproblems, sofern es existiert, in einer Ecke angenommen wird. Dazu benötigen wir den folgenden Hilfssatz.

Satz 4 *Für jeden Punkt $x + \lambda y$ auf einer Geraden im $\mathbb{R}^n$ gilt entweder*

- *alle Punkte $x + \lambda' y$ mit $\lambda' > \lambda$ haben einen größeren Abstand zum Nullpunkt als $x + \lambda y$ (d.h. $\|x + \lambda' y\| > \|x + \lambda y\|$) oder*
- *alle Punkte $x + \lambda' y$ mit $\lambda' < \lambda$ haben einen größeren Abstand zum Nullpunkt als $x + \lambda y$.*

Beweis: Da $\|y\| > \|y'\| \Leftrightarrow \|y\|^2 > \|y'\|^2$ können wir den Beweis für den quadratischen Abstand führen. Wir definieren dazu eine Funktion

$$f(\lambda') = \|x + \lambda' y\|^2 = (x_1 + \lambda' y_1)^2 + \ldots + (x_n + \lambda' y_n)^2 = c_1 \lambda'^2 + c_2 \lambda' + c_3$$

für Konstanten c_1, c_2, c_3. Da $y \neq 0$ (sonst wäre es keine Gerade) gilt $y_i^2 > 0$ für mindestens ein i und damit auch $c_1 > 0$. Damit ist $f(\lambda')$ eine nach oben offene Parabel und $f(\lambda') > f(\lambda)$ für $\lambda' > \lambda$ oder für $\lambda' < \lambda$, je nachdem, wo wir uns auf der Parabel befinden. □

Der folgende Satz zeigt, dass auf kompakten konvexen Mengen maximale Punkte Extrempunkte sein müssen.

Satz 5 *Sei $\emptyset \neq M \subset \mathbb{R}^n$ konvex und kompakt (d.h. abgeschlossen und beschränkt). Dann existiert $\max_{x \in M} \|x\|$ und jeder Punkt x, für den dieses Maximum angenommen wird, ist ein Extrempunkt.*

Beweis: Das Maximum existiert, da $\|.\|$ eine stetige Funktion ist und damit auf kompakten Mengen ein Maximum annimmt. Sei x der Punkt, für den das Maximum angenommen wird, d.h. der Abstand zum Nullpunkt maximal ist.
Angenommen x sei kein Extrempunkt, dann gibt es ein y, so dass die Strecke von $x - y$ nach $x + y$ vollständig in M liegt. Nach Satz 4 haben die Punkte auf der Strecke von $x - y$ nach x oder auf der Strecke nach x nach $x + y$ einen größeren Abstand zum Nullpunkt als x. Dies ist ein Widerspruch zur Wahl von x, das maximalen Abstand haben soll.

□

Die beiden folgenden Sätze behandeln die Schnittmenge konvexer Mengen und die Lage von Randpunkten in den resultierenden Mengen.

Satz 6 *Seien $M_1, \ldots, M_k$ konvexe Mengen und $M = M_1 \cap \ldots \cap M_k$. Dann ist M konvex, und wenn x ein Randpunkt von M ist, so gibt es ein i, so dass x Randpunkt von M_i ist.*

Beweis: Aus $x, y \in M$ folgt $x, y \in M_i$ für alle $i \in \{1, \ldots, k\}$. Da alle M_i konvex sind, gilt dann auch $z = \lambda x + (1 - \lambda)y \in M_i$ und, da dies für alle $i \in \{1, \ldots, k\}$ gilt, folgt auch $z \in M$.
Falls x Randpunkt von M aber kein Randpunkt von M_i wäre, so gäbe es für jedes i ein $\epsilon_i > 0$, so dass alle Punkte in einer ϵ_i-Kugel um x in M_i liegen. Für $\epsilon = \min_{i=1,\ldots,k}(\epsilon_i)$ wären alle Punkte in einer ϵ-Kugel um x in M und x könnte kein Randpunkt von M sein.

□

Satz 7 *Sei M eine konvexe Menge und K eine Kugel. Dann ist jeder Extrempunkt von $M \cap K$ ($\neq \emptyset$) ein Extrempunkt von K oder ein Extrempunkt von M.*

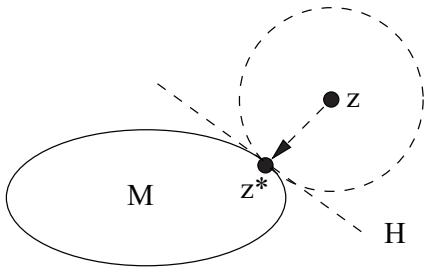
Beweis: Eine Kugel ist konvex und jeder Randpunkt ist auch Extrempunkt. Sei x von $M \cap K$. Nach Satz 6 ist x Randpunkt von M oder K und damit, falls Randpunkt von K auch Extrempunkt.
Sei deshalb x Randpunkt von M und nicht von K. Wir nehmen an, dass x kein Extrempunkt von M sei. Dann gibt es eine Strecke von $x - y$ nach $x + y$, die ganz in M liegt. Da x kein Randpunkt von K ist, gibt es eine ϵ-Kugel um x, die ganz in K liegt. Der Schnitt der Strecke mit dieser Kugel gibt eine Verbindungsstrecke von $x - y'$ nach $x + y'$, die ganz in $M \cap K$ liegt. Damit kann x kein Extrempunkt von $M \cap K$ sein.

□

Es soll im Folgenden gezeigt werden, dass das Minimum, falls es existiert, in einem Extrempunkt angenommen wird. Dazu benötigen wir einige Hilfssätze.

Satz 8 *Sei $\emptyset \neq M \subset \mathbb{R}^n$ konvex und abgeschlossen und sei $z \in \mathbb{R}^n \setminus M$. Dann existiert ein Punkt $z^* \in M$,der zu z am nächsten liegt (d.h. $z^* = argmin_{x \in M}(\|x - z\|)$. Die Hyperebene $H = \{y | uy = uz^*\}$ mit $u = (z^* - z)/(\|z^* - z\|) \in \mathbb{R}^n$verläuft durch z^* und hat alle Punkte von M auf einer Seite (d.h. es gilt $ux \geq uz^*$ für alle $x \in M$).*

Beweis: Das Vorgehen wird in der folgenden Abbildung veranschaulicht.



z^* existiert, da $D(x) = \|x - z\|$ $(x \in M)$ auf einer abgeschlossenen Menge stetig und nach unten beschränkt ist. Damit nimmt $D(x)$ ein Minimum in einem Punkt z^* an. Es gilt $\|u\| = 1$ und $u(z^* - z) > 0 \Rightarrow uz^* > uz$. Zu zeigen ist, $ux \geq uz^*$ für alle $x \in M$.
Wir betrachten einen Punkt $x \in M$ und die Verbindungsstrecke $w_\lambda = \lambda x + (1-\lambda)z^* = \lambda(x - z^*) + z^* \in M$ $(0 \leq \lambda \leq 1)$. $h(\lambda) = \|w_\lambda - z\|^2$ ist eine stetige Funktion, die das Quadrat des Abstands der Punkte auf der Geraden w_λ vom Punkt z misst. Es gilt

$$h(\lambda) = (w_\lambda - z)^2 = (\lambda(x - z^*) + (z^* - z))^2 = \lambda^2(x - z^*)^2 + 2\lambda(x - z^*)(z^* - z) + (z^* - z)^2$$

Die Ableitung dieser Funktion nach λ lautet

$$h'(\lambda) = 2\lambda(x - z^*)^2 + 2(x - z^*)(z^* - z)$$

Für $\lambda = 0$, also im Punkt z^* haben wir $h'(0) = 2(x - z^*)(z^* - z) = \nu u(x - z^*)$ für $\nu > 0$. Falls $h'(0) < 0$, dann gäbe es ein $\mu > 0$, so dass $h(\mu) < h(u)$. Dies würde bedeuten, dass der zugehörige Punkt w_μ einen geringeren Abstand zu z hat als z^*. Dies ist aber ein Widerspruch zur Wahl von z^*. Damit muss $h'(0) \geq 0$ gelten, woraus $u(x - z^*) \geq 0$ und damit $ux \geq uz^*$ folgt.

□

Satz 9 (Trennende Hyperebenen) *Sei M konvex und abgeschlossen und x^0 sei ein Randpunkt von M, dann existiert eine Hyperebene $H = \{y | uy = ux^0\}$ durch x^0, so dass alle $x \in M$ im gleichen Halbraum bzgl. H liegen (d.h. $ux \geq ux^0$ für alle $x \in M$).*

Beweis: Im Unterschied zum vorherigen Satz betrachten wir nun einen Punkt aus M. Die Beweisidee besteht darin, eine Folge von Punkten zu betrachten, die gegen x^0 konvergieren aber außerhalb von M liegen.
Da x^0 ein Randpunkt von M ist, enthält jede ϵ-Kugel um x^0 Punkte, die nicht in M liegen. Es gibt also eine Folge von Punkten $z^i \notin M$, die gegen x^0 konvergiert. Nach Satz 8 können wir für jedes z^i ein z^{i*} finden, das den geringsten Abstand zu z^i hat und in M liegt. Für dieses z^{i*} können wir eine Hyperebene H_i finden, die durch $u^i = (z^{i*} - z^i) / \left\|z^{i*} - z^i\right\|$ beschrieben wird und alle Punkte von M auf einer Seite hat.
Da $\left\|u^i\right\| = 1$ für alle i gibt es nach dem Satz von Bolzano-Weierstraß ("Jede beschränkte Folge enthält eine konvergente Teilfolge") eine konvergente Teilfolge u^{i_j}, die gegen einen Vektor u konvergiert. Betrachten wir die zugehörigen z^{i_j}und z^{i_j*}. Da z^i gegen x^0 konvergiert, konvergieren auch die Teilfolgen $z^{i_j)}$ und z^{i_j*} gegen x^0. Da ferner $u^{i_j} x \geq u^{i_j} z^{i_j*}$ für alle $x \in M$, folgt aus der Konvergenz auch $ux \geq ux^0$, so dass

$$H = \{y | uy = ux^0\}$$

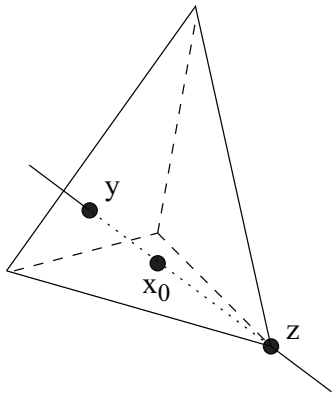
die gesuchte Hyperebene ist.

□

Der folgende Satz stellt für kompakte konvexe Mengen einen Zusammenhang zwischen beliebigen Punkten und den Extrempunkten her.

Satz 10 (von Krein und Milman) *Sei $\emptyset \neq M \subset \mathbb{R}^n$ konvex und kompakt. Dann lässt sich jeder Punkt $x \in M$ als konvexe Linearkombination von höchstens $n + 1$ Extrempunkten von M darstellen.*

Beweis: Der Beweis erfolgt per Induktion über n.
Für $n = 1$ ist M von der Form $[a, b]$ mit $a < b$. Offensichtlich ist jeder Punkt $c \in [a, b]$ in der Form $c = \lambda a + (1 - \lambda)b$ $(0 \leq \lambda \leq 1)$ darstellbar.
Sei nun $n > 1$. Nach Satz 5 enthält M Extrempunkte. Sei z ein Extrempunkt von M. Falls $x = z$, so sind wir fertig und haben eine Darstellung für x. Ansonsten betrachten wir eine Gerade durch z und x. Diese Gerade kann zu einem Randpunkt y verlängert werden, da M abgeschlossen und beschränkt ist. Das Vorgehen kann im dreidimensionalen wie folgt skizziert werden.



Offensichtlich kann x als konvexe Linearkombination von z und y dargestellt werden, wobei z ein Extrempunkt ist. y liegt auf einer Hyperebene $H = \{w | uw = uy\}$ der Dimension $n - 1$, die alle Punkte von M auf einer Seite hat, d.h. $uv \geq uy$ für alle $v \in M$. H ist abgeschlossen und konvex, daher ist $H \cap M$ kompakt, konvex und liegt in einem Raum der Dimension $n - 1$. Nach Induktionsvoraussetzung kann y als konvexe Linearkombination der Extrempunkte von $H \cap M$ dargestellt werden.
Damit bleibt zu zeigen, dass jeder Extrempunkt von $H \cap M$ auch Extrempunkt von M ist. Angenommen v sei Extrempunkt von $H \cap M$ aber nicht von M. Dann existiert eine Strecke von $v - w$ nach $v + w$ $(w \neq 0)$ in M. Dies bedeutet auch, dass $u(v - w) \geq uy$ und $u(v + w) \geq uy$, da beide Punkte in M liegen. Angenommen es würde sogar $u(v + w) > uy$ gelten, dann

$$uv = \frac{1}{2}u(v + w + v - w) = \frac{1}{2}u((v + w) + (v - w)) > \frac{1}{2}(uy + uy) = uy$$

Damit wäre aber $v \notin H$ und kann damit auch kein Extrempunkt von $H \cap M$ sein. Also muss $u(v - w) = u(v + w) = uy$ sein und die Strecke von $v - w$ nach $v + w$ liegt in H und in M, so dass v kein Extrempunkt von $H \cap M$ sein kann. Damit folgt, dass jeder Extrempunkt von $H \cap M$ auch Extrempunkt von M sein muss.

□

Wir haben bereits in Satz 2 gezeigt, dass der zulässige Bereich eines linearen Optimierungsproblems konvex ist. Der folgende Satz zeigt, dass er auch noch Extrempunkte enthält.

Satz 11 *Sei W der zulässige Bereich eines linearen Optimierungsproblems. Es gilt*
$W \neq \emptyset \Rightarrow$ W hat mindestens einen Extrempunkt.
$W \neq \emptyset \Rightarrow$ jeder Punkt $x^0 \in W$ lässt sich als konvexe Linearkombination von Punkten $x^1, \ldots, x^p \in W$ darstellen, wobei $\lambda_1 > 0$ und x^1 ein Extrempunkt von W ist.

Beweis: Die erste Behauptung folgt offensichtlich aus der zweiten. Wenn W beschränkt ist, so ist er nach Satz 2 auch kompakt. Nach Satz 10 kann man sogar eine Linearkombination aus Extrempunkten angeben und der Satz gilt.
Sei also W unbeschränkt. Falls x^0 der Nullvektor ist, so ist x^0 wegen der Nebenbedingungen $x_i \geq 0$ auch Extrempunkt. Ansonsten betrachten wir eine Kugel K mit Radius $(n + 2)\left\|x^0\right\|$ um den Nullpunkt. Da $W \cap K$ konvex und kompakt ist, lässt sich x^0 nach Satz 10 als konvexe

Linearkombination von Extrempunkten $x^1, \ldots, x^p$ mit $p \leq n+1$ als $x^0 = \sum_{i=1}^{p} \lambda_i x^i$ darstellen. Nach Satz 7 ist jedes x^i Extrempunkt von W oder K. Wir müssen nun nachweisen, dass nicht alle x^i Extrempunkte von K sind.
Angenommen alle x^i wären Extrempunkte von K, dann wäre $\left\|x^i\right\| = (n+2)\left\|x^0\right\|$ für alle i. Wähle ein $\lambda_i \geq 1/(n+1)$. Ein solches muss existieren, da $\lambda_i > 0$, $\sum_{i=1}^{p} \lambda_i = 1$ und $p \leq n+1$. Dann gilt

$$\begin{aligned} \left\|x^0\right\| &= \left\|\lambda_1 x^1 + \ldots + \lambda_p x^p\right\| &&= \sqrt{\left(\sum_{j=1}^{p} \lambda_j x_1^j\right)^2 + \ldots + \left(\sum_{j=1}^{p} \lambda_j x_n^j\right)^2} \\ &\geq \sqrt{\left(\lambda_i x_1^i\right)^2 + \ldots + \left(\lambda_i x_n^i\right)^2} &&= \lambda_i \left\|x^i\right\| \geq \tfrac{n+2}{n+1}\left\|x^0\right\| \qquad > \left\|x^0\right\| \end{aligned}$$

Die vorletzte Ungleichung gilt, weil alle x^i auf dem Kugelrand liegen und führt zum Widerspruch. Also muss mindestens ein x^i ein Extrempunkt von W sein. Dieses kann dann durch Umnummerierung zu x^1 werden.

□

Damit können wir zeigen, dass das Minimum der Zielfunktion in einem Extrempunkt des zulässigen Bereichs angenommen wird.

Satz 12 *Nimmt die Zielfunktion $f(x)$ ihr Minimum auf dem zulässigen Bereich an, dann wird das Minimum auch in einem Extrempunkt angenommen.*

Beweis: Sei $x^0 \in W$ und $f(x^0) = \min_{x \in W}(f(x))$. Falls x^0 kein Extrempunkt ist, so kann x^0 als konvexe Linearkombination $x^0 = \sum_{j=1}^{p} \lambda_j x^j$ dargestellt werden, wobei mindestens ein x^i Extrempunkt von W ist (siehe Satz 11). Aus der Linearität der Zielfunktion und der Minimalität von x^0 folgt

$$f(x^0) = \lambda_1 f(x^1) + \ldots + \lambda_p f(x^p) \geq \lambda_1 f(x^0) + \ldots + \lambda_p f(x^0) = f(x^0)$$

Also muss aus dem $\geq$ ein $=$ werden und $f(x^i) = f(x^0)$ gilt für einen Extrempunkt $x^i \in W$.

□

Im vorherigen Abschnitt haben wir schon die Anzahl der Eckpunkte als Schnittpunkte von Hyperebenen charakterisiert. Dies wird im folgenden Satz formalisiert.

Satz 13 *Jeder Extrempunkt des zulässigen Bereichs W eines linearen Optimierungsproblems mit den Nebenbedingungen $Ax \leq b$ und $x \geq 0$ ist Schnittpunkt von n linear unabhängigen Hyperebenen aus der Menge der $n+m$ Hyperebenen $H_1, \ldots, H_{n+m}$.*

Es gibt maximal $\binom{m+n}{n}$ Extrempunkte in W.

Beweis: Sei x^0 Extrempunkt von W. Damit ist x^0 auch Randpunkt von W und liegt auf mindestens einer Hyperebene. Sei $I = \left\{i | x^0 \text{ liegt auf } H_i\right\} \subseteq \{1, \ldots, n+m\}$ damit gilt auch $\{x^0\} = \cap_{i \in I} H_i \neq \emptyset$.
Um zu zeigen, dass I n linear unabhängige Hyperebenen enthält, nehmen wir an, dass höchstens $n-1$ Hyperebenen $H_i (i \in I)$ linear unabhängig sind. Dann hat der Durchschnitt $\cap_{i \in I} H_i$ mindestens Dimension 1 und enthält eine Gerade g durch x^0. Von den anderen Hyperebenen H_j mit $j \notin I$ hat x^0 einen endlichen Abstand, der für ein geeignet gewähltes $\epsilon > 0$ größer als ϵ ist. Sei K die ϵ-Kugel um x^0. Dann gilt $g \cap K \subseteq W$ und x^0 liegt auf einer Strecke, die ganz in W liegt. Damit kann x^0 kein Extrempunkt sein.
Die maximale Anzahl der Extrempunkte ergibt sich aus den Auswahlmöglichkeiten von n aus $n+m$ Hyperebenen.

□

Damit haben wir hergeleitet, dass das Minimum, sofern es existiert, in einem Eckpunkt angenommen wird. Es kann aber ein leerer zulässiger Bereich vorliegen oder die Zielfunktion kann unbeschränkt sein. Wir haben bisher noch nicht gezeigt, dass ein lokales Optimum auch global optimal ist. Nur wenn dies der Fall wäre, können wir einen Suchalgorithmus abbrechen, wenn wir eine Lösung gefunden haben, die sich lokal nicht mehr verbessern lässt.

Satz 14 *Ein lokales Optimum der Zielfunktion $f(x)$ eines linearen Optimierungsproblems auf dem zulässigen Bereich W ist auch ein globales Optimum.*

Beweis: Sei x^0 ein lokales Minimum und $f(x^1) < f(x^0)$ für $x^1 \in W$. Da W konvex ist, liegt die Verbindungsstrecke von x^0 nach x^1 ganz in W. Im Inneren der Strecke gilt für Punkte $x = \lambda x^0 + (1-\lambda)x^1 (0 < \lambda < 1)$

$$f(x) = \lambda f(x^0) + (1-\lambda) f(x^1) < f(x^0)$$

Dies gilt auch für $\lambda \to 1$ und damit $x \to x^0$, so dass entweder x^0 kein lokales Optimum ist oder kein x^1 mit $f(x^1) < f(x^0)$ existieren kann.

□

Damit haben wir die Grundlagen für den Simplexalgorithmus gelegt. Im Prinzip läuft der Algorithmus von Extrempunkt zu Extrempunkt und verbessert dabei den Zielfunktionswert in jedem Schritt, so lange, bis keine Verbesserung mehr möglich ist. Gleichzeitig sollten Spezialfälle wie leere zulässige Bereiche oder unbeschränkte Zielfunktionen abgefangen werden.

10.3 Prinzip des Simplexverfahrens

Das Simplexverfahren benutzt die folgenden Schritte:

- Ausgehend von einem Eckpunkt des zulässigen Bereichs (d.h. einem Schnittpunkt von n Hyperebenen)
- werden Nachbareckpunkte aufgesucht, indem eine der n Hyperebenen gegen eine der restlichen m Hyperebenen ausgetauscht wird, so dass ein Eckpunkt erreicht wird, der
  - im zulässigen Bereich liegt und
  - einen verbesserten Zielfunktionswert aufweist.
  - Falls ein solcher Eckpunkt nicht existiert, wurde ein lokales Optimum erreicht.
- Während der Berechnung sollten leere zulässige Bereiche und unbeschränkte Zielfunktionen erkannt werden.

Für das Vorgehen muss ferner die Komplexität hergeleitet werden, und die Terminierung muss gezeigt werden.

Bevor der Simplexalgorithmus hergeleitet wird, soll das prinzipielle Vorgehen an einem einfachen Beispiel erläutert werden. Wir betrachten dazu das folgende Beispiel:

$$\min(c^T x) = \min(-3,-5) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \text{ udN } Ax = \begin{pmatrix} -1 & 1 \\ 2 & -3 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 2 \\ 3 \\ 12 \end{pmatrix} = b \qquad (10.3)$$

Durch die Einführung der nichtnegativen Schlupfvariablen x_3, x_4 und x_5 entsteht ein Gleichungssystem mit 3 Gleichungen und 5 Variablen.

$$Ax = \begin{pmatrix} -1 & 1 & 1 & 0 & 0 \\ 2 & -3 & 0 & 1 & 0 \\ 2 & 3 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 12 \end{pmatrix} = b$$

Zusätzlich muss noch gelten $x \geq 0$ und $c_i = 0$ für $i > n = 2$. Abbildung 10.3 veranschaulicht die Struktur des zulässigen Bereichs für x_1 und x_2. Falls ein Punkt auf einer begrenzenden Geraden angenommen wird, so ist der Wert der zugehörigen Schlupfvariable 0.

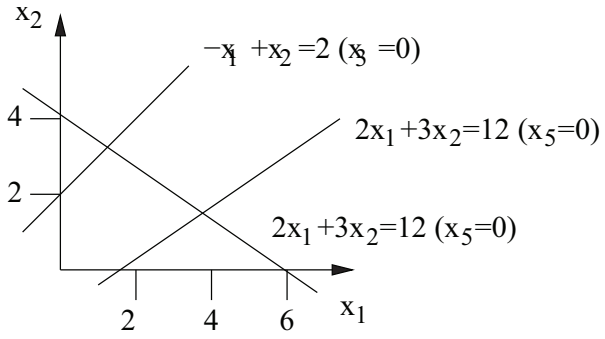


Abbildung 10.3: Zulässiger Bereich des Beispielmodells.

Wir haben im vorherigen Abschnitt m Gleichungen bzw. Ungleichungen und $n+m$ Variablen betrachtet. Zur Vereinfachung der Schreibweise setzen wir $q = n+m$ und unterscheiden nicht mehr zwischen Schlupfvariablen und "normalen" Variablen.

Bevor wir zum Simplexalgorithmus kommen, soll die Lösung linearer Gleichungssysteme kurz in Erinnerung gerufen werden. Sei A eine $m \times q$ Matrix mit $rg(A) = m < q$. Sei B eine $m \times m$ Matrix, die aus m linear unabhängigen Spalten von A besteht. Wir definieren dazu zwei disjunkte Mengen $\mathcal{B}, \mathcal{N} \subset \{1, \ldots, q\}$, so dass $\mathcal{B} \cup \mathcal{N} = \{1, \ldots, q\}$ und $\mathcal{B}$ m Elemente beinhaltet. Ferner nehmen wir an, dass die zu Indizes in $i \in \mathcal{B}$ gehörigen Spalten $A_{\bullet i}$ von A linear unabhängig sind. m unabhängige Spalten existieren in A, da $rg(A) = m$. Sei $B = (A_{\bullet i})_{i \in \mathcal{B}}$ und $N = (A_{\bullet j})_{j \in \mathcal{N}}$. Sei weiterhin $rg(A) = rg(A, b)$, so dass $Ax = b$ eine Lösung hat. Dann hat das lineare Gleichungssystem $Bx^B = b$ eine eindeutige Lösung, und $\bar{x}$ mit $\bar{x}_i = x_i^B$ falls $i \in \mathcal{B}$ und 0 sonst, ist eine Basislösung für $Ax = b$. $(x_i)_{i \in \mathcal{B}}$ sind die Basisvariablen und $(x_j)_{j \in \mathcal{N}}$ sind die Nichtbasisvariablen. Das Setzen auf 0 ist nur eine geschickte Wahl, den Nichtbasisvariablen Werte zuzuweisen, die wir im Folgenden nutzen werden. Es existieren aber auch andere Lösungen von $Ax = b$, die eine identische Basislösung nutzen. Ist die Basislösung nichtnegativ, so spricht man von einer zulässigen Basislösung, falls zusätzlich der Wert der Zielfunktion minimal ist, so spricht man einer optimalen Basislösung.

Wir definieren nun eine erweiterte Form unseres Gleichungssystems, die wir später im Simplexalgorithmus nutzen werden, indem die Zielfunktion mit einbezogen wird.

$$\hat{A} = \begin{pmatrix} A & 0 \\ c^T & -1 \end{pmatrix}, \ \hat{b} = \begin{pmatrix} b \\ 0 \end{pmatrix} \text{ und } \hat{x} = \begin{pmatrix} x \\ z \end{pmatrix}$$

z ist eine freie Variable ohne Vorzeichenbeschränkung, die den Wert der Zielfunktion wiedergibt (d.h. $z = c^T x$). Damit erhalten wir ein äquivalentes lineares Programm

$$\min \left\{ z | \hat{A} \begin{pmatrix} x \\ z \end{pmatrix} = \hat{b}, x \geq 0 \right\}$$

mit dem zulässigen Bereich

$$\hat{W} = \left\{ \hat{x} = \begin{pmatrix} x \\ z \end{pmatrix} | \hat{A}\hat{x} = \hat{b}, x \geq 0 \right\}$$

Wenn $(x_{j_1}, \ldots, x_{j_q})$ eine Basislösung von A ist, so ist der erweiterte Vektor $(x_{j_1}, \ldots, x_{j_q}, c^T x)$ eine Basislösung von $\hat{A}$ und umgekehrt.

Satz 15 *Zu jedem Extrempunkt $\bar{x}$ des zulässigen Bereichs W gibt es eine zulässige Basis $\mathcal{B}$, so dass $y = (x_i)_{i \in \mathcal{B}}$ Basislösung von $Bx^B = b$ ist und umgekehrt. B ist dabei die Matrix der zur Basis $\mathcal{B}$ gehörenden Spaltenvektoren.*

Beweis: Sei $\bar{x}$ ein Extrempunkt des zulässigen Bereichs. Sei $\mathcal{S}(\bar{x}) = \{i | \bar{x}_i > 0\}$ die Menge der Indizes, die zu Nichtnullelementen in $\bar{x}$ gehören. Wir zeigen, dass die Indizes in $\mathcal{S}(\bar{x})$ zu linear unabhängigen Spalten von A gehören und damit eine Basis bilden oder zu einer Basis ergänzt werden

können. Angenommen die Spalten wären nicht linear unabhängig, dann gäbe es Koeffizienten α_i so dass

$$\sum_{i \in \mathcal{S}(\bar{x})} \alpha_i A_{\bullet i} = 0 \ .$$

Wir definieren einen Vektor $z \in \mathbb{R}^n$ mit

$$z_i = \begin{cases} \alpha_i & \text{falls } i \in \mathcal{S}(\bar{x}) \\ 0 & \text{sonst} \end{cases}$$

Offensichtlich gilt dann $Az = 0$ und damit auch $A(\bar{x} \pm \epsilon z) = b$ für $\epsilon > 0$. Damit wäre die Gerade von $\bar{x} - \epsilon z$ nach $\bar{x} + \epsilon z$ in W, und $\bar{x}$ kann kein Extrempunkt sein.
Sei $\mathcal{B}$ eine Basis und $\bar{x}$ die zugehörige Basislösung. Es gelte $\bar{x}_i = 0$ für $i \in \mathcal{N}$, so dass für jeden zugehörigen Extrempunkt $\bar{\bar{x}}$ gelten muss $\mathcal{S}(\bar{\bar{x}}) \subseteq \mathcal{B}$. Wir nehmen nun an, dass $\bar{\bar{x}}$ kein Extrempunkt sei. Dann existieren Punkte $y, z \in W$, so dass $\bar{\bar{x}} = \lambda y + (1 - \lambda) z$ für $0 < \lambda < 1$. Zu zeigen ist, dass $\bar{\bar{x}} = y = z$ gelten muss und damit $\bar{\bar{x}}$ ein Extrempunkt sein muss. Betrachten wir zuerst die Indizes $i \in \mathcal{N}$.

$$\bar{\bar{x}}_i = \lambda y_i + (1 - \lambda) z_i = 0 \Rightarrow y_i = z_i = 0$$

Die Folgerung gilt, da aus $y, z \in W$ $y \geq 0$ und $z \geq 0$ folgt.
Für $i \in \mathcal{S}(\bar{\bar{x}})$ können wir die folgende Darstellung wählen.

$$0 = b - b = Ay - Az = A(y - z) = \sum_{i \in \mathcal{S}(\bar{\bar{x}})} \alpha_i A_{\bullet i}(y_i - z_i)$$

Da $\mathcal{S}(\bar{\bar{x}}) \subseteq \mathcal{B}$, sind die Spalten $A_{\bullet i}$ linear unabhängig, und die einzige Lösung ist $y_i = z_i$.

□

Falls $\bar{x}$ eine Ecke des zulässigen Bereichs W ist, so ist $\hat{\bar{x}} = \left(\bar{x}, c^T \bar{x}\right)^T$ eine Ecke von $\hat{W}$, wie man leicht zeigen kann.

Es stellt sich nun die Frage, inwieweit ein Extrempunkt eine Basis eindeutig definiert. Wie man aus dem Beweis des vorherigen Satzes schon erahnen kann, ist die Antwort nein. Betrachten wir dazu folgendes Beispiel.

$$A = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \ b = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Der einzige Extrempunkt ist $(0, 0, 1)^T$. Für diesen Punkt kann man die Basen $\{1, 3\}$ und $\{2, 3\}$ wählen. Das Problem tritt immer dann auf, wenn ein Extrempunkt mit weniger als n Nichtnullelementen charakterisiert werden kann oder anders ausgedrückt, wenn sich mehr als n Hyperebenen in einem Punkt schneiden. Man spricht dann von einem entarteten oder degenerierten Extrempunkt.

Wir kommen nun zurück zum Beispiel aus (10.3) und schreiben es etwas anders auf, um den Ablauf des Austauschschrittes zu erläutern.

x_1	x_2	x_3	x_4	x_5			
-1	1	1	0	0	=	2	(1)
2	-3	0	1	0	=	3	(2)
2	3	0	0	1	=	12	(3)
-3	-5				=	0	(Z)

Die erste Zeile beinhaltet die Variablennamen, die folgenden drei Zeilen die Elemente aus Matrix A und Vektor b und die letzte Zeile die Zielfunktion und den Funktionswert, unter der Bedingung, dass die letzten drei Variablen die Basis bilden. Damit haben wir die Basislösung $(0, 0, 2, 3, 12)^T$, bei der nur die Schlupfvariablen Werte ungleich 0 sind. Um zu einem benachbarten Extrempunkt zu gelangen, müssen wir eine Basisvariable gegen eine Nichtbasisvariable austauschen. Da beide Nichtbasisvariablen ein negatives Vorzeichen in der Zielfunktion haben, wird dieser Austausch, der dazu führt, dass die entsprechende Nichtbasisvariable einen Wert > 0 annimmt, zu einer Verkleinerung des Wertes der Zielfunktion führen. Wir wählen x_2 zur Aufnahme in die Basis

aus. Es bleibt zu untersuchen, welche Variable aus der Basis entlassen werden kann, so dass ein Extrempunkt des zulässigen Bereichs entsteht. Die folgenden drei Fälle können auftreten.

$$\begin{array}{lll}
x_1 = 0,\ x_3 = 0 & \Rightarrow (1) x_2 = 2;\ (2) - 3x_2 \leq 3;\ (3) 3x_2 \leq 12 & \text{zulässig} \\
x_1 = 0,\ x_4 = 0 & \Rightarrow (1) x_2 \leq 2;\ (2) - 3x_2 = 3;\ (3) 3x_2 \leq 12 & \text{nicht zulässig wegen (2)} \\
x_1 = 0,\ x_5 = 0 & \Rightarrow (1) x_2 \leq 2;\ (2) - 3x_2 \leq 3;\ (3) 3x_2 = 12 & \text{nicht zulässig wegen(1)} \wedge \text{(3)}
\end{array}$$

Damit wird x_2 für x_3 in die Basis aufgenommen. Dazu werden die Spalten für x_2 und x_3 ausgetauscht, und anschließend werden die Werte in den letzten drei Zeilen in der Spalte für x_2 zu Null gesetzt, wie bei der Lösung linearer Gleichungssysteme üblich. Dies führt zu folgender Darstellung.

$$\begin{array}{rrrrrcrl}
x_1 & x_3 & x_2 & x_4 & x_5 & & & \\
-1 & 1 & 1 & 0 & 0 & = & 2 & (1') = (1) \\
-1 & 3 & 0 & 1 & 0 & = & 9 & (2') = (2) + 3 \cdot (1) \\
5 & -3 & 0 & 0 & 1 & = & 6 & (3') = (3) - 3 \cdot (1) \\
-8 & 5 & & & & = & -10 & (Z') = (Z) + 5 \cdot (1)
\end{array}$$

Damit lautet die neue Basislösung $(0, 2, 0, 9, 6)^T$ und der Zielfunktionswert ist -10. Für den nächsten Austauschschritt müssen wir x_1 in die Basis aufnehmen, da x_1 einen negativen Koeffizienten in der Zielfunktion hat und damit den Wert der Zielfunktion verkleinert, während x_3 einen positiven Koeffizienten hat und damit den Wert der Zielfunktion vergrößert. Auch hier haben wir wieder drei Möglichkeiten, Variablen aus der Basis zu entfernen, deren Zulässigkeit wir prüfen müssen.

$$\begin{array}{lll}
x_3 = 0,\ x_2 = 0 & \Rightarrow (1) x_1 = -2;\ (2) x_1 \geq -9;\ (3) 5x_1 \leq 6 & \text{nicht zulässig wegen (1)} \\
x_3 = 0,\ x_4 = 0 & \Rightarrow (1) x_1 \geq -2;\ (2) x_1 = -9;\ (3) 5x_1 \leq 6 & \text{nicht zulässig wegen (2)} \\
x_3 = 0,\ x_5 = 0 & \Rightarrow (1) x_1 \geq -2;\ (2) x_1 \geq -9;\ (3) 5x_1 = 6 & \text{zulässig}
\end{array}$$

Damit wird x_5 aus der Basis entlassen und x_1 in die Basis aufgenommen. Die erste und fünfte Spalte werden ausgetauscht und die Werte in der ersten, zweiten und letzten Zeile der fünften Spalte auf Null gesetzt, so dass die folgende Darstellung entsteht.

$$\begin{array}{rrrrrcrl}
x_5 & x_3 & x_2 & x_4 & x_1 & & & \\
1/5 & 2/5 & 1 & 0 & 0 & = & 16/5 & (1'') = (1') + (3')/5 \\
1/5 & 13/5 & 0 & 1 & 0 & = & 51/5 & (2'') = (2') + (3')/5 \\
1/5 & -3/5 & 0 & 0 & 1 & = & 6/5 & (3'') = (3')/5 \\
8/5 & 1/5 & & & & = & -98/5 & (Z'') = (Z') + 8/5 \cdot (3')
\end{array}$$

Da die Koeffizienten der Zielfunktion nun alle positiv sind, kann die Zielfunktion nicht mehr lokal verbessert werden, und wir haben das globale Minimum gefunden. Die Basislösung lautet $(6/5, 16/5, 0, 51/5, 0)^T$ und liefert den Zielfunktionswert $-98/5$.

Das vorgestellte Vorgehen entspricht im Prinzip dem Simplexalgorithmus, muss aber noch etwas formalisiert und platzsparender aufgeschrieben werden.

10.4 Der Simplexalgorithmus

Der Simplexalgorithmus nutzt das so genannte Simplextableau als standardisierte Form der Darstellung, die von zugehörigen Optimierungsprogrammen unterstützt wird. Wir gehen aus von einem Optimierungsproblem

$$\min(f(x)) = \min(c^T x) \text{ udN } Ax = b,\ x \geq 0,\ b \geq 0 \text{ aus.}$$

A ist eine $m \times (n+m)$ Matrix, und die letzten m Variablen sind Schlupfvariablen, die nicht explizit ausgewiesen werden. Die Darstellung lässt sich aber auch nutzen, wenn einzelne Nebenbedingungen als Gleichungen vorliegen und die Variablenzahl dann entsprechend kleiner ist. Wir werden die dann unter Umständen auftretenden Probleme im nächsten Abschnitt behandeln.

Die im vorherigen Abschnitt verwandte Darstellung können wir in ein Tableau bringen. Man beachte dabei, dass nicht b sondern $-b$ gespeichert wird.

x_1	$\cdots$	x_n	x_{n+1}	$\cdots$	$\cdots$	x_{n+m}	$-b$
a_{11}	$\cdots$	a_{1n}	1	0	$\cdots$	0	$-b_1$
$\vdots$		$\vdots$	0	$\ddots$		$\vdots$	$\vdots$
$\vdots$		$\vdots$	$\vdots$		$\ddots$	0	$\vdots$
a_{m1}	$\cdots$	a_{mn}	0	$\cdots$	0	1	$-b_m$
c_1	$\cdots$	c_n	c_{n+1}	$\cdots$	$\cdots$	c_{n+m}	Z

Wenn die letzten m Variablen Schlupfvariablen sind, so sind die zugehörigen Koeffizienten der Zielfunktion natürlich gleich 0. Das Tableau enthält redundante Information, da zu den Basisvariablen immer eine Einheitsmatrix gehört, die wir nicht abspeichern müssen. Es muss allerdings in jeder Zeile die zugehörige Basisvariable gespeichert werden. Dies führt zu einem reduzierten Tableau, auf dem der Simplexalgorithmus basiert.

x_1	$\cdots$	x_n	$-b$	
a_{11}	$\cdots$	a_{1n}	$-b_1$	$-x_{n+1}$
$\vdots$		$\vdots$	$\vdots$	$\vdots$
a_{m1}	$\cdots$	a_{mn}	-b_m	$-x_{n+m}$
c_1	$\cdots$	c_n	Z	

Während des Ablaufs werden nun Variablen der ersten Zeile und der letzten Spalte ausgetauscht. Für unser Beispiel aus dem letzten Abschnitt liegt das folgende Starttableau vor.

x_1	x_2	$-b$	
-1	1	-2	$-x_3$
2	-3	-3	$-x_4$
2	3	-12	$-x_5$
-3	-5	0	

Die Basislösung ist zulässig, da $b_i \geq 0$ gilt. Zur Beschreibung des Algorithmus sei $\mathcal{B}$ die Indexmenge der Basisvariablen, also der Variablen, die in der letzten Spalte stehen. $\mathcal{N}$ sei die Indexmenge der Nichtbasisvariablen, also der Variablen, die an den ersten n Positionen der ersten Zeile stehen. Der Simplexalgorithmus besteht aus den folgenden Schritten:

1. Falls alle $c_i \geq 0$, dann wurde das Optimum gefunden, ansonsten fahre bei 2. fort.
2. Wähle eine Pivotspalte $l = \text{argmin}_{i \in \mathcal{N}}(c_i)$, falls das Minimum nicht eindeutig ist, wähle zufällig eine der Spalten mit minimalen Werten.
3. Bestimme eine Pivotzeile $k = \text{argmin}_{i \in \mathcal{B}}\,(b_i/a_{il} | a_{il} > 0)$. Falls kein $a_{il} > 0$ existiert, so hat das Problem keine Lösung bzw. die Lösung wäre $-\infty$. Falls k nicht eindeutig ist, so wähle zufällig eine der Zeilen aus.
4. Führe folgende Schritte durch, um ein neues Tableau mit Einträgen a'_{ij}, b'_i, c'_j und Z' zu erhalten:
   (a) Vertausche die Indizes k und l in der letzten Spalte und ersten Zeile. Setze die neuen Mengen $\mathcal{B}' = \{l\} \cup \mathcal{B} \setminus \{k\}$ und $\mathcal{N}' = \{k\} \cup \mathcal{N} \setminus \{l\}$
   (b) Für die Werte der Pivotzeile $a'_{kl} = 1/a_{kl}$, $a'_{kj} = a_{kj}/a_{kl}$ und $b'_k = b_k/a_{kl}$ für $j \in \mathcal{N} \setminus \{l\}$.
   (c) Für die restlichen Werte der Pivotspalte $a'_{il} = -a_{il}/a_{kl}$ und $c'_l = -c_l/a_{kl}$ für $i \in \mathcal{B} \setminus \{k\}$.
   (d) Für die restlichen Werte im Tableau $a'_{ij} = a_{ij} - a_{il}a_{kj}/a_{kl}$, $b'_i = b_i - a_{il}b_k/a_{kl}$, $c'_j = c_j - c_l a_{kj}/a_{kl}$ und $Z' = Z + c_l b_k/a_{kl}$.
5. Nutze die neu berechneten Werte im Tableau und fahre bei 1. fort.

In Schritt 2 ist der Wert von $c_l < 0$, sonst wäre in Schritt 1 abgebrochen worden. Die Auswahl des kleinsten Koeffizienten der Zielfunktion als Kandidat für den Basiswechsel ist eine naheliegende Heuristik, es könnte aber ebenso eine andere Spalte mit negativem c-Wert ausgewählt werden. Wenn im Schritt 3 alle $a_{il} < 0$ sind, so gibt es keine Schranke für x_l, und da der Koeffizient der Zielfunktion c_l negativ ist, kann die Situation $x_l \to \infty$ und damit $Z \to -\infty$ erreicht werden. In allen anderen Fällen beschränkt b_k/a_{kl} der Wert der neuen Basisvariable. Interessant ist ferner der Fall $b_k = 0$ in Schritt 3. In diesem Fall ändert sich der Wert der Zielfunktion durch den Basiswechsel nicht und $b'_k = 0$ gilt ebenfalls, wie man sich leicht an den Berechnungsschritten verdeutlichen kann. Wir haben es mit einem degenerierten Extrempunkt zu tun, und der Basiswechsel führt nicht von diesem Extrempunkt weg. Man kann sich nun überlegen, dass durch zyklisches Tauschen der Elemente in der Basis der Algorithmus immer am selben Extrempunkt bleibt und damit nicht terminiert. Durch die zufällige Auswahl einer neuen Basisvariablen ist die Wahrscheinlichkeit eines unendlichen langen Verweilens in einem Extrempunkt allerdings 0. Der Aufwand eines Durchlaufs durch Schritt 4 ist offensichtlich $O(nm)$.

Auf unser Beispiel angewendet würde im ersten Schritt x_2 in die Basis aufgenommen und x_3 aus de Basis entlassen, wie schon im vorherigen Abschnitt beschrieben. Daraus resultiert das folgende Simplextableau.

x_1	x_3	$-b$	
-1	1	-2	$-x_2$
-1	3	-3	$-x_4$
5	-3	-12	$-x_5$
-8	5	-10	

Die Lösung ist noch nicht optimal, da $c_1 < 0$ ist. Der Basiswechsel von x_5 zu x_1 führt zu einem neuen Tableau.

x_5	x_3	$-b$	
0.2	0.4	-3.2	$-x_2$
0.2	2.4	-10.2	$-x_4$
0.2	-0.6	-1.2	$-x_1$
1.6	0.2	-19.6	

Dieses Tableau ist optimal, da alle Koeffizienten der Zielfunktion nichtnegativ sind. Die Lösung entspricht natürlich der im vorherigen Abschnitt ermittelten.

Im nächsten Abschnitt werden wir uns noch einige Sonderfälle anschauen und Korrektheit, Endlichkeit und Aufwand etwas genauer untersuchen.

10.5 Allgemeines Simplexverfahren

Die bisherige Anwendung des Simplexverfahrens beruht auf der Standardform, wie in (10.2) definiert. Andere Formen sind in die Standardform überführbar, wie wir es am Beispiel der Normalform gesehen haben. Darüber hinaus können folgende Modifikationen vorgenommen werden, um ein Problem in Standardform zu bringen.

- Eine Maximierungsaufgabe kann in eine Minimierung transformiert werden, d.h. $\max(f(x)) \to \min(-f(x))$.
- Eine nach unten beschränkte Ungleichung kann in eine von oben beschränkte Ungleichung überführt werden, d.h. $A_{i\bullet}x \geq b_i \to -A_{i\bullet}x \leq -b_i$
- Der Wertebereich der Variablen kann in den folgenden Fällen transformiert werden, indem jeweils zusätzliche Variablen definiert werden, die nicht in der Zielfunktion auftauchen bzw. Koeffizient 0 bekommen.

- Falls $x_i \geq u_i > 0$ gelten muss, so wird eine neue Variable $y_i = x_i - u$ mit der Nebenbedingung $y_i \geq 0$ eingeführt, wodurch automatisch auch $x_i \geq u_i$ sichergestellt ist.
- Falls $x_i \leq o_i$ gilt, wird eine neue Variable $y_i = o_i - x_i$ eingeführt.
- Falls x_i unbeschränkt ist, so kann man x_i durch zwei Variablen y_i und z_i ersetzen, so dass $x_i = y_i - z_i$ und $y_i, z_i \geq 0$.[1]

Mit allen diesen Änderungen lässt sich aber nicht $b \geq 0$ in der Standardform erzwingen. Wenn $b_j < 0$ gilt oder wenn schon in der Normalform einige Ungleichungen als Gleichungen vorliegen, so dass keine Schlupfvariablen für diese Gleichungen notwendig sind, treten offensichtlich Schwierigkeiten bei der Aufstellung des initialen Tableaus auf. Im initialen Tableau wurden bisher die ursprünglich vorhandenen Variablen auf 0 gesetzt und die Werte der Schlupfvariablen so gewählt, dass die Gleichungen gelten. Wenn nun $b_j < 0$ ist, so kann die zugehörige Schlupfvariable nicht auf b_j gesetzt werden, da dann die Nebenbedingung $x_j \geq 0$ verletzt wäre. Es muss also eine initiale Basislösung gefunden werden. Dies geschieht durch Einführung einer zusätzlichen Vorphase, deren Aufgabe es ist, ausgehend von einem nicht zum zulässigen Bereich gehörenden Schnittpunkt der begrenzenden Hyperebenen, einen Extrempunkt des zulässigen Bereichs als Startpunkt des Simplexalgorithmus zu finden oder festzustellen, dass der zulässige Bereich leer ist. Interessanterweise kann auch für diesen Schritt der Simplexalgorithmus, in leicht veränderter Form, angewandt werden.

Das Vorgehen ist sehr ähnlich zum bereits beschriebenen Algorithmus. In jedem Schritt wird ausgehend von einem Schnittpunkt von n Hyperebenen eine Hyperebene weggelassen und eine neue hinzugenommen. Dabei wird nicht auf eine Verbesserung des Zielfunktionswertes geachtet, sondern schlechte Zeilen $b_j < 0$ sollen verbessert (d.h. $b'_j > b_j$) oder beseitigt bzw. konsolidiert (d.h. $b'_j \geq 0$) werden.

Das Vorgehen soll am folgenden Beispiel verdeutlicht werden.

$$f(x) = c^T x = (-5, -2) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \text{ udN } Ax = \begin{pmatrix} -3 & -1 \\ -2 & -3 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} -3 \\ -6 \\ 4 \end{pmatrix} = b$$

Das Problem kann im folgenden Tableau dargestellt werden.

x_1	x_2	$-b$	
-3	-1	3	$-x_3$
-2	-3	6	$-x_4$
2	1	-4	$-x_5$
-5	-2	0	

Der zum Start gewählte Extrempunkt gehört nicht zum zulässigen Bereich, da der erste und zweite Wert im Vektor b kleiner 0 sind und damit x_3 und x_4 kleiner 0 würden. Im Allgemeinen haben wir eine Menge $G = \{i | i = 1, \ldots, m; b_i \geq 0\}$ von guten Zeilen und eine Menge $S = \{i | i = 1, \ldots, m; b_i < 0\}$ von schlechten Zeilen. Ziel ist es, einen Extrempunkt zu suchen, bei dem $S = \emptyset$ gilt. Dazu konzentrieren wir uns jeweils auf eine schlechte Zeile $s \in S$, die so lange modifiziert wird, bis sie gut ist. Dies geschieht mit dem folgenden modifizierten Simplexschritt.

1. Wähle $s = \max_{s' \in S} \{s'\}$
2. Überprüfe auf Unlösbarkeit, d.h. der zulässige Bereich ist leer, falls $a_{s1}, \ldots, a_{sn} > 0$, da in diesem Fall die Gleichung nur mit negativen Werten im Vektor x erfüllbar ist.
3. Suche nach einer Pivotspalte l zur Aufnahme in die Basis.
   Da der Wert von b_s vergrößert wird, wenn j mit $a_{sj} < 0$ in die Basis aufgenommen wird, wählen wir willkürlich l aus $\{j | j = 1, \ldots, n; a_{sj} < 0\}$.

[1] Bei r nicht beschränkten Variablen reicht es aus, $r+1$ neue Variablen einzuführen, wie in Neumann and Morlock [1993, Kap. 1.5] gezeigt wird.

4. Suche nach einer Pivotzeile k zur Entlassung aus der Basis.
   Die Zeilen aus G sollen gut bleiben. Falls eine Zeile i mit $a_{il} > 0$ gewählt wird, so wird b_i/a_{il} von den anderen Elementen des Vektors b abgezogen. Wähle also $k = \text{argmin}_{i \in G} \{b_i/a_{il} | a_{il} > 0\}$, falls die Menge leer ist, setze $k = s$.

5. Führe Schritt 4 aus dem Simplexalgorithmus (auf Seite 163) durch.

Satz 16 *Falls $b_k > 0$, d.h. der gewählte Extrempunkt nicht degeneriert ist, so wird Zeile s besser (d.h. $b'_s > b_s$), und alle Zeilen $i \in G$ sind auch nach Ausführung des Schritts in G.*

Beweis: Wir unterschieden zwischen $s \neq k$ und $s = k$.
Sei zuerst $s \neq k$:
Für $i \in G$ und $i \neq k$ gilt: $b'_i = b_i - a_{il}b_k/a_{kl} \geq b_i - a_{il}b_i/a_{il} > 0$, da $b_i \geq 0$ und $b_k/a_{kl} \leq b_i/a_{il}$. Damit bleibt Zeile i gut.
Für k ($k \in G$ folgt aus der Auswahl und $s \neq k$) gilt: $b'_k = b_k/a_{kl} \geq 0$, da $b_k \geq 0$ und $a_{kl} > 0$. Damit bleibt auch k in G.
Für Zeile s gilt: $b'_s = b_s - a_{sl}b_k/a_{kl} \geq b_s$, da $a_{sl} < 0$, $b_k \geq 0$ und $a_{kl} > 0$. Falls der Extrempunkt nicht degeneriert ist, gilt sogar $b_k > 0$, so dass $b_s$verbessert (d.h. vergößert) wird.
Sei nun $s = k$:
Für $i \in G$ gilt: $b'_i = b_i - a_{il}b_s/a_{sl} \geq b_i$, da $b_i \geq 0$, gilt $b_s, a_{sl} < 0$, sonst wäre $i = k$. Damit bleibt Zeile i in G.
Für s gilt: $b'_s = b_s/a_{sl} > 0$, da $b_s, a_{sl} < 0$. Damit wird s eine gute Zeile, da $b'_s > 0$ gilt.

□

Es werden keine Aussagen über die Werte anderer *schlechter* Zeilen gemacht, diese können sich auch weiter verschlechtern. Da aber die ausgewählte schlechte Zeile s verbessert, sofern der Austauschschritt nicht in einem degenerierten Eckpunkt erfolgt, wird s nach endlich vielen Schritten entweder gut oder die Unlösbarkeit wird festgestellt. Da eine gute Zeile durch Schritte des Algorithmus nicht schlecht wird, können alle schlechten Zeilen beseitigt werden, sofern der zulässige Bereich nicht leer ist.

Der Austauschschritt soll nun auf das Beispiel angewendet werden. Nach obigem Vorgehen erhalten wir Pivotspalte $l = 1$, Pivotzeile $k = 3$ und $s = 2$. Es entsteht folgendes Tableau:

x_2	x_5	$-b$		
1/2	3/2	−3	$-x_3$	$(1') = (1) + 3/2(3)$
-2	1	2	$-x_4$	$(2') = (2) + (3)$
1/2	1/2	−2	$-x_1$	$(3') = 1/2(3)$
1/2	5/2	−10		$(z') = (z) + 5/2(3)$

Interessanterweise haben wir durch die Verbesserung von Zeile 2 die erste Zeile auch gleich verbessert, so dass nun nur noch Zeile 2 schlecht ist. Der erreichte Zielfunktionswert ist natürlich nicht gültig, da der Punkt wegen Gleichung 2 immer noch außerhalb des zulässigen Bereichs liegt. Im nächsten Schritt ist $l = 1$, $k = 3$ und $s = 2$. Dieser Schritt liefert folgendes Tableau:

x_1	x_5	$-b$		
−1	1	−1	$-x_3$	$(1'') = (1') - (3')$
4	3	−6	$-x_4$	$(2'') = (2') + 4(3)$
2	1	−4	$-x_2$	$(3'') = 2(3')$
−1	2	−8		$(Z'') = (Z') - (3')$

Damit ist eine zulässige Basislösung erreicht. Der Zielfunktionswert von -8 ist noch nicht optimal, da $c_1 < 0$ und durch den Austausch der Basis-/Nichtbasis-Variablen x_1 und x_4 ($l = 1$, $k = 2$) eine Verbesserung der Zielfunktion erreicht wird.

x_4	x_5	$-b$		
$1/4$	$7/4$	$-5/2$	$-x_3$	$(1''') = (1'') - (2'')/4$
1	$3/4$	$-3/2$	$-x_1$	$(2''') = (2'')/4$
$-1/2$	$-1/2$	-1	$-x_2$	$(3''') = (3'') - (2'')/2$
$1/4$	$11/4$	$-19/2$		$(Z''') = (Z'') + (2'')/4$

Da alle c_i nun nichtnegativ sind, ist $(1.5, 1, 2.5, 0, 0)^T$ eine optimale Lösung mit Zielfunktionswert -9.5.

Damit haben wir einen Zweiphasenalgorithmus, und die Bedingung $b \geq 0$ kann fallengelassen werden. In der ersten Phase wird ein Extrempunkt des zulässigen Bereichs gesucht, von dem aus dann in der zweiten Phase das Optimum gesucht wird. Die folgenden Sätze fassen unsere Ergebnisse über den Simplexalgorithmus zusammen.

Satz 17 (Korrektheit des Simplexalgorithmus) *Der Simplexalgorithmus ist partiell korrekt, d.h. er liefert ein korrektes Resultat, wenn er terminiert.*

Beweis: Aus der Entwicklung des Algorithmus wissen wir:
Die Vorphase des Algorithmus endet entweder mit der Feststellung der Unlösbarkeit oder mit einer zulässigen Basislösung. Ausgehend von einer Basislösung endet die Hauptphase entweder mit der Feststellung der Unbeschränktheit oder der optimalen Basislösung.

□

Satz 18 (Endlichkeit bei nicht degenerierten Problemen) *Der Simplexalgorithmus ist für nicht degenerierte Probleme endlich.*

Beweis: Für nicht degenerierte Probleme wird in der Vorphase in jedem Schritt ein Wert im Vektor b verbessert oder konsolidiert (d.h. von negativ zu positiv verändert). Damit kann keine Basislösung mehrfach besucht werden. Da Basislösungen aus dem Schnitt von n aus $n + m$ Hyperebenen entstehen, kann es nur endlich viele geben.
Auch die Hauptphase wird in endlich vielen Schritten überwunden, da in jedem Schritt der Zielfunktionswert verbessert wird und damit keine Basislösung mehrfach besucht wird.

□

Satz 19 (Aufwand des Simplexalgorithmus) *Der zeitliche Aufwand eines Schrittes des Simplexalgorithmus ist $O(nm)$.*

Die Zahl der Schritte ist im schlechtesten Fall bei nicht degenerierten Problemen $\binom{n+m}{n}$ und ist damit exponentiell in n und m.

Beweis: Je Schritt sind alle Einträge im Simplex-Tableau zu ändern. Es gibt $O(nm)$ Einträge.
Bei nicht degenerierten Problemen werden maximal alle möglichen Basisvektoren (Schnittebenen von n Hyperebenen) besucht. Es müssen damit n aus $n + m$ mögliche Schnittmengen ausgewählt werden, wofür es die angegebene Anzahl Möglichkeiten gibt.

□

Der letzte Satz ist ernüchternd, da er zeigt, dass die Laufzeit schon für relativ kleine Probleme immens sein kann. In der Praxis ist der Simplexalgorithmus aber einer der wenigen exponentiellen Algorithmen, der sich für praktisch alle relevanten Probleme gutmütig verhält. Die Laufzeit ist oft linear in n und m. Damit ist der Simplexalgorithmus meistens schneller als bekannte alternative Algorithmen mit deutlich besserer "worst case" Komplexität (siehe auch Abschnitt 10.9). Um das exponentielle Laufzeitverhalten zu beobachten, muss man mühsam konstruierte "worst case" Beispiele verwenden.

Wir betrachten zum Abschluss des Abschnitts noch einmal die optimale Lösung des LP-Problems als optimale Basislösung eines linearen Gleichungssystems. Diese Darstellung werden wir in den folgenden Abschnitten für einige Beweise nutzen.

Sei $\min c^T x$ udN $Ax = b, x \geq 0$ das LP-Problem mit m Gleichungen und $m+n$ Variablen. Zur Vereinfachung nehmen wir an, dass die Basis der optimalen Lösung aus den Variablen $x_{n+1}, \ldots, x_{n+m}$ besteht. Bei bekannter Lösung kann man diese Darstellung immer durch Variablenumordnung erreichen. $A = (N, B)$ wobei B eine nicht singuläre $m \times m$ Basismatrix ist. Sei ferner $M = B^{-1}(N, B) = \left(B^{-1}N, I\right)$, $x^* = \begin{pmatrix} x_N^* \\ x_B^* \end{pmatrix} = \begin{pmatrix} 0 \\ x_B^* \end{pmatrix}$ die optimale Lösung, bei der Nichtbasisvariablen auf den Wert 0 gesetzt wurden, und $c = \begin{pmatrix} c_N \\ c_B \end{pmatrix}$ der Vektor der Zielfunktionskoeffizienten nach Nichtbasis- und Basisvariablen unterteilt. Für den optimalen Zielfunktionswert gilt $Z^* = c^T x^* = c_B^T x_B^*$ und

$$(N, B) \begin{pmatrix} 0 \\ x_B^* \end{pmatrix} = B x_B^* = b \Rightarrow x_B^* = B^{-1} b.$$

Matrix $\Gamma = B^{-1}N$ enthält die Werte in den Zeilen $2, \ldots, m+1$ und den Spalten $1, \ldots, n$ im Simplextableau bei Beendigung des Simplexalgorithmus mit Lösung x^*. Vektor x_B^* beinhaltet die Werte den Zeilen $2, \ldots, m+1$ und Spalte $n+1$ des Simplextableaus. Sei $d^T = c_B^T M = c_B^T B^{-1} A$ und $\zeta = \begin{pmatrix} \zeta_N \\ \zeta_B \end{pmatrix} = c - d$. Der Vektor ζ enthält die modifizierten Koeffizienten der Zielfunktion nach der Berechnung der Basislösung. Offensichtlich gilt $\zeta_B = 0$. Vektor $(\zeta_N)^T$ beinhaltet die Werte in den Spalten $1, \ldots, n$ in der letzten Teile des Simplextableaus. Da x^* optimal ist gilt $\zeta_N \geq 0$.

10.6 Typische Anwendungsbeispiele

Nachdem der formale Rahmen der linearen Optimierung eingeführt wurde, sollen jetzt typische Problemstellungen kurz vorgestellt werden, die teilweise zu speziellen Strukturen in den Matrizen und Vektoren führen, die sich für eine effiziente Analyse nutzen lassen. In der Praxis wurden und werden Probleme mit mehreren 1000 Variablen gelöst. Viele praktische Probleme, die eigentlich nicht linear sind, lassen sich sehr gut durch lineare Probleme approximieren.

Eine Klasse von Problemen sind die so genannten Produktionsplanungsprobleme. Es gibt m Produktionsfaktoren $(R_1, \ldots, R_m)$ mit deren Hilfe n Produkte $(P_1, \ldots, P_n)$ hergestellt werden. Für die Erzeugung einer Einheit von P_j benötigt man a_{ij} Einheiten von R_i. Von Produktionsfaktor R_i sind nur b_i Einheiten vorhanden. Für die Produktion einer Einheit von P_j fallen Kosten k_j an, und es werden Erlöse g_j erzielt. Ziel ist es, einen Produktmix zu finden, der einen möglichst hohen Gewinn erzielt. Dieses Problem kann wie folgt als lineares Optimierungsproblem formalisiert werden.

$$\min f(x_1, \ldots, x_n) = -\sum_{j=1}^{n} (g_j - k_j) x_j \text{ udN } \sum_{j=1}^{n} a_{ji} x_j \leq b_i \ (i = 1, \ldots, m) \text{ und } x_j \geq 0 \ (j = 1, \ldots, n)$$

Dies ist ein klassisches lineares Optimierungsproblem, wie wir es kennengelernt haben.

Eine weitere typische Problemklasse sind Transportprobleme. Probleme dieser Art haben Dantzig zur Entwicklung der Simplexmethode motiviert. Die einfachste Variante des Transportproblems betrachtet nur ein Gut, das an n Orten $(P_1, \ldots, P_n)$ produziert wird und an m Orten $(V_1, \ldots V_m)$ verbraucht wird. Der Transport von P_i nach V_j kostet c_{ij}. In P_i werden p_i Einheiten gelagert bzw. produziert, und in V_j werden v_j Einheiten benötigt. Ziel ist es, möglichst kostengünstig, jeden Zielort mit genügend Einheiten des produzierten Gutes zu versorgen. Sei x_{ij} die Menge, die von P_i nach V_j gebracht wird. Damit die Lösung zulässig ist, dürfen nur so viele Güter von P_i genommen werden, wie dort vorhanden sind, d.h. $\sum_{j=1}^{m} x_{ij} \leq p_i$ für alle $i = 1, \ldots, n$. Gleichzeitig müssen zu V_j genügend Güter gebracht werden, d.h. $\sum_{i=1}^{n} x_{ij} \geq v_j$ für alle $j = 1, \ldots, m$. Damit hat das Problem nur eine Lösung wenn $\sum_{i=1}^{n} p_i \geq \sum_{j=1}^{m} v_j$, d.h. es wird mindestens so viel produziert, wie verbraucht wird. Wenn mehr produziert als verbraucht wird, so können Mehrmengen im Prinzip im Produktionsort verbleiben. Um dies im Algorithmus zu handhaben, führen wir in

diesem Fall einen zusätzlichen virtuellen Verbraucher V_{m+1} mit $v_{m+1} = \sum_{i=1}^{n} p_i - \sum_{j=1}^{m} v_j$ und Kosten $c_{i,m+1} = 0$ für $i = 1, \ldots, n$ ein. Dieser Verbraucher nimmt die Reste auf. Die auftretenden Gesamtkosten sind $\sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$. Damit liegt das folgende Optimierungsproblem vor.

$$\begin{aligned} &\min_{x_{ij}} \left(\sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} \right) && \text{udN } x_{ij} \geq 0 \text{ und} \\ &\sum_{j=1}^{m} x_{ij} = p_i && \text{für } 1 \leq i \leq n \\ &\sum_{i=1}^{n} x_{ij} = v_i && \text{für } 1 \leq j \leq m \end{aligned}$$

Die Darstellung ist äquivalent zur Standardform (10.2). Die Zielfunktion umfasst $n \cdot m$ Variablen, und A ist eine $n + m \times n \cdot m$ Matrix. Im Gegensatz zu den bisherigen Modellen enthält Matrix A nur Elemente gleich 0 oder 1. Insgesamt sind nur $2nm$ der $(n + m)nm$ Elemente der Matrix ungleich 0. Für größere Werte von n und m ist die Matrix damit spärlich besetzt. Diese spezifischen Eigenschaften des Problems können in einer speziellen Form des Simplexalgorithmus genutzt werden.

Weitere Beispiele findet man unter anderem in der netlib (*http://www.numerical.rl.ac.uk/cute/netlib.html*). Es gibt zahlreiche freie und kommerzielle Implementierungen von Lösungsmethoden, die auf dem Simplexalgorithmus basieren. Eine Übersicht ist unter *http://lionhrtpub.com/orms/surveys/LP/LP-survey.html* zu finden.

10.7 Dualität

Die Idee der Dualität in der linearen Optimierung soll zuerst an einem einfachen Beispiel erläutert werden, bevor wir mit der theoretischen Herleitung beginnen. Dazu betrachten wir das folgende Beispiel aus Domschke and Drexl [1993].

$$\begin{aligned} \max \quad & f(x) = 6x_1 + 4x_2 \\ \text{udN} \quad & x_1 + 2x_2 \leq 8 \\ & 3x_1 + x_2 \leq 9 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Mit Hilfe der Nebenbedingungen lassen sich obere Schranken für den Wert der Zielfunktion bestimmen. Wenn wir zum Beispiel die erste Nebenbedingung mit 6 multiplizieren, so gilt $6x_1 + 4x_2 \leq 6x_1 + 12x_2 \leq 48$, so dass 48 eine obere Schranke für den Wert der Zielfunktion bildet. Ähnlich erhält man durch Multiplikation der zweiten Nebenbedingung mit 4 die Ungleichung $6x_1 + 4x_2 \leq 12x_1 + 4x_2 \leq 36$. Diese Idee kann man weiterführen indem wir Linearkombinationen der Nebenbedingungen betrachten.

$$f(x) = 6x_1 + 4x_2 \leq y_1 (x_1 + 2x_2) + y_2 (3x_1 + x_2) = x_1 (y_1 + 3y_2) + x_2 (2y_1 + y_2) \leq 8y_1 + 9y_2$$

Damit die Ungleichung gilt, müssen die Koeffizienten der Zielfunktion erreicht werden, was zu folgenden Nebenbedingungen bei der Wahl von y_1 und y_2 führt.

$$y_1 + 3y_2 \geq 6 \text{ und } 2y_1 + y_2 \geq 4$$

Da Linearkombinationen von Ungleichungen gebildet werden, dürfen wir nur Koeffizienten mit gleichen Vorzeichen verwenden. Wir fordern deshalb $y_1, y_2 \geq 0$. Zur Bestimmung der kleinst möglichen Werte von y_1, y_2 muss das Optimierungsproblem $\min (8y_1 + 9y_2)$ unter den obigen Nebenbedingungen gelöst werden. Die kleinst möglichen Werte liefern gleichzeitig die kleinste obere Schranke für den Wert der ursprünglichen Zielfunktion. Zusammenfassend haben wir damit die folgenden

beiden Probleme vorliegen:

$$\begin{array}{lll|lll} \max & f(x) = 6x_1 + 4x_2 & & \min & g(y) = 8y_1 + 9y_2 \\ \text{udN} & x_1 + 2x_2 \leq 8 & & \text{udN} & y_1 + 3y_2 \geq 6 \\ & 3x_1 + x_2 \leq 9 & & & 32y + y_2 \geq 4 \\ & x_1, x_2 \geq 0 & & & y_1, y_2 \geq 0 \end{array}$$

Wir sprechen von einem dualen Problem. Sei P' das primale Problem und $\bar{P}'$ das zugehörige duale Problem. Es gilt dann

Primales Problem P'	Duales Problem $\bar{P}'$
$\min f(x) = c^T x$	$\max g(y) = b^T y$
udN $Ax \geq b$	udN $A^T y \leq c$
$x \geq 0, x \in \mathbb{R}^n$	$y \geq 0, y \in \mathbb{R}^m$

Man bezeichnet x als Primalvariablen und y als Dualvariablen. Wenn man von Gleichungen statt Ungleichungen bei den Nebenbedingungen ausgeht, so besteht folgender Zusammenhang zwischen primalem und dualem Problem.

Primales Problem P	Duales Problem $\bar{P}$
$\min f(x) = c^T x$	$\max g(y) = b^T y$
udN $Ax = b$	udN $A^T y \leq c$
$x \geq 0, x \in \mathbb{R}^n$	$y \in \mathbb{R}^m$

Dabei ist zu beachten, dass y in diesem Fall nicht vorzeichenbeschränkt ist. In beiden Fällen gilt, dass die zweifache Anwendung der Dualität wieder auf das primale Modell führt, d.h. $P = \bar{\bar{P}}$. Allgemein besteht folgender Zusammenhang zwischen den einzelnen Teilen des primalen und dualen Problems.

Primales Problem	Duales Problem
Minimierungsproblem	Maximierungsproblem
Variable	Echte Nebenbedingung
Vorzeichenbeschränkte Variable	Ungleichung ($\geq$ bei max, $\leq$ bei min)
Nicht vorzeichenbeschränkte Variable	Gleichung
Koeffizienten der Zielfunktion	Rechte Seite der Nebenbedingungen
Koeffizientenmatrix	Transponierte Koeffizientenmatrix

Der folgende Satz stellt einen interessanten Zusammenhang zwischen den Lösungen des primalen und dualen Problems her.

Satz 20 *Sei x eine zulässige Lösung des primalen Problems P und y eine zulässige Lösung des dualen Problems $\bar{P}$, dann gilt $f(x) \geq g(y)$.*

Beweis:
Da y zulässig ist, gilt $A^T y \leq c \Leftrightarrow c^T \geq y^T A$. Ferner gilt, da auch x zulässig ist, $x \geq 0$ und $Ax = b$. Damit gilt auch

$$f(x) = c^T x \geq y^T A x = y^T b = g(y).$$

□

Voraussetzung für den vorherigen Satz ist, dass beide Probleme zulässige Lösungen haben. Der zulässige Bereich von $\bar{P}$ leer, wenn $f(x)$ auf dem zulässigen Bereich von P unbeschränkt ist. Andererseits ist der zulässige Bereich von P leer, wenn $g(y)$ auf dem zulässigen Bereich von $\bar{P}$ unbeschränkt ist. Damit kann man folgern, dass wenn x eine Lösung von P ist, y eine Lösung von $\bar{P}$ ist und $f(x) = g(y)$, dann sind beide Lösungen für die jeweiligen Optimierungsprobleme optimal. Der folgende Satz der linearen Optimierung wird auch als starkes Dualitätstheorem bezeichnet.

Satz 21 *Wenn das primale Problem P und das duale Problem $\bar{P}$ zulässige Lösungen besitzen, so besitzen die Probleme auch optimale Lösungen und es gilt*

$$\min\left\{c^T x | Ax = b, x \geq 0\right\} = \max\left\{b^T y | A^T y \leq c\right\}.$$

Beweis:
Wir skizzieren den Beweis und nehmen an, dass beide Probleme zulässige Lösungen besitzen. Wir nehmen an, dass $x^* = \begin{pmatrix} 0 \\ x_B^* \end{pmatrix}$ mit $x_B^* = B^{-1}b$ die optimale Lösung von P ist. Sei nun $y^* = \left(B^{-1}\right)^T c_B$. Wir müssen zeigen, dass y^*eine Optimallösung von $\bar{P}$ ist und $f(x^*) = g(y^*)$. Betrachten wir zuerst die Zulässigkeit von y^*. Es gilt

$$\left(y^*\right)^T A = c_B^T B^{-1} A = c_B^T M \leq c^T \Rightarrow A^T y^* \leq c.$$

Die Gültigkeit de Ungleichung $c_B^T M \leq c^T$ folgt aus der Optimalität von x^* (siehe auch die Herleitung am Ende von Abschnitt 10.5)
Die Optimalität folgt aus

$$b^T y^* = \left(y^*\right)^T b = c_B^T B^{-1} b = c_B^T x_B^* = c^T x^*$$

Da $c^T x^*$optimal für P ist und $b^T y^*$ eine Lösung mit identischem Zielfunktionswert ist, kann die obige Folgerung angewendet werden und beide Lösungen sind optimal.

□

Wenn Z^* der optimale Zielfunktionswert ist, so gilt damit $g(y) \leq Z^* \leq f(x)$ für beliebige zulässige Lösungen. Es tritt immer einer der folgenden vier Fälle ein:

1. Beide Probleme haben Lösungen und die optimalen Lösungen haben den gleichen Zielfunktionswert.
2. Die Zielfunktion von P ist nach unten unbeschränkt und der zulässige Bereich von $\bar{P}$ ist leer.
3. Die Zielfunktion von $\bar{P}$ ist nach oben unbeschränkt und der zulässige Bereich von P ist leer.
4. Die zulässigen Bereiche der Probleme P und $\bar{P}$ sind leer.

Betrachten wir nun das primale Problem P' und das zugehörige duale Problem $\bar{P}'$. Durch Erweiterung der Vektoren um Schlupfvariablen gilt

$$\begin{array}{ll} x^T = (x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+m}) & \text{der Vektor des primalen Problems,} \\ y^T = (y, \ldots, y_m, y_{m+1}, \ldots, y_{n+m}) & \text{der Vektor des dualen Problems.} \end{array}$$

Man kann zeigen, ohne dass wir es tun werden, dass für zulässige optimale Lösungen x^* und y^* gilt

$$\begin{array}{ll} x_i^* \cdot y_{m+1}^* = 0 & \text{für alle } i = 1, \ldots, n \\ y_j^* \cdot x_{n+j}^* = 0 & \text{für alle } j = 1, \ldots, m \end{array}$$

Diesen Aussage bezeichnet man auch als Satz vom komplementären Schlupf. Eine Ungleichung ist ohne Schlupf erfüllt, wenn der Wert der zugehörigen Schlupfvariablen 0 ist. Der obige Zusammenhang zeigt, dass $x_i^* \neq 0$ bedeutet, dass die i-te Nebenbedingung für die optimale Lösung des dualen Problems ohne Schlupf erfüllt ist. Ebenso bedeutet $y_j^* \neq 0$, dass die j-te Nebenbedingung für die optimale Lösung des primalen Problems ohne Schlupf erfüllt ist.

Man kann das Simplexverfahren in primaler und dualer Form realisiert werden kann und die jeweils passendste Variante für die Lösung eines Problems verwendet werden kann. Wir diesen Punkt aber nicht weiter vertiefen und es kurz noch der Interpretation der Dualität in einem spezifischen Umfeld widmen.

Dazu betrachten wir das Produktionsproblem, das bereits im vorherigen Abschnitt vorgestellt wurde. Folgende Interpretation der Primal- und Dualvariablen liegt hier vor.

Primales Problem P'	Duales Problem $\bar{P}'$
$\min f(x) = c^T x$	$\max g(y) = b^T y$
udN $Ax \leq b$	udN $A^T y \geq c$
$x \geq 0, x \in \mathbb{R}^n$	$y \geq 0, y \in \mathbb{R}^m$
$c_i = -(g_i - k_i)$ (Gewinn - Kosten)	y_j Schattenpreis Faktor j
b_j Menge Faktor j	
x_i Produktionsmenge i	

Schattenpreise oder auch Knappheitskosten sind in der Ökonomie ein weiterverbreitetes Maß. Sie drücken aus, welchen Gewinn die Erhöhung einer Faktormenge bringen würde. Falls die zu einem Faktor gehörende Ungleichung mit Schlupf erfüllt ist, so wird nicht die komplette Menge dieses Faktors benötigt und der Schatten preis ist 0. Falls die Formelmenge voll ausgeschöpft wird, so würde ihre Erhöhung einen zusätzlichen Gewinn bringen, der dem Schattenpreis entspricht.

10.8 Sensitivitätsanalyse

Bisher sind wir davon ausgegangen, dass alle Daten des Modells exakt bekannt sind und das Optimum auf Basis dieser Informationen bestimmt wird. In der Praxis ist dies nicht unbedingt der Fall. So können Erlöse oder Kosten in der Regel nicht exakt über einen längeren Zeitraum vorausgesagt werden, sondern schwanken in gewissen Grenzen. Ähnlich sieht es bei der Verfügbarkeit oder auch dem Verbrauch von Ressourcen aus. Beide Größen sind oft nicht exakt quantifizierbar und werden auf Basis vorhandener Daten geschätzt. Man könnte die nicht exakt bekannten Größen durch Zufallsvariablen modellieren und so ein stochastisches Modell erzeugen. Dies würde zu neuen Fragestellungen und auch deutlich schwierigeren Optimierungsproblemen führen, die wir hier nicht behandeln werden. Statt dessen wollen wir analysieren, wie sich das Optimum bei kleinen Änderungen der Parameter verändert.

Formal betrachten wir folgendes Problem

$$\begin{aligned} &\min \left(c + \Delta_c\right)^T x \\ &\text{udN } \left(A + \Delta_A\right) x = \left(b + \Delta_b\right),\ x \geq 0 \end{aligned}$$

Die Werte Δ_c, Δ_A und Δ_b beschreiben die Änderungen in den einzelnen Parametern. Man kann potenziell zwei unterschiedliche Effekt beobachten:

1. Die Lösung bleibt qualitativ identisch (d.h. die optimale Lösung liegt im selben Eckpunkt des zulässigen Bereichs) und ändert sich nur quantitativ.

2. Die Lösung ändert sich auch qualitativ (d.h. das Optimum liegt in einem neuen Eckpunkt).

Fall 1 wird in der Sensitivitätsanalyse behandelt, während Fall 2 in der parametrischen Analyse behandelt wird. Wir beschäftigen uns nur mit der Sensitivitätsanalyse.

Das Vorgehen bei der Sensitivitätsanalyse wird an dem folgenden Beispiel aus Neumann and Morlock erläutert.

$$\begin{aligned} \max f(x) = \max \left(x_1 + x_2\right) \\ \text{udN} \quad -x_1 + x_2 \leq 2 \\ x_1 + 2x_2 \leq 6 \\ 2x_1 + x_2 \leq 6 \\ x_1, x_2 \geq 0 \end{aligned}$$

In Abbildung 10.4 sieht man die Struktur des zweidimensionalen Optimierungsproblems. Der optimale Punkt lautet $(2, 2)$ mit dem optimalen (hier maximalen) Zielfunktionswert $Z^* = 4$.

Das erweiterte Simplextableau nach der Berechnung des Optimum hat folgendes Struktur.

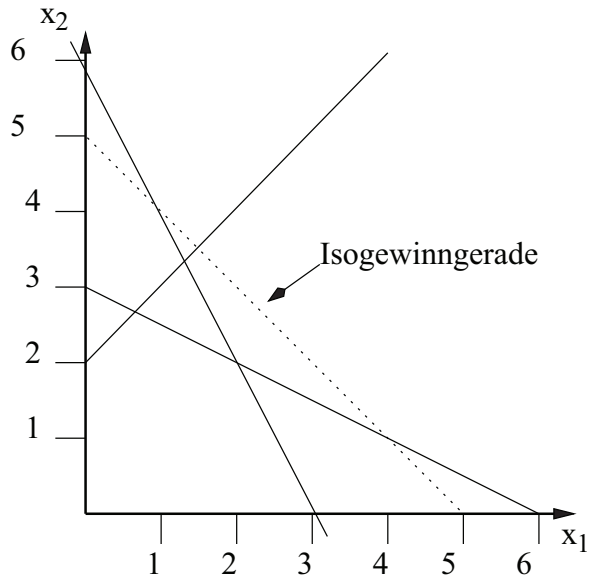


Abbildung 10.4: Struktur des Beispielmodells zur Sensitivitätsanalyse.

x_1	$\cdots$	x_n	x_{n+1}	$\cdots$	$\cdots$	x_{n+m}	$-b$
M_{11}	$\cdots$	M_{1n}	1	0	$\cdots$	0	$-x^*_{n+1}$
$\vdots$		$\vdots$	0	$\ddots$		$\vdots$	$\vdots$
$\vdots$		$\vdots$	$\vdots$		$\ddots$	0	$\vdots$
M_{m1}	$\cdots$	M	0	$\cdots$	0	1	$-x^*_{n+m}$
ζ_1	$\cdots$	ζ_n	0	$\cdots$	$\cdots$	0	Z^*

Mit Hilfe der Werte im optimalen Tableau kann analysiert werden, welche Auswirkungen Änderungen in den Parametern hervorrufen.

Wir beginnen mit Änderungen in den Koeffizienten der Zielfunktion. Statt c wird der Vektor $c + \Delta_c$ in der Zielfunktion verwendet. Die Änderung eines Zielfunktionskoeffizienten führt im zweidimensionalen zu einer Drehung der Isogewinn-Geraden. Wenn diese Drehung zu stark wird, verlässt die optimale Lösung den bisherigen optimalen Punkt. Im Beispiel aus Abbildung 10.4 würde bei einer zu starken Drehung der Isogewinn-Geraden das Optimum nicht mehr im Punkt $(2, 2)$ sondern einer der Punkte $\left(\frac{2}{3}, \frac{8}{3}\right)$ oder $(3, 0)$ optimal. Falls der Eckpunkt nicht verlassen wird, so gilt für den optimalen Zielfunktionswert nach Änderung

$$f^{\Delta_c}(x^*) = (c + \Delta_c)^T x^*$$

Damit x^* Minimalpunkt bleibt, müssen die Koeffizienten ζ_j für alle Nichtbasisvariablen nichtnegativ bleiben. Einsetzen der neuen Koeffizienten der Zielfunktion in die Berechnung der ζ_j liefert

$$c_j + \Delta_{c_j} - \sum_{k=1}^{m} M_{kj} \left(c_k + \Delta_{c_k}\right) \geq 0 \Rightarrow \Delta_{c_j} - \sum_{k=1}^{m} M_{kj} \Delta_{c_k} \geq -\zeta_j$$

Änderungen des Vektors der rechten Seite, der die Verfügbarkeit der Ressourcen beschreibt, führen zu einer Parallelverschiebung der zugehörigen Hyperebenen (im zweidimensionalen den zugehörigen Begrenzungsgeraden). Daraus folgt unmittelbar, dass der Punkt des Optimums sich verschiebt, auch wenn der Eckpunkt, in dem das Optimum liegt, qualitativ unverändert bleibt. Sei $x^*\left(\Delta_b\right)$ die Lage des Eckpunkts mit der selben Basis nach Änderung des Vektors b zu $(b + \Delta_b)$. Es gilt $x^*\left(\Delta_b\right) = B^{-1}\left(b + \Delta_b\right)$. Damit $x^*\left(\Delta_b\right)$ zulässig ist muss gelten

$$x^*\left(\Delta_b\right) = B^{-1}\left(b + \Delta_b\right) \geq 0 \Rightarrow B^{-1}\Delta_b \geq -B^{-1}b = -x^*$$

der Zielfunktionswert lautet dann

$$f\left(x^*\left(\Delta_b\right)\right) = c_B^T B^{-1}\left(b+\Delta_b\right) = f(x^*) + c_B^T B^{-1}\Delta_b.$$

Für unser Beispiel bedeutet dies, dass bei einer Änderung von b_3 sich die zwischen den Punkten $(0,3)$ und $(6,0)$ liegende Gerade verschiebt, bei positivem Δ_{b_3} nach rechts und sonst nach links. Für $-2 \leq \Delta_{b_3} \leq 6$ bleibt die Lösung qualitativ erhalten, es gilt dann $f\left(\Delta_{b_3}\right) = 4 + \frac{1}{3}\Delta_{b_3}$.

Zum Abschluss betrachten wir noch Änderungen der Koeffizienten in Matrix A. Da diese Änderungen schwieriger als die beiden bisherigen Fälle zu handhaben sind, beschränken wir uns auf den Fall, dass sich nur Koeffizienten in einer Spalte der Matrix A ändern. Weitergehende Änderungen kann man im Prinzip durch mehrfache Ausführung der folgenden Schritte berücksichtigen. Die Änderung in einer Spalte bedeutet, dass sich die Koeffizienten bzgl. eines Faktors ändern. Dadurch wird eine Drehung der betroffenen Hyperebenen (Begrenzungsgeraden im Zweidimensionalen) hervorgerufen. Sei Δ_j der Änderungsvektor der j-ten Spalte von Matrix A, d.h. aus $A_{\bullet j}$ wird $A_{\bullet j} + \Delta_j$. Δ_j ist ein Spaltenvektor der Länge m. Sei e_j ein Zeilenvektor der Länge $m+n$, der an Position j eine 1 und sonst nur Nullen enthält. Es gilt dann $A\left(\Delta_j\right) = A + \Delta_j e_j$ ist die geänderte Koeffizientenmatrix. Um den Effekt der Änderung zu analysieren müssen wir unterschieden, ob die geänderte Spalte zu einer Basis- oder Nichtbasisvariablen gehört.

Wir beginnen mit dem Fall, dass Spalte j zu einer Basisvariablen gehört (d.h. $n < j \leq n+m$). Damit entsteht eine neue Basismatrix $B\left(\Delta_j\right) = B + \Delta_j e_{j-n}$, die natürlich auch eine andere Inverse hat als die ursprüngliche Matrix B. Da es sich um eine Änderung der ursprünglichen Matrix durch Addition einer Matrix mit Rang 1 handelt, kann die inverse Matrix der geänderten Matrix aus der inversen Matrix der ursprünglichen Matrix mit Hilfe der Sherman-Morrison-Woodbury-Formel [Hager, 1989] bestimmt werden.

$$\left(B\left(\Delta_j\right)\right)^{-1} = B^{-1} - \frac{B^{-1}\Delta_j e_{j-n} B^{-1}}{1 + \Delta_j B^{-1} e_{j-n}}$$

Falls der Nenner des Bruchs 0 wird, so führt die Modifikation zu einer singulären Matrix. Die bisherige Basis liefert eine zulässige Lösung für das modifizierte Problem, wenn $\left(B\left(\Delta_j\right)\right)^{-1} b \geq 0$. Wir definieren $M\left(\Delta_j\right) = \left(\left(B\left(\Delta_j\right)\right)^{-1} N, I\right)$.

Wenn j nicht zur Basis gehört (d.h. $1 \leq j \leq n$), dann gilt $M\left(\Delta_j\right) = \left(B^{-1}\left(N + \Delta_j e_j\right), I\right)$. Sei in beiden Fällen $K\left(\Delta_j\right) = M\left(\Delta_j\right) - M$.

Damit ist Δ_j eine Modifikation, für die die Basis des Optimums unverändert bleibt, wenn

$$\zeta\left(\Delta_j\right)_i = c_i - \sum_{k=1}^{m} M\left(\Delta_j\right)_{ki} c_{n+k} \geq 0 \Rightarrow \zeta_i \geq \sum_{k=1}^{m} K\left(\Delta_j\right)_{ki} c_{n+k}$$

für alle $1 \leq i \leq n$. Falls j eine Spalte aus der Basis ist, muss die inverse Matrix existieren und die ermittelte Lösung muss zulässig sein, wie oben beschrieben.

10.9 Weitere Aspekte der linearen Optimierung

Die lineare Optimierung ist insgesamt ein weites Feld, über das man problemlos eine ganze Vorlesung halten könnte. In diesem Kapitel wurde nur ein erster Einstieg gegeben. Zum Abschluss sollen noch einige Aspekte, die nicht behandelt wurden, kurz angerissen werden. Für weitere Details sei auf die Literatur z.B. Neumann and Morlock [1993, Kap. 1], Jarre and Stoer [2004, Kap. 2-5] oder Bertsimas and Tsitsiklis [1997] verwiesen.

Da die Simplexmethode so breit eingesetzt wird, gibt es natürlich zahlreiche Varianten, die für spezifische Probleme besonders effizient sind. So beobachtet man gerade bei sehr großen Problemen, dass nur sehr wenige Matrixelemente ungleich 0 sind, die Matrix ist also spärlich besetzt. Damit könnte man die Matrix kompakt abspeichern, indem nur die Nichtnullelemente und deren Positionen gespeichert werden. Durch die Austauschschritte werden allerdings einzelne Elemente,

die vorher 0 waren, mit neuen Werten belegt. Dies nennt man auch *fill in*. Für eine möglichst effiziente Realisierung des Algorithmus ist es erstrebenswert, dass möglichst wenig *fill in* erzeugt wird. Für bestimmte Matrixstrukturen kann man durch geschickte Wahl der Pivotelemente den *fill in* reduzieren. und

Da die worst case Komplexität der Simplexmethode sehr schlecht ist, stellt sich natürlich die Frage, inwieweit dies am Problem oder an der Methode liegt. Konkret also, gibt es Algorithmen mit besserer worst case und vielleicht sogar besserer average case Komplexität? Tatsächlich gibt es zwei Ansätze, die Ellipsoid-Methode und die Projektionsmethode von Karmarkar, die eine polynomielle worst case Komplexität haben. Bei der Ellipsoid-Methode wird ausgehend von unzulässigen Lösungen so lange eine Folge von Lösungen konstruiert, bis eine zulässige Lösung erreicht ist. Bei der Methode von Karmarkar startet man im Inneren des zulässigen Bereichs und verbessert iterativ die gefundene Lösung bis die geforderte Genauigkeit erreicht ist. Praktische Untersuchungen zeigen, dass die Methode von Karmarkar ähnliche Laufzeiten wie der Simplexalgorithmus hat, diesen aber in der Regel nicht unterbietet. Deshalb wird in der Praxis meistens der Simplexalgorithmus genutzt.

Kapitel 11

Ganzzahlige und kombinatorische Optimierung

Bisher haben wir uns mit Optimierungsproblemen beschäftigt, bei denen die Parameter reellwertig sind, so dass die optimale Lösung aus einem Kontinuum ausgewählt werden kann. Bei vielen praktischen Problemen sind die Parameter aber aus unterschiedlichen Gründen ganzzahlig. So

- können aus physikalischen Gründen nur ganzzahlige Werte realisierbar sein (z.B. Anzahl der Pufferplätze, Anzahl Maschinen),
- sind nur endlich viele Realisierungen möglich (z.B. Übertragungskabel für ein Rechnernetz, LKWs),
- sind kontinuierliche Werte nur mit beschränkter Genauigkeit einstellbar.

In allen diesen Fällen, ist das Ziel die Bestimmung einer optimalen Lösung mit ganzzahligen Parametern von Interesse. Typische Problemklassen sind u.a.

- Zuordnungsprobleme (z.B. Stundenplanerstellung),
- Reihenfolgeprobleme (z.B. Rundreise, Maschinenbelegung),
- Auswahlprobleme (z.B. Rucksackproblem),
- und Kombinationen davon.

Man spricht von ganzzahliger Optimierung, wenn die Menge der Parameterbelegungen abzählbar ist und von kombinatorischer Optimierung, wenn aus einer endlichen Menge von Alternativen gewählt wird. Anordnungsprobleme sind zum Beispiel typische kombinatorische Optimierungsprobleme.

Formaler sei $x \in \mathbb{Z}^n$ oder auch $x \in \mathbb{Z}_+^n$ der Parametervektor. Zu minimieren ist die Zielfunktion $Z = f(x)$ unter den Nebenbedingungen $h(x) \leq 0$, wobei $h(x) : \mathbb{Z}^n \to \mathbb{R}^m$ ein Vektor von Nebenbedingungen ist. Der zulässige Bereich $W \subseteq \mathbb{Z}^n$ ist definiert als

$$W = \{x | h(x) \leq 0\} \ .$$

Falls $W \subseteq \mathbb{B}^n$, so spricht man auch von einem binärem Optimierungsproblem. Falls $f(x)$ und $h(x)$ lineare Funktionen sind, so spricht man von ganzzahliger (linearer) Programmierung. Gemischte Probleme zeichnen sich durch eine Zweiteilung des Parametervektors aus, so dass $x = (x_D, x_K)$ mit $x_D \in \mathbb{Z}^{n_1}$ und $x_K \in \mathbb{R}^{n_2}$ gilt.

Die meisten ganzzahligen Optimierungsprobleme sind *schwer* zu lösen. So ist die lineare Optimierung mit polynomiellem Aufwand lösbar, während ein ganzzahliges lineares Optimierungsproblem schon NP-schwer ist. Auch wenn dieses Ergebnis nicht gerade ermutigend ist, sind ganzzahlige und kombinatorische Optimierungsprobleme von immenser praktischer Bedeutung und es

gibt unterschiedliche Ansätze, die Lösung handhabbar zu machen. Ansätze gehen von geschickten Zerlegungsansätzen über Schrankenberechnungen bis hin zu verschiedenen Heuristiken, die zwar nicht zwangsläufig das Optimum, aber mit großer Wahrscheinlichkeit einen Wert nahe beim Optimum berechnen. Wir können an dieser Stelle die ganzzahlige und kombinatorische Optimierung nur in Ansätzen behandeln. Dazu werden wir zuerst lineare ganzzahlige Probleme untersuchen und anschließend einige generelle Lösungsansätze betrachten, die insbesondere bei kombinatorischen Optimierungsproblemen mit einer endlichen Menge von Alternativen zum Einsatz kommen. Im abschließenden Abschnitt werden zwei spezielle Klassen von kombinatorischen Optimierungsproblemen, nämlich Rucksackprobleme und Maschinenbelegungs-Probleme, näher erläutert. Die Darstellungen in diesem Kapitel orientieren sich stark an Neumann and Morlock [1993, Kap. 3], weitere Quellen, in denen weitere Resultate über ganzzahlige und kombinatorische Optimierung zu finden sind, sind Korte and Vygen [2002], Nemhauser and Wolsey [1988], Papadimitriou and Steiglitz [1998].

11.1 Ganzzahlige Programmierung

Die einfachste Form der ganzzahligen Optimierung ist die lineare ganzzahlige Programmierung (ILP).

$$\min_{x\in\mathbb{Z}_+^n} f(x) = c^T x \text{ udN } Ax \leq b$$

Es gibt n Entscheidungsvariablen und m Nebenbedingungen. Wir nehmen an, dass $c \in \mathbb{Z}^n$, $b \in \mathbb{Z}^m$ und $A \in \mathbb{Z}^{m,n}$. Die Ganzzahligkeit von c wird nicht unbedingt benötigt, stellt aber auch keine große Einschränkung dar. Wie bei der linearen Optimierung können wir durch Einführung von Schlupfvariablen eine erweiterte Form definieren, so dass

$$\min_{x\in\mathbb{Z}_+^n} f(x) = c^T x \text{ udN } Ax = b$$

und $m < n$ gilt. Es wird dann üblicherweise $rg(A) = m$ gefordert, um redundante Gleichungen zu vermeiden.

Wenn man bei einem ILP auf Ganzzahligkeit verzichtet, so entsteht ein LP. Man spricht dann von einer relaxierten (relaxiert = gelockert/gestrichen) Form des ILP. Sei $x^* \in \mathbb{R}^n$ die Lösung des LPs, das aus dem ILP dadurch entsteht, dass die Bedingung der Ganzzahligkeit fallen gelassen wird. $x' \in \mathbb{Z}^n$ sei die Lösung des ILPs. Es stellt sich die Frage, welcher Zusammenhang zwischen x^* und x' besteht? Abbildung 11.1 zeigt unterschiedliche Situationen, die beim Übergang von reellwertigen zu ganzzahligen Parametern eintreten können. Rechts oben sieht man ein Beispiel, bei dem die Ecken das zulässigen Bereichs teilweise ganzzahlig und teilweise reellwertig sind. Damit kann die Lösung des ILP der Lösung des LP entsprechen, falls diese ganzzahlig ist. Wenn die Lösung des LP aber nicht ganzzahlig ist, so hat das ILP offensichtlich eine andere Lösung. Links unten sehen wir ein Beispiel, bei dem alle Ecken des zulässigen Bereichs ganzzahlig sind, so dass offensichtlich $x^* = x'$ gilt. Rechts unten sehen wir dagegen eine Situation, bei der der zulässige Bereich des ILPs leer ist, während der zulässige Bereich des LPs nicht leer ist.

Wie aus den einfachen Beispielen hervorgeht, lässt sich allgemein x' nicht aus x^* ableiten. Es gilt allerdings $f(x^*) \leq f(x')$, so dass die Lösung des LPs eine untere Schranke bildet. Ein einfacher Lösungsansatz besteht besteht darin, x^* zu bestimmen und anschließend eine ganzzahlige Lösung durch Runden zu erreichen. Falls $A \geq 0$ reicht es aus x_i^* durch $x_i = \lfloor x_i^* \rfloor$ zu ersetzen. $f(x) - f(x^*)$ ist dann eine obere Schranke für den Fehler. Grundsätzlich erreicht man durch Runden nicht das Optimum, in manchen Fällen ist der auftretende Fehler relativ klein, während in anderen Fällen, zum Beispiel bei Booleschen Problemen, Runden zu vollkommen inakzeptablen Resultaten führt.

Wir betrachten das Beispiel

$$\min_{x\in\mathbb{Z}_+} f(x) = \min_{x\in\mathbb{Z}_+} (-2x_1 - 3x_2) \text{ udN } \begin{array}{l} x_1 + x_2 \leq 8 \\ 2x_1 + x_2 \leq 9 \end{array}.$$

Abbildung 11.2 zeigt den zulässigen Bereich und die verschiedenen Lösungen für das Beispielproblem. Das lineare Optimierungsproblem ohne Ganzzahligkeit liefert die Lösung $(10/3, 7/3)$ mit

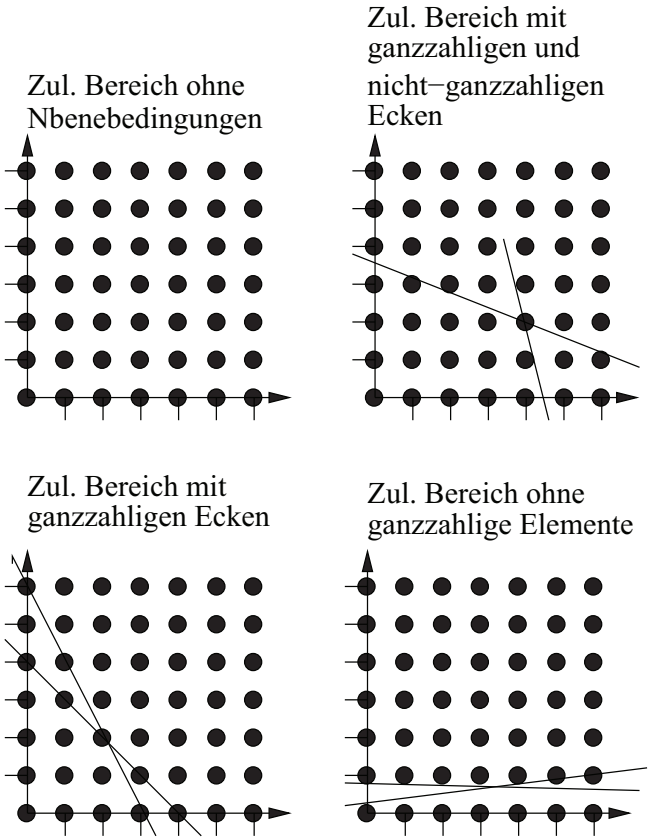


Abbildung 11.1: Zulässige Bereiche für ILPs.

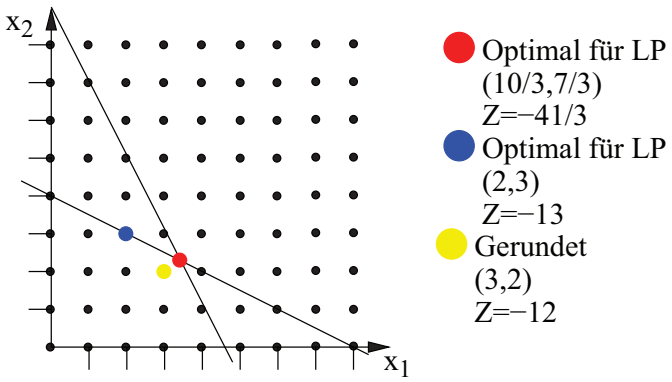


Abbildung 11.2: Zulässiger Bereich und Lösungen für das Beispielproblem.

dem Zielfunktionswert -13.5. Rundung führt zur Lösung $(3, 2)$ mit Zielfunktionswert -12, während die optimale Lösung des ILP $(2, 3)$ mit Zielfunktionswert -13 ist.

Damit tauchen zwei Fragen auf, nämlich wann x^* ganzzahlig ist, so dass $x^* = x'$ gilt und wie man x' in den übrigen Fällen bestimmt. Diese beiden Fragen sollen in diesem Abschnitt beantwortet werden. Wir beginnen mit der ersten.

Definition 7 *Eine Matrix $B \in \mathbb{Z}^{n,n}$ heißt unimodular, falls $|det\ B| = 1$.*

Eine Matrix $A \in \mathbb{Z}^{m,n}$ heißt total unimodular, falls die Determinante jeder quadratischen Untermatrix 0, -1 oder 1 ist (d.h. jede quadratische Untermatrix ist singulär oder unimodular).

Beispiel:
Die Matrix $A = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}$ ist unimodular, da $det\ A = 1 \cdot 1 - 2 \cdot 1 = -1$. Sie ist aber nicht total unimodal, da $det\ A_{12} = 2$. Es folgt, dass eine total unimodale Matrix nur Elemente 0, -1 und 1 enthalten darf. Die Matrix $A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ ist total unimodular, da die beiden 2×2 Untermatrizen $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ und $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ Determinanten von 0 bzw. 1 haben und die einzelnen Elemente gleich 0 oder 1 sind.

Den folgenden Satz benötigen wir für den Nachweis der Ganzzahligkeit von Lösungen bei total unimodularer Matrix A, die wir anschließend betrachten.

Satz 22 *Sei $B \in \mathbb{R}^{n,n}$ eine nicht singuläre Matrix, denn gilt*

$$B^{-1} = \frac{B^{adj}}{det\ B}$$

wobei B^{adj} die adjungierte der Matrix B ist, d.h. $B^{adj}(i, j) = (-1)^{i+j} M_{ij}$ mit M_{ij} der Determinante der Matrix, die aus B durch Streichen der i-ten Zeile und j-ten Spalte entsteht.

Der Beweis des Satzes folgt aus der Cramerschen Regel. Der folgende Satz definiert eine Klasse von ILPs mit ganzzahligen Basislösungen.

Satz 23 *Falls A total unimodular und Vektor b ganzzahlig ist, so hat das ILP $\min c^T x$ udN $Ax = b$ nur ganzzahlige Basislösungen.*

Beweis:
Sei $B = (A_{k\bullet})$ eine Basismatrix und x_B eine Basislösung (der Ordnung m). Es gilt dann $Bx_B = b$ und damit auch $x_B = B^{-1}b$. Da B total unimodular ist, gilt $det\ B = 1$ oder -1 und alle Elemente in B^{adj} sind ganzzahlig. Damit ist bei ganzzahligem b auch x_B ganzzahlig. Dies gilt für alle Basismatrizen, die aus A resultieren, da A total unimodular ist.

□

Der Test, ob eine Matrix unimodular ist, ist zwar mit einem polynomiellen Algorithmus möglich, aber in der Praxis nicht zu empfehlen, da der Aufwand relativ hoch ist. Manche Zuordnungsprobleme führen allerdings zu unimodularen Matrizen. Diese wollen wir aber hier nicht weiter behandeln.

11.1.1 Schnittebenenverfahren für ganzzahlige Programmierungsprobleme

Während wir bisher die Problemstellung formalisiert haben und eine Unterklasse von Problemen kennen gelernt haben, für die ein ILP durch die Lösung des relaxierten LPs gelöst werden kann, soll in diesem Abschnitt ein allgemeines Verfahren zur Lösung von ILPs vorgestellt werden. Es handelt sich um das Schnittebenenverfahren von Gomory [2002], das wir hier nur in seiner einfachsten Variante kennen lernen werden. Das Verfahren besteht aus den folgenden zwei Schritten, die iteriert werden, bis die optimale Lösung des ILPs gefunden wurde.

1. Löse das relaxierte LP mit dem Simplex-Algorithmus.
2. Wenn die ermittelte optimale Lösung nicht ganzzahlig ist, so führe neue Nebenbedingungen ein, die dafür sorgen, dass
   (a) die gefundene optimale (nicht ganzzahlige) Lösung nicht mehr zulässig ist und
   (b) alle ganzzahligen Lösungen zulässig bleiben.

Die zentrale Frage ist nun, wie man Nebenbedingungen findet, die den Anforderungen genügen. Dazu betrachten wir unser Beispiel aus Abbildung 11.2. Das Ausgangstableau und das Endtableau nach zwei Simplex-Schritten sehen wie folgt aus:

Ausgangstableau

x_1	x_2	$-b$	
1	2	-8	x_3
2	1	-9	x_4
-2	-3	0	Z

Endtableau

x_3	x_4	$-b$	
$-1/3$	$2/3$	$-7/3$	x_2
$2/3$	-1/3	$-10/3$	x_1
$1/3$	$4/3$	$41/3$	Z

Die ermittelte Lösung des relaxierten LPs ist nicht ganzzahlig und deshalb keine Lösung für das ILP. Zur Einführung neuer Nebenbedingungen definieren wir zwei Mengen.

$$B := \{i | x_i \text{ Basisvariable}\} \text{ und } N := \{i | x_i \text{ Nichtbasisvariable}\}$$

Bei jeder Lösung des LPs mit Simplexalgorithmus sind die Werte der Nichtbasisvariablen gleich 0. Jede Zeile des Endtableaus liefert eine Gleichung der Form

$$-b_k + \sum_{l \in N} a_{kl} \cdot x_l = -x_k \quad (k \in B)$$

Da $x_l = 0$ für $l \in N$, gilt auch $x_k = b_k$. Sei nun $b_k = \lfloor b_k \rfloor + r_k$ und $a_{kl} = \lfloor a_{kl} \rfloor + r_{kl}$ mit $0 \leq r_k, r_{kl} < 1$. Die vorherige Gleichung kann damit umgeschrieben werden.

$$x_k + \sum_{l \in N} \lfloor a_{kl} \rfloor \cdot x_l - \lfloor b_k \rfloor = -\sum_{l \in N} r_{kl} \cdot x_l + r_k$$

Wenn in diese Gleichung eine ganzzahlige Lösung einsetzen, so ist die linke Seite offensichtlich ganzzahlig und damit auch die rechte Seite. Ferner gilt

$$-\sum_{l \in N} r_{kl} \cdot x_l + r_k \leq r_k$$

für zulässige Lösungen $x_l \geq 0$. Da gleichzeitig $r_k < 1$, muss für ganzzahlige Lösungen

$$-\sum_{l \in N} r_{kl} \cdot x_l + r_k \leq 0$$

gelten. Wir führen nun die folgende neue Nebenbedingung mit einer neuen Variablen x_{n+1} ein.

$$x_{n+1} = -r_k + \sum_{l \in N} r_{kl} \cdot x_l$$

Für ganzzahlige Lösungen ist $x_{n+1} \geq 0$, wie wird gerade hergeleitet haben. Damit werden ganzzahlige Lösungen durch die neue Nebenbedingung nicht beeinflusst. Für x^* gilt $x_{n+1} < 0$, $x_l^* = 0$ für $l \in N$, womit die Summe 0 ist und da x_k^* nicht ganzzahlig ist, muss$r_k > 0$ gelten.

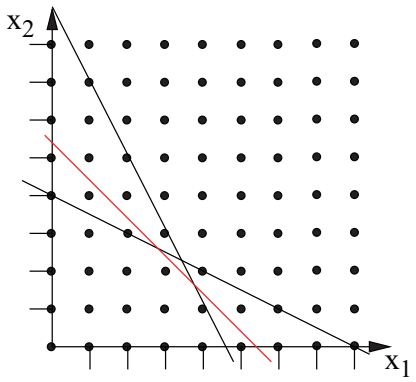


Abbildung 11.3: Zulässiger Bereich des Beispielproblems nach Einführung der neuen Nebenbedingung

Für unser Beispiel wird eine neue Restriktion

$$x_5 = -\frac{1}{3} + \frac{2}{3} \cdot x_3 + \frac{2}{3} \cdot x_4 \text{ und } x_5 \geq 0$$

eingeführt. Da $x_3 = 8 - x_1 - 2 \cdot x_2$ und $x_4 = 9 - 2 \cdot x_1 - x_2$ (siehe Initialtableau) erhalten wir die Nebenbedingung

$$2 \cdot x_1 + 2 \cdot x_2 \leq 11$$

mit der Schlupfvariablen x_5. Die durch die Gleichung $2x_1 + 2x_2 = 11$ definierte Gerade bezeichnet man als Schnittebene.

Abbildung 11.3 zeigt den zulässigen Bereich nach Einführung der neuen Nebenbedingung. Offensichtlich wird die bisherige optimale Lösung des relaxierten LPs abgeschnitten, alle ganzzahligen Lösungen bleiben aber erhalten. Ausgehend von der neuen Nebenbedingung erhalten wir das unten angegebene Starttableau. Die durch das Tableau repräsentierte Basislösung ist nicht zulässig, da $x_5 < 0$. Man kann sogar zeigen, dass dies allgemein für die Einführung von Schnittebenen gilt. Üblicherweise nutzt man deshalb für die Lösung des resultierenden Problems das duale Simplexverfahren Neumann and Morlock [1993, Kap. 1.4] und Abschnitt 10.7. Wir werden deshalb den umständlicheren und damit in der Praxis nicht empfehlenswerten Weg gehen, und zuerst per Simplex-Algorithmus eine neue zulässige Basislösung bestimmen.

Starttableau

x_4	x_3	$-b$	
$-1/3$	$2/3$	$-7/3$	x_2
$2/3$	$-1/3$	$-10/3$	x_1
$-2/3$	$-2/3$	$1/3$	x_5
$1/3$	$4/3$	$-41/3$	Z

Tableau mit zul. Basislsg.

x_1	x_3	$-b$	
$1/2$	$1/2$	-4	x_2
$3/2$	$-1/2$	-5	x_4
1	-1	-33	x_5
$-1/2$	$3/2$	-12	Z

Endtableau

x_5	x_3	$-b$	
$-1/2$	1	$-5/2$	x_2
$-3/2$	1	$-1/2$	x_4
1	-1	-3	x_1
$1/2$	1	-13.5	Z

Nach dem allgemeinen Simplex-Algorithmus wird Pivotzeile 2 und Pivotspalte 1 gewählt. Nach Ausführung des zugehörigen Simplexschrittes entsteht das Tableau in der Mitte. Die zugehörige Basislösung ist zulässig und sogar ganzzahlig, aber leider nicht optimal, da $c_1 = -1/2 < 0$. Ein weiterer Simplexschritt mit Pivotzeile 3 und Pivotspalte 1 liefert eine neue Basislösung $(3, 5/2)$, die optimal aber nicht ganzzahlig ist.

Es wird eine neue Schnittebene eingeführt, um das nicht ganzzahlige Optimum abzuschneiden. Dem vorgestellten Ansatz folgend ergibt sich die Schnittebene

$$x_6 = -\frac{1}{2} + \frac{1}{2} \cdot x_5 \;\Rightarrow\; x_1 + x_2 = 5$$

und das folgende Starttableau:

Starttableau

x_5	x_3	$-b$	
$-1/2$	1	$-5/2$	x_2
$-3/2$	1	$-1/2$	x_4
1	-1	-3	x_1
$-1/2$	0	$1/2$	x_6
$1/2$	1	-13.5	Z

Tableau mit zul. Basislsg.

x_1	x_3	$-b$	
$1/2$	$1/2$	-4	x_2
$3/2$	$-1/2$	-5	x_4
1	-1	-3	x_5
$1/2$	$-1/2$	-1	x_6
$-1/2$	$3/2$	-12	Z

Endtableau

x_6	x_3	$-b$	
-1	1	-3	x_2
-3	1	-2	x_4
-2	0	-2	x_5
2	-1	-2	x_1
1	1	-13	Z

Da x_6 negativ ist, ist die Basislösung nicht zulässig. Ein Schritt des Simplex-Algorithmus mit Pivotzeile 3 und Pivotspalte 1 liefert das Tableau in der Mitte. Die erreichte Basislösung ist zulässig und ganzzahlig, aber nicht optimal. Ein weiterer Simplexschritt mit Pivotzeile 4 und Pivotspalte 1 liefert schließlich das Endtableau mit der optimalen und ganzzahligen Basislösung $(2, 3)$.

Man kann das Verfahren so realisieren, dass es nach endlichen vielen Schritten mit der korrekten Lösung terminiert, falls diese existiert. Die Komplexität bleibt natürlich exponentiell, da schon der Simplex-Algorithmus im *worst case* exponentielle Komplexität hat.

In reiner Form wird das Verfahren nicht angewendet, da es bei größeren Problemen durch die Einführung vieler zusätzlicher Nebenbedingungen zu komplexen Simplexlösungen führt und darüber numerisch nicht stabil ist. D.h. es ist kaum prüfbar, ob eine Lösung ganzzahlig ist. Deshalb wurde eine Variante des Verfahrens entwickelt, bei der nur mit ganzzahligen Werten gerechnet wird. Außerdem gibt es eine Variante für gemischt reellwertige und ganzzahlige Probleme. In der Praxis wird das Schnittebenenverfahren oft mit so genannten *branch and bound* Verfahren, die in Abschnitt 11.2 vorgestellt werden, kombiniert. Dies führt zu problemspezifischen Ansätzen.

11.1.2 Darstellung verschiedener Optimierungsprobleme als ILP

In diesem Abschnitt betrachten wir einige typische Optimierungsprobleme und ihre Formulierung als ILP oder besser als binäre Optimierungsprobleme, da wir mit binären Entscheidungsvariablen arbeiten. Für die Lösung der vorgestellten Probleme eignen sich insbesondere die im nachfolgenden Abschnitt vorgestellten branch and bound Verfahren.

Das erste Problem ist das bekannte Rucksackproblem. Ziel ist es, eine Menge von Gegenständen, die aus n Gegenständen ausgewählt werden in einen Rucksack zu packen. Jeder Gegenstand hat ein Gewicht $a_j > 0$ und einen Wert $c_j > 0$. Der Rucksack darf ein Maximalgewicht von A haben und sein Wert soll möglichst groß sein. Zur Formalisierung führen wir für jeden Gegenstand eine Entscheidungsvariable $x_j \in \mathbb{B}$ ein. $x_j = 1$ bedeutet, dass Gegenstand j eingepackt wird, $x_j = 0$ bedeutet entsprechend, dass der Gegenstand nicht eingepackt wird. Damit kann man das Problem wie folgt formulieren:

$$\min\left(-\sum_{i=1}^{n} x_i \cdot c_i\right) \text{ udN } \sum_{i=1}^{n} x_i \cdot a_i \leq A \text{ und } x_i \in \{0,1\} \ (i = 1, \ldots, n)$$

Man kann die Problemstellung noch erweitern, indem man Abhängigkeiten zusätzlich berücksichtigt. So lässt sich die Forderung, dass Gegenstand j immer mitgenommen werden muss, wenn Gegenstand i mitgenommen wird, durch die Ungleichung $x_i \leq x_j$ formalisieren.

Das zweite Problem, das wir hier betrachten wollen, ist das Problem des Handlungsreisenden. Ziel ist es, eine möglichst kurze Rundreise zu bestimmen, bei der m Städte besucht werden sollen und die Kosten der Fahrt von i nach j $(i, j = 1, \ldots n)$ c_{ij} betragen. Kosten können wahlweise als wirkliche Kosten, Entfernungen oder Zeit interpretiert werden. Probleme dieser Art haben eine große Bedeutung in der Praxis.

Wir definieren m^2 Entscheidungsvariablen $y_{ij} \in \{0, 1\}$, die jeweils angeben, ob nach der Stadt i die Stadt j besucht wird. Auf dieser Basis lässt sich das Problem wie folgt formalisieren.

$$\min\left(\sum_{i=1}^{m}\sum_{j=1}^{m} y_{ij} \cdot c_{ij}\right) \text{ udN } \sum_{j=1}^{m} y_{ij} = 1 \text{ und } \sum_{i=1}^{m} y_{ij} = 1 \ (i, j = 1, \ldots, m)$$

Die beiden Gruppen von Nebenbedingungen sorgen dafür, dass genau ein Nachfolger und ein Vorgänger pro Stadt existiert. Dies stellt zusammen sicher, dass alle Städte besucht werden. Für die exakte ILP Formulierung sind die zweidimensionalen Indizes ij noch zu linearisieren, so dass $n = m^2$ Variablen genutzt werden.

Das letzte Problem, das wir hier betrachten ist das Versorgungsproblem, welches ebenfalls in unterschiedlichen Varianten in praktischen Anwendungen vorkommt. Es gibt m Orte, an denen Bedienzentren eröffnet werden können. Die Kosten, um ein Bedienzentrum am Ort $i \in \{1, \ldots, m\}$ zu eröffnen betragen c_i. Es gibt insgesamt k Kunden, die aus den Bedienzentren versorgt werden sollen. Um den Kunden j aus dem Bedienzentrum am Ort i zu versorgen treten Kosten d_{ij} auf.

Wir definieren m Entscheidungsvariablen $z_j \in \{0, 1\}$, die angeben, ob am Ort j ein Bedienzentrum eröffnet wird oder nicht. Weitere $k \cdot m$ Entscheidungsvariablen $y_{ij} \in \{0, 1\}$ ($i = 1, \ldots, k$, $j = 1, \ldots, m$) geben an, ob Kunde i von Ort j aus beliefert wird. Mit diesen Variablen kann das Problem formalisiert werden.

$$\min \left(\sum_{i=1}^{k} \sum_{j=1}^{m} y_{ij} \cdot d_{ij} + \sum_{j=1}^{m} z_j \cdot c_j \right) \text{ udN } y_{ij} \leq z_j, \sum_{j=1}^{m} y_{ij} = 1 \text{ für alle } i \in \{1, \ldots, k\} \text{ und } j \in \{1, \ldots, m\}$$

Die ersten Nebenbedingungen stellen sicher, dass Kunden nur aus existierenden Standorten beliefert werden, die zweiten Nebenbedingungen garantieren, dass jeder Kunde von genau einem Standort beliefert wird. Wie schon im vorherigen Beispiel ist die Indizierung zu linearisieren, um zur Standardproblemformulierung zu gelangen.

Versorgungsprobleme können erweitert werden. So können einzelne Bedienzentren nur eine bestimmte Anzahl von Kunden bedienen oder haben nur eine feste Bedienkapazität und die Kunden unterschiedliche Bedienwünsche. Im letzteren Fall kann man auch Lösungen anvisieren, bei denen ein Kunde von mehreren Bedienzentren aus bedient wird, wodurch ein gemischtes Problem entsteht.

11.2 Lösungsansätze für kombinatorische Optimierungsprobleme

In diesem Abschnitt betrachten wir Methoden für schwere kombinatorische Optimierungsprobleme. Die Menge zulässiger Lösungen ist bei diesen Problemen zwar endlich, in der Regel existieren aber sehr viele mögliche Lösungen. Es werden zuerst grundsätzliche Lösungsansätze vorgestellt, die prinzipiell auch über die kombinatorische Optimierung hinaus anwendbar sind. Für praktische Probleme sind diese allerdings problemangepasst zu verfeinern. Deshalb werden in Abschnitt 11.3 Verfahren für zwei spezielle Klassen von Beispielproblemen vorgestellt.

Da wir es mit einer endlichen Menge potentieller Lösungen zu tun haben, ist die vollständige Enumeration aller zulässigen Punkte ein Ansatz, der offensichtlich zum Ziel führt. Dieser Ansatz benötigt auf jeden Fall einen exponentiellen Aufwand. Als einfaches Beispiel können wir ein binäres Entscheidungsproblem mit $n = 50$ Variablen betrachten. Der zulässige Bereich umfasst $2^{50} \approx 10^{15}$ Elemente und die vollständige Enumeration würde schon bei einem Aufwand von 1 msec pro Punkt ungefähr 3200 Jahre dauern. Dies zeigt, dass auch mit fortschreitender Leistungsfähigkeit der zur Verfügung stehenden Rechner die vollständige Enumeration nicht praktikabel ist. Anderseits ist es aber auch klar, dass bei NP-schweren Problemen der exponentielle Aufwand im worst case wohl nicht zu vermeiden ist.

Man unterscheidet zwei Klassen von Verfahren, nämlich *exakte Verfahren* und *heuristische Verfahren*. Exakte Verfahren haben im worst case einen exponentiellen Aufwand, für viele praktische Probleme ist der Aufwand aber polynomiell oder zumindest tolerabel. Exakte Verfahren zeichnen sich dadurch aus, dass sie immer die optimale Lösung finden, falls diese existiert. Sie beruhen auf den folgenden Prinzipien:

- Begrenzung des Lösungsraums: Es werden schrittweise Restriktionen eingeführt, so dass Teile des Lösungsraums, in denen das gesucht Optimum nicht liegen kann, abgeschnitten werden. Schnittebenenverfahren sind ein typisches Beispiel für diese Klasse von Verfahren.

- Divide and conquer-Ansätze: Nach dem Prinzip des "Teile und Herrsche" wird das Problem in kleinere Subprobleme zerlegt, so dass
  - aus den Lösungen der Teilprobleme die Lösung des Gesamtproblems herleitbar ist und
  - durch die Zerlegung der Lösungsaufwand i.d.R. reduziert wird.

Zur zweiten Klasse von Ansätzen gehören branch and bound Verfahren, die gleich vorgestellt werden und in gewissem Sinne gehört auch die dynamische Optimierung dazu, die wir in Kapitel 12 untersuchen werden. Es existieren auch Kombinationen aus den genannten Ideen, die z.B. unter dem Stichwort branch and cut Algorithmen zu finden sind.

Heuristische Verfahren weisen in der Regel einen polynomiellen Aufwand auf oder sind iterativ, so dass schrittweise bessere Lösungen berechnet werden. Sie berechnen aber nicht zwangsläufig die optimale Lösung sondern nur eine Approximation und im Idealfall Schranken. Die Klasse der heuristischen Verfahren ist sehr groß und umfasst viele unterschiedliche Ansätze. Viele dieser Ansätze lassen sich in die folgenden zwei Schritte unterteilen.

1. Den Eröffnungsschritt zur Bestimmung einer initialen Lösung, die im zulässigen Bereich liegt.
2. Dem Verbesserungsschritt zum Finden besserer Lösungen. Oft geschieht dies durch eine lokale Suche in der Nachbarschaft.

Beide Schritte können unterschiedlich realisiert und kombiniert werden und führen so zu unterschiedlichen Verfahren. Es gibt eine Reihe weiterer Aspekte, die bei der Konstruktion heuristischer Verfahren berücksichtigt werden müssen. So kann die Suche nach einem deterministischen Algorithmus oder auf Basis von Zufallsentscheidungen erfolgen, es können vergangene Entscheidungen einbezogen werden oder es kann nur auf Basis des aktuellen Punktes entschieden werden. Weiterhin ist festzulegen, wann ein neuer Punkt akzeptiert wird. Eine Akzeptanzbedingung, die einen Punkt nur akzeptiert, wenn er einen besseren Funktionswert aufweist, führt zu einem so genannten "hill climbing" (auch wenn wir eigentlich ein Tal bei der Minimierung suchen) und sorgt dafür, dass Verfahren in lokalen Optima stecken bleiben. Deshalb gibt es eine zahlreiche andere Ansätze, die auch schlechtere Lösungen nach vorgegebenen Kriterien akzeptieren. Schließlich ist bei vielen heuristischen Verfahren zu entscheiden, wann das Verfahren abzubrechen ist. Da nicht zwangsläufig die optimale Lösung gefunden wird und meistens auch kein Kriterium existiert, um festzustellen, ob das Optimum gefunden wurde, müssen andere Kriterien zum Abbruch gewählt werden. Üblicherweise erfolgt der Abbruch nach einer vorgegebenen Zeit oder wenn keine Verbesserung der Lösung über einen längeren Zeitraum mehr beobachtet wird. Falls Schranken bestimmt werden, so kann ein Abbruch auf dieser Basis erfolgen. Wir werden uns hier nicht weiter mit heuristischen Verfahren beschäftigen, einen interessanten Überblick über diese Verfahren findet man in Michalewicz and Fogel [2004].

11.2.1 Branch-and Bound Verfahren

Wir betrachten im Folgenden ein Maximierungsproblem[1] P_0, um daran die Klasse der branch-and-bound Verfahren (B&B-Verfahren) zu erläutern. Formal liege damit folgendes Problem vor:

$$P_0 = \max_{x \in W_0} (f(x)) \text{ mit } W_0 \subseteq \{0,1\}^n$$

Wir haben bereits gesehen, dass eine vollständige Enumeration für zulässige Mengen großer Kardinalität hoffnungslos ist. B&B-Verfahren benutzen einen Ansatz, den man auch als implizite Enumeration bezeichnen könnte. Die Idee ist dabei auf möglichst effiziente Art und Weise Teilmengen des zulässigen Bereiches zu identifizieren, in denen die optimale Lösung nicht liegen kann. Diese Teilmengen brauchen dann nicht weiter untersucht zu werden. Bei B&B-Verfahren wird dazu

[1] Minimierungsprobleme, die wir bisher untersucht haben, lassen sich im Prinzip völlig analog behandeln. Die Beschreibung der Maximierung scheint hier etwas intuitiver zu sein.

ein Suchbaum genutzt. Sei W_0 der zulässige Bereich oder die Lösungsmenge von P_0. Im branching-Schritt eines B&B-Verfahrens wir P_0 in Teilprobleme $P_1, \ldots, P_k$ mit Lösungsmengen $W_1, \ldots, W_k$ so zerlegt, dass

$$W_0 = \cup_{i=1}^{k} W_i \text{ und } W_i \cap W_j = \emptyset \ (i \neq j) \ .$$

Diesen Schritt kann man anschließend auch auf jedes Teilproblem wieder anwenden. Dadurch entsteht ein Baum mit Wurzel P_0.

Im bounding-Schritt geht es darum, Schranken für Teilproblem zu bestimmen. Sei Z^* das gesucht Optimum und U ($\leq Z$) eine untere Schranke. Ein triviale untere Schranke ist offensichtlich $-\infty$. Sei $W' \subset W_0$ die Menge der Lösungen die untersucht wurden, dann gilt offensichtlich auch $\max_{x \in W'}(f(x)) \leq \max_{x \in W_0}(f(x))$, so dass $U = \max_{x \in W'}(f(x))$ eine untere Schranke ist. Offensichtlich kann U als untere Schranke auch für die Bewertung der Teilprobleme $P_1, \ldots, P_k$ eingesetzt werden. Neben der globalen unteren Schranke werden für die einzelnen Teilprobleme obere Schranken O_i ermittelt. Eine obere Schranke für Teilproblem P_i wird entweder direkt mit Hilfe einer Heuristik ermittelt oder P_i wird in die Teilprobleme $P_{i_1}, \ldots, P_{i_k}$ zerlegt (branching Schritt) und anschließend wird

$$O_i = \max_{j=1,\ldots,k}(O_{i_k}) \geq \max_{j=1,\ldots,k} \left(\max_{x \in W_{i_k}} (f(x)) \right)$$

bestimmt. Eine triviale obere Schranke $O_i = \infty$ existiert natürlich für jedes Teilproblem. Wir definieren ferner

$$O_i^* = \max_{x \in W_i} (f(x))$$

als den maximalen Wert, der in diesem Teilproblem erreicht wird. Wir können vier Fälle unterschieden.

1. Ein Teilproblem ist *weiter zu untersuchen*, wenn noch keine Information über O_i^* bekannt ist und es nicht ausgeschlossen werden kann, dass das Maximum im Teilproblem erreicht wird.

2. Wenn O_i^* nicht bekannt ist und $O_i > U$ gilt, so wird das Problem zerlegt und die Werte für die Teilprobleme sind zu ermitteln, um daraus die Werte für P_i zu bestimmen.

3. Falls O_i^* bekannt ist so ist der entsprechende Teilbaum abgearbeitet. Falls $O_i^* > U$, kann damit $U = O_i^*$ gesetzt werden.

4. Wenn $O_i < U$ dann folgt $O_i^* < U \leq Z^*$, damit braucht der Teilbaum nicht weiter untersucht werden, da das Optimum nicht im Teilbaum liegt.

Der zweite Punkt beschreibt den Verzweigungsfall, die Punkte drei und vier den Auslotungsfall. Die Effizienz des Verfahrens hängt davon ab, wie oft die vierte Bedingung greift. Wenn es gelingt, relativ schnell eine gute untere Schranke zu bestimmen und für Teilprobleme mit wenig Aufwand obere Schranken bestimmt werden können, so können viele Teilprobleme ohne nähere Untersuchung ausgeschlossen werden. Damit bestimmen die Qualität der Schrankenberechnung und die Reihenfolge der Auswertung der Teilprobleme die Laufzeit der Verfahren. Beides ist im Wesentlichen problemspezifisch zu optimieren.

Neben dem geschilderten Ansatz, der mit einer globalen unteren Schranke U und lokalen oberen Schranken O_i arbeitet, gibt es Verfahrensvarianten die zusätzlich eine globale obere Schranke O und lokale untere Schranken U_i nutzen.

Die Zerlegung des Problems hängt natürlich stark von der Problemstruktur ab. Eine natürliche Zerlegung für binäre Probleme entsteht durch die Festlegung einzelner Variablen. So kann das Ausgangsproblem in zwei Probleme disjunkt zerlegt werden, wenn die erste Variable auf 0 und auf 1 festgesetzt wird. Diesem Schema folgend können die resultierenden Probleme weiter zerlegt werden. Insgesamt entsteht damit ein binärer Baum (siehe auch die Beispiele im folgenden Abschnitt). Für andere Probleme, die nicht binär sind, müssen andere Zerlegungen gefunden werden. Die Effizienz des Verfahrens wird natürlich von der Zerlegung mit beeinflusst.

11.3 Lösungsmethoden für typische Beispielprobleme

Wir haben im letzten Abschnitt gesehen, dass Algorithmen für die kombinatorische Optimierung viele problemspezifische Komponenten beinhalten. Dieser Abschnitt betrachtet zwei typische Problemklassen und stellt Optimierungsalgorithmen für diese Problemklassen vor. Für das Rucksackproblem wird ein typischer B&B-Algorithmus eingeführt, während die anschließend behandelten Maschinenbelegungs-Probleme sehr viel vielschichtiger sind, so dass je nach Variante unterschiedliche Algorithmen existieren.

11.3.1 Das Rucksackproblem

Das Rucksackproblem wurde bereits in Abschnitt 11.1.2 formalisiert. Es liegt das folgende Optimierungsproblem vor.

$$\min\left(-\sum_{i=1}^{n} x_i \cdot c_i\right) \text{ udN } \sum_{i=1}^{n} x_i \cdot a_i \leq A \text{ und } x_i \in \{0,1\} \ \ (i = 1, \ldots, n)$$

mit $c_i, a_i > 0$. Sinnvollerweise sei $\sum_{i=1}^{n} a_i > A$, sonst ist die Lösung trivial. Falls $\max_i a_i > A$ könnten wir die Menge der Gegenstände einfach verkleinern. Deshalb gehen wir auch davon aus, dass jeder Gegenstand mitgenommen werden kann. Offensichtlich ist $x = 0$, d.h. der leere Rucksack, eine zulässige Lösung.

Das Problem ist theoretisch gut untersucht und dient oft als Testfeld für Lösungsmethoden, da es relativ einfach zu beschreiben und trotzdem NP-schwer ist.

Jeder Gegenstand hat einen spezifischen Wert c_i/a_i, der seinen Wert relativ zum Gewicht ausdrückt. Wir nehmen an, dass $c_1/a_1 \geq c_2/a_2 \geq \ldots \geq c_n/a_n$ gilt. Dies lässt sich immer durch Umsortieren erreichen. Für das relaxierte LP wird aus $x_j \in \{0,1\} \Rightarrow 0 \leq x_j \leq 1$. Dieses Problem können wir lösen, die Lösung ist sogar direkt, ohne einen Optimierungsalgorithmus bestimmbar. Sei dazu

$$k: \sum_{j=1}^{k-1} a_j \leq A \text{ und } \sum_{j=1}^{k} a_j > A$$

Da wir von positiven Gewichten ausgehen, kann k eindeutig bestimmt werden. Sei x^{LP} die Lösung des relaxierten LPs, Z_{LP} dessen Zielfunktionswert und A_{LP} das Gewicht des Rucksacks. Es gilt

$$x_j^{LP} = \begin{cases} 1 & \text{falls } j < k \\ \frac{A - \sum_{j=1}^{k-1} a_j}{a_k} & \text{falls } j = k \\ 0 & \text{falls } j > k \end{cases}, \quad Z_{LP} = \sum_{j=1}^{k-1} c_j + \left(A - \sum_{j=1}^{k-1} a_j\right) \cdot \frac{c_k}{a_k} \text{ und } A_{LP} = A \ .$$

Sei Z_{ILP}^* der optimale Zielfunktionswert des Rucksackproblems. Eine Näherungslösung x^{ILP} kann man aus x^{LP} gewinnen.

$$x_j^{ILP} = \begin{cases} x_j^{LP} & \text{falls } j \neq k \\ 0 & \text{sonst} \end{cases}, \quad Z_{ILP} = \sum_{j=1}^{k-1} c_j \text{ und } A_{ILP} = \sum_{j=1}^{k-1} a_j \ .$$

Es gilt dann $Z_{ILP} \leq Z_{ILP}^* \leq Z_{LP}$. Diese Lösung lässt sich mit einer einfachen Greedy-Heuristik verbessern. Die Idee besteht darin, den Rest $A' = A - A_{ILP}$ mit weiteren Gegenständen aus der Menge $\{j | j = k+1, \ldots, n\}$ zu füllen. Der folgende Algorithmus berechnet eine neue Lösung x mit Zielfunktionswert Z_H und Gewicht A'.

$A' = A - A_{ILP}$;
$Z_H = Z_{ILP}$;
for $j = k+1, \ldots, n$ do
  if $a[j] \leq A'$ then

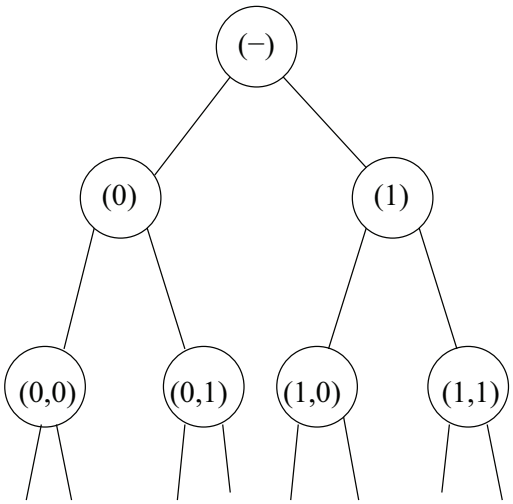


Abbildung 11.4: Erste Knoten des binären Entscheidungsbaums für ein Rucksackproblem.

begin
$x[j] = 1$;
$Z_H = Z_H + c[j]$;
$A' = A' - a[j]$;
end
$A_H = A - A'$;

Es gilt $Z_H \leq Z^*_{ILP} \leq Z_{LP}$. Der Aufwand zur Berechnung von Z_H liegt in $O(n \log n)$, da die relativen Gewichte zuerst sortiert werden müssen. Der obige Algorithmus hat nur einen linearen Aufwand. Es gibt weitere Verfahren zur Berechnung besserer Schranken, die wir aber nicht weiter betrachten wollen.

Stattdessen soll am Beispiel des Rucksackproblems ein B&B-Algorithmus entwickelt werden. Der Lösungsraum wird durch einen binären Baum dargestellt. Wir nummerieren die Ebenen des binären Baums von 0 bis n. Ein Knoten der Ebene s ($\leq n$) ist durch einen Vektor $t = (x_1, \ldots, x_s)$ mit $x_i \in \{0, 1\}$ charakterisiert. Damit wird im Vektor festgelegt, welche der ersten s Gegenstände im Rucksack sind. Die Entscheidung über die restlichen Gegenstände $s + 1, \ldots, n$ ist noch offen.

Mit Hilfe der vorgestellten Greedy-Heuristik kann eine Lösung x_H mit Zielfunktionswert Z_H effizient berechnet werden. Offensichtlich ist $U = Z_H$ eine untere Schranke. Sei (Z^+, x^+) (initial $= (Z_H, x_H)$) die jeweils beste zur Zeit bekannte Lösung und der zugehörige Funktionswert.

Für $t = (x_1, \ldots, x_s)$ sei

$$A_t = \sum_{i=1}^{s} x_i \cdot a_i \text{ und } Z_t = \sum_{i=1}^{n} x_i \cdot c_i$$

das aktuelle Gewicht und der Wert des Rucksacks. Es können die folgenden drei Fälle auftreten.

1. $A_t > A$: Die Lösungsmenge ist leer, d.h. es gibt keine Lösungen x, so dass die ersten s Vektorkomponenten von x, denen von t entsprechen. Der Fall kann offensichtlich nur bei $x_s = 1$ auftreten. Das Teilproblem ist damit ausgelotet und der entsprechende Teilbaum muss nicht mehr untersucht werden.

2. $A_t = A$: Der Rucksack ist voll, so dass t durch das Anfügen von $n - s$ Nullen zu einem vollständigen Vektor über alle Gegenstände ergänzt werden kann. Der Wert des resultierenden Rucksacks ist dann Z_t. Falls $Z_t > Z^+$, wurde eine bessere Lösung gefunden. Diese wird gespeichert, d.h. $(Z^+, x^+) = (Z_t, x_t)$. Der Fall kann ebenfalls nur bei $x_s = 1$ auftreten.

Das Teilproblem ist damit ausgelotet und der entsprechende Teilbaum muss nicht weiter untersucht werden.

3. $A_t < A$: Das Optimum liegt potentiell im Teilbaum. Wir müssen zwei Fälle unterscheiden

   (a) $s = n$: Eine weitere Aufteilung ist nicht mehr möglich, da für alle Gegenstände feststeht, ob sie im Rucksack sind oder nicht. Falls $Z_t > Z^+ \Rightarrow (Z^+, x^+) = (Z_t, x_t)$.

   (b) $s < n$:Das Teilproblem kann möglicherweise weiter aufgeteilt werden. Um festzulegen, ob dies sinnvoll ist, berechnen wir zuerst eine obere Schranke O_t für den Zielfunktionswert. Falls

      i. $O_t \leq Z^+$ liegt das Optimum nicht in diesen Teilbaum, d.h. das Teilproblem ist ausgelotet.

      ii. $O_t > Z^+$dann liegt das Optimum potentiell im Teilbaum. Um dies weiter zu analysieren, wird das Teilproblem in $(x_1, \ldots, x_s, 0)$ und in $(x_1, \ldots, x_s, 1)$ zerlegt und beide Falle werden separat weiter untersucht (Rekursionsfall).

Die Berechnung von O_t im letzten Fall erfolgt auf Basis eines relaxierten LPs, mit Maximalgewicht $A' = A - A_t$ und den freien Variablen $(x_{s+1}, \ldots, x_n)$. Die Lösung dieses LPs kann, wie die Lösung des relaxierten LPs für das Gesamtproblem, direkt angegeben werden. Wenn kein Teilproblem mehr zu untersuchen ist, so entspricht x^+ der gesuchten Lösung und Z^+ ist der zugehörige Zielfunktionswert.

Die Beschreibung des Vorgehens lässt offen, in welcher Reihenfolge die Teilprobleme abzuarbeiten sind. Es gibt verschiedene Varianten und Vorschläge. Weit verbreitet sind die beiden folgenden Ansätze.

- Die Teilprobleme werden nach der Größe von O_t bearbeitet. Zuerst wird immer das Teilproblem mit der größten oberen Schranke weiter bearbeitet. Dieses Vorgehen ist mit der Hoffnung verbunden, Z^+ möglichst frühzeitig zu erhöhen, um so viele Teilprobleme schnell ausloten zu können.

- Die Teilprobleme werden nach dem *last in first out*-Prinzip behandelt. Dies entspricht einem Tiefendurchlauf durch den Baum. Der Vorteil dieses Vorgehens ist, dass nur relativ wenige Knoten gespeichert werden müssen.

Als Erweiterung des binären Rucksackproblems existiert das ganzzahlige Rucksackproblem, bei dem mehrere Gegenstände eines Typs vorkommen. Im Prinzip kann man den Algorithmus vom binären auf den ganzzahligen Fall übertragen werden. Weitere Lösungsverfahren findet man in Martello and Toth [1990, Kap. 3].

11.3.2 Maschinenebelegungs-Probleme

In diesem Abschnitt beschäftigen wir uns mit einer Klasse von kombinatorischen Optimierungsproblemen, die eine große praktische Bedeutung in Bereichen haben. In dieser Problemklasse ist eine Menge von Jobs (bzw. Aufträgen, Aufgaben, ...) von einer Menge von Maschinen (bzw. Prozessoren, Personen, ...) zu bearbeiten. Jobs können aus mehreren Tasks (d.h. Arbeitsschritten) bestehen, die in vorgegebener Reihenfolge bearbeitet werden müssen. Die zugehörigen Probleme bezeichnet man als Maschinenbelegungs-Probleme oder Scheduling-Probleme. Probleme dieser Art haben eine immense praktische Bedeutung. Die folgende Liste umfasst nur einige relevante Anwendungsgebiete:

- In einer Fertigung müssen die einzelnen Fertigungsschritte von Maschinen in teilweise festgelegter, teilweise frei wählbarer Reihenfolge bearbeitet werden.

- Auf Flughäfen müssen An- und Abflüge in eine Reihenfolge gebracht werden.

- Auf einem Rechner muss der Prozessor unter verschiedenen Jobs aufgeteilt werden.

- In einem hochparallelem Rechnersystem müssen die Jobs auf die Prozessoren verteilt werden.
- In einem Call-Center müssen Anrufe auf Agenten verteilt werden.

Man kann zahlreiche weitere Beispiele dieser Art finden. Abstrakt ist in allen Fällen die Frage zu beantworten:
Wann *soll* ***welcher*** *Job (bzw. welcher seiner Tasks) auf* ***welcher*** *Maschine/* ***welchen*** *Maschinen bearbeitet werden, wenn eine bestimmte Zielfunktion gegeben und zu optimieren ist?*

Für die einzelnen Parameter des Problems gibt es zahlreiche Freiheitsgrade. So ist die folgende Aufzählung von möglichen Zielfunktionen bei weitem nicht vollständig:

- Minimierung der Zeitspanne bis zur Fertigstellung des letzten Jobs.
- Minimierung der mittleren Wartezeit (Zeit ohne Bearbeitung) von Jobs.
- Minimierung der maximalen Überschreitung der vorgegebenen Fertigstellungsstermine.
- Minimale mittlere Überschreitung der vorgegebenen Fertigstellungstermine.
- Minimierung der Gesamtkosten.

Neben der Zielfunktion sind verschiedene weitere Annahmen über das Verhalten von Maschinen und Jobs notwendig. So können zum Beispiel

- alle Jobs zu Beginn vollständig bekannt sein oder es können neue Jobs während der Bearbeitung hinzukommen.
- die Bearbeitungszeiten der Jobs vollständig bekannt sein oder es sind nur statistische Werte bekannt.
- Jobs von unterschiedlicher Wichtigkeit sein.
- Jobs unterbrechbar oder ununterbrechbar sein. Wenn sie unterbrechbar sind, so können für die Unterbrechung Kosten entstehen.

Es gibt eine breite Theorie über Scheduling-Probleme alle Art. Dieser Abschnitt basiert im Wesentlichen auf Neumann and Morlock [1993, Kap. 3.6], detailliertere Ergebnisse findet man unter anderem in Blazewicz et al. [2001], Chretienne et al. [1995]. Gerade im Bereich des Scheduling treten Probleme sehr unterschiedlicher Schwierigkeitsgrade. So können leichte Modifikationen aus einem leichten (d.h. in polynomieller Zeit lösbarem) Problem ein schweres (d.h. nur in exponentieller Zeit lösbares) Problem machen. Auf Grund der Vielzahl der Varianten können hier nur einige wenige Beispiele vorgestellt werden.

Grundlegende Notationen und Resultate

Wir betrachten einen von n Jobs $JM = \{J_1, \ldots, J_n\}$ und m Maschinen $MM = \{M_1, \ldots, M_m\}$. Falls keine Verwechselungsgefahr besteht, werden wir nur den Index zur Kennzeichnung des Jobs oder der Maschine verwenden. Folgende Größen charakterisieren die Jobs und Maschinen:

- Bearbeitungsdauer (engl. processing time): p_{ij} für Job J_j auf Maschine M_i.
- Bereitstellungstermin (engl. release time): r_j von Job J_j, d.h. die Zeit, zu der mit der Bearbeitung von Job J_j frühestens begonnen werden kann.
- Fälligkeitstermin (engl. due date): d_j von Job J_j, d.h. der Termin, zu dem der Job spätestens fertig sein soll.
- Gewicht (engl. weight): w_j von Job J_j, d.h. die Wichtigkeit/Priorität von Job J_j, wenn $w_j > w_k$, dann ist J_j wichtiger als J_k.

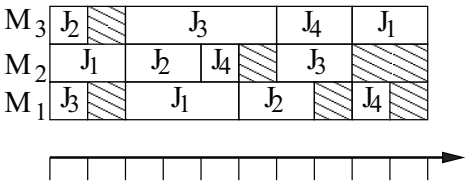


Abbildung 11.5: Gantt-Diagramm für ein Schedule für $n = 4$ Jobs auf $m = 3$ Maschinen.

Wir nehmen an, dass alle Werte nichtnegativ und ganzzahlig sind. Dies ist keine große Einschränkung, da wir immer entsprechend skalieren können. Wir betrachten zuerst einmal die folgende einfache Klasse von Problemen:

- Alle Jobs sind zu Beginn bekannt und alle Parameter der Jobs sind bekannt.
- Jeder Job kann zu einem Zeitpunkt von maximal einer Maschine bearbeitet werden.
- Jede Maschine kann zu einem Zeitpunkt maximal einen Job bearbeiten.

Wenn die Zeitintervalle, auf denen die Jobs auf den Maschinen bearbeitet werden festgelegt sind, so spricht man von einem Schedule (bzw. Bearbeitungsplan). Ein Schedule legt damit die Zeiten fest, zu denen ein Job bzw. ein Task auf einer bestimmten Maschine bearbeitet wird. Schedules kann man durch so genannte Balken- oder Gantt-Diagramme grafisch darstellen. Abbildung 11.5 zeigt ein Beispiel-Diagramm. Die schraffierten Flächen bezeichnen Intervalle, in denen die Maschine untätig ist.

Ein Schedule heißt zulässig, wenn alle Restriktionen erfüllt sind, er heißt optimal, wenn er zulässig ist und eine vorgegeben Zielfunktion optimiert wird. Restriktionen ergeben sich aus den obigen Annahmen, dass eine Maschine zu einem Zeitpunkt nur einen Job bearbeiten kann und ein Job zu einem Zeitpunkt nur von einer Maschine bearbeitet werden kann.

Wenden wir uns kurz den Zielfunktionen zu. Eine monoton wachsende (bzw. nicht fallende) Funktion $f_j : \mathbb{R}_+ \to \mathbb{R}$ beschreibt die für Job J_j anfallenden Kosten. D.h. $f_j(t)$ sind die Kosten, die anfallen, wenn Job $J_j$zum Zeitpunkt t fertig wird. Folgende Beurteilungsgrößen für Jobs sind üblich:

- Der Abschlußzeitpunkt C_j (engl. completion time)
- Die Durchlaufzeit oder Verweilzeit $V_j = C_j - r_j$(engl. turnaround time, residence time, sojourn time)
- Die Verspätung $L_j = C_j - d_j$ (engl. lateness) und als Variante $D_j = \max(C_j - d_j, 0)$ (engl. tardiness).

Auf Basis dieser Größen können unterschiedliche Zielfunktionen definiert werden. Es gibt einige Standardfunktionen mit spezieller Kennung, die wir gleich kurz vorstellen werden.

Für die Kennzeichnung von Scheduling-Problemen ist eine Kurznotation verbreitet Graham et al. [1979].

Scheduling-Probleme werden durch ein Tripel $A|B|C$ beschrieben, wobei

- A die Maschinenkonfiguration beschreibt und aus zwei Symbolen a_1a_2 besteht. a_2 ist eine ganze Zahl für die Anzahl Maschinen. a_1 kann aus folgenden Alternativen gewählt werden:
  - Wenn a_1 weggelassen wird, so haben wir es mit einem Einprozessorsystem zu tun.
  - $a_1 = P$ identische Maschinen
  - $a_1 = Q$ uniforme Maschinen, die alle Jobs mit identischer aber maschinenspezifischer Geschwindigkeit bearbeiten.
  - $a_1 = R$ heterogene Maschinen.

  - $a_1 = O$ Maschinen in einem Open-Shop (siehe unten)
  - $a_1 = F$ Maschinen in einem Flow-Shop (siehe unten)
  - $a_1 = J$ Maschinen in einem Job-Shop (siehe unten)

- B beschreibt die Beabreitungsspezifika und kann durch bis zu 8 Symbolen bestehen (siehe Blazewicz et al. [2001, S. 68-69]). Wir betrachten hier die folgenden Beispiele:
  - Weglassen von B bedeutet, dass alle Jobs zum Zeitpunkt 0 bereit stehen, dass es keine Restriktionen bzgl. der Abarbeitung gibt und dass Jobs nicht unterbrochen werden dürfen.
  - *pmtn* bedeutet, dass Jobs unterbrochen werden dürfen.
  - *prec* bedeutet, dass es eine Präzendezrelation zwischen den Jobs gibt und *tree* zeigt, dass diese durch einen Baum definiert ist.

- C beschreibt die Zielfunktion. Folgende standardisierte Zielfunktionen sind von besonderem Interesse:
  - Min-max-Probleme sind durch Zielfunktion der Form $C_{\max} = \max_{j=1,\ldots,n} C_j$, $L_{\max} = \max_{j=1,\ldots,n} L_j$ oder $f_{\max} = \max_{j=1,\ldots,n} f_j(C_j)$ charakterisiert.
  - Min-sum-Probleme sind durch Zielfunktionen der Form $\sum C_j$, $\sum w_j C_j$ oder $\sum w_j U_j$ mit $U_j = \delta(C_j > d_j)$charakterisiert.

Einige Beispiele:
$1|tree|\sum w_j C_j$ beschreibt ein Problem auf einer Maschine, bei dem eine durch einen Baum definierte Präzedenzrelation zwischen den Jobs besteht und als Zielfunktion die gewichtete Summe der Abschlußzeitpunkte dient.
$1|pmtn, r_j|f_{max}$ beschreibt ein Problem auf einer Maschine, bei dem Jobs unterbrochen werden dürfen und erst zu vorgegebenen Bereitstellungsterminen verfügbar sind. Die Zielfunktion ist durch das Maximum einer Funktion über die Abschlußzeiten gegeben.
$P2||C_{max}$ beschreibt ein Problem mit zwei identischen Prozessoren und den voreingestellten Eigenschaften der Jobs. Das Ziel ist die Minimierung der maximalen Abschlußzeit.

Ein-Maschinen-Probleme

Wir betrachten Probleme, deren Jobs sich nicht weiter in Tasks zerlegen lassen und die alle auf einer Maschine bearbeitet werden müssen. Dadurch kann der Maschinenindex weggelassen werden. D.h. aus p_{ij} wird p_j. Folgender fundamentale Satz hilft uns bei der Lösung dieser Probleme.

Satz 24 *Jedes Ein-Maschinenproblem ohne Bereitstellungstermine besitzt einen optimalen Schedule ohne Leerzeiten.*

Dabei ist jeder optimale Schedule eines Problems ohne Unterbrechungen auch optimal für das entsprechende Problem mit zugelassenen Unterbrechungen.

Beweis: Wir haben vorausgesetzt, dass f_j monoton steigend in t ist. Gleichzeitig sind die Zielfunktionen (MinMax, MinSum) monoton steigend in f_j und sollen minimiert werden. Wenn wir nun ein Schedule mit Lücken betrachten, so können diese Lücken dadurch ausgefüllt werden, dass Bearbeitungen vorgezogen werden. Auf Grund der Monotonie kann sich dadurch die Zielfunktion nicht verschlechtern.
Den zweiten Teil kann man ähnlich beweisen. Angenommen in einem Schedule wird J_j durch J_k unterbrochen. Dann kann man auf Grund der Monotonie der Gewichte leicht zeigen, dass einer der Schedules, bei denen $J_j$und J_k nacheinander ausgeführt werden mindestens so gut sein muss wie der Schedule mit Unterbrechung. Auf diese Art können alle Unterbrechungen aus dem Schedule entfernt werden, ohne dass sich die Zielfunktion verschlechtert.

□

Das Problem $1||C_{max}$ ist offensichtlich einfach zu lösen, da für jede Reihenfolge der Bearbeitung ohne Leerzeiten $C_{\max} = \sum p_j$ gilt. Das Problem $1||L_{\max}$ wird mit der *Earliest Due Date Regel* (EDD) gelöst. Dazu werden Jobs nach nicht fallenden Fälligkeitsterminen d_j bearbeitet. Der Aufwand zur Bestimmung des Schedules ist $O(n \log n)$, da die Jobs nach Fälligkeitsterminen sortiert werden müssen.

Wir wollen kurz begründen, warum EDD optimal ist. Wir können einen Schedule in diesem einfachen Fall durch die Reihenfolge der Jobs vollständig definieren. Sei Π^* der EDD-Schedule und Π ein beliebiger anderer nicht-EDD-Schedule. Es müssen Jobs J_j und J_k existieren, die Π direkt aufeinander folgen (d.h. $\Pi = (\ldots, j, k, \ldots)$), für die aber gilt $d_k \leq d_j$ und die in Π^* in umgekehrter Reihenfolge auftauchen (d.h. $\Pi^* = (\ldots, k, \ldots, j, \ldots)$). Wenn wir J_j und J_k in Π vertauschen, so wird $L_{\max}$ nicht verändert oder verringert, da aus $d_k \leq d_j$ und $C_k \geq C_j$ auch $C_k - d_k \geq C_j - d_j$ folgt. Mit endlichen vielen Vertauschungen dieser Art, kann Π in Π^* transformiert werden. Da bei jeder Vertauschung $L_{\max}$ nicht verschlechtert wird, gilt auch $L_{\max}(\Pi) \geq L_{\max}(\Pi^*)$.

Minmax-Probleme der Form $1|prec|f_{\max}$ lassen mit der Regel von Lawler [1973] lösen. Die Präzendezrelation zwischen den einzelnen Jobs sei durch einen Digraphen beschrieben, dessen Knoten die Jobs beschreiben und dessen Kanten die Präzendezrelation darstellen.

Die Regel von Lawler lautet:
Unter allen noch nicht eingeplanten Jobs ohne Nachfolger, setze denjenigen an die letzte Stelle, der die geringsten Kosten verursacht."

Die Nutzung de Regel führt zu folgendem Algorithmus:

$S = \emptyset$; /* leerer Schedule */
$J = \{1, \ldots, n\}$; /* Jobs, die noch nicht zugeordnet wurden */
while $J \neq \emptyset$ do
  $L = \{j | j \in J,\ j$ hat keinen Nachfolger in $J\}$;
  $p_J = \sum_{j \in J} p_j$;
  $k = \arg\min_{j \in L} f_j(p_J)$;
  Job k wird an den Anfang von S gesetzt ;
  $J = J \setminus \{k\}$;

Wir argumentieren kurz, warum diese Lösung optimal ist. Sei f_J^* das Maximum der Jobabschlusskosten für eine optimale Reihenfolge. Bei Auswahl von $k \in L$ gemäß der Regel von Lawler ist der minimale Zielfunktionswert $\max\left(f_k(p_J), f^*_{J\setminus\{k\}}\right)$. Da $f_k(p_J) \leq f_J^*$ und $f^*_{J\setminus\{k\}} \leq f_J^*$ findet der Algorithmus die optimale Lösung bzgl. der Auswahl ersten auszuführenden Jobs aus der Menge J. Wenn man die Argumentation nun rekursiv auf die Menge $J \setminus \{k\}$ anwendet, so folgt daraus die Korrektheit des Algorithmus.

Zur Analyse des Aufwands setzen wir voraus, dass der Aufwand der Auswertung von $f(.)$ unabhängig von n ist. Dies ist normalerweise realistisch. Im Algorithmus werden n Mengen betrachtet und die Werte $f_k(p_J)$ berechnet der Aufwand dazu ist linear in der Anzahl der Jobs in J und der Gesamtaufwand ist damit $O(n^2)$.

Auf Grund des vorherigen Ergebnisses wird es nicht überraschen, dass auch das Problem $1||L_{\max}$ einfach zu lösen ist. Dagegen ist das Problem $1|r_j|L_{\max}$ schwer. Vorgegebene Bereitstellungszeiten verhindern, dass die Regel von Lawler anwendbar ist. Man kann in diesem Fall Varianten haben, bei denen ein optimalen Schedules Leerzeiten auftreten, da es besser ist, auf die Bereitstellung eines Jobs zu warten, als einen anderen bereitgestellten Job direkt zu bearbeiten. Einen gewissen Sonderfall stellt das Problem $1|r_j, p_j = 1|L_{\max}$ dar. In diesem Fall beträgt die Bearbeitungdauer jedes Jobs genau eine Zeiteinheit. Das Problem lässt sich mit einer einfachen Erweiterung der EDD-Regel lösen. Man wählt zu jedem Zeitpunkt t unter den verfügbaren Jobs (d.h. Jobs J_j mit $r_j \leq t$) denjenigen mit dem kleinsten Fälligkeitstermin d_j aus. Da die Bearbeitung nur eine Zeiteinheit dauert und die nächsten Jobs erst nach einer Zeiteinheit bereit stehen, beeinflusst diese Entscheidung keine der erst später bereitsgestellten Jobs. Der Aufwand des Verfahrens ist natürlich wieder $O(n \log n)$.

Ebenfalls einfacher zu lösen ist das Problem $1|pmtn, prec, r_j|f_{\max}$. Unterbrechungen machen also die Lösung einfacher. Der Aufwand für den Algorithmus ist $O(n^2)$ für Details sei auf die Literatur Blazewicz et al. [2001, Kap. 4.5] verwiesen.

Minsum-Probleme der Form $1||\sum C_j$ lassen sich nach der Shortest-Processing-Time-First (SPT) Regel behandelt. Dies bedeutet, dass Jobs nach nichtfallenden Bearbeitungsdauern bearbeitet werden. Folgende Überlegung zeigt, dass SPT optimal ist. Seien J_j und J_k zwei Jobs, die im SPT-Schedule direkt nacheinander bearbeitet werden und sei $p_j < p_k$. Falls J_j zum Zeitpunkt A startet, dann entstehen durch diese beiden Jobs folgende Kosten: $C = C_j + C_k = (A + p_j) + (A + p_j + p_k)$. Falls wir J_j und J_k umordnen, so entstehen Kosten $C' = C'_k + C'_j = (A + p_k) + (A + p_k + p_j) > C$. Falls $p_j = p_k$ gilt $C' = C$. Falls wir nun ein Schedule Π haben, welches nicht die SPT Bedingung erfüllt, so gibt es zwei aufeinander folgende Jobs J_k und J_j mit $p_k > p_j$. Nach obiger Herleitung würde das Vertauschen der Jobs die Zielfunktion verkleinern. Wenn wir so lange vertauschen, bis keine aufeinander folgenden Jobs dieser Art existieren, so verkleinern wir in jedem Schritt die Zielfunktion und erreichen schließlich ein SPT-Schedule.

Da multiplikative und additive Konstanten die Optimalität der Zielfunktion nicht ändern, ist SPT auch für $1||\sum(C_j - r_j)/n$ (d.h. mittlere Durchlaufzeiten) optimal. Für $1||\sum w_j C_j$ kann eine Version von SPT definiert werden, die nicht die Bearbeitungzeiten p_j sondern die gewichteten Bearbeitungszeiten $q_j = p_j/w_j$ berücksichtigt.

Mehrere parallele Maschinen

Der nächste Schritt ist die Untersuchung von Problemen, die auf mehreren Maschinen abgearbeitet werden. Wir betrachten zuerst Probleme der folgenden Art:

- Jobs werden nicht in Tasks unterteilt.
- Jeder Job kann irgendeiner Maschine bearbeitet werden. Allerdings kann ein Job jeweils nur von maximal einer Maschine bearbeitet werden, d.h. nicht von mehreren parallel wohl aber, wenn Unterbrechungen erlaubt sind, von mehreren Maschinen nacheinander.
- Es gibt m Maschinen (sinnvollerweise $m \leq n$), die parallel arbeiten. Wir unterschieden folgende Fälle:
  - Die Maschinen haben unterschiedliche jobspezifische Geschwindigkeiten s_{ij}, so dass die Bearbeitung von Job J_j auf Maschine M_i $s_{ij} \cdot p_{ij}$ Zeiteinheiten dauert. Die Kennzeichnung dieser Variante ist R.
  - Die Maschinen haben eine feste Geschwindigkeit s_i, die nicht vom Job abhängt. Die Bearbeitungszeit von Job J_j auf Maschine M_i lautet $s_i \cdot p_{ij}$. Man spricht von uniformen Maschinen und nutzt die Bezeichnung Q.
  - Alle Maschinen sind identisch, so dass die Geschwindigkeit nicht ausgewiesen werden muss, bzw. in die Bearbeitungszeiten einbezogen wird. Die Bearbeitungszeit von Job J_j auf Maschine M_i ist dann $p_j = p_{ij}$. Die Bezeichnung dieser Variante ist P.

Das Scheduling paralleler Maschinen ist von großer praktischer Bedeutung, da bei der Mehrzahl der Anwendungen mehrere Maschinen zur Verfügung stehen. Leider sind die meisten Probleme aber schwer, d.h. exakt nur in exponentieller Zeit lösbar. Allgemein kann man feststellen, dass Unterbrechungen, die keine zusätzlichen Kosten verursachen, in der Regel das Problem vereinfachen. Ohne Unterbrechung sind schon die Probleme $P2||C_{\max}$ und $P2||\sum w_j C_j$ nur mit exponentiellem Aufwand exakt lösbar.

Das Problem $P||\sum C_j$ ist dagegen für endlich viele Maschinen mit dem Ansatz von Conway exakt lösbar. Das Verfahren entspricht der SPT-Regel. Die Jobs werden nach Bearbeitungszeit geordnet, d.h. $p_1 \leq p_2 \leq \ldots \leq p_n$ und werden in dieser Reihenfolge auf den Maschinen eingeplant. Falls eine Maschine frei wird, wird darauf der nächste zu bearbeitende Job bearbeitet. Abbildung zeigt den daraus entstehenden Schedule. Wenn zwei Maschinen gleichzeitig frei werden, so wird zuerst der Maschine mit dem kleineren Index ein Job zugewiesen.

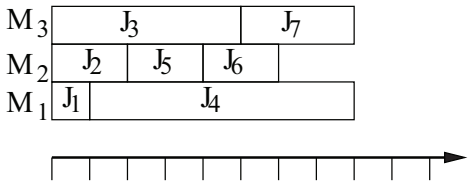


Abbildung 11.6: Schedule für $P||\sum C_j$ nach Conway.

Wir werden nun kurz plausibel machen, warum der so erzeugte Schedule optimal ist. Sei n_i die Anzahl der Jobs, die Maschine M_i zugewiesen wurden. Dann gilt $\sum_{i=1}^{m} n_i = n$. Sei $j_1, \ldots, j_{n_i}$ die Sequenz der Job-Nummern, die in dieser Reihenfolge auf Maschine M_i bearbeitet werden. Für die Summe der Abschlusszeitpunkte dieser Jobs gilt dann

$$C^i_{j_1} + C^i_{j_2} + \ldots + C^i_{j_{n_i}} = p_{j_1} + (p_{j_1} + p_{j_2}) + \ldots (p_{j_1} + \ldots + p_{j_{n_i}}) = kp_{j_1} + (k-1)p_{j_2} + \ldots + p_{j_{n_i}} \quad (11.1)$$

Sei $C^i = \sum_{k=1}^{n_i} C^i_{j_k}$ die Summe der Bearbeitungszeiten der Jobs auf Maschine M_i. Für die Summe der Bearbeitungszeiten über alle Maschinen gilt dann

$$\sum_{i=1}^{m} C^i = \sum_{i=1}^{m} \sum_{k=1}^{n_i} (n_i - k - 1) \cdot p^i_{j_k} \quad (11.2)$$

mit $p^i_{j_k}$ der Bearbeitungszeit des k-ten Jobs auf Maschine i. In der Summe kommt jede Bearbeitungszeit genau einmal vor und wird mit ganzzahligen Faktoren multipliziert. Die Summe wird minimiert, wenn die Bearbeitungszeiten bzgl. der Faktoren nicht fallend angeordnet werden. Der Aufwand des Verfahrens ist $O(n \log n)$.

Satz 25 *Ist ein Schedule optimal für $P||\sum w_j C_j$, dann ist er auch optimal für $P|pmtn|\sum w_j C_j$.*

Der Satz ist plausibel, da man auf identischen Maschinen, die gewichtete Summe der Abschlusszeiten nicht dadurch verringern kann, dass man einen Job unterbricht und später auf der selben oder einer anderen Maschine weiterführt.

Damit ist auch $P|pmtn|\sum C_j$ mit Hilfe der Conway-Regel lösbar. Da $1||\sum C_j$ und $P||\sum C_j$ mit Hilfe der SPT-Regel gelöst werden können, ist es naheliegend sich zu fragen, inwieweit eine ähnliche Regel MinMax-Probleme der Form $P||C_{\max}$ gefunden werden kann. Eine naheliegende Übertragung ist die Definition einer Largest-Processing-Time-First (LPT) Regel, bei der große Jobs zuerst bearbeitet werden. Dieser Strategie liegt die Überlegung zu Grunde, dass man zuerst große Jobs bearbeitet, um am Ende die kleinen Jobs möglichst gut auf die m Maschinen zu verteilen und so den maximalen Abschlusszeitpunkt zu minimieren. Folgender Algorithmus setzt die Idee um.

Ordne die Jobs nach Bearbeitungszeit, so dass $p_1 \geq p_2 \geq \ldots \geq p_n$;
for $i = 1, \ldots, m$ do $t_i = 0$;
for $j = 1, \ldots, m$ do
  bestimme kleinstes $l \in \{1, \ldots, m\}$ mit $t_l = \min_{i=1,\ldots,m} t_i$;
  bearbeite Job j auf M_l im Intervall $[t_l, t_l + p_j]$;
  $t_l = t_l + p_j$;

Leider liefert der Algorithmus nicht die optimale Lösung, sondern nur eine relativ gute Approximation. Man kann zeigen, dass der maximale relative Fehler $(1 - 1/m)/3$ ist. Die exakte Lösung von $P||C_{\max}$ erfordert wieder einen exponentiellen Aufwand.

Durch Unterbrechungen wird das Problem wieder einfacher. So ist das Problem $P|pmtn|C_{\max}$ in $O(n)$ exakt lösbar, wie wir im Folgenden zeigen werden. Offensichtlich ist $C_{\max}$größer gleich

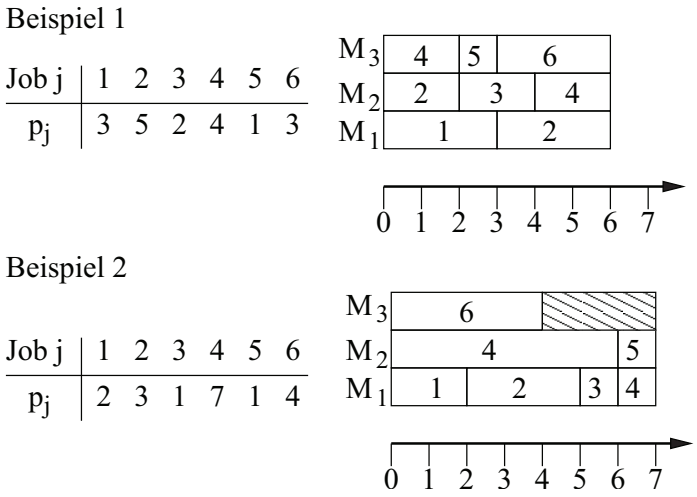


Abbildung 11.7: McNaughton-Schedules für zwei Beispielprobleme.

der maximalen Bearbeitungszeit eines Jobs und der mittleren Belegungszeit aller Jobs auf allen Maschinen.

$$C_{\max} \geq C' = \max\left(\max_{j=1}^{n} p_j, \frac{1}{m}\sum_{j=1}^{m} p_j\right)$$

Der folgende Ansatz von McNaughton liefert einen Schedule mit $C_{\max} = C'$.

- Verweise die Jobs in beliebiger Reihenfolge lückenlos und nacheinander an die Maschinen.
- Wenn für eine Maschine C' erreicht wurde, verweise den Rest des Jobs auf die nächste Maschine.

Da $C' \geq \max p_j$ gilt, wird ein Job zu einem Zeitpunkt immer nur auf maximal einer Maschine bearbeitet. Offensichtlich gibt es in diesem Schedule maximal $m-1$ Unterbrechungen. Die Leerzeiten sind $L = m \cdot \max_{j=1}^{n} p_j - \sum_{j=1}^{n} p_j$, wenn $\max_{j=1}^{n} p_j \geq (1/m)\sum_{j=1}^{n} p_j$ gilt. Im anderen Fall treten keine Leerzeiten auf.

Abbildung 11.7 zeigt zwei Beispiele (aus Neumann and Morlock [1993]) für optimale Schedules. Im erste Beispiel ist die durchschnittliche Maschinenbelegungszeit größer als die maximale Bearbeitungszeit, so dass eine Schedule ohne Leerzeiten entsteht. Im zweiten Beispiel ist dies nicht der Fall und es entstehen Lehrzeiten auf der letzten Maschine.

Beim Übergang von identischen zu uniformen Maschinen ist zu beachten, dass ein schweres Problem für identische Maschinen auch für uniforme Maschinen schwer ist. Bei uniformen Maschinen bringt in der Regel eine Jobunterbrechung Vorteile, da u.U. eine schnellere Maschine gewählt werden kann.

Für das Problem $Q||\sum C_j$benötigt Job j auf Maschine i eine Bearbeitungszeit $p_{ij} = p_j/s_i$. Wenn die Jobs $j_i, \ldots, j_k$ in dieser Reihenfolge auf Maschine i bearbeitet werden, so beträgt die Dauer

$$C^i = C^i_{j_1} + C^i_{j_2} + \ldots + C^i_{j_k} = \frac{1}{s_i}\sum_{l=1}^{k}(k+1-l)p_{j_l} \ .$$

Die Formel ist ähnlich zu (11.1) für $P||\sum C_i$ und auch die Summe der Bearbeitungszeiten lässt sich wieder als $\sum_{i=1}^{m} C^i$ darstellen. Die Koeffizienten der p_j haben jetzt die Form $k/s_i (k = 1, \ldots, n; l = 1, \ldots, m)$. Von diesen mn Koeffizienten werden die kleinsten n ausgewählt und beginnend mit dem kleinsten Koeffizienten für den Job mit der längsten Bearbeitungszeit, den Jobs zugewiesen. Der folgende Algorithmus von Horowitz und Sahni realisiert dieses Vorgehen.

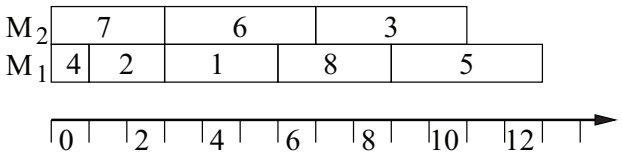


Abbildung 11.8: Schedule für das $Q||\sum C_i$-Beispielproblem.

Ordne die Jobs so, dass $p_1 \leq p_2 \leq \ldots \leq p_n$;
Speichere die Werte $1/s_1, 1/s_2, \ldots, 1/s_m$ auf einem Heap H ;
for $i = 1, \ldots, m$ do $k_i = 1$;
for $j = n, n-1, \ldots, 1$ do
entferne minimales Element aus H (sei k/s_i minimal) ;
füge $(k+1)/s_i$ in H ein ;
setze $\nu_{ik_i} = j$ und $k_i = k_i + 1$;

Nach Beendigung enthält ν_{ik} die Nummer des Jobs, der als k-letzter auf Maschine M_i bearbeitet wird. Der Aufwand des Algorithmus ist $O(n \log n)$ zum Sortieren der Bearbeitungszeiten und zum Einfügen der Elemente.

Betrachten wir ein kleines Beispiel für $m = 2$ Maschinen und $n = 8$ Jobs, das ebenfalls aus Neumann and Morlock [1993] entnommen wurde.

Job j	1	2	3	4	5	6	7	8
p_j	9	6	10	3	12	8	4	9

Umordnen der Jobs nach Bearbeitungszeit liefert.

Job j	4	7	2	6	1	8	3	5
p_j	3	4	6	8	9	9	10	12

Die folgende Tabelle gibt die Geschwindigkeiten der Maschinen und die daraus resultierenden Koeffizienten und teilt die Jobs konsekutiv den Koeffizienten zu.

i	s_i	$1/s_i$	$2/s_i$	$3/s_i$	$4/s_i$	$5/s_i$
1	3	1/3 (5)	2/3 (8)	1 (1)	4/3 (2)	5/3 (4)
2	2	1/2 (3)	1 (6)	3/2 (7)	2	5/2

Das zugehörige Gantt-Diagramm des Schedules wird in Abbildung 11.8 gezeigt.

Wenn wir Unterbrechungen erlauben, also Probleme der Form $Q|pmtn|\sum C_j$ betrachten, so lässt sich eben falls ein optimaler Schedule mit Hilfe einer ähnlichen Idee finden. Man geht wieder von einer Ordnung der Jobs aus, so dass $p_1 \leq p_2 \leq \ldots \leq p_n$ gilt. In dieser Reihenfolge plant man Jobs immer so ein, dass sie so schnell wie möglich fertig werden. Wenn wir annehmen, dass $s_1 \geq s_2 \geq \ldots \geq s_m$ gilt, so werden die ersten m Jobs den Maschinen $1, \ldots, m$ zugewiesen. Wenn der erste Job aus Maschine M_1fertig geworden ist. Wird der zweite Job von Maschine M_2 auf M_1 verlagert, der dritte von M_3 auf M_2 usw. Maschine M_m beginnt mit der Bearbeitung des $m+1$-ten Jobs.

Openshop-, Flowshop- und Jobshop-Probleme

Probleme dieser Art sind noch komplexer. Man geht von den folgenden Annahmen aus.

$m > 1$ Maschinen $M_1, \ldots, M_m$.

Jobs J_j, die aus Tasks O_{ij} mit Bearbeitungsdauern p_{ij} bestehen.

Auf jeder Maschine wird weiterhin nur 1 Job bearbeitet und jeder Job wird von höchstens einer Maschine zu einem Zeitpunkt bearbeitet.

Die Reihenfolgevorschriften der Tasks in den Jobs bestimmen die Modellklasse.

Wenn die Reihenfolge nicht vorgeschrieben ist, so spricht man von einem Openshop-Problem (Bezeichnung O).

Wenn die Reihenfolge vorgeschrieben und für alle Jobs identisch ist, so spricht man von einem Flowshop-Problem (Bezeichnung F)

Wenn die Reihenfolge vorgeschrieben ist und für die einzelnen Jobs unterschiedlich sein darf, so spricht man von einem Jobshop-Problem (Bezeichnung J).

Es wird nicht überraschen, dass für diese Klassen die Lösung der meisten Probleme exponentiellen Aufwand erfordert. Es gibt einige wenige Ausnahmen, für die auf die Literatur verwiesen sei (Blazewicz et al. [2001], Neumann and Morlock [1993]).

Natürlich können auch für Scheduling-Probleme die vorgestellten B&B-Verfahren eingesetzt werden, da es sich um kombinatorische Optimierungsprobleme handelt. Der Lösungsaufwand bleibt aber in vielen Fällen sehr hoch.

In der Praxis kommen auch noch zusätzliche erschwerende Aspekte hinzu. So sind die Bearbeitungszeiten oft nicht oder nur als stochastische Größen bekannt, es kommen während der Bearbeitung noch zusätzliche Jobs hinzu und es treten zusätzliche Kosten auf, wenn Jobs unterbrochen werden oder wenn eine Maschine von einem Job-Typ auf einen anderen wechselt.

Kapitel 12

Dynamische Optimierung

Die bisher vorgestellten Optimierungsprobleme waren dadurch charakterisiert, dass für eine Funktion eine zulässige Parameterbelegung gefunden werden sollte, so dass die (skalare) Resultatgröße möglichst klein wird. Es handelt sich um statische Probleme, bei denen eine Lösung für einen Zeitpunkt bestimmt wird. In vielen realen Systemen ändert sich das Verhalten über die Zeit. Solche Systeme waren die Grundlage unserer Simulationsmodelle im ersten Teil der Vorlesung und führten zu dynamischen Modellen. Wenn für solche dynamischen Modelle optimale Parameterbelegungen gefunden werden sollen, so muss der sich mit der Zeit ändernde Systemzustand berücksichtigt werden. Dies bedeutet, dass die Entscheidung in Schritten getroffen werden muss. Immer wenn neue Informationen vorliegen, werden die Parameter so angepasst, dass unter der gegenwärtigen Informationslage eine möglichst gute Lösung erreicht wird. Diese Sichtweise schließt auf natürliche Weise stochastische Probleme ein, bei denen zu jedem Zeitpunkt Wahrscheinlichkeitsaussagen über das zukünftige Modellverhalten vorliegen.

Ein typisches Beispiel für die genannte Problemklasse sind Lagerhaltungsprobleme. Im Lager gibt es zu jedem Zeitpunkt einen Bestand, es werden Auslieferungsaufträge erteilt und es können neue Waren zur Einlagerung bestellt werden. Ziel ist es, eine optimale Bestellstrategie zu finden, wenn die Auslieferungsaufträge erst im Laufe der Zeit eintreffen und vorab höchstens Wahrscheinlichkeitsverteilungen über Bestellmengen bekannt sind. Optimal bedeutet, dass alle Auslieferungsaufträge möglichst schnell erfüllt werden können (Bedeutung von "möglichst schnell" ist festzulegen) und gleichzeitig der Lagerbestand und damit die Kapitalbindung möglichst gering ist (auch dies ist genauer zu spezifizieren). Jedes Mal, wenn ein neuer Auslieferungsauftrag eintrifft, ändert sich die Informationslage und es kann sinnvoll sein, die Bestellungen anzupassen. Weitere typische Anwendungsszenarien findet man im Bereich der Finanzmathematik, der Wartung technischer Systeme oder der Steuerung von Anlagen.

Eine andere Klasse von Problemen, die mit den im Folgenden vorzustellenden Methoden behandelt werden können, sind statische Probleme, deren Lösung schrittweise ermittelt werden kann. Ein Beispiel für ein solches Problem ist das Rucksackproblem (siehe 11.3.1). Beim Rucksackproblem ist zwar sämtliche Information vorhanden, man kann es aber sequentialisieren indem Gegenstände einer nach dem anderen eingepackt werden und man so zur einer optimalen Einpackreihenfolge gelangt, die einen optimal gepackten Rucksack liefert.

Die beschriebenen Optimierungsprobleme fasst man im Operations Research unter dem Begriff dynamische Optimierung zusammen. Unter den Begriffen optimale Steuerung oder schrittweise Entscheidungsfindung werden ähnliche oder identische Problemstellungen in anderen Kontexten behandelt. Wir werden die Grundlagen dieses Ansatzes kennenlernen und uns dabei insbesondere auf Neumann and Morlock [1993, Kap. 5] und Domschke and Drexl [1993, Kap. 7] beziehen. Das Themengebiet ist aber sehr viel breiter und umfassender. Für eine detaillierte Übersicht sei auf die Bücher Bertsekas [2005] und Bertsekas [2007], sowie Puterman [2005] verwiesen.

Im Folgenden werden zuerst einige typische Beispiele der dynamischen Programmierung vorgestellt. Daran anschließend wird die Problemstellung für deterministische Modelle formal definiert und das Optimierungsproblem formuliert. Eine weit verbreitete Lösungsmethode wird danach vorgestellt. Die dann folgenden Abschnitte widmen sich stochastischen dynamischen Optimierungsproblemen. Es werden zuerst allgemeine stochastische Probleme betrachtet und anschließend Markovsche Entscheidungsprozesse, zu de-

nen optimale Entscheidungssequenzen numerisch berechnet werden können, vorgestellt. Für Markovsche Entscheidungsprozesse betrachten wir optimale Entscheidungen über endliche und unendliche Zeithorizonte, sowie das spezielle Problem des optimalen Stoppens, welches im Kontext von Gewinnspielen und Finanzentscheidungen Anwendung findet. Das Kapitel schließt mit einer Übersicht über weitergehende Ansätze aus dem Bereich der dynamischen Optimierung.

12.1 Beispiele und Einführung

In diesem Abschnitt werden typische Beispiele für dynamische Optimierungsprobleme vorgestellt.

12.1.1 Ein Lagerhaltungsproblem

Wir betrachten hier eine sehr einfache Variante aus der Klasse der Lagerhaltungs- oder Bestellprobleme. Wir nehmen an, dass es nur ein einziges Gut gibt, das jeweils zu Beginn einer Periode geliefert wird und direkt nach der Belieferung unter Umständen ausgeliefert wird. Das Modell wird über n Perioden untersucht. Sei $u_j \geq 0$ die Liefermenge in Periode j ($j = 1, \ldots, n$) und r_j die Nachfrage (= Auslieferungsmenge) in Periode j. Der Lagerbestand zu Beginn von Periode j, vor Belieferung und Auslieferung der Güter, wird mit x_j bezeichnet. Es gilt damit folgende Gleichung

$$x_{j+1} = x_j + u_j - r_j$$

mit der Nebenbedingung $x_j \geq 0$. Wir nehmen ferner an, dass das Lager zu Beginn leer ist und auch am Ende leer sein soll, also $x_1 = x_{n+1} = 0$. Für eine Optimierung benötigen wir eine Kostenfunktion. Es fallen Kosten $h_B(u_j)$ für die Belieferung mit u_j Einheiten in der $j$ten Periode an. Ferner fallen Kosten $h_L(x_j)$ für die Lagerung von x_j Einheiten in der $j$ten Periode an. Seien $g_j(u_j, x_j)$ die Kosten der $j$ten Periode. Ziel ist die Festsetzung der Werte u_j, so dass die Kosten bei gegebener Nachfrage minimal werden und die Nebenbedingungen eingehalten werden. Formal lautet die Problemstellung

$$\min_{u_j \geq 0} \left(\sum_{j=1}^{n} g_j(x_j, u_j) \right) \text{ udN } x_{j+1} = x_j + u_j - r_j,\ x_1 = x_{n+1} = 0,\ x_j, u_j, r_j \geq 0 \ .$$

Es muss also sichergestellt sein, dass immer genügend Ware im Lager ist bzw. geliefert wird, um die Nachfrage abzudecken. Da die Liefermengen vorab bekannt sind, handelt es sich eigentlich nicht um ein dynamisches Problem, es kann aber mit Methoden der dynamischen Programmierung gelöst werden.

12.1.2 Ein Erneuerungsproblem

Auch das zweite behandelte Problem ist eine sehr einfache Variante einer bedeutsamen Klasse von Problemen. Es geht um die Erneuerung von Komponenten in einem System, um hohe Wartungskosten oder gar Systemausfälle zu vermeiden. Werden Komponenten zu früh ausgetauscht, so ist dies mit hohen Kosten verbunden. Werden sie nicht früh genug ausgetauscht, so entstehen hohe Wartungskosten und Kosten durch Systemstillstand, wenn die Komponente während des laufenden Betriebs ausfällt und ersetzt werden muss. Erneuerungsprobleme können sehr komplex sein, da in der Regel mehrere Komponenten in einem System existieren, die Kosten für eine Ersetzung stark zustandsabhängig sind und der Zustand einer Komponente nur statistisch beschrieben werden kann.

Wir betrachten hier eine sehr einfache Variante, bei der wir von einer Komponente und diskreten Erneuerungszeitpunkten ausgehen. Der Planungszeitraum ist wieder endlich und umfasst n Perioden. Zu Beginn jeder Periode wird die Komponente ersetzt oder bleibt erhalten. Die Variable u_j gibt an, ob die Komponente zu Beginn der $j$ten Periode ersetzt wird. Falls $u_j = 1$ so wird die Komponente ersetzt, $u_j = 0$ zeigt an, dass sie nicht ersetzt wurde. Sei x_j das Alter der Komponente am Ende von Periode j, es gelte $x_1 = 1$, d.h. wir beginnen mit einer neuen Komponente. Ferner gilt dann

$$x_{j+1} = x_j(1 - u_j) + 1 \ .$$

Die auftretenden Kosten setzen sich aus den Kosten für den Ersatz der Komponente $h_E(x_j)$ und den Wartungs- und Betriebskosten $h_W(x_j)$ zusammen. Beide Kosten hängen vom Alter der Komponente ab. Da der Restwert älterer Komponenten geringer ist und die Betriebskosten höher sind, kann man davon ausgehen, dass die Kosten mit dem Alter steigen. In unserem Fall ergeben sich folgende Kosten in Periode j

$$g(x_j, u_j) = u_j h_E(x_j) + (1 - u_j) h_W(x_j) \ .$$

Ziel ist die Minimierung der Gesamtkosten über das Betriebsintervall $[0, n]$. Die Ersatzzeitpunkte j sollen also so gewählt werden, dass die Gesamtkosten minimal sind. Formal lässt sich das Problem wie folgt formulieren.

$$\min_{u_j \in \{0,1\}} \left(\sum_{j=1}^{n} g(x_j, u_j) \right) \text{ udN } x_{j+1} = x_j(1 - u_j) + 1, \ x_1 = 1, u_j \in \{0, 1\} \ (j = 1, \ldots, n)$$

12.1.3 Das Rucksackproblem

Das Rucksackproblem haben wir bereits in Abschnitt 11.3.1 detailliert behandelt. Durch die Darstellung als Entscheidungsbaum (siehe Abb. 11.4) wurde bereits eine Ordnung auf den Gegenständen eingeführt. Von dieser Ordnung ausgehend formulieren wir nun ein dynamisches Optimierungsproblem. Die Problemstellung lautet wie folgt: Es liegen n Gegenstände mit Gewichten $a_j > 0$ und Werten $c_j > 0$ vor. Der Rucksack darf ein Maximalgewicht von A haben und soll so gepackt werden, dass der Wert der Gegenstände im Rucksack maximal wird.

Die (i.d.R. willkürliche) Ordnung der Gegenstände nutzen wir für die künstliche Dynamisierung des Problems. In Periode j wird entschieden, ob Gegenstand j eingepackt wird. Die binäre Variable u_j beschreibt dies ($u_j = 1$ Gegenstand wird eingepackt, $u_j = 0$ Gegenstand wird nicht eingepackt). Ferner wird eine Variable $x_j$definiert, die die Gewichtsreserve im Rucksack vor der Entscheidung über den $j$ten Gegenstand beschreibt. Es gilt damit

$$x_{j+1} = x_j - u_j a_j \text{ und } x_1 = A \ .$$

Damit kann folgendes Optimierungsproblem definiert werden.

$$\max_{u_j \in \{0,1\}} \left(\sum_{j=1}^{n} c_j u_j \right) \text{ udN } x_{j+1} = x_j - u_j a_j, \ x_1 = A, \ x_j \geq 0, u_j \in \{0, 1\} \ (j = 1, \ldots, n) \ .$$

Damit hat auch das Rucksackproblem eine Darstellung, die mittels dynamischer Optimierung behandelt werden kann.

12.1.4 Ein Maschinenenbelegungsproblem

Maschinenbelegungsprobleme (bzw. Schedulingprobleme) wurden in Abschnitt 11.3.2 detailliert behandelt. Hier wird anhand eines konkreten Beispiels gezeigt, wie sich diese Probleme als dynamische Programmierungsprobleme auffassen lassen. Wir betrachten dazu folgendes Beispiel aus Bertsekas [2005]: Es müssen 4 Operationen *A, B, C, D* durchgeführt werden. *A* muss vor *B* und *D* immer nach *B* durchgeführt werden. Für die Bearbeitungsschritte steht eine Maschine zur Verfügung, die zwischen den Schritten umgerüstet werden muss. Für die Umrüstung von Schritt x nach y ($x, y \in \{A, B, C, D\}$) entstehen Kosten $K_{x,y}$. Ferner entstehen für die initiale Einrichtung der Maschine für Produkt *A* oder *C* (dies sind die einzigen Schritte mit denen begonnen werden kann) Kosten S_A bzw. S_C.

Die Zustandsbeschreibung für dieses Problem ist deutlich komplexer, da der Zustand die bisherigen Bearbeitungsschritte beinhalten muss. Zum Zeitpunkt 0 entspricht der Zustand $\emptyset$. Der Zustand zum Zeitpunkt $j \leq 4$ ist eine Vektor der Länge j, dessen Elemente aus $\{A, B, C, D\}$ gewählt wurden. Es gilt $x_{j+1} = (x_j(1), \ldots, x_j(j), u_j)$. Da in einem gültigen Schedule alle Arbeitsschritte genau einmal vorkommen dürfen und die gerade definierten Abhängigkeiten einzuhalten sind, hängen die Wahlmöglichkeiten u_j von

x_j ab. Sei $U_j(x_j)$ die Menge der möglichen Entscheidungen (d.h. nächsten Arbeitsschritte im Zustand x_j). Es gilt

$$\begin{array}{ll} A \in U_j(x_j) & \text{falls } \forall i = 1, \ldots, j : x_j(i) \neq A \\ B \in U_j(x_j) & \text{falls } \forall i = 1, \ldots, j : x_j(i) \neq B \wedge \exists i = 1, \ldots, j : x_j(i) = A \\ C \in U_j(x_j) & \text{falls } \forall i = 1, \ldots, j : x_j(i) \neq C \\ D \in U_j(x_j) & \text{falls } \forall i = 1, \ldots, j : x_j(i) \neq D \wedge \exists i = 1, \ldots, j : x_j(i) = B \end{array}$$

Damit können wir folgendes Optimierungsproblem formulieren

$$\min_{u_j \in U_j(x_j)} \left(S_{u_1} + \sum_{j=2}^{4} K_{x_j(j-1), u_j} \right).$$

Die Nebenbedingungen sind in der Definition der Mengen $U_j(x_j)$ kodiert. Das einfache Problem zeigt, dass die formale Beschreibung dynamischer Optimierungsprobleme sehr komplex werden kann, wenn Nebenbedingungen auftreten.

12.2 Problemstellung der dynamischen Optimierung

Die bisher vorgestellten Beispiele waren alle ähnlich, weisen aber spezifische Unterschiede auf. Bevor wir uns der Optimierung widmen, soll eine allgemeine Problemformulierung definiert werden. Die Formulierung dynamischer Optimierungsprobleme ist zwar oft sehr problemspezifisch, es lässt sich aber eine recht breit einsetzbare Formulierung finden, die viele, wenn auch bei weitem nicht alle, mit der dynamischen Optimierung behandelbaren Problemstellungen abdeckt. Wir machen dazu folgende Annahmen:

- Das Verhalten des Systems lässt sich in diskreten Schritten/Stufen/Perioden beschreiben.
- Die Systembeobachtung erfolgt über einen endlichen Planungszeitraum, der die Perioden $j = 1, \ldots, n$ umfasst[1].
- Zu Beginn von Periode j (d.h. direkt nach dem Ende von Periode $j-1$) befindet sich das System im Zustand x_j.
- Der Zustand zu Beginn der Planungsperiode $x_1 = x_a$ ist gegeben.
- In Periode j wird die Entscheidung u_j getroffen (u_j ist die Entscheidungs- oder Steuerungsvariable).
- Entscheidung u_j im Zustand x_j resultiert in Folgezustand $x_{j+1} = f_j(x_j, u_j)$.
- Es treten Kosten/Erlöse $g_j(x_j, u_j)$ auf.

Es ist zu beachten, dass Folgezustand und Kosten jeweils nur von der Periode, dem aktuellen Zustand und der getroffenen Entscheidung abhängen dürfen. Eine potenzielle Lösung muss unter Umständen Restriktionen berücksichtigen. $U_j(x_j) \neq \emptyset$ sei der (nichtleere) Steuerungsbereich, wenn das System in Periode j im Zustand x_j ist. X_j ist die Menge potenzieller Zustände in Periode j, die wie folgte definiert ist:

$$X_1 = \{x_1\} \text{ und } X_{j+1} = \bigcup_{x_j \in X_j} \bigcup_{u_j \in U(x_j)} \{f_j(x_j, u_j)\} \quad (j = 1, \ldots, n)$$

Wir nehmen an, dass $X_j \neq \emptyset$ gilt. Dies bedeutet, dass die Funktionen g_j, f_j auf
$D_j = \{(x, u) | x \in X_j, u \in U_j(x)\}$ entsprechend definiert sind. Damit kann das Optimierungsproblem formuliert werden.

$$\min_{u_j \in U_j(x_j)} (g_j(x_j, u_j)) \text{ udN } x_{j+1} = f_j(x_j, u_j), \; x_1 = x_a$$

Auch wenn die Darstellung relativ einfach aussieht, kann sie bei entsprechender Definition der Funktionen g_j und f_j sehr komplex werden. Für unsere Beispielmodelle fasst Tabelle 12.1 die Zustands- und Entscheidungsvariablen zusammen.

[1] Wir werden uns später in diesem Kapitel auch mit dem Fall $n = \infty$ befassen.

	Lagerhaltung	Erneuerung	Rucksack	Maschinenbelegung
x_j X_j	reellwertig Bestand $\mathbb{R}_+$für $j = 2, \ldots, n$ $X_0 = X_{n+1} = \{0\}$	ganzzahlig Alter $\{1, \ldots, j-1\}$ für $j = 1, \ldots, n+1$, $X_1 = \{1\}$	reellw. Kapazität $[0, A]$ für $j = 2, \ldots, n+1$ $X_1 = \{A\}$	Vektor über $\{A..D\}$ Vektor der Länge j $X_0 = \emptyset$
u_j $U_j(x_j)$	reellwertig $\mathbb{R}_+$	binär $\{0, 1\}$	binär $\{0, 1\}$	Element aus $\{A..D\}$ abhängig von x_j
$f_j(x_j, u_j)$ $g_j(x_j, u_j)$	$x_j + u_j - r_j$ $h_B(u_j) + h_L(x_j)$	$x_j(1-u_j)+1$ $u_j h_E(x_j)$ $+(1-u_j)h_W(x_j)$	$x_j - a_j u_j$ $-c_j u_j$	$(x_j(1), \ldots, x_j(j), u_j)$ $g_{j-1}(x_{j-2}, u_{j-1})$ $+K_{u_{j-1}, u_j}$
	Kosten	Kosten	Nutzen (maximieren!)	Kosten

Tabelle 12.1: Struktur der Beispielprobleme (Neumann and Morlock [1993, S. 509]).

Bei der Entscheidungsfindung wird schrittweise vorgegangen. Am Ende einer Periode wird eine Entscheidung getroffen, die den nachfolgenden Zustand und damit die zukünftige Entwicklung des Modells beeinflusst.

Die beschriebene Problemstellung deckt nicht den allgemeinsten Fall ab. So könnte man sich folgende Generalisierungen vorstellen:

- f_j und g_j dürfen nur vom aktuellen Zustand und der aktuellen Entscheidung abhängig sein. Man bezeichnet dies auch als Markovsches Modell. Durch eine entsprechende Definition des aktuellen Zustandes lässt sich aber die Vergangenheit im Zustand kodieren, so dass die Sichtweise nur wenig einschränkt. Der Preis kann allerdings ein sehr komplexer Zustandsraum sein. Ein Beispiel für eine solche Zustandskodierung ist das Maschinenbelegungsproblem, bei dem der Zustand alle bisherigen Entscheidungen widerspiegelt.

- f_j und g_j sind deterministisch definiert. Die Einbeziehung stochastischer Funktionen führt zu stochastischen Modellen, wie wir sie in Abschnitt 12.4 behandeln werden.

- Die Betrachtung der Zeit ist eingeschränkt. So wird ein zeitdiskretes Modell vorausgesetzt, bei dem nur zu diskreten Zeitpunkten Änderungen eintreten. Eine allgemeinere Sichtweise erlaubt Zustandswechsel und Steuerungsentscheidungen zu beliebigen Zeitpunkten. Diesen Aspekt berücksichtigen wir hier nicht.

- Die vorgestellten Beispiele betrachten Systeme über einen endlichen Zeitraum (endlicher Horizont). Man könnte sich auch vorstellen unendliche Zeiträume (unendliche Horizonte) zu untersuchen, muss dann aber die Kostenberechnung anders formulieren. Wir werden dies in Abschnitt 12.6 tun.

Wir bleiben bei unserem einfachen Modell und betrachten eine Entscheidungsfolge (Politik, Steuerung) $u_1, u_2, \ldots, u_n$, die zu einer Zustandsfolge $x_1, x_2, \ldots, x_n, x_{n+1}$ führt. Eine Politik heißt *zulässig*, wenn sie den Nebenbedingungen genügt und eine zulässige Lösung generiert. Eine Politik heißt *optimal*, wenn sie (die Existenz vorausgesetzt) zulässig ist und das Optimierungsziel erreicht. D.h. eine optimale Politik erreicht einen Funktionswert, der von keiner anderen Politik unterschritten wird.

Bei bekannten Funktionen f_j und g_j hängt die optimale Lösung, sofern sie überhaupt existiert, vom Anfangszustand x_1 ab. Wir benutzen deshalb die Bezeichnung $P_1(x_1)$ für das Gesamtproblem. Ebenso kann man Problem $P_j(x_j)$ definieren, indem man nur die Perioden $i = j, \ldots, n$ betrachtet und in Periode j vom Zustand x_j startet. Da f_j und g_j nur vom aktuellen Zustand und der getroffenen Entscheidung abhängen, hängt die Lösung $P_j(x_j)$ nur vom Startzustand x_j und den Entscheidungen in den Perioden $i = j, \ldots, n$ ab. Dies Erkenntnis führt zu dem folgenden Satz.

Satz 26 *Existiert für ein Teilproblem $P_j(x_j)$ eine optimale Politik $(u_j^*, u_{j+1}^*, \ldots, u_n^*)$ mit resultierendem Minimalwert der Zielfunktion $v_j^*(x_j)$, dann ist $(u_{j+1}^*, \ldots, u_n^*)$ eine optimale Politik für das Problem $P_{j+1}(x_{j+1}^*)$ mit $x_{j+1}^* = f_j(x_j, u_j^*)$ und $v_{j+1}^*(x_{j+1}^*) = v_j^*(x_j) - g_j(x_j, u_j^*)$ ist der zugehörige Wert der Zielfunktion.*

Beweis: Für den Beweis nehmen wir an, dass für $P_{j+1}(x_{j+1})$ eine Politik $(u_{j+1}^{'}, \ldots, u_n^{'})$ existiere, die zu einem Funktionswert $v_{j+1}^{'}(x_{j+1}^*) < v_{j+1}^*(x_{j+1}^*)$ führt. Da $v_j^*(x_j) = g_j(x_j, u_j^*) + v_{j+1}^*(x_{j+1}^*)$ würde auch $v_j^{'}(x_j) = g_j(x_j, u_j^*) + v_{j+1}^{'}(x_{j+1}^*) < g_j(x_j, u_j^*) + v_{j+1}^*(x_{j+1}^*) = v_j^*(x_j)$ gelten und $(u_j^*, u_{j+1}^*, \ldots, u_n^*)$ kann nicht optimal sein, was zum Widerspruch unserer Annahme führt. □

Der folgende Satz ist die Basis für die Optimierung, wie sie im nächsten Abschnitt vorgestellt wird.

Satz 27 (Bellmansches Optimalitätsprinzip) *Sei $(u_1^*, \ldots, u_j^*, \ldots, u_n^*)$ eine optimale Politik für $P_1(x_1)$ und x_j^* der Zustand, der zu Beginn von Periode j unter diese Politik erreicht wird, dann ist $(u_j^*, \ldots, u_n^*)$ eine optimale Politik für $P_j(x_j^*)$. Dies bedeutet, dass optimale Folgeentscheidungen ab Periode j unabhängig von Vorentscheidungen vor Periode j sind, allerdings vom Anfangszustand x_j abhängen.*

Beweis: Der Beweis für den Satz folgt wieder aus der Darstellung $v_j^*(x_j^*) = g_j(x_j^*, u_j^*) + v_{j+1}^*(x_{j+1}^*)$ und der damit verbundenen Minimalität von $v_j^*(x_j^*)$ und $v_{j+1}^*(x_{j+1}^*)$. □

Mit Hilfe des vorherigen Satzes kann die Bellmansche Funktionsgleichung aufgestellt werden. Es gilt

$$v_j^*(x_j) = \min_{u \in U_j(x_j)} \big(g_j(x_j, u_j) + v_{j+1}^*(f_j(x_j, u_j))\big) \text{ mit } x_j \in X_j,\ 1 \le j \le n+1,\ v_{n+1}(x_{n+1}) = 0 \quad (12.1)$$

Auf Basis dieser Beziehung kann für jedes $x_j \in X_j$ $v_j^*(x_j)$ bestimmt werden, sofern $v_{j+1}^*(x_{j+1})$ bekannt ist. Damit kann rekursiv das Optimum in der ersten Periode bestimmt werden, was letztendlich die Grundlage der Bellmanschen Funktionsgleichungsmethode im nächsten Abschnitt ist. Bevor wir dazu kommen, betrachten wir einige Modifikationen, die sich leicht integrieren lassen. Offensichtlich kann das Minimum durch das Maximum ersetzt werden, dann beschreibt (12.1) die Maximierung. In ähnlicher Weise kann in der Zielfunktion die Summe durch das Produkt ersetzt werden und man erhält bei einer Maximierung

$$v_j^*(x_j) = \max_{u_j \in U_j(x_j)} \big(g_j(x_j, u_j) \cdot v_{j+1}^*(f(x_j, u_j))\big) \ .$$

Diese Darstellung wird u.a. benutzt, um die Zuverlässigkeit von Bauteilen in einer Serienschaltung zu maximieren. Weitere Erweiterungen wären die Betrachtung einer Menge von Anfangszuständen X_1 sowie das Hinzufügen von initialen Kosten für den Anfangszustand und terminalen Kosten für den Endzustand.

12.3 Bellmansche Funktionsgleichungsmethode

Auf Basis der Bellmanschen Funktionsgleichung (12.1) lässt sich nun ein Lösungsansatz entwickeln, den wir in zwei unterschiedlichen Varianten vorstellen. Im Folgenden wird vorausgesetzt, dass eine optimale Lösung existiert. Diese muss aber nicht eindeutig sein.

Variante I:

Algorithmus 1 (Rückwärtsentwicklung)
Initialisiere $v_{n+1}^*(x_{n+1}) = 0$ für alle $x_{n+1} \in X_{n+1}$;
for ($j = n, n-1, \ldots, 1$ do
berechne $v_j^*(x_j) = \min_{u_j \in U_j(x_j)} \big(g_j(x_j, u_j) + v_{j+1}^*(f(x_j, u_j))\big)$
und $z_j^*(x_j) = \arg\min_{u_j \in U_j(x_j)} \big(g_j(x_j, u_j) + v_{j+1}^*(f(x_j, u_j))\big)$ für alle $x_j \in X_j$;

Die Rückwärtsentwicklung liefert für jeden Zustand x_j eine Entscheidung $z_j^*(x_j)$ die zu einer optimalen Lösung ausgehend von x_j führt. Diese Entscheidung muss nicht eindeutig sein, es wird aber mittels arg min eine der möglichen Entscheidungen gespeichert. Um die optimale Zustandsfolge $(x_1^*, \ldots, x_n^*)$ und die optimale Politik $(u_1^*, \ldots, u_n^*)$ zu ermitteln wird die folgende Vorwärtsentwicklung genutzt.

Algorithmus 2 (optimale Rückkopplungssteuerung)
Wähle $x_1^* = x_a$;
for ($j = 1, \ldots, n$ do
$u_j^* = z_j^*(x_j^*)$ und $x_{j+1}^* = f_j(x_j^*, u_j^*)$;

Die errechnete Zustandsfolge und Politik ist optimal für das Problem $P_1(x_a)$. In dieser Variante müssen für jeden Zustand der optimale Funktionswert und die optimale Entscheidung gespeichert werden.

Variante II: Wie in der ersten Variante werden in einer Rückwärtsphase die Werte $v_j^*(x_j)$ für $j = n+1, \ldots, 1$ und $x_j \in X_j$ bestimmt. $v_1^*(x_a)$ ist dann der minimale Funktionswert. Zur Ermittlung der optimalen Politik muss in der anschließenden Vorwärtsphase ausgehend von $v_1^*(x_a)$ jeweils die optimale Entscheidung und der daraus resultierende Zustand ermittelt werden.

Also $u_j^* = \arg\min_{u_j \in U_j(x_j)} \left(g_j(x_j^*, u_j) + v_{j+1}^*(f_j(x_j, u_j))\right)$ und $x_{j+1}^* = f_j(x_j^*, u_j^*)$.

Variante I benötigt etwas mehr Speicherplatz, da die optimalen Entscheidungen neben den optimalen Funktionswerten gespeichert werden. In beiden Versionen werden keine Annahmen bzgl. der Funktionen f_j und g_j gemacht.

Im Prinzip berechnen die beiden Varianten des Algorithmus einen kürzesten Weg durch den Graph der alle möglichen Abläufe beschreibt. Die Komplexität hängt von der Knotenzahl und der Zahl der aus jedem Zustand ausgehenden Kanten ab, nicht aber von der Anzahl der Pfade im Graph. Dies liegt darin begründet, dass das Bellmansche Optimalitätsprinzip gilt und Optimalität lokal festgestellt werden kann. Das Problem des Findens kürzester Wege in einem gerichteten Graphen mit Kantengewichten lässt sich so auch als dynamisches Optimierungsproblem formulieren und mit den vorgestellten Algorithmen lösen. Dieser Ansatz ist aber in den meisten Fällen weniger effizient als die Nutzung spezifischer Algorithmen für dieses Problem (siehe auch Bertsekas [Kap. 2 2005]). Wir betrachten zwei kleine Beispiele aus dem Bereich der Lagerhaltung.

Das nun folgende Beispiel wurde aus Domschke and Drexl [1993] entnommen. Wir betrachten ein einfaches Bestellmengenproblem, bei dem nur ein einziges Gut bestellt und ausgeliefert werden soll. Das System soll über $n = 4$ Perioden analysiert werden. Es steht ein Lager der Kapazität 2 zur Verfügung, das zu Beginn leer ist. Es können ferner maximal 2 Einheiten des Guts pro Periode geliefert werden. Die Kosten pro Einheit in Periode j lauten q_j, es treten keine Kosten für die Lagerung auf. Der Bedarf in Periode j beträgt b_j. Die Werte für q_j und b_j können aus der folgenden Tabelle entnommen werden.

Periode j	1	2	3	4
Preis q_j	7	9	12	10
Bedarf b_j	1	1	1	1

Der Steuerbereich lautet damit $U_j(x_j) = \{0, 1, 2\}$ und wird u.U. durch Lagerbeschränkungen oder Bestellungen weiter eingeschränkt. Der Zustandsraum kann erst einmal grob als $X_j = \{0, 1, 2\}$ festgelegt werden, wobei abweichend $X_0 = X_5 = \{0\}$ gelten sollte. Die Zustandsübergangsfunktion und die Kostenfunktion lauten

$$f_j(x_j, u_j) = x_j + u_j - b_j \text{ und } g_j(x_j, u_j) = u_j q_j \ .$$

Damit ist das Problem vollständig spezifiziert und der vorgestellte Algorithmus kann angewendet werden.

Dynamische Optimierungsprobleme lassen sich mittels attributierter Digraphen visualisieren. Dazu werden die Zustände einer Periode in einer Spalte als Knoten dargestellt. Kanten beschreiben jeweils den Übergang von x_j in Spalte j zu $f_j(x_j, u_j)$ in Spalte $j+1$. Die Kanten sind mit u/g beschriftete, wobei $u = u_j$ und $g = g_j(x_j, u_j)$ gilt.

Die resultierenden Graphen werden natürlich schnell sehr groß. Für unser einfaches Beispiel entsteht aber eine übersichtliche Darstellung. Abbildung 12.1 zeigt den Graphen. Der optimale Pfad ausgehend vom Startzustand fett und in rot gedruckt. Es ist also am besten, in den ersten 2 Perioden jeweils 2 Teil zu ordern und diese danach abzubauen. Die Gesamtkosten sind dann $2 \cdot 7 + 2 \cdot 9 = 32$. Die jeweils optimalen Entscheidungen ausgehen von einem beliebigen Knoten im Graphen sind jeweils fett gedruckt.

Die Berechnung der optimalen Lösung in einem deterministischen dynamischen Programmierungsproblem entspricht der Berechnung des kürzesten Pfades in einem azyklischen Graphen mit Kantengewichten. Die Komplexität ergibt sich aus der Größe des Graphen. Das Rucksackproblem zeigt, dass Probleme exponentiellen Aufwand erfordern können. Der Zusammenhang zwischen dynamischer Programmierung und kürzesten Wegen in Graphen bedingt, dass sich jedes kürzeste Wege Problem in einem Graphen mit nicht negativen Kosten als dynamisches Programmierungsproblem formulieren und mit den vorgestellten Algorithmen lösen lässt. In vielen Fällen gibt es allerdings effizientere Algorithmen zur Bestimmung kürzester Wege in Graphen.

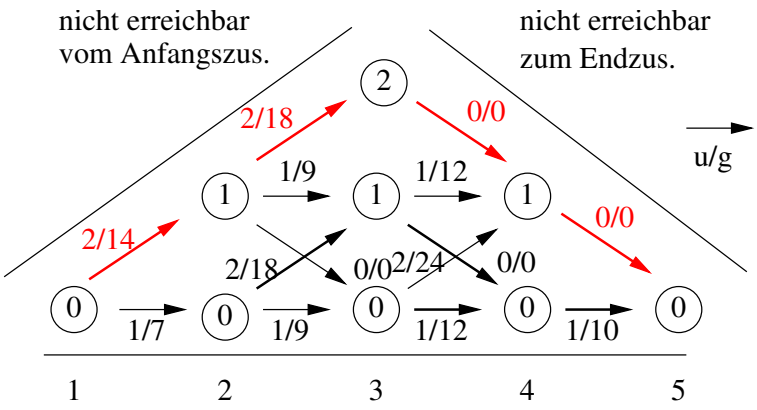


Abbildung 12.1: Zustandsübergangsgraph für das Beispielmodell.

Das zweite Beispiel wurde aus Neumann and Morlock [1993] entnommen und zeigt eine allgemeinere Problemstellung, für deren Lösung der Algorithmus etwas erweitert werden muss. Auch dieses Beispiel ist aber sehr einfach und stark idealisiert. Das Beispiel untersucht die Verkaufsplanung für ein Gut, das beliebig teilbar ist. Ein Flussgut wie Öl oder Gas wären Beispiele für mögliche Güter dieser Art. Die Verkaufsplanung soll über $n = 3$ Perioden erfolgen, es kann zu Beginn jeder Periode j eine Menge $u_j \in \mathbb{R}_+$ verkauft werden. Wenn x_j die zu Beginn von Periode j verfügbare Menge ist, so ist $U_j(x_j) = [0, x_j]$. Die Verfügbarkeitsmenge des Gutes verdoppelt sich in einer Periode, so dass

$$x_{j+1} = f_j(x_j, u_j) = 2 \cdot (x_j - u_j) \ .$$

Zu Anfang sei genau eine Einheit unseres Gutes vorhanden, d.h. $x_1 = 1$. Die Erlöse ergeben sich aus

$$g_j(x_j, u_j) = \sqrt{u_j} \ ,$$

wachsen also unterlinear mit der Verkaufsmenge. Ziel ist es, den Verkaufserlös $\sqrt{u_1} + \sqrt{u_2} + \sqrt{u_3}$ zu maximieren. Die Maximierungsbedingung der Bellmanschen Funktionsgleichungsmethode lautet dann

$$v_j^*(x_j) = \max_{u_j \in U_j(x_j)} \left(g_j(x_j, u_j) + v_{j+1}^*(f(x_j, u_j))\right) = \max_{0 \le u_j \le x_j} \left(\sqrt{u_j} + v_{j+1}^*(2 \cdot (x_j - u_j))\right) \ .$$

Im Gegensatz zum vorherigen Beispiel können nun nicht mehr alle Zustände aufgezählt werden, da der Zustandsraum in den Perioden $j > 1$ überabzählbar ist. Trotzdem ist die Bellmansche Funktionsgleichung gleichwohl zur Optimierung einsetzbar.

Bevor wir zur Optimierung kommen, ist eine kurze Vorüberlegung angebracht. Sei

$$w(u) = \sqrt{u} + \sqrt{c \cdot (x - u)} \text{ mit } c > 1, \ 0 \le u \le x \text{ und } x \ge 0 \text{ (fest gewählt)} \ .$$

Die Funktion $w(u)$ beschreibt den Verlauf der Erlöse, wenn u Einheiten sofort und der Rest zu Beginn der nächsten Periode verkauft wird. Abbildung 12.2 zeigt den Verlauf von $w(u)$ und der beiden Summanden, die die Funktion bilden. Offensichtlich ist $w(u)$ streng konkav (dies gilt für alle $x > 0$ und $c > 1$) und besitzt damit ein Maximum auf $[0, x]$. Die Extremstelle u^* kann durch Nullsetzen der ersten Ableitung bestimmt werden.

$$w'(u) = \frac{1}{2\sqrt{u}} - \frac{c}{2\sqrt{c(x-u)}} \ \Rightarrow \ c\sqrt{u} = \sqrt{c(x-u)} \ \Rightarrow \ u^* = \frac{x}{c+1}$$

Ferner gilt

$$w(u^*) = \sqrt{\frac{x}{c+1}} + \sqrt{c\frac{x(c+1)-x}{c+1}} = \sqrt{(c+1)x} \ .$$

Damit kann Variante I des Algorithmus eingesetzt werden.

Rückwärtsentwicklung

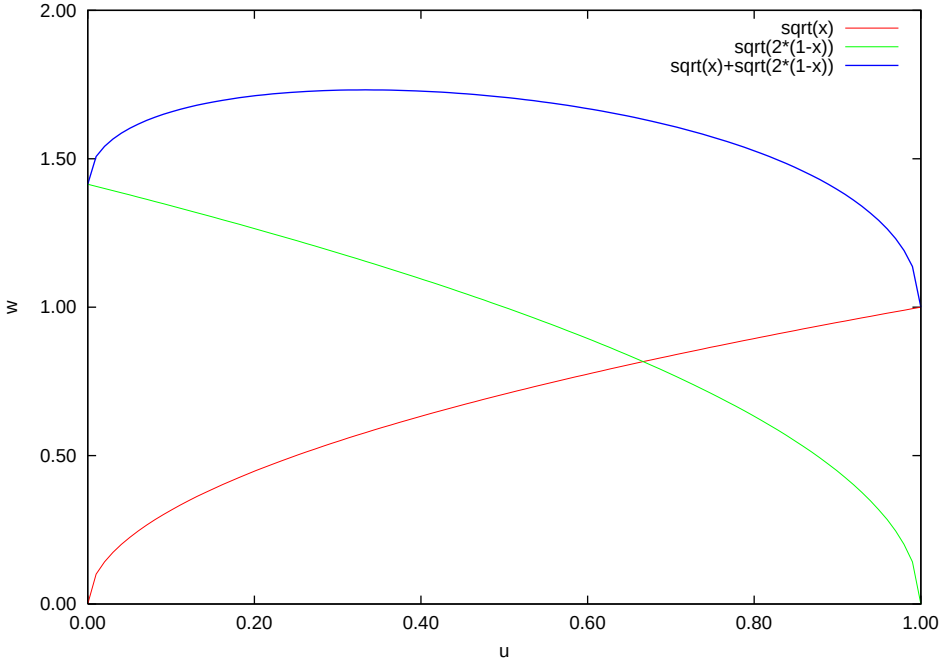


Abbildung 12.2: Verlauf der Funktion $w(u)$ für $x = 1$ und $c = 2$.

- Periode 3:
  $v_4^*(x_4) = 0 \;\Rightarrow\; v_3^*(x_3) = \max_{0 \le u_3 \le x_3} \left(\sqrt{u_3} + 0\right)$
  daher gilt $v_3^*(x_3) = \sqrt{x_3},\; z_3^*(x_3) = x_3$ (alles verkaufen)

- Periode 2:
  $v_2^*(x_2) = \max_{0 \le u_2 \le x_2} \left(\sqrt{u_2} + \sqrt{2(x_2 - u_2)}\right)$
  nach den Vorüberlegungen gilt für die Maximalstelle (mit $c = 2$)
  $v_2^*(x_2) = \sqrt{3x_2},\; z_2^*(x_2) = \frac{x_2}{3}$

- Periode 1:
  $v_1^*(x_1) = \max_{0 \le u_1 \le x_1} \left(\sqrt{u_1} + \sqrt{6(x_1 - u_1)}\right)$
  nach den Vorüberlegungen gilt für die Maximalstelle (mit $c = 6$)
  $v_1^*(x_1) = \sqrt{7x_1},\; z_1^*(x_1) = \frac{x_1}{7}$

Vorwärtsentwicklung

$$
\begin{array}{lll}
x_1^* = 1 & \Rightarrow\; u_1^* = 1/7 & \rightarrow \\
x_2^* = 2(x_1^* - u_1^*) = 12/7 & \Rightarrow u_2^* = x_2^*/3 = 4/7 & \rightarrow \\
x_3^* = 2(x_2^* - u_2^*) = 16/7 & \Rightarrow u_3^* = x_3^* = 16/7 & \rightarrow \\
x_4^* = 2(x_3^* - u_3^*) = 0 & &
\end{array}
$$

Damit lautet die optimale Politik $(1/7, 4/7, 16/7)$, diese führt zur optimalen Zustandsfolge $(1, 12/7, 16/, 0)$ und dem optimalen Ergebnis $v_1^*(1) = \sqrt{7} = 2.65$.

Das Beispiel zeigt, dass sich auch Probleme mit dem dynamischen Programmierungsansatz lösen lassen, deren Zustandsraum nicht endlich und nicht einmal abzählbar ist. In diesen Fällen ist das Standardverfahren, das erst den Zustandsraum aufbaut nicht anwendbar. Für das vorherige Beispiel konnten wir uns zu Nutze machen, dass die optimale Verkaufsmenge symbolisch berechenbar ist und somit die Rückwärtsberechnung ohne Kenntnis des Zustandsraums symbolisch erfolgen kann. Dies ist einer von vielen problemspezifische Tricks, die angewendet werden, um dynamische Programmierungsprobleme zu lösen.

Im allgemeinen Fall mit kontinuierlichem Zustandsraum, bei dem das Minimum sich nicht symbolisch berechnen lässt, sind in den meisten Fällen nicht exakt sondern nur approximativ zu lösen. Wir werden die zugehörigen Methoden kurz im Kontext der stochastischen dynamischen Programmierung betrachten.

12.4 Stochastische dynamische Programmierung

Nachdem wir uns bisher mit deterministischen Problemstellungen befasst haben, betrachten wir nun Probleme, die stochastische Komponenten beinhalten. Dazu wird zuerst die allgemeine Problemstellung definiert. Wir werden sehen, dass in diesen Fällen die Bestimmung der Lösung sehr schwierig sein kann und deshalb oft auf Approximationen bzw. Simulation zurückgegriffen wird. Deshalb betrachten wir in den nachfolgenden Kapiteln eine eingeschränkte Klasse von Problemen, die eine einfachere Optimierung erlauben aber gleichwohl praktisch breit eingesetzt werden, nämlich die Markovschen Entscheidungsprobleme.

Im stochastischen Kontext, kann man zwar die vorgestellten Prinzipien der Bellmanschen Funktionsgleichungen oft übertragen. Trotzdem sind die auftretenden Problemstellungen in der Regel deutlich komplexer. Insbesondere muss man sich vor der Entwicklung eines Lösungsansatzes mit der Messbarkeit der auftretenden Funktionen und Mengen auseinandersetzen. Da an dieser Stelle nur eine kurze Einführung gegeben werden soll, gehen wir davon aus, dass die Funktionen messbar sind und die im Folgenden verwendeten Integrale existieren und ausgewertet werden können. Für weitere Details und die theoretischen Grundlagen sei auf die Literatur verwiesen (Bertsekas [2005, 2007]).

Wenn wir die bisherige Problemstellung um stochastische Elemente erweitern, so wird bei einer Entscheidung u_j im Zustand x_j nicht ein eindeutiger Folgezustand x_{j+1} angenommen, sondern der Folgezustand ist von einem Zufallsparameter w_j abhängig. In gleicher Weise sind die Kosten von einem Zufallsparameter w_j abhängen. Es gilt also $x_{j+1} = f(x_j, u_j, w_j)$ mit Kosten $g(x_j, u_j, w_j)$. Wenn wir davon ausgehen, dass Folgezustand und Kosten von w_j in identischer Weise beeinflusst werden, so reicht es aus, die Kosten vom Folgezustand abhängig zu machen, d.h. $g(x_j, u_j, x_{j+1}) = g(x_j, u_j, f(x_j, u_j, w_j))$. Folgezustände werden über bedingte Verteilungen $pX_j(.|x_j, u_j)$ oder Dichtefunktionen $fX_j(.|x_j, u_j)$ definiert. Dies bedeutet, dass der Zufallseinfluss vom aktuellen Zustand und der Entscheidung, nicht aber von vorherigen Zuständen, Entscheidungen oder Zufallseinflüssen abhängen darf.

So können zum Beispiel in unserem ersten Beispiel die Auslieferungsmengen oder die Belieferungskosten stochastisch schwanken. Diese Schwankungen dürfen von der Periode und dem Lagerbestand abhängen, nicht aber von vorherigen Auslieferungsmengen oder Belieferungskosten, auch wenn dies im Beispiel durchaus begründbar wäre. Die Voraussetzung für die Nutzung der dynamischen Programmierungsansätze ist die Beschränkung der Abhängigkeit auf den aktuellen Zustand und die aktuelle Entscheidung. Allgemein wird Stochastik, wie schon bei der Modellierung erläutert, immer dann eingesetzt, wenn Abläufe deterministisch nicht beschreibbar sind oder Komplexität vermieden werden soll.

Aus dem stochastischen Zustandübergangsverhalten folgt, dass auch der Wert der Zielfunktion eine Zufallsvariable ist. Damit kann i.d.R. kein minimaler Wert an sich durch eine optimale Politik erreicht werden, sondern es können nur Maße der Zufallsvariable minimiert werden. Üblicherweise wird der Erwartungswert minimiert. Gesucht wird damit eine optimale Politik, so dass der Erwartungswert von $V_1(x_1)$ minimal wird. Wie schon im deterministischen Fall eingeführt, werden die Größen der optimalen Politik mit * bezeichnet. Im Gegensatz zum deterministischen Fall folgt aber nun aus einer Entscheidung in einem Zustand nicht mehr automatisch der Folgezustand, so dass die optimale Politik nicht a priori errechnet werden kann, sondern sich nach der aktuellen Trajektorie richten muss. Da Folgezustand und auftretende Kosten weiterhin nur vom aktuellen Zustand, der aktuellen Entscheidung und nun zusätzlich der Realisierung des Zufalls abhängen, kann man das Prinzip der Bellmanschen Funktionsgleichung beibehalten. D.h. $E(V_j^*(x_j))$ hängt von $g_j(x_j, u_j, x_{j+1})$ und $E(V_{j+1}^*(x_{j+1}))$ ab. Für den diskreten Fall ergibt sich damit

$$E(V_j^*(x_j)) = \min_{u_j \in U_j(x_j)} \left(\sum_{x \in X_{j+1}} \left(\left(g_j(x_j, u_j, x) + E(V_{j+1}^*(x)) \right) \cdot pX_j(x|x_j, u_j) \right) \right) .$$

So lange die Summe und die Entscheidungsmenge endlich sind, kann der im vorherigen Abschnitt vorge-

stellte Lösungsalgorithmus zum Einsatz kommen. Im kontinuierlichen Fall gilt

$$E(V_j^*(x_j)) = \min_{u_j \in U_j(x_j)} \left(\int_{x \in X_{j+1}} \left(\left(g_j(x_j, u_j, x) + E(V_{j+1}^*(x)) \right) \cdot fX_j(x|x_j, u_j) \right) dx \right) .$$

Ferner gilt für den Erwartungswert der auftretenden Kosten

$$E(g_j(x_j, u_j)) = \int_{x \in X_{j+1}} (g_j(x_j, u_j, x) \cdot fX_j(x|x_j, u_j))\, dx .$$

Eingesetzt führt dies zu der folgenden Darstellung.

$$E(V_j^*(x_j)) = \min_{u_j \in U_j(x_j)} \left(E(g_j(x_j, u_j)) + \int_{x \in X_{j+1}} \left(E(V_{j+1}^*(x)) \cdot fX_j(x|x_j, u_j) \right) dx \right) . \tag{12.2}$$

Die Auswertung von (12.2) kann sehr aufwändig sein, so dass der allgemeine Fall nur per Simulation analysierbar ist. Dies bedeutet, dass optimale Entscheidungen nicht mehr exakt ermittelt werden können, sondern geschätzt und statistisch validiert werden müssen. Wir werden kurz einige mögliche Lösungsansätze betrachten.

Eine einfache Variante besteht darin, den Nachfolgezustand einfach durch den Erwartungswert zur ersetzen. Wir nehmen dazu an, dass der Nachfolge Zustand durch einen Zufallsparameter w beeinflusst wird, der in Periode j aus einer Menge W_j gewählt wird. Sei $f(x_j, u_j, w)$ eine Funktion, die den Nachfolgezustand bestimmt. Damit das Verhalten nicht verändert wird, muss für alle $Y_{j+1} \subseteq X_{j+1}$gelten

$$\int_{w \in W_j} \delta(f(x_j, u_j, w) \in Y_{j+1}) dw = \int_{x \in Y_{j+1}} fX_j(x|x_{j,u_j}) dx,$$

wobei $\delta(b) = 1$ für b gleich true und $\delta(b) = 0$ für b gleich false. Es gilt dann für den Erwartungswert

$$\bar{x}_{j+1} = \int_{x \in X_{j+1}} x \cdot fX_j(x|x_j, u_j) dx = \int_{w \in W_j} f(x_j, u_j, w) dw$$

Da $\bar{x}_{j+1}$ damit eindeutig bestimmt ist, liegt wieder ein deterministisches Modell vor, das mit dem Methoden aus dem letzten Abschnitt analysiert werden kann. In der Regel ist das beschriebene Vorgehen aber zu grob, da die Zufallseinflüsse einfach aus dem Model entfernt werden. Außerdem ist nicht sicher, dass $\bar{x}_j \in X_j$ gilt.

Ein deutlich besserer aber auch aufwändigerer Lösungsansatz nutzt eine Approximation von $\int_{X_{j+1}} E(V_{j+1}^*(x)) fX_j(x|x_j, u_j) dx$, die wir als $\bar{V}_{j+1}(x_j, u_j)$ bezeichnen wollen. Die Bestimmung von $\bar{V}_j(x_j, u_j)$ muss problemspezifisch erfolgen, es gibt allerdings einige grundsätzliche Vorgehensweisen. Oft werden die approximierten Kosten eines Folgezustands durch die Kosten der nachfolgenden Entscheidungen beschrieben. Im einfachsten Fall gilt damit

$$\bar{V}_j(x_j, u_j) = \int_{x \in X_{j+1}} \min_{u_{j+1} \in U(x)} (E(g_{j+1}(x, u_{j+1})) \cdot fX_j(x|x_j, u_j) dx. \tag{12.3}$$

Wenn man weiter in die Zukunft blickt erhält man

$$\bar{V}_j(x_j, u_j) = \int_{x \in X_{j+1}} \min_{u_{j+1} \in U(x)} \left(E(g_{j+1}(x, u_{j+1})) + \int_{y \in X_{j+2}} \min_{u_{j+2} \in U(y)} (E(g_{j+2}(y, x_{j+2}) \cdot fX_{j+1}(y|x, u_{j+1}) dy) \right) \cdot fX_j(x|x_j, u_j) dx. \tag{12.4}$$

Man spricht in diesem von einer Vorausschau (engl. *lookahead*) über zwei Intervalle. Im Fall diskreter und endlicher Zustandsräume werden aus den Integralen endliche Summe, die sich berechnen lassen. Die allgemeine Form ist in den meisten Fällen nicht analytisch auswertbar, so dass auf Simulation zurückgegriffen wird. Für die Simulation können wir annehmen, dass w_j durch eine Pseudozufallszahlen $w_j^{(k)}$ im

$k$ten Simulationslauf ersetzt wird. $x_{j+1} = f(x_j, u_j, w_j^{(k)})$ wird damit zu einer deterministischen Größe. Damit kann das Integral in (12.3) approximiert werden durch

$$\bar{V}_j(x_j, u_j) \approx \frac{1}{K} \sum_{k=1}^{K} \min_{u_{j+1} \in f(x_j, u_j, w_j^{(k)})} \left(E(g_{j+1}(f(x_j, u_j, w_j^{(k)}), u_{j+1}) \right).$$

K ist die Anzahl der Simulationsläufe, die durchgeführt wird. Um (12.4) per Simulation zu analysieren müssen für jeden simulierten Wert $x_{j+1}^{(ik)} = f(x_j, u_j, w_j^{(k)})$ Simulationen zur Bestimmung des inneren Integrals durch geführt werden, so dass der Aufwand ohne weitere Vereinfachungen in $O(K^2)$ liegt.

Eine andere Approximationsmethode besteht darin, anhand von Heuristiken für jeden Zustand vorab eine Entscheidung $\sigma_j(x_j) \in U(x_j)$ festzulegen. Die Werte von $\bar{V}_j(x_j, u_j)$ lauten dann

$$\bar{V}_j(x_j, u_j) = \int_{x \in X_{j+1}} V_{j+1}^{\sigma}(x) f X_j(x|x_j, u_j) dx$$

wobei $V_{j+1}^{\sigma}(x)$ die Kosten sind, die auftreten, wenn in den Schritten $j+1, \ldots, n$ die durch σ definierte Politik verwendet wird. Oft müssen diese Kosten per Simulation bestimmt werden, d.h. wir bestimmen per Simulation

$$V_{j+1}^{\sigma}(x) \approx \frac{1}{K} \sum_{k=1}^{K} \sum_{i=j+1}^{n-1} g(y_i^{(k)}, \sigma(y_i^{(k)}), y_{i+1}^{(k)})$$

mit $y_{j+1}^{(k)} = x$ und $y_{i+1}^{(k)} = f(y_i^{(k)}, \sigma(y_i^{(k)}), w_i^{(k)})$ für $i = j+2, \ldots, n-1$ und $k = 1, \ldots, K$. Es werden also K Simulationsläufe durchgeführt, die jeweils die Perioden $j+1, \ldots, n$ unter Politik σ abdecken. Ähnlich zur Simulation ereignisdiskreter Systeme können Konfidenzintervalle für $V_{j+1}^{\sigma}(x)$ bestimmt und damit die Anzahl der Replikationen K dynamisch gesteuert werden. Wir werden diesen Aspekt nicht weiter vertiefen.

Neben dem Erwartungswert können auch andere Zielgrößen per dynamischer Programmierung behandelt werden. Wir betrachten dazu als Beispiel die Minimierung der maximalen Kosten. Es soll also eine Politik gefunden werden, die die maximal auftretenden Kosten möglichst gering hält und damit den Verlust begrenzt.

$$V_j^*(x_j) = \min_{u_j \in U_j(x_j)} \left(\max_{w_j \in W_j} \left(g(x_j, u_j, f(x_j, u_j, w_j)) + V_{j+1}^*(f(x_j, u_j, w_j)) \right) \right). \tag{12.5}$$

Die Menge W_j beinhaltet wieder alle möglichen Zufallseinflüsse, die in Periode j auf das System einwirken können. Im Prinzip lässt sich das Problem wie die Probleme zur Minimierung des Erwartungswertes lösen, die Bestimmung des Erwartungswertes wird durch die Bestimmung des Maximums ersetzt. Falls das Maximum nicht analytisch bestimmbar ist, muss Simulation eingesetzt, dazu sind bereits vorgestellten Ansätze entsprechend zu modifizieren. Die Minimierung des Maximums ist in stochastischen Systemen unrealistisch, wenn die maximalen Werte mit sehr kleiner Wahrscheinlichkeit auftreten. Oft ist es sinnvoller den zu minimieren, der mit vorgegebener Wahrscheinlichkeit, zum Beispiel 95%, erreicht wird. Das zugehörige Optimierungsproblem erfüllt aber leider nicht das Bellmansche Optimalitätsprinzip, da der maximale Wert, der mit vorgegebener Wahrscheinlichkeit in Periode j erreicht wird sich in einem stochastischen Modell nicht aus Werten die mit fester Wahrscheinlichkeit in Periode $j+1$ erreicht wurden errechnen lässt, so dass die Rückwärtsrechnung ohne weitere Annahmen nicht möglich ist.

Wir untersuchen zum Abschluss nun noch kurz die Kosten über einen unendlichen Zeithorizont. Offensichtlich sind diese Kosten unendlich, wenn die Kosten der einzelnen Schritte positiv sind. Deshalb wählt man oft eine Sichtweise, die davon ausgeht, dass Kosten in der Zukunft weniger ins Gewicht fallen als in der Gegenwart. Sei α ($0 < \alpha < 1$) ein Diskontierungsfaktor, so dass zukünftige Kosten c (von denen wir annehmen, dass sie endlich sind), die erst in j Perioden auftreten, aktuell nur einen Wert $\alpha^j c$ haben. Damit wird aus (12.2) die folgende Gleichung

$$E(V_j^*(x_j)) = \min_{u_j \in U_j(x_j)} \left(E(g_j(x_j, u_j)) + \alpha \int_{x \in X_{j+1}} \left(E(V_{j+1}^*(x)) \cdot f X_j(x|x_j, u_j) \right) dx \right).$$

Ziel ist die Bestimmung von $E(V^*) = \lim_{j\to\infty}\left(E\left(V_j^*\right)\right)$ und der zugehörigen Politik. Bei der Berechnung der optimalen Politik kann man sich folgendes Ergebnis zu nutze machen.
Sei $X = X_j = X_{j+k}$ für ein $j \geq 0$ und alle $k > 0$ der Zustandsraum und sei $U(x)$ so gewählt, dass $fX(.|x,u)$ für alle $x \in X$ und $u \in U(x)$ eine messbare Funktion ist, dann gilt

$$E(V^*(x)) = \min_{u\in U(x)}\left(E(g(x,u)) + \alpha\int_X \left(E(V^*(y))\cdot fX(y|x,u)\right)dy\right) . \tag{12.6}$$

Der Beweis ist zum Beispiel in Wakuta [1987] zu finden. Die Gleichung zeigt, dass die optimale Politik nur vom aktuellen Zustand abhängt, man spricht von einer stationären Politik und die Periode bei der Berechnung keine Rolle mehr spielt. Wir werden im nächsten Abschnitt Methoden kennen lernen, um die optimale Politik für diesen Fall in einer eingeschränkten Modellklasse zu berechnen. Bei der Berechnung macht man sich zunutze, dass (12.6) eigentlich keine rekursive Darstellung sondern eine Gleichung ist, die mit angepassten Methoden gelöst werden kann. Für den allgemeinen Fall ist man allerdings in der Regel auf die Simulation angewiesen.

12.5 Markovsche Entscheidungsprobleme über endlichen Horizonten

In diesem Abschnitt schränken wir die Klasse der stochastischen Probleme weiter ein, um zu analysierbaren Modellen zu gelangen. Sei dazu $X = \{1,\ldots,m\}$ der endliche Zustandsraum und $U(i) = \{u_{i1},\ldots,u_{is_i}\}$ für $i \in X$ die endliche Entscheidungsmenge. Zustände und mögliche Entscheidungen sind damit unabhängig von der Periode. Das Modellverhalten wird durch die Übergangswahrscheinlichkeiten festgelegt. Sei der Zustand in Periode j $x_j = i$ und der Zustand in Periode $j+1$ $x_{j+1} = j$. Die Zustände werden jeweils zu Beginn der Periode angenommen. Sei ferner $\sigma_j(i) \in U(i)$ eine Entscheidung, die in Periode j im Zustand i getroffen werden kann. Dann ist $p_j(i,\sigma_j(i),k)$ die Übergangswahrscheinlichkeit von i nach k in Periode j und es gilt $\sum_{k=1}^{m} p_j(i,\sigma_j(i),k) = 1$ für alle $j = 1,2,\ldots$, $i \in X$ und $\sigma_j(i) \in U(i)$. Eine Politik für die Perioden $1,\ldots,n$ ist definiert durch eine Sequenz von Vektoren $\sigma_1,\ldots,\sigma_n$ mit $\sigma_j(i) \in U(i)$. Für jeden Entscheidungsvektor $\sigma_j = (\sigma_j(1),\ldots,\sigma_j(m))$ beschreibt das Modell einen inhomogenen Markov Prozess. Aus diesem Grund spricht man von einem Markovschen Entscheidungsproblem, der zugehörige Entscheidungsprozess heißt Markovscher Entscheidungsprozess (engl. *Markov decision process* (MDP)).

Bevor wir uns detailliert mit Markovschen Entscheidungsprozessen beschäftigen, werden in einem kurzen Einschub die zugrundeliegenden zeitdiskreten Markov Prozesse kurz eingeführt (siehe dazu die Einführungen in Puterman [2005], Bertsekas [2005] sowie die grundlegende Darstellung in Kemeny and Snell [1976]). Sei Y_j $(j = 1,2,\ldots)$ eine Sequenz von Zufallsvariablen, die Werte aus einem endlichen oder abzählbaren Zustandsraum X annehmen. Y_j ist ein Markov Prozess falls gilt

$$P\left[Y_j = i_j | Y_1 = i_1, Y_2 = i_2, \ldots, Y_{j-1} = i_{j-1}\right] = P\left[Y_j = i_j | Y_{j-1} = i_{j-1}\right].$$

Die obige Bedingung bezeichnet man als Markov Eigenschaft oder Gedächtnislosigkeit des Prozesses. Die zukünftige Entwicklung hängt nur vom Zustand, nicht aber von der Vergangenheit ab. Die Übergangswahrscheinlichkeiten kann man abgekürzt als

$$p_j(i,k) = P\left[Y_{j+1} = k | Y_j = i\right]$$

bezeichnen, wobei der Index j die Periode beschreibt. Wir beschränken uns hier auf endliche Zustandsräume X mit $m < \infty$ Zuständen. Zustände werden von 1 bis m nummeriert. Die Übergangswahrscheinlichkeiten in Periode j können in einer $m \times m$ Matrix P_j zusammengefasst werden. Alle Elemente in Matrix P_j sind nicht negativ und die Zeilensumme jeder Zeile ist 1. Man spricht von einer *stochastischen Matrix*. Falls die Übergangswahrscheinlichkeiten nicht von der Periode abhängen, so lassen wir den Index j weg. Man spricht dann von einem homogenen Markov Prozess, ansonsten von einem inhomogenen Markov Prozess.

Das Verhalten von Markov Prozessen kann mit Hilfe von Matrizen und Vektoren analysiert werden. Sei $\pi_j = (\pi_j(1),\ldots,\pi_j(m))$ einer Zeilenvektor der Länge m, der eine Wahrscheinlichkeitsverteilung über

die Zustände aus X in Periode j beinhaltet (d.h. $\forall u \in X : \pi_j(i) \geq 0$ und $\sum_{i=1}^{m} \pi_j(i) = 1$). Vektor π_j beschreibt die Zustandsverteilung in Periode j. Die Zustandsverteilung in Periode $j+1$ kann bei bekannter Zustandsverteilung in Periode j durch $\pi_{j+1} = \pi_j P_j$ berechnet werden. Es ist einfach zu zeigen, dass auch π_{j+1} eine Verteilung beinhaltet. Allgemein kann bei bekannter Zustandsverteilung in Periode 1 die Verteilung in Periode $j+1$ durch $\pi_{j+1} = \pi_1 \prod_{k=1}^{j} P_k$ berechnen.

Wir betrachten nun einige Resultate für homogene Markov Prozesse mit Matrix P. Für einen homogenen Markov Prozess gilt $\pi_{j+h} = \pi_j P^h$ für alle $j, h \geq 1$, wobei P^h die $h$te Potenz der Matrix P ist. Sei $P^h(i,k)$ das Element (i,k) in der Matrix P^h. Es gilt $P\left[Y_{j+h} = k | Y_j = i\right] = P^h(i,k)$. Zwei Zustände i und k *kommunizieren*, falls Indizes $1 \leq h_1, h_2 < m$ existieren, so dass $P^{h_1}(i,k) > 0$ und $P^{h_2}(k,i) > 0$. Man kann die Kommunikation auch graphtheoretisch interpretieren. Ein Markov Prozess definiert einen gerichteten Graphen mit Knotenmenge X und einer Kante von i nach k genau dann wenn $P(i,jk) > 0$. Die Zustände i und k kommunizieren, falls ein gerichteter Weg von i nach k und k nach i im Graphen existiert. Falls $X^c \subseteq X$, so dass alle $i, k \in X^c$ kommunizieren und für alle $i \in X^c$, $k \in X \setminus X^c$ und alle $h \geq 1$ $P^h(i,k) = 0$ gilt, so spricht man von einer *rekurrenten* Klasse. Der Zustandsraum eines endlichen Markov Prozesses lässt sich in eine Menge rekurrenter Klassen und die restlichen Zustände, die zu keiner rekurrenten Klasse gehören und in der Klasse der transienten Zustände zusammengefasst werden, disjunkt unterteilen. Transiente Zustände werden nur endlich oft besucht, da der Prozess irgendwann eine rekurrente Klasse betreten wird und diese nicht wieder verlässt. Besteht eine rekurrente Klasse aus nur einem Zustand, so spricht man von einem absorbierenden Zustand und es gilt $P(i,i) = 1$ für einen absorbierenden Zustand i. Ein Zustand $i \in X$ gehört zur Klasse der transienten Zustände genau dann wenn $\lim_{n\to\infty} P^n(i,i) = 0$. Falls alle Zustände eines Markov Prozesses zu einer rekurrenten Klasse gehören, so spricht man von einem *irreduziblen* Markov Prozess. Ein Markov Prozess mit endlichem Zustandsraum X ist *aperiodisch*, wenn der Grenzwert $\lim_{n\to\infty} P^n$ existiert. Anderenfalls ist der Markov Prozess periodisch. Wir betrachten ferner den Grenzwert

$$P^* = \lim_{n\to\infty} \frac{1}{n} \sum_{j=1}^{n} P^j$$

der unter den hier verwendeten Voraussetzungen immer existiert und durch die stochastische Matrix P^* beschrieben wird. Matrix P^* hat die folgenden Eigenschaften:

- Falls k ein transienter Zustand ist, so gilt für alle $i \in X$: $P^*(i,k) = 0$.
- Falls i und k zu unterschiedlichen rekurrenten Klassen gehören, so gilt $P^*(i,k) = P^*(k,i) = 0$.
- Falls i und k zur selben rekurrenten Klasse gehören, so gilt für alle $k \in X$: $P^*(i,k) = P^*(j,k)$.
- Falls der Zustandsraum X aus einer rekurrenten Klasse besteht, so sind alle Zeilen von P^* identisch und gleich einem Vektor π. Es gilt $\pi P = \pi$. π ist die stationäre Verteilung des Markov Prozesses.

Nach diesem kurzen Einschub über Markov Prozesse, betrachten wir nun wieder Markovsche Entscheidungsprozesse. Für vorgegebene Entscheidungsvektoren $\sigma_1, \sigma_2, \ldots$ beschreibt der Markovsche Entscheidungsprozess einen Markov Prozess mit Zustandsübergangsmatrix $P_j^{\sigma_j}$ in Periode j, wobei $P_j^{\sigma_j}(i,k) = p_j(i, \sigma_j(i), k)$. Dabei ist $p_j(i,u,k)$ die Wahrscheinlichkeit, dass der Prozesse von Zustand i nach k wechselt, wenn in Periode j Entscheidung $u \in U(i)$ getroffen wird.

Seien $g_j(i, \sigma_j(i), k)$ die Kosten, die auftreten, wenn in Periode j Entscheidung $\sigma_j(i)$ im Zustand i getroffen wird und der Zustandsübergang nach k führt. Der Erwartungswert der Kosten von Entscheidung $\sigma_j(i)$ in Periode j lautet damit

$$E(g_j(i, \sigma_j(i))) = \sum_{k=1}^{m} p_j(i, \sigma_j(i), k) \cdot g_j(i, \sigma_j(i), k) \ .$$

Damit lassen sich die Bellmanschen Funktionsgleichungen als Sonderfall von (12.2) formulieren und entsprechend vereinfachen.

$$E(V_j^*(i)) = \min_{u \in U(i)} \left(E(g_j(i,u)) + \sum_{k=1}^{m} p_j(i,u,k) \cdot E(V_{j+1}^*(k)) \right)$$

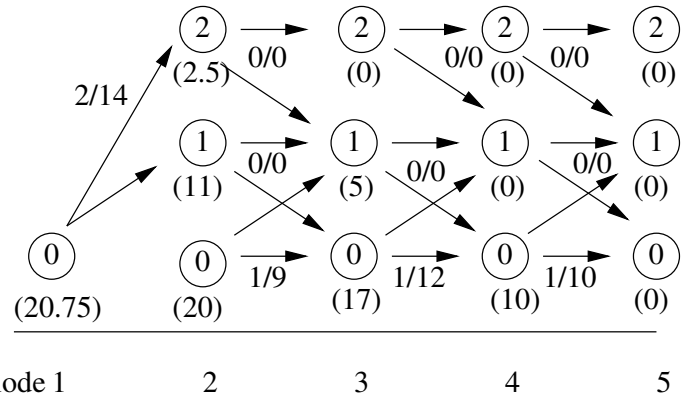


Abbildung 12.3: Optimale Politik für das stochastische Bestellproblem.

Sei $\sigma_j^*(i)$ eine Entscheidung, die im Zustand i in Periode j zum Minimum $E(V_j^*(i))$ führt. $\sigma_j^*(i)$ und $E(V_j^*(i))$ können mit einem Algorithmus bestimmt werden, der sehr ähnlich zu den Algorithmen für den deterministischen Fall (Algorithmen 1 und 2) ist. In der Rückwärtsphase werden ausgehend von $E(V_{j+1}^*(i)) = 0$ $(i = 1, \ldots, m)$ für jede Periode und jeden Zustand die Werte $E(V_j^*(i))$ und $\sigma_n^*(i)$ ermittelt. In der Vorwärtsphase werden anschließend die optimale Politik und die dabei auftretenden Zustandswahrscheinlichkeiten ermittelt. Im Gegensatz zum deterministischen Fall bestimmt man nun nicht eine Politik mit n Entscheidungen, sondern eine Entscheidung für jeden potenziellen Nachfolgezustand.

Zur Veranschaulichung erweitern wir das Bestellproblem um eine stochastische Komponente und nehmen an, dass mit Wahrscheinlichkeit 0.5 ein Teil ausgeliefert werden muss und mit Wahrscheinlichkeit 0.5 keine Auslieferung erfolgt. Bei der Bestellung ist natürlich noch nicht bekannt, ob ein Teil ausgeliefert werden muss oder nicht. Die Kosten für die Belieferung bleiben wie im deterministischen Fall definiert.

Die Einführung von Stochastik sorgt dafür, dass nun nicht mehr gefordert werden kann, dass das Lager in Periode 5 leer ist. Weiterhin muss bei einem leeren Lager immer mindestens ein Teil bestellt werden, da sonst eine mögliche Lieferung nicht garantiert werden kann. Abbildung 12.3 zeigt den Zustandsgraphen des stochastischen Bestellproblems. Im Gegensatz zum deterministischen Fall sind nun nur die mit einer optimalen Entscheidung verbundenen Zustandsübergänge dargestellt. Von einem Zustand ausgehende Kanten haben jeweils eine Wahrscheinlichkeit von 0.5, da Wahrscheinlichkeit 0.5 jeweils ein oder kein Teil bestellt wird. Unter jedem Zustand steht der Wert von $E(V_j^*(i))$. Die minimalen Kosten $E(V_1(1))$ betragen 20.75 Geldeinheiten. Für die optimale Politik werden zu Beginn zwei Einheiten bestellt, es wird nur nachbestellt, wenn das Lager leer ist und dann wird nur eine Einheit nachbestellt. Auch unter der optimalen Politik ist lediglich Zustand 0 in Periode 2 nicht erreichbar, alle anderen Zustände sind erreichbar, so dass Entscheidungen für diese berechnet und gespeichert werden müssen.

Aus der Problemstellung ergibt sich, dass eine optimale Politik vom aktuellen Zustand und der Periode abhängt, nicht aber von vorherigen Entscheidungen. Die optimale Politik muss nicht eindeutig sein, es kann vorkommen, dass in einem Zustand mehrere Entscheidungen zum Optimum führen. Es bringt aber keine Vorteile, Entscheidungen zu mischen indem etwa mit vorgegebener Wahrscheinlichkeit aus mehreren Entscheidungen eine ausgewählt wird.

12.6 Markovsche Entscheidungsprobleme über unendlichen Horizonten

Die bisherigen Betrachtungen beschränkten sich auf endliche Planungshorizonte. In vielen Anwendungen ist man aber an Strategien interessiert, die über einen langen Zeitraum eingesetzt werden können. In diesen Fällen geht man davon aus, dass das Modell Stationarität erreicht, wie wir es schon bei Simulationsmodellen vorausgesetzt haben. Offensichtlich macht es in diesem Fall keinen Sinn, die akkumulierten Kosten zu analysieren, da die Kosten immer unendlich werden, wenn die Kosten für die einzelnen Entscheidungen positiv sind. Um zu analysierbaren Modellen zu gelangen wird eine der folgenden zwei Interpretationen

verwendet:

- Es wird der Erwartungswert der Kosten oder Gewinne in einem Schritt als Optimierungskriterium verwendet.
- Es wird ein Diskontierungsfaktor $\alpha < 1$ eingesetzt und ein Gewinn bzw. Kosten x in k Perioden ist nur noch $x \cdot \alpha^k$ wert.

Je nachdem, ob man Kosten oder Gewinne betrachtet, führt dies zu den entsprechenden Minimierungs- oder Maximierungsproblemen.

Für die Optimierung werden folgende Methoden eingesetzt, die kurz vorgestellt werden.

- Werteiteration
- Lineare Programmierung
- Politikiteration

Wir beginnen mit einer Formalisierung des Problems und betrachten zuerst die Bestimmung des Erwartungswertes pro Schritt.

Seien $v_j^*(i)$ die minimalen Kosten im Zustand i nach j Schritten und $c_j^+(i) = v_j^*(i)/j$ der mittlere Wert pro Schritt. Ferner sei $\sigma_j^+ = (\sigma_j^+(1), \ldots, \sigma_j^+(m))$ eine Politik, die im $j$ten Schritt angewendet wird, um die minimalen Kosten $v_j^*(i)$ zu erreichen. Für eine vorgegebene Schrittzahl n können die Werte für $v_j^* = (v_j^*(1), \ldots, v_j^*(m))$ und σ_j^+ $(j = 1, \ldots, n)$ mit dem bereits vorgestellten Algorithmen bestimmt werden. Um die Existenz einer Lösung auch für $n \to \infty$ zu sichern machen wir folgende Annahme.

Annahme 1 *Die Kosten und Übergangswahrscheinlichkeiten hängen nicht von der Periode ab und es existiert ein Zustand $i \in X$ und eine ganze Zahl $n > 0$, so dass, ausgehend von jedem Startzustand $k \in X$ und unter jeder möglichen stationären Politik, i innerhalb von n Schritten mit positiver Wahrscheinlichkeit erreicht wird. Eine Politik ist stationär, wenn Entscheidungen nur vom aktuellen Zustand abhängen.*

Falls die Annahme erfüllt ist, umfasst der Zustandsraum jedes Markov Prozesses, der aus einer festen Politik entsteht, genau eine rekurrente Zustandsklasse plus eventuell einigen transienten Zuständen.

Sei $\sigma = (\sigma(1), \ldots, \sigma(m))$ eine stationäre Politik. Aus der Theorie der Markov Prozesse (vgl. Stewart [1994]) folgt dann, dass das Gleichungssystem

$$\pi_\sigma(i) = \sum_{k=1}^{m} \pi_\sigma(k) P^\sigma(k,i) \text{ und } \sum_{k=1}^{m} \pi_\sigma(k) = 1.0$$

unter Annahme 1 eine eindeutige Lösung hat. P^σ ist die Übergangsmatrix des Markov Prozesses mit Entscheidungsvektor σ, die bei einer stationären Politik nicht von der Periode abhängt. Der Vektor π_σ beinhaltet die stationäre Verteilung des Markov Prozesses unter Politik σ. Es gilt

$$\pi_\sigma(i) = \lim_{j\to\infty} \pi_\sigma^j(i) \text{ mit } \pi_\sigma^j(i) = \sum_{k=1}^{m} \pi_\sigma^{j-1}(k) P_\sigma(k,i) \text{ für beliebige } \pi_\sigma^0(k) \geq 0 \text{ mit } \sum_{k=1}^{m} \pi_\sigma^0(k) = 1.$$

falls der Markov Prozess aperiodisch (siehe Stewart [1994, Kap. 1.3]) ist. Man kann nun zeigen (siehe Bertsekas [2007, Kap. 4]), dass eine optimale stationäre Politik $\sigma^+ = (\sigma^+(1), \ldots, \sigma^+(m))$ existiert, für die gilt:

- $\lim_{j\to\infty} \sigma_j^+ = \sigma^+$ und $\lim_{j\to\infty} c_j^+(i) = c^+(i) = E(C^+)$.

Damit wird das Optimum mit einer stationäre Politik erreicht und es gilt

$$E(C^+) = \sum_{i=1}^{m} \pi_{\sigma^+}(i) E(g(i, \sigma^+(i))) \ .$$

Es folgt auch, dass die mittleren Kosten/Gewinne unabhängig vom Anfangszustand sind.

Ein erster Ansatz zur Bestimmung des Optimums wäre nun die Berechnung von c_j^+ und σ_j^+ für ein großes j bzw. für $j \to \infty$. Auf Grund des obigen Konvergenzresultates für die optimale Politik reicht es aus, die Rückwärtsphase unseres bisherigen Algorithmus zu nutzen. Es ergibt sich die folgende Iteration.

$$v_j(i) = \min_{\sigma(i) \in U(i)} \left(E(g(i, \sigma(i))) + \sum_{k=1}^{m} P^{\sigma}(i,k) v_{j-1}(k) \right)$$

Für einen beliebig initialisierten initialen Kostenvektor v_0 gilt $\lim_{j\to\infty} v_j(i)/j = E(C^+)$ und die dabei berechnete Politik konvergiert gegen σ^+. Um die exakten Werte zu berechnen können aber unendlich viele Iterationen nötig sein. Das Vorgehen bezeichnet man als Werteiteration. Typischerweise beginnt man mit $v_0 = 0$ und iteriert so lange, bis sich die Politik sich nicht mehr ändert und

$$\left| \frac{j v_{j+1}(i)}{\max(\eta, (j+1) v_j(i))} - 1 \right| < \epsilon$$

für $1 \gg \eta > 0$ und alle $i = 1, \ldots, m$ gilt. Diese Abbruchbedingung ist eine Heuristik, es kann also vorkommen, dass die Schranke unterschritten wird, obwohl die berechneten Werte noch weit vom Minimum entfernt sind. Der Algorithmus hat den Nachteil, dass in der Regel $\lim_{j\to\infty} v_j(i) = \infty$ gilt. Damit wird die Berechnung von $v_j(i)/j$ für große j numerisch instabil. Man benutzt deshalb oft die relative Werteiteration. Dazu wird ein Zustand $s \in X$ ausgewählt und die folgende Iteration ausgeführt.

$$h_j(i) = \min_{\sigma(i) \in U(i)} \left(E(g_{\sigma(i)}(i)) + \sum_{k=1}^{m} P^{\sigma}(i,k)(\sigma_i) h_{j-1}(k) \right) - \min_{\sigma(s) \in U(s)} \left(E(g_{\sigma(s)}(s)) + \sum_{k=1}^{m} P^{\sigma}(s,k) h_{j-1}(k) \right)$$

Wenn wir $h_0(i) = v_0(i) - v_0(s)$ wählen, gilt $h_j(i) = v_j(i) - v_j(s)$. Im Gegensatz zu v_j sind die Werte in h_j beschränkt. Als Abbruchbedingung wird deshalb wieder eine sich konstante Politik und die Bedingung

$$\left| \frac{h_j(i) - h_{j-1}(i)}{\max(\eta, h_j(i))} \right| < \epsilon$$

für ein $1 \gg \eta > 0$ und alle $i = 1, \ldots, m$ gewählt. Sei $h = h_j$, der Vektor nach Beendigung der Iteration, dann gilt

$$E(C^+) = \min_{\sigma(s) \in U(s)} \left(E(g_{\sigma(s)}(s)) + \sum_{k=1}^{m} P^{\sigma}(s,k) h(k) \right) .$$

Man kann das Problem auch als lineares Programmierungsproblem auffassen und zum Beispiel mittels des Simplexalgorithmus (siehe Kap. 10.4) lösen. Sei dazu y_{iu} ($i \in X$, $u \in U(i)$) eine Entscheidungsvariable, die die Wahrscheinlichkeit beinhaltet, im Zustand i zu sein und Entscheidung σ zu treffen. Dann lassen sich folgende lineare Nebenbedingungen formulieren:

$$\sum_{i=1}^{m} \pi(i) = \sum_{i=1}^{m} \sum_{u \in U(i)} y_{iu} = 1 \text{ und } \sum_{u \in U(i)} y_{iu} = \sum_{k=1}^{m} \sum_{u' \in U(k)} y_{ku'} p(k, u', i) \text{ für alle } i \in X$$

Ferner gilt $y_{iu} \geq 0$. Es muss die folgende Zielfunktion minimiert werden.

$$\min \left(\sum_{i=1}^{m} \sum_{u \in U(i)} y_{iu} E(g(i,u)) \right)$$

Aus den bisherigen Ergebnissen folgt, dass für eine optimale Lösung $y_{iu} = \pi(i) > 0$ für genau ein $u \in U(i)$ gilt. Für alle anderen $u' \in U(i) \setminus \{u\}$ gilt $y_{iu'} = 0$. Das resultierende lineare Optimierungsproblem umfasst $\sum_{i=1}^{m} s_i$ Variablen und $m + 1$ Nebenbedingungen.

Als drittes Lösungsverfahren wollen wir die Politikiteration betrachten. Für eine feste Politik $\sigma = (\sigma(1), \ldots, \sigma(m))$ gilt

$$G(\sigma) + v(i) = E(g(i, \sigma(i))) + \sum_{k=1}^{m} P(i, \sigma(i), k)v(k) \ (i = 1, \ldots, m) \text{ mit } G(\sigma) = \sum_{k=1}^{m} E(g(k, \sigma(k))\pi_\sigma(k) \ . \tag{12.7}$$

Diesen Zusammenhang kann man einfach herleiten. Für eine feste Politik σ und ein großes j gilt

$$v_j(i) \approx j \cdot G(\sigma) + v(i) \text{ und damit auch } v_j(i) - v_j(j) \approx v(i) - v(j) \ .$$

Wenn wir diese Beziehung in (12.7) einsetzen, so ergibt sich die folgende Beziehung, wobei nun von j ausgehend rückwärts gerechnet wird.

$$\begin{aligned} v_j(i) &= E(g(i, \sigma(i))) + \sum_{k=1}^{m} P^\sigma(i, k)v_{j-1}(k) && \Leftrightarrow \\ jG(\sigma) + v(i) &= E(g(i, \sigma(i)) + P^\sigma(i, k)\left((j-1)G(\sigma) + v(k)\right) && \Leftrightarrow \\ G(\sigma) + v(i) &= E(g(i, \sigma(i)) + \sum_{k=1}^{m} P^\sigma(i, k)v(k) \end{aligned}$$

Mit Hilfe von (12.7) kann damit jede Politik bewertet werden. Dazu ist ein Gleichungssystem mit m Variablen zu lösen. Ein naiver Ansatz wäre nun das Ausprobieren aller Politiken und die anschließende Ausgabe der besten Politik. Leider gibt es aber $\prod_{i=1}^{m} s_i$ Politiken, so dass dieses Vorgehen schon für kleine Modelle kaum durchführbar ist. Besser ist eine schrittweise Verbesserung der Politik, die in den folgenden 4 Schritten zusammengefasst ist und als Algorithmus unter dem Namen Politikiteration gefasst wird.

1. Starte mit einer Politik $\sigma = (\sigma(1), \ldots, \sigma(m))$, die z.B. durch $\sigma(i) = \arg\min_{u \in U(i)} (E(g(i, u)))$ oder aus einer bekannte Approximation der optimalen Politik gebildet wird.

2. Setze $v(m) = 0$ und löse (12.7), so dass Vektor v und $G(\sigma)$ berechnet werden. Durch die Festlegung eines Wertes im Vektor ist die Lösung für Markov Prozesse, die Annahme 1 erfüllen, eindeutig.

3. Bestimme eine verbesserte Politik σ' aus der Minimierung $\sigma'(i) = \arg\min_{u \in U(i)} (E(g(i, u) + \sum_{k=1}^{m} p(i, u, k)v(k))$ für $i = 1, \ldots, m$. Falls das Minimum nicht eindeutig ist und $\sigma(i)$ den minimalen Wert liefert, so wird $\sigma'(i) = \sigma(i)$ gesetzt.

4. Falls $\sigma = \sigma'$, so wurde eine optimale Politik gefunden und es wird abgebrochen. Ansonsten gilt $G(\sigma') < G(\sigma)$, setze $\sigma = \sigma'$ und fahre bei 2. fort.

Im Gegensatz zur Werteiteration findet die Politikiteration in endlichen vielen Schritten eine optimale Politik. Es muss dafür aber in jedem Schritt eine Gleichungssystem der Dimension m gelöst werden.

Zum Abschluss des Abschnitts betrachten wir noch die Minimierung der Kosten bei Multiplikation mit einem Diskontierungsfaktor $\alpha < 1$. Man kann für diese Problemstellung wieder die Bellmanschen Funktionsgleichungen aufstellen.

$$v_j^+(i) = \min_{u \in U(i)} \left(E(g(i, u)) + \alpha \cdot \sum_{k=1}^{m} p(i, u, k)v_{j+1}^+(k) \right)$$

Ziel ist es, eine optimale stationäre Politik σ^+ zu bestimmen, so dass $v^+(i) = \lim_{j \to \infty} v_j^+(i)$ minimiert wird (vgl. Bertsekas [2007, Kap. 1]). Durch die Diskontierung bleiben die Kosten endlich.

Wir betrachten Werteiteration und Politikiteration zur Berechnung von σ^+ und v^+. Die Existenz einer stationären Politik die zum Minimum führt und des Grenzwertes für v_j^+ sorgen dafür, dass die folgende Gleichung für die optimale Politik und die optimalen Kosten gilt.

$$v^+(i) = E(g(i, \sigma^+(i))) + \alpha \cdot \sum_{k=1}^{m} P^{\sigma^+}(i, k)v^+(k)$$

Die Werteiteration besteht aus den folgenden drei Schritten.

1. Initialisiere $v_0^+(i) = 0$ für alle $i = 1, \ldots, m$ und setze $j = 1$.

2. Bestimme für jeden Zustand i:
   $v_j^+(i) = \min_{u \in U(i)} \left(E(g(i,u)) + \alpha \cdot \sum_{k=1}^m p(i,u,k) v_{j-1}^+(k) \right)$ und speichere die gewählte Politik σ_j^+.

3. Falls $\left| (v_j^+(i) - v_{j-1}^+(i)) / \max(\eta, v_j^+(i)) \right| < \epsilon$ für alle $i = 1, \ldots, m$, setze $v^+ = v_j^+$ und $\sigma^+ = \sigma_j^+$ und stoppe. Ansonsten setze $j = j + 1$und fahre bei 2. fort.

Auch für diese Problemklasse bedeutet das Erreichen des Abbruchkriteriums nicht, dass die optimale Politik erreicht wurde. Im Gegenteil, man kann Beispiele konstruieren, bei denen die Werteiteration nie die optimale Politik erreicht. Da die Iterationen bei der Werteiteration durch Vektor-Matrix-Produkte und nicht durch die Lösung von Gleichungssystemen realisiert werden, ist eine einzelne Iteration sehr effizient und das Verfahren ist insbesondere für große Probleme geeignet.

Als nächsten Algorithmus stellen wir die Politikiteration für diskontierte Probleme vor. Der Algorithmus besteht aus den folgenden Schritten.

1. Starte mit einer Politik $\sigma = (\sigma(1), \ldots, \sigma(m))$, die z.B. durch $\sigma(i) = \arg\min_{u \in U(i)} (E(g(i,u)))$ oder aus einer bekannte Approximation der optimalen Politik gebildet wird.

2. Berechne den zugehörigen Vektor v durch Lösung des Gleichungssystems
   $v(i) = E(g(i, \sigma(i))) + \alpha \cdot \sum_{k=1}^m P^\sigma(i,k) v(k)$ für $i = 1, \ldots, m$.

3. Bestimme eine verbesserte Politik σ'
   $\sigma'(i) = \arg\min_{u \in U(i)} (E(g(i,u) + \alpha \cdot \sum_{k=1}^m p(i,u,k) v(k))$ für $i = 1, \ldots, m$. Falls das Minimum nicht eindeutig ist und $\sigma(i)$ den minimalen Wert liefert, so wird $\sigma'(i) = \sigma(i)$ gesetzt.

4. Falls $\sigma = \sigma'$, so wurde eine optimale Politik gefunden, setze $\sigma^+ = \sigma$, $v^+ = v$ und stoppe. Ansonsten gilt setze $\sigma = \sigma'$ und fahre bei 2. fort.

Wie schon bei der Bestimmung der optimalen durchschnittlichen Kosten, findet die Politikiteration immer eine optimale Politik nach endlich vielen Schritten. Ein einzelner Schritt ist allerdings sehr aufwändig, da jeweils ein Gleichungssystem zu lösen ist. Deshalb wendet man die Politikiteration immer dann an, wenn der Zustandsraum klein ist oder eine gute initiale Politik bekannt ist. Darüber hinaus gibt es zahlreiche Kombinationen von Werte- und Politikiteration, auf die wir allerdings nicht weiter eingehen.

12.7 Optimales Stoppen einer Markov-Kette

Die in diesem Abschnitt behandelten Probleme unterscheiden sich von den bisherigen Problemen dadurch, dass kein oder nur ein maximaler Zeithorizont vorgegeben wird. In jedem Zustand gibt es vielmehr eine Entscheidung, die zum Abbruch führt. Viele praktische Entscheidungsprobleme sind von dieser Art. In der Finanzmathematik ist eine typische Entscheidung, eine Anlage zu halten oder zu verkaufen. Diese Entscheidung muss typischerweise getroffen werden, ohne dass die zukünftige Entwicklung bekannt ist. Ähnlich sieht es beim Mieten einer Wohnung aus. Nach einem Besichtigungstermin muss man sich in der Regel sehr schnell entscheiden, ob man die Wohnung nimmt oder nicht. Es ist zum Entscheidungszeitpunkt nicht bekannt, welche weiteren Angebote in Zukunft verfügbar sein werden.

Wir definieren das Problem des optimalen Stoppens einer Markov Kette jetzt formal. Wir betrachten einen Markov Markovschen Entscheidungsprozess mit endlichem Zustandsraum $X = \{0, \ldots, m\}$. Zustand 0 ist ein besonderer Zustand, in dem der Prozess gestoppt wurde. Wir definieren $X' = X \setminus \{0\}$, die Menge aller Zustände, in denen die Markov Kette nicht gestoppt wurde. In Zustand 0 ist absorbierend und es können keine Entscheidungen getroffen werden und keine Kosten auftreten. Wir nehmen zur Vereinfachungen, dass $U(0) = \{S\}$ gilt. In den anderen Zuständen gibt es jeweils zwei Entscheidungen, Stoppen (S) oder Weitermachen (W). Die Transitionen der Markov Kette können nur dadurch beeinflusst werden, dass die Kette gestoppt wird. Wir nehmen zuerst an, dass wir das Modell in den Perioden $1, \ldots, n < \infty$ untersuchen.

Im Zustand i fallen in Periode j Kosten/Gewinne $c_j(i)$ an, wenn Entscheidung W getroffen wird. Wird Entscheidung S getroffen, so fallen Kosten/Gewinne $s_j(i)$ an, wobei $s_j(0) = 0$. Ferner seien die

Kosten/Gewinne im Zustand i in Periode n, also am Ende des untersuchten Intervalls, gleich $g_n(i)$. Die Übergangswahrscheinlichkeiten sind gegeben durch $p(0,S,0) = 1$ für alle $u \in U(0)$. $p(i,S,0) = 1$ für alle $i \in X'$, sowie $p(i,W,k)$ für $k \in X'$ und $p(i,W,0) = 0$. Damit können wir das Problem als normales dynamische Markovsches Entscheidungsproblem über einem endlichen Horizont formulieren und es gilt

$$E(V_j^*(i)) = \begin{cases} \min\left(c_j(i) + \sum\limits_{k \in X'} p(i,W,k)E(V_j^*(k)), s_j(i) + E(V_{j+1}^*(0)\right) & \text{falls } j < n \wedge i \neq 0 \\ E(V_{j+1}^*(0)) & \text{falls } j < n \wedge i = 0 \\ g_n(i) & \text{falls } j = n \end{cases}$$

Als einfaches Beispiel betrachten wir den Verkauf einer Ware. Wir nehmen dazu an, dass die Ware innerhalb einer bestimmten Zeit, hier nach n Perioden verkauft sein muss. Es werden innerhalb einer Periode Gebote abgegeben, deren Höhe vom Zustand des Markov Prozesses abhängt. Um zu einem endlichen Zustandsraum zu gelangen, nehmen wir an, dass eine endliche Menge von Geboten existiert, die wir von 1 bis m nummerieren. Sei w_i der Wert des Gebots im Zustand i. Da wir die Einnahmen maximieren wollen, hier aber Minimierungsprobleme betrachten, setzen wir $c_j(i) = -w_i$. Wenn die Ware nicht verkauft wird, treten Fixkosten K pro Periode auf, so dass $s_j(i) = K$ ist. Da in Periode n verkauft werden muss gilt hier $g_n(i) = -w_i$. Die Wahrscheinlichkeiten $p(i,W,k)$ ergeben sich aus der Wahrscheinlichkeit, Gebot w_k, wenn in der vorherigen Periode Gebot w_i abgegeben wurde. Falls Gebote nicht von den Geboten der Vorperiode abhängen, so ist die Wahrscheinlichkeit unabhängig vom Startzustand. In diesem Fall bekommen wir jede Periode einen globalen Wert B_j. Die optimale Politik besteht dann darin zu stoppen, wenn der Gewinn in Periode j größer als B_j und ansonsten weiter zu machen.

Wenn wir das Stoppen einer Markov Kette für einen unendlichen Zeithorizont untersuchen, geht man in der Regel davon aus, dass die Kostenfunktionen $c(i)$ und $s(i)$ nicht von der Periode j abhängen. Ferner können wir $\infty > c(i) \geq 0$ und $s(i) < \infty$ voraus und nehmen an, dass ein Zustand i mit $c(i) > 0$ existiert und dieser Zustand von allen anderen Zuständen in X' erreichbar ist. Diese Annahme schließt aus, dass die optimale Strategie darin besteht, nie zu stoppen. Für $i \in X'$ definieren wir

$$v^*(i) = \min\left(s(i), c(i) + \sum_{k \in X'} p(i,W,k) \cdot v^*(k)\right)$$

Es gilt offensichtlich $\min_{k \in X'}(v^+(k)) \leq v^*(i) \leq s(i)$ für alle $i \in X'$. Man kann nun zeigen, ohne dass wir es tun werden, dass die Lösung eindeutig existiert und zum Beispiel per Werteiteration berechnet werden kann. Dazu sein $v_0(i) = s(i)$ und

$$v_j(i) = \min\left(s(i), c(i) + \sum_{k \in X'} p(i,W,k) \cdot v_{j-1}(k)\right).$$

Die Iteration bricht ab, wenn $v_j(i) = v_{j-1}(i)$ für alle $i \in X'$ gilt.

12.8 Erweiterungen

Wir haben nur einen kleinen Teil des weiten Gebiets der stochastischen dynamischen Optimierung behandelt. Es gibt zahlreiche weitere Aspekte, von denen einige kurz angerissen aber nicht vertieft werden sollen.

- Die Dynamik der hier betrachtete Modellklasse basiert auf Perioden zu deren Beginn Entscheidungen getroffen werden können. Es liegt also ein zeitdiskretes Modell vor. Wir haben schon bei der Systemmodellierung im ersten Teil der Vorlesung gesehen, dass viele praktische Probleme besser mit zeitkontinuierlichen Modellen beschrieben werden können. Der Einsatz zeitkontinuierlicher Modelle in der dynamischen Optimierung bedeutet, dass nicht nur über die Entscheidung sondern auch deren Zeitpunkt so gewählt werden muss, dass die Kosten minimal sind. Um zu realisierbaren Lösungen zu gelangen, muss die Anzahl der Änderungen der verwendeten Politik endlich sein und

darf in der Regel nicht zu groß werden. Zur Bestimmung der Politik wird üblicherweise eine Diskretisierungsansatz verwendet. In einigen wenige Fällen kann das zeitkontinuierliche Modell in ein bzgl. der Lösung äquivalentes zeitdiskretes Modell transformiert werden.

- Das sehr breite Gebiet der approximativen dynamischen Programmierung haben wir hie nur in ersten Ansätzen kennengelernt. Es gibt zahlreiche weitere Ansätze, die sich damit beschäftigen, mit Hilfe von Approximationen und insbesondere Simulation die optimale Politik möglichst gut zu approximieren. Die Techniken werden unter unterschiedlichen Namen wie *approximate dynamic programming*, *neuro-dynamic programming*, *reinforcement learning* oder *Q learning* in der Literatur behandelt.

- In den hier vorgestellten Ansätzen wurde immer davon ausgegangen, dass der Zustand des Systems vollständig bekannt ist. Dies muss in der Praxis nicht der Fall sein. So kann es vorkommen, dass nur ein Teil des Zustands beobachtbar ist und Entscheidungen auf Grund der bekannten Teilzustandsinformation und der bisherigen Interaktion mit dem System bzw. der Beobachtung des Systems getroffen werden. Dies führt dazu, dass optimale Entscheidungen nicht nur vom aktuellen (Teil-)Zustand und eventuell dem Zeitpunkt, sondern auch von der Vergangenheit abhängen.

- In vielen realen Anwendungen sind die Modellparameter nicht genau bekannt. In manchen Fällen definiert man Intervalle für Parameter, wie Übergangswahrscheinlichkeiten oder Kosten. Die dann auftretenden Fragestellungen beschäftigen sich damit, welche Kosten minimal oder maximal auftreten können und welche zugehörigen Politiken zu wählen sind, um die entsprechen Werte zu erreichen.

Literaturverzeichnis

Hierarchical performance evaluation tool. http://ls4-www.informatik.uni-dortmund.de/HIT/HIT.html.

Omnet++ community site. http://www.omnetpp.org.

Papers presented at the winter simulation conferences. http://www.informs-cs.org/wscpapers.html.

S. Asmussen, P. W. Glynn, and H. Thorisson. Stationarity detection in the initial transient problem. *ACM Transactions on Modeling and Computer Simulation*, 2(2):130–157, 1992.

J. Banks, J. S. Carson II, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation*. Prentice-Hall, 2000.

F. Bause, H. Beilner, M. Fischer, P. Kemper, and M. Völker. The Proc/B toolset for the modeling and analysis of process chains. In T. Field, P. G. Harrison, J. Bradley, and U. Harder, editors, *Computer Performance Evaluation Modeling Techniques and Tools*, pages 51–70. Springer LNCS 2324, 2002.

H. Beilner. Folienkopien zur Vorlesung Simulation. Informatik IV, Universität Dortmund, 2002.

E. Beltrami. *What is Random? Chance and Order in Mathematics and Life*. Copernicus, 1999.

D. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, 3 edition, 2005.

D. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 3 edition, 2007.

D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

G. M. Birwistle. *DEMOS a system for discrete event modelling on Simula*. Macmillan, 1985.

J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling Computer and Manufacturing Processes*. Springer, 2001.

J. F. Bonnans, J. C. Gilbert, C. Lemarechal, and C. A. Sagastizabal. *Numerical Optimization*. Springer, 2006.

H. Bossel. *Simulation dynamischer Systeme*. Vieweg, 1989.

L. D. Brown, T. T. Cai, and A. DasGupta. Interval estimation for a binomial proportion. *Statistical Sciences*, 16(2):101–133, 2001.

C. G. Cassandras and S. Lafortune. *Introduction to Discrete Events Systems*. Springer, 2007.

F. E. Cellier. *Continuous System Modeling*. Springer, 1991.

P. Chretienne, Jr. E. G. Coffman, J. K. Lenstra, and Z. Liu. *Scheduling Theory and its Applications*. Wiley, 1995.

W. Domschke and A. Drexl. *Einführung in Operations Research*. Springer, 1993.

DynaSys. http://www.freeware-archiv.de/Dynasys-Simulation.htm, 2003. Simulationssoftware.

I. Gerdes, F. Klawonn, and R. Kruse. *Evolutionäre Algorithmen*. Vieweg, 2004.

R. Gomory. Early integer programming. *Operations Research*, 50(1):78–81, 2002.

R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and appraximation in deterministic sequencing and scheduling theory: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

W. K. Grassmann. Warm-up periods in simulation can be detrimental. *Probability in the Engineering and Informational Science*, 22:415–429, 2008.

F. S. Hillier and G. L. Lieberman. *Introduction to Operations Research*. McGraw Hill, 1995.

D. M. Imboden and S. Koch. *Systemanalyse*. Springer, 2003.

F. Jarre and J. Stoer. *Optimierung*. Springer, 2004.

Z. Karian and E. Dudewicz. *Modern statistical systems and GPSS simulation*. CRC Press, 1998.

W. D. Kelton, R. P. Sadowski, and D. T. Sturrock. *Simulation with Arena*. Mc Graw Hill, 2004.

W. D. Kelton, R. P. Sadowski, and D. T. Sturrock. *Simulation with Arena*. McGraw-Hill, 2007.

S. Klein. *Alles Zufall*. Rowohlt, 2005.

T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some clasical and modern methods. *SIAM review*, 45(3):385–482, 2003.

B. Korte and J. Vygen. *Combinatorial Optimization, Theory and Algorithms*. Springer, 2002.

S. Krüger. *Simulation: Grundlagen, Techniken und Anwendungen*. de Gruyter, 1975.

A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. Mc Graw Hill, 2000.

N. Law. Making physics concepts accessible and explorable using worldmaker - an iconic modelling tool. In *Proc. of the International Conference on Physics Teaching*, 1999.

E. L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Sciences*, 19:544–546, 1973.

P. L'Ecuyer. Good parameter sets for combined multiple recursive random number generators. *Operations Research*, 47(1):159–164, 1999.

P. L'Ecuyer and F. Panneton. Fast random number generation based on linear recurrence modulo 2: Overview and comparison. In *Proc. of the 2005 Winter Simulation Conference*, pages 110–119, 2005.

S. Martello and P. Toth. *Knapsack Problems*. Wiley, 1990. VerfÃŒgbar unter http://www.or.deis.unibo.it/knapsack.html.

M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transaction on Modeling and Computer Simulation*, 8(1):3–30, 1998.

Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2004.

D. C. Montgomery. *Design and Analysis of Experiments*. Wiley, 2005.

D. C. Montgomery and G. C. Runger. *Applied Statistics and Probability for Engineers*. Wiley, 2007.

B. Müller-Clostermann. Javademos user manual. Verfügbar unter http://sysmod.icb.uni-due.de/, 2005.

R. H. Myers and D. C. Montgomery. *Response Surface Methodology*. Wiley, 2002.

G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

K. Neumann and M. Morlock. *Operations Research*. Hanser, 1993.

B. Page. *Diskrete Simulation – Eine EinfÃŒhrung mit Modula-2*. Springer, 1991.

B. Page and W. Kreutzer. *Simulating discrete event systems with UML and Java*. Shaker, 2005.

B. Page, T. Lechler, and S. Claassen. *Objektorientierte Simulation in Java*. Libri Books, 2000.

C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization : algorithms and complexity*. Dover Publications, 1998.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C++*. Cambridge University Press, 2002.

K. E. E. Raatikainen. Sequential procedure for simultaneous estimation of several percentiles. *Trans. of the Society for Comp. Sim.*, 7:21–24, 1990.

R. Y. Rubinstein and B. Melamed. *Modern Simulation and Modeling*. Wiley, 1998.

T. J. Santner, B. J. William, and W. I. Notz. *The Design and Analysis of Computer Experiments*. Springer, 2003.

R. E. Shannon. *Systems Simulation: The Art and the Science*. Engelwood Cliffs, 1975.

W. J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994.

SwitcherCAD. http://www.linear.com/designtools/software, 2008.

K. Wakuta. The Bellman's principle of optimality in discounted dynamic programming. *Journal of Mathematical Analysis and Applications*, 125:213–217, 1987.

I. Wegener and T. Hofmeister. Skript zur Vorlesung Operations Research. Informatik II, Universität Dortmund, 2000.

K. Weicker. *Evolutionäre Algorithmen*. Teubner, 2002.

WorldMaker. http://worldmaker.cite.hku.hk/worldmaker/pages/, 1999. Simulationssoftware.