Using Input-Output Hidden Markov Models for the Aggregation of Performance Models

Falko Bause
Informatik IV, Universität Dortmund
D-44221 Dortmund, Germany*
falko.bause@udo.edu

Abstract

This paper describes a new idea how to use Input-Output Hidden Markov Models (IOHMMs) for the aggregation of performance models and reports about corresponding first experiments.
The parameters of the IOHMM are calculated from observation sequences obtained from the input/output behaviour of the original model.

Keywords: Performance Evaluation, Aggregation, Input-Output Hidden Markov Models, Simulation, Optimisation.

1 Introduction

It is well-known that the performance analysis of models might be computationally intensive. Especially simulation of models of todays systems is often time-consuming [1, 15]. The use of (decomposition and) aggregation techniques might reduce the analysis effort. Aggregation can be applied on different levels of description. E.g. for Markov models, aggregation techniques can be applied at state space level (cf. [19]) and their use is often appropriate when the Markov chain is nearly completely decomposable. Another possibility is to define them at the level of model description, which is often supported by a modular and/or hierarchical model specification.

The basic idea is to decompose the model into sub-models. One (or more) of these sub-models is analysed in isolation and the analysis results are used to construct a less detailed substitute representation. Afterwards the substitute is used instead of the original sub-model for the purpose of evaluating the overall model hoping to reduce the analysis effort compared to the analysis of the original model. Here we assume that analysis is done by simulation, so that a possible reduction is achieved in generating less events when simulating the aggregate model. Certainly, aggregation only makes sense when interesting performance figures are not changed significantly.

Aggregation is a well-known technique in the area of Queueing Networks (QNs) [9], since classes of models are known where this technique does not only reduce the analysis effort (for parametric analyses), but also leads to exact results. The technique is based on Norton's theorem for (closed, single class) product-form QNs [11, 13] and has been extended to other classes of nets, see e.g. [5, 6, 10]. In case of non-product form nets the results are often approximative [2]. These approaches have in common that the type of aggregate is given and analysis in isolation is used to provide the actual parameters of the aggregate.

In this paper we assume that the type of aggregate is not explicitly given, but described by a Hidden Markov Model (HMM) [17] and that we try to identify the Markov model by observing the behaviour at the interface of the to be aggregated sub-model.

Several papers consider the use of HMMs for supporting performance evaluation [12, 18, 20]. All approaches have in common that HMMs are used for the specification of the load of a performance model. The parameters of the HMM are calculated from observed or assumed arrival patterns of customers/processes.

*This research was supported by the Deutsche Forschungsgemeinschaft as part of the Collaborative Research Centre "Modelling of large logistics networks"(559).

By definition HMMs do not react on inputs and therefore can not be directly used for aggregation, since a single HMM is useless whenever model parameters are changed or the HMM is plugged into a different environment. [8] extends HMMs to so-called Input-Output Hidden Markov Models (IOHMMs). Here the output of the HMM does not only depend on the current state of the Markov model, but also considers a read input symbol. Thus IOHMMs might be a suitable type of aggregate for performance analysis, because there is some hope that they will also be useful in other environments due to their ability to take inputs into account. When trying to use IOHMMs for aggregation, several questions have to be answered concerning the semantics of the input and output, and how the output should be put into effect with respect to the dynamics of the model. In essence, this is what this paper is about.

The outline is as follows: In Sect. 2 we define Input-Output Hidden Markov Models (IOHMMs) and Sect. 3 presents the key idea of aggregation based on IOHMMs. Sects. 4 to 8 present several aggregate definitions and corresponding experiments with the intention to improve the accuracy of the results. The article ends with some conclusions in Sect. 9.

The paper is based on a report [3], where the reader can find additional information.

2 IOHMMs

An IOHMM [8] is a Markov model with transition probabilities dependent on input symbols. The behaviour of an IOHMM is as follows: The (hidden) Markov chain "reads" a symbol from an input stream, changes its internal state and afterwards outputs a symbol both according to the read input symbol and dependent on each current state. The (hidden) Markov chain starts in initial state i according to an initial state distribution π_i at time $t = 0$.

The notion "hidden" is motivated from common applications of Hidden Markov Models, especially in speech recognition. It is assumed that the observed outputs are from a Markov chain whose states are hidden to the observer. The main concern is to find a Markov chain which best fits to the observed data.
We use the following notation for describing IOHMMs:

N	number of (hidden) states
Q	set of states $Q = \{1, \ldots, N\}$
M	number of symbols
V	set of symbols $V = \{1, \ldots, M\}$
q_t	state at step/time t, $\quad t = 0, \ldots, T$
x_t	input symbol at step/time t, $\quad t = 1, \ldots, T$
y_t	observed output symbol at time t, $t = 1, \ldots, T$
A	transition probability matrix with $a_{kij} = P[q_t = j \| q_{t-1} = i, x_t = k]$
B	observation probability distribution with $b_{ljk} = P[y_t = k \| q_t = j, x_t = l]$ i.e. the conditional probability of observing symbol k given the Markov chain is in state j and the input symbol l is read
π	initial state distribution $\pi_i = P[q_0 = i]$

Matrix A describes the transitions of a Markov chain and thus the entries a_{kij} are independent of time t. This independence is also supposed for B. We require that the set of states and symbols is finite, so that an integer encoding is sufficient. A and B have to satisfy the following conditions ("stochastic submatrices"):

$$\sum_{j=1}^{N} a_{kij} = 1, \forall k = 1, \ldots, M \text{ and } i = 1, \ldots, N \quad (1)$$

$$\sum_{k=1}^{M} b_{ljk} = 1, \forall l = 1, \ldots, M \text{ and } j = 1, \ldots, N \quad (2)$$

$\lambda := (A, B, \pi)$ denotes the entire IOHMM.
Let $T \in \mathbb{N}$ denote the length of an observation sequence. If two finite observation sequences $X = (x_1, \ldots, x_T), x_i \in \mathbb{N}_0$ and $Y = (y_1, \ldots, y_T), y_i \in \mathbb{N}_0$ are given, where X is the sequence of observed input symbols and Y the corresponding sequence of output symbols, our intention is to find an IOHMM which best fits to the observed sequences of symbols or describing this optimisation problem more precisely:

(**) *Given* $X = (x_1, \ldots, x_T)$ *and* $Y = (y_1, \ldots, y_T)$, *estimate model parameters* $\lambda = (A, B, \pi)$ *that maximise* $P[Y|X, \lambda]$.

Algorithms for this optimisation problem are well-known, e.g. the Baum-Welch-Algorithm (cf. [8, 14]). Since the Baum-Welch-Algorithm might get stuck in a local optimum, we use a combination of the Baum-Welch-Algorithm and an $(1, 1)$-evolutionary strategy (cf. [16]) in our experiments.

For more details the reader is referred to [3]. In the following we assume that an appropriate algorithm is available to tackle the optimisation problem (**) giving an adequate solution.

3 Aggregates based on IOHMMs

In this section we show how a template for IOHMM aggregates can be defined. For notation we use the *ProC/B* paradigm described in [4].

3.1 General idea

If we want to use IOHMMs for the construction of an aggregate, we surely have to define an interpretation of the (observation) sequences X and Y. With other words we have to define the meaning of these sequences and how we can extract them from an observation of the model.

Furthermore it is desirable that, at least in principle, our aggregate can be adjusted such that the analysis effort is reduced and/or the resultant model gives more accurate results. Surely, we want both: a reduction of the analysis effort and accurate results.

It is a matter of designing an aggregate based on IOHMMs how to handle these two points.

Concerning analysis we assume that we simulate (the original and) the aggregated model and that efficiency and (hopefully) accuracy of the simulation can be controlled in a simple way (by a parameter $\Delta t \in \mathbb{R}^+$ in the following).

Concerning the observation sequences, one has to decide whether internal information on the to be aggregated sub-model is included or not. In our approach no information on the internal state of the sub-model is used and we presume that only the behaviour at the sub-model's interface is known to the analyst[1]. With respect to the *ProC/B* paradigm (or think of queueing networks) we thus observe when a process (or customer) "arrives" at and when it or she "departs" from the sub-model. To make life simple, let us neglect the types and service call-specific parameters of the individual processes giving a single class of unparameterised processes. So our observations look like

```
...
ARRIVAL    5.016166E+003  1
DEPARTURE  5.018955E+003  1
ARRIVAL    5.019071E+003  2
ARRIVAL    5.020196E+003  1
...
```

[1]Only in Sect. 8 we will also consider information on the internal state.

where the first column denotes the type of event, the second column gives the event time and the third column denotes single or batch arrivals/departures by specifying the batch size.

Probably, this information is too detailed and one possibility is to aggregate the data for specific time intervals. Again, this is a design decision. We select a fixed time interval of length Δt and account for the number of arrivals/departures in these time intervals. Thus suppose that, for a given time interval length $\Delta t \in \mathbb{R}^+$, the following data has been collected from an appropriate observation of the sub-model:

- an array $X = (x_1, \ldots, x_T)$ where $x_i \in \mathbb{N}_0$ denotes the number of arrivals in the i-th Δt time interval, and
- an array $Y = (y_1, \ldots, y_T)$ where $y_i \in \mathbb{N}_0$ denotes the number of departures in the i-th Δt time interval.

Using some optimisation procedure, we assume that an IOHMM $\lambda = (A, B, \pi)$ has been determined, which hopefully describes the observed input/output of the sub-model.

The behaviour of an aggregate for this sub-model can then be defined as follows:

The aggregate counts the number of arrivals in a time interval of length Δt. All arriving processes are blocked in the aggregate, i.e. they are not allowed to leave. At the end of the time interval one step of the IOHMM is emulated giving an output value y for that time interval. This output symbol y can be interpreted as the number of observed departures. Thus, from the set of blocked jobs, y jobs are released and depart from the aggregate. If y exceeds the number of blocked jobs only the currently present number of jobs is released.

A more precise description of an IOHMM template, employing the *ProC/B* notation, is given in the next subsection.

3.2 Template IOHMM aggregate

Fig. 1 shows the general form of an aggregate employing IOHMMs and the aggregation idea described in Sect. 3.1. For simplicity this template aggregate only offers a single service and serves a single class of processes. Service *Service* counts the number of arrivals and also keeps track of the currently number of blocked processes. The integer variable *max_stopped_procs* ensures that at most the number of blocked processes is released after receiving the answer from the functional unit (FU) modelling the behaviour of the IOHMM. A counter in *ProC/B*, like the counter *BlockProcs* in Fig. 1, supports the manipulation of passive resources. A change to a counter is immediately granted if and only if its result accounts for specified upper and lower bounds (here 0 and 10000); otherwise, the requesting process gets blocked until the change becomes possible.

The second process chain in Fig. 1 models the control process which is started every Δt time units. This control process first calls the FU *IOHMM* which emulates one step of the IOHMM. The resultant output is interpreted as the number of processes which have to leave and the corresponding number of processes (at most the number of blocked processes) is released by setting the counter *BlockProcs* appropriately. Setting of this counter is controlled by several conditions (see the "IF-ELSE" construct in Fig. 1) ensuring that the counter value is always ≤ 0 in tangible states. Finally the control process resets the variable for arrivals.

The definition of the FU *IOHMM* (cf. Fig. 2) is straight-forward. The only interesting case is what should be outputted if an input symbol is not in the assumed range of symbols. Note that we have determined this range by an observation of the original sub-model. Whenever this aggregate is plugged into a different environment (or even in the original environment, if the aggregate does not behave identical to the original sub-model), it might happen that we "see" an unknown input symbol, i.e. an unknown number of arrivals. This design decision has to be based on the intended use of the aggregate. Here we assume that we are interested in an analysis of the system's steady state behaviour and that the original sub-model does not create or destroy processes. Following these assumptions the "flow in" of processes into the aggregate has to equal the "flow out". Assuming that the IOHMM aggregate will indirectly follow this flow balance condition concerning observed input/output symbols, it seems natural to define an identity mapping for unknown/unobserved input symbols. This simple extension gives us the possibility to use the IOHMM aggregate in different environments.

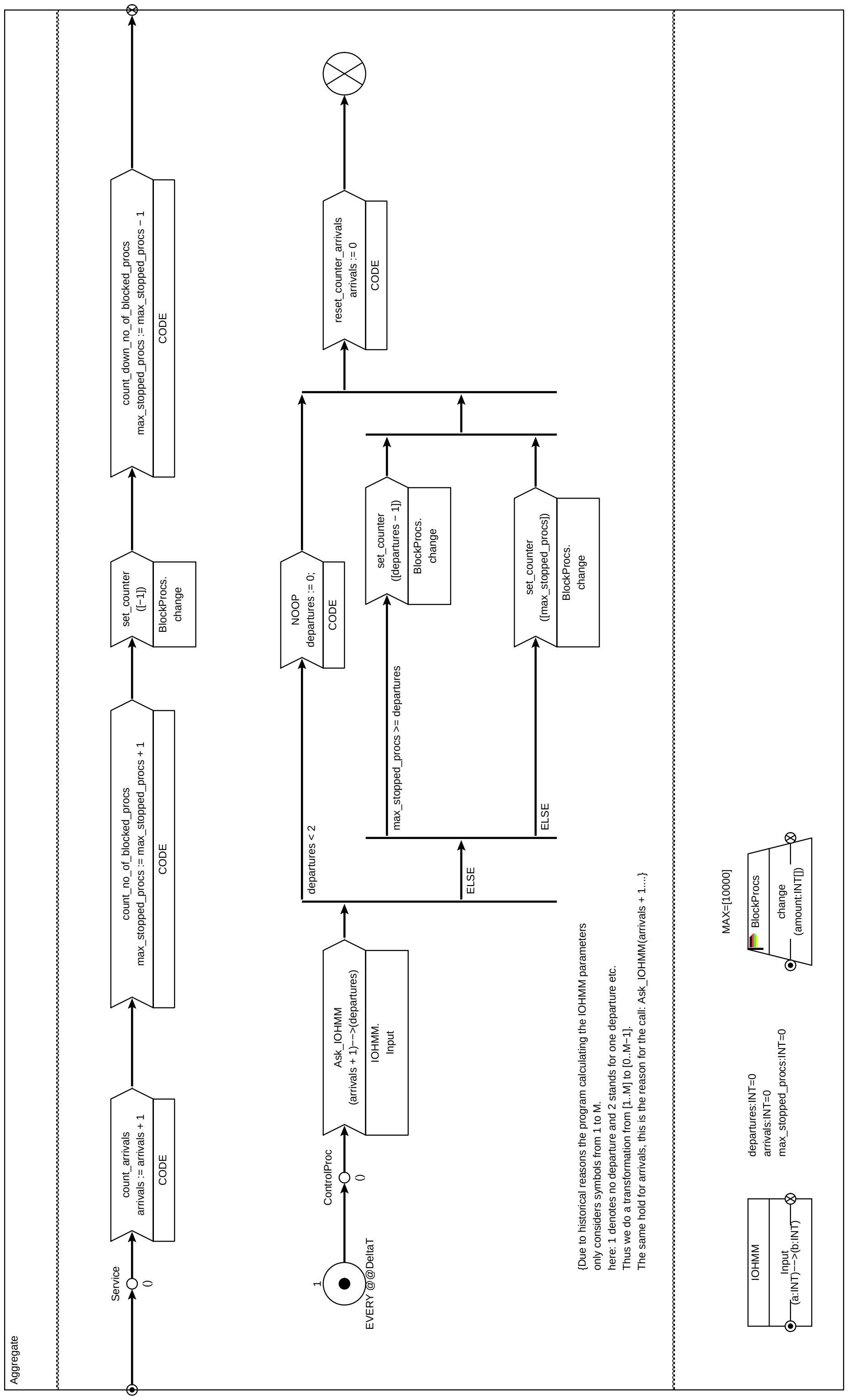
Aggregate
Service
0
count_arrivals
arrivals := arrivals + 1
CODE
count_no_of_blocked_procs
max_stopped_procs := max_stopped_procs + 1
CODE
set_counter
([-1])
BlockProcs.
change
count_down_no_of_blocked_procs
max_stopped_procs := max_stopped_procs − 1
CODE
1
EVERY @@DeltaT
ControlProc
0
Ask_IOHMM
(arrivals + 1)-->(departures)
IOHMM.
Input
departures < 2
ELSE
max_stopped_procs >= departures
ELSE
NOOP
departures := 0;
CODE
set_counter
([departures − 1])
BlockProcs.
change
set_counter
([max_stopped_procs])
BlockProcs.
change
reset_counter_arrivals
arrivals := 0
CODE
{Due to historical reasons the program calculating the IOHMM parameters
only considers symbols from 1 to M.
here: 1 denotes no departure and 2 stands for one departure etc.
Thus we do a transformation from [1..M] to [0..M−1].
The same hold for arrivals, this is the reason for the call: Ask_IOHMM(arrivals + 1,...}
IOHMM
Input
(a:INT)-->(b:INT)
departures:INT=0
arrivals:INT=0
max_stopped_procs:INT=0
MAX=[10000]
BlockProcs
change
(amount:INT[])

Figure 1. Template FU *Aggregate* (@@DeltaT set to concrete values $\in \mathbb{R}^+$ for experiments)

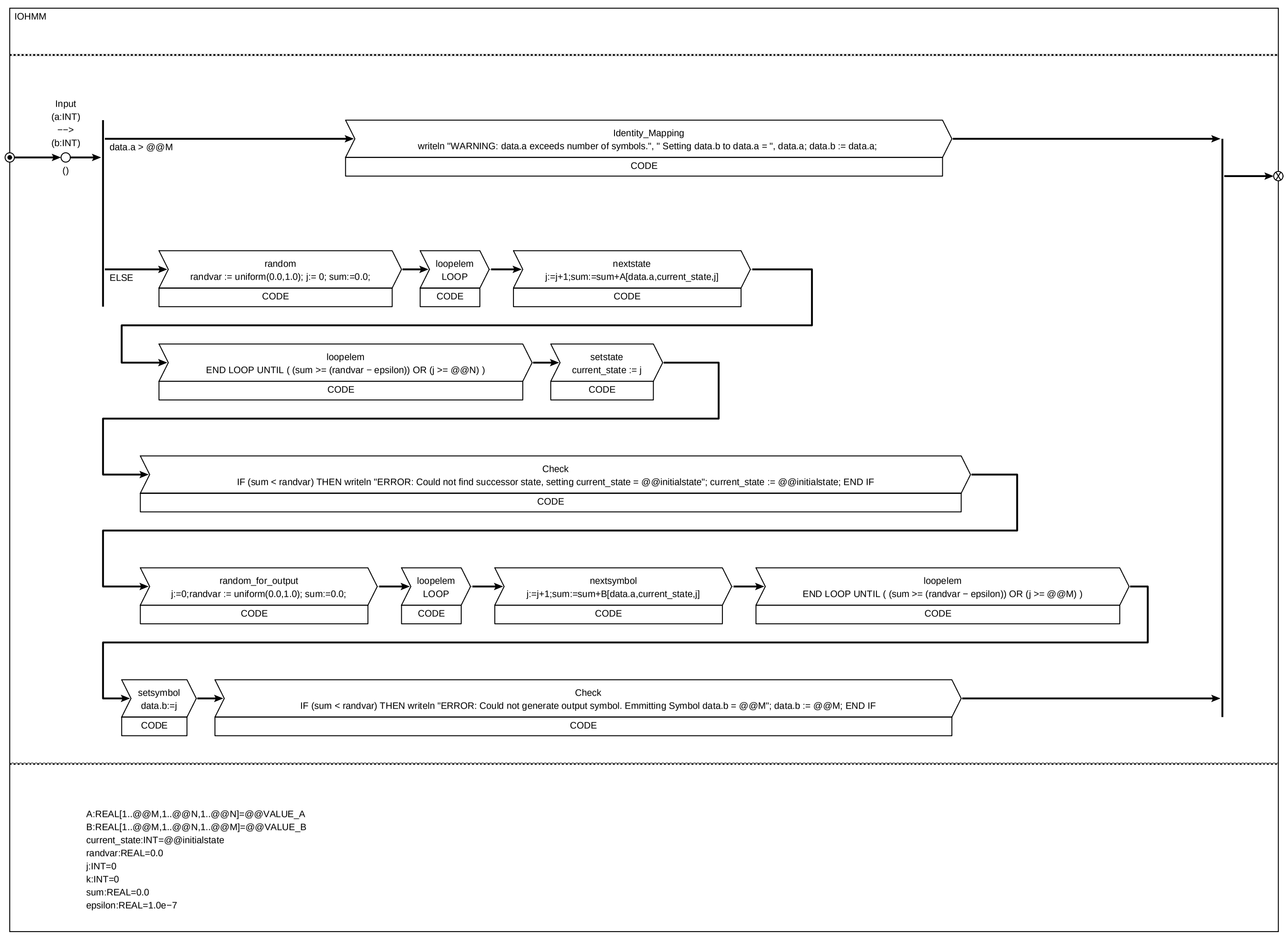
IOHMM
Input
(a:INT)
-->
(b:INT)
()
data.a > @@M
Identity_Mapping
writeln "WARNING: data.a exceeds number of symbols.", " Setting data.b to data.a = ", data.a; data.b := data.a;
CODE
ELSE
random
randvar := uniform(0.0,1.0); j:= 0; sum:=0.0;
CODE
loopelem
LOOP
CODE
nextstate
j:=j+1;sum:=sum+A[data.a,current_state,j]
CODE
loopelem
END LOOP UNTIL ((sum >= (randvar − epsilon)) OR (j >= @@N))
CODE
setstate
current_state := j
CODE
Check
IF (sum < randvar) THEN writeln "ERROR: Could not find successor state, setting current_state = @@initialstate"; current_state := @@initialstate; END IF
CODE
random_for_output
j:=0;randvar := uniform(0.0,1.0); sum:=0.0;
CODE
loopelem
LOOP
CODE
nextsymbol
j:=j+1;sum:=sum+B[data.a,current_state,j]
CODE
loopelem
END LOOP UNTIL ((sum >= (randvar − epsilon)) OR (j >= @@M))
CODE
setsymbol
data.b:=j
CODE
Check
IF (sum < randvar) THEN writeln "ERROR: Could not generate output symbol. Emmitting Symbol data.b = @@M"; data.b := @@M; END IF
CODE
A:REAL[1..@@M,1..@@N,1..@@N]=@@VALUE_A
B:REAL[1..@@M,1..@@N,1..@@M]=@@VALUE_B
current_state:INT=@@initialstate
randvar:REAL=0.0
j:INT=0
k:INT=0
sum:REAL=0.0
epsilon:REAL=1.0e−7

Figure 2. Template FU *IOHMM* (“@@”-values set to concrete values for experiments)

4 Experiment scenario and first results

We consider a simple experiment where we "aggregate" a M/M/1 queue.

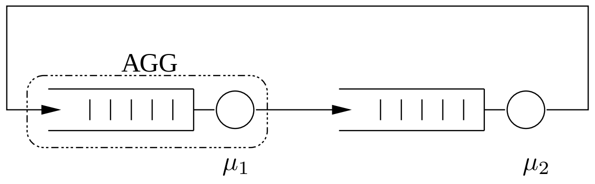

Figure 3. Closed Tandem Queueing Network

Fig. 3 depicts the original model. The model is a closed queueing network comprising two M/M/1 queues. The network is a product-form queueing network and performance figures can be calculated efficiently, e.g. employing Mean Value Analysis (MVA) [13].

For our experiments we use the tool HIT [7] and select the following set of parameters: $\mu_1 = 1.2, \mu_2 = 1.3$ and a total network population of 5 giving the results shown in Table 1.

queue	population	throughput	sojourn time
1	2.73254	1.03779	2.63304
2	2.26746	1.03779	2.18489

Table 1. Results (mean values) of QN of Fig. 3

We choose queue *AGG* for aggregation and queue 2 for validation. In particular we do the following:

1. We simulate the QN of Fig. 3 for the given parameter set and record the arrival and departure events at queue *AGG*. Recording starts after a long warm-up phase of 5000 model time units thus trying to make sure that the output information belongs to the steady-state phase of the model.

2. The simulation output is afterwards transformed (for a given time interval Δt and observation length T) to input values ($X = (x_1, \ldots, x_T)$ and $Y = (y_1, \ldots, y_T)$) for the optimisation procedure giving an IOHMM specification. The length of the observation sequences is chosen such that they account for flow balance, i.e. $\sum_{i=1}^{T} x_i = \sum_{i=1}^{T} y_i$ holds. The calculation of an IOHMM is performed for several number of states, N. The number of symbols, M, is always determined by $M = max_{i=1}^{T}(max(x_i, y_i))$.

3. Finally, we substitute queue *AGG* by the template IOHMM aggregate (cf. Sect. 3.2) plugging in the calculated IOHMM specification. The resultant model is simulated, giving results for queue 2.

Δt	N	T	population	throughput	sojourn
EXACT			**2.267**	**1.038**	**2.185**
0.1	6	1004	1.87	0.94	1.99
0.1	15	1004	1.85	0.94	1.97
0.1	15	3003	1.79	0.93	1.91
0.1	30	277[2]	1.51	0.86	1.76
0.3	6	102	1.31	0.80	1.64
0.3	6	1003	1.72	0.93	1.85
0.3	10	90	1.98	0.94	2.10
* 0.3	10	1003	2.21	1.04	2.13
0.3	15	102	1.73	0.90	1.93
* 0.3	15	1003	2.20	1.03	2.12
* 0.3	25	1003	2.20	1.04	2.13
0.8	6	1002	1.82	0.93	1.96
0.8	15	202	1.76	0.93	1.90
0.8	15	1002	1.72	0.91	1.90

Table 2. Results (mean values) for queue 2 in the model with substituted queue *AGG*

Table 2 shows results for different values of $\Delta t, N$ and T. The columns denote the following: Δt is the length of the time interval (cf. with "@@DeltaT" in Fig. 1), N denotes the number of states, T denotes the length of the observation sequence and *population, throughput, sojourn* are the mean values of population, throughput and sojourn time at queue 2 after simulating 10000 time units. The given mean values are always calculated taking all values of the time interval $[0, 10000]$ into account, which is the standard evaluation option of the tool HIT [7]. All simulation re-

[2] Since the running time of the Baum-Welch-Algorithm is in our case governed by the factor (N^2T) only short observation sequences are used in case of a larger number of states.

sults have been obtained for a confidence interval level of 90% and the interval width was at most 5% of the mean value. The theoretically exact steady-state mean values are shown in the first line. Certainly, due to Little's law [13], the table contains redundant data, but we keep all values in order to have a simple check for typos or transcription errors.

Most results depicted in Table 2 show a significant difference to the exact values. Only those lines marked by an asterisk give satisfactory results, keeping in mind the very low complexity of our example.

All shown results indicate that the proposed aggregate nearly always underestimates the theoretical values. Having a closer look at our aggregate (cf. Fig. 1), we realise that, if only a few processes are blocked, we will discard information from the IOHMM by releasing only the number of blocked processes. Since we assume that the IOHMM considers flow balance, this construct will violate the flow balance condition for the overall aggregate. The current aggregate will then probably block more processes than the corresponding original sub-model, which also explains the low population and other values for the non-aggregated queue 2. In the next section we will revise our template aggregate and redo some of the above described experiments in the now changed setting.

5 Considering infeasible answers

The revised form of the aggregate differs from the former one in the definition of the control process. Irrespective of the number of currently blocked processes, the counter *BlockProcs* is always set to the corresponding output value of the IOHMM. Thus the counter keeps track of IOHMM answers of former time intervals which could not be realised. The value of the counter might now exceed 0. Thus a call to service *Service* might result in no blocking of the calling process, so that the process leaves the aggregate immediately.

We use the IOHMM definitions obtained from the experiments described in Sect. 4 for new simulation experiments. The results are shown in Table 3. The last column ($\sum$ diff) indicates infeasible answers of the IOHMM. Whenever the IOHMM output exceeds the number of blocked processes, we stored the difference. The result shown in the last column is the sum of all these positive values for the total simulation run of 10000 model time units.

Δt	N	T	population	throughput	sojourn	$\sum$ diff
EXACT			**2.267**	**1.038**	**2.185**	
0.1	6	1004	2.840	1.052	2.700	3837
0.1	15	1004	2.786	1.057	2.636	3836
0.1	15	3003	2.552	1.020	2.503	2984
0.1	30	277	1.878	0.911	2.060	1513
0.3	6	102	1.524	0.848	1.797	779
0.3	6	1003	2.224	0.994	2.238	2173
0.3	10	90	3.782	1.150	3.290	7868
0.3	10	1003	3.783	1.183	3.195	7477
0.3	15	102	2.778	1.033	2.689	4172
0.3	15	1003	3.715	1.179	3.149	7254
0.3	25	1003	3.727	1.175	3.169	7231
0.8	6	1002	2.417	1.014	2.384	2476
0.8	15	202	2.264	0.994	2.278	2099
0.8	15	1002	2.269	0.994	2.282	2138

Table 3. Results (mean values) for queue 2 (infeasible answers)

The results show that also this form of aggregate has its deficits and the performance figures do only conform to the exact ones in few cases. Now we overestimate the theoretical values, which might be caused by service of processes in zero time. Having a closer look at column $\sum$ *diff* we notice that this difference might be caused by the different views both functional units have on the state of the overall aggregate. The FU modelling the *IOHMM* has some internal state (cf. *current_state* in Fig. 2) and calculates a corresponding response. Since this response might not conform to the number of blocked processes, i.e. to the internal state of the FU *Aggregate* (cf. Fig. 1), we adjust the answer of the IOHMM somehow. But this adjustment does not influence the internal state (*current_state*) of FU *IOHMM*. So it might happen that the assumed state encoded in the IOHMM and the "real" state of the aggregate differ more and more as time goes by.

6 Accepting only feasible answers

For a next series of experiments we decide to redesign the FU *IOHMM* in such a way that the real state of the aggregate and the IOHMM's internal knowledge on the state do not diverge. The simplest such way is to accept only that output of the IOHMM which conforms to the number of blocked processes. I.e. we do only accept those answers of the IOHMM which do not exceed the number of currently blocked processes.

Technically speaking, we emulate one step of the IOHMM and check whether the generated output exceeds the number of currently blocked processes. In that case we reset the *current_state* variable to its previous value and redo the emulation step until the answer is less than or equal to the number of blocked processes.

The results of some experiments with this revised definition of FU IOHMM are shown in Table 4. Still the results are not satisfactory.

Δt	N	T	population	throughput	sojourn
EXACT			**2.267**	**1.038**	**2.185**
0.1	15	3003	1.774	0.930	1.906
0.1	30	277	1.488	0.862	1.728
0.3	6	102	1.302	0.816	1.595
0.3	6	1003	1.676	0.924	1.814
0.3	10	90	1.957	0.946	2.070
0.3	10	1003	2.156	1.029	2.096
0.3	15	102	1.663	0.899	1.849
0.3	15	1003	2.147	1.027	2.091
0.3	25	1003	2.130	1.028	2.072
0.8	6	1002	1.721	0.918	1.875
0.8	15	202	1.697	0.923	1.839
0.8	15	1002	1.649	0.905	1.822

Table 4. Results (mean values) for queue 2 (accepting only feasible answers)

In tendency they are similar to the results of our first series of experiments (see Table 2), and even differ more from the theoretical exact values.

As we see, disregarding the IOHMM's answers does not lead to satisfactory results. The chosen strategy implicitly changes the IOHMM giving an IOHMM, which presumably does not conform to the observed sequences X, Y having been used for its definition.

7 Virtual arrivals

For a next series of experiments we redesign the FU *Aggregate* again and keep the old (i.e. the first) version of the definition of FU *IOHMM* (see Fig. 2). As mentioned, this might give us some answers which exceed the current number of blocked processes. But now, we will not forget these additional departures, which can not be realised due to a low population of blocked processes. We also do not store these additional departures in the counter *BlockProcs* possibly resulting in zero time delays. Instead, we now count those additional departures as "virtual arrivals" for the next Δt time interval by setting the variable *arrivals* before termination of process *ControlProc* (cf. Fig. 1). In the next Δt time intervals the IOHMM is asked with a larger number of arrivals (actual and virtual) and thus should tend to output a larger number of departures. Since we assume that the IOHMM considers flow balance, the formerly infeasible departures would hopefully have been performed after several Δt time intervals, so that the overall aggregate satisfies flow balance on the long run. More precisely, let x_i denote the number of arrivals in the i-th Δt time interval, d_i the corresponding number of (realised) departures and v_i the number of virtual arrivals in the i-th time interval. Since the IOHMM should consider flow balance in the observed T time intervals and assuming $v_T = 0$ (i.e. in the end all formerly infeasible departures have been realised), we have

$$\sum_{i=1}^{T} x_i + v_{i-1} = \sum_{i=1}^{T} d_i + v_i \tag{3}$$

with $v_0 := 0$. Eq. (3) implies

$$\sum_{i=1}^{T} x_i = \sum_{i=1}^{T} d_i$$

showing that the overall aggregate also satisfies flow balance in the T time intervals.

The results of some experiments with this redesigned FU *Aggregate* are shown in Table 5. Unfortunately, again the results are not satisfactory.

In summary, the resultant mean values of almost all experiment configurations are very close. Note that the reason for all the different aggregate definitions is to

Δt	N	T	popu-lation	through-put	sojourn	$\sum$ diff
EXACT			**2.267**	**1.038**	**2.185**	
0.1	15	3003	1.791	0.934	1.917	870
0.1	30	277	1.542	0.868	1.777	512
0.3	6	102	1.309	0.820	1.597	293
0.3	6	1003	1.721	0.930	1.849	651
0.3	10	90	2.010	0.956	2.103	1946
0.3	10	1003	2.214	1.042	2.125	1548
0.3	15	102	1.727	0.898	1.921	1060
0.3	15	1003	2.196	1.035	2.121	1361
0.3	25	1003	2.204	1.035	2.129	1387
0.8	6	1002	1.838	0.934	1.968	883
0.8	15	202	1.848	0.938	1.970	791
0.8	15	1002	1.782	0.922	1.933	750

Table 5. Results (mean values) for queue 2 (virtual arrivals)

cope with infeasible answers of the IOHMM. The experiments of Sects. 4-7 indicate that the different views on the state of the aggregate might cause inaccurate results. We therefore suspect that we need a mechanism which ensures a single view on the "state" of the aggregate.

8 Adding internal information

In order to eliminate the different (implicitly) encoded information on the current number of blocked processes, we decide to give the IOHMM more information. The IOHMM now does not only get the information on the number of arrivals during a time interval Δt, but also gets the information on the current state of the aggregate, which is (in our case) the number of blocked processes. This implies that also the sequences X and Y, giving the base of the IOHMM determination, include information on the internal state. For our experiments we use input symbols (a, s), encoded as a single integer, where a denotes the number of arrivals in the last observed Δt interval and s the number of currently present customers/processes at queue *AGG*. With that all answers of the IOHMM are feasible and we do not have to take care of situations where the answer of the IOHMM exceeds the number of blocked processes, since such answers will never be given.

Therefore this design of using IOHMMs for building aggregates reduces our problems on interpreting the IOHMM's answers. The only decision left is what to do when the IOHMM is asked with a symbol not occurring in the sequences X and Y. As before, we use an identity mapping for those cases. In order to reduce also the occurrences of those situations, we collect the sequences X and Y now from a simulation of queue *AGG* in an open environment using a Poisson arrival stream with rate $\lambda = 1.0$. Again, the sequences X and Y comprise only situations observed after 5000 time units with the additional restriction that observations are only allowed to start after queue *AGG* has been emptied. The experiments are performed for the closed QN of Fig. 3.

Table 6 shows that the new approach leads to satisfactory results for some parameters. Especially the results for $\Delta t \in [0.2, 0.3]$ for large observation sequences ($T > 700$) give a good approximation for the performance measures of queue 2.

In particular, the results now show a tendency which is expected and desired! Accuracy of the results seems to increase with decreasing Δt. Also the selection of the observation sequences influence the accuracy as expected. A larger value for T seems to give more accurate results, probably because the IOHMM encodes more behavioural characteristics of the original submodel.

9 Conclusions

This paper describes first results of various experiments where Input-Output Hidden Markov Models have been used to build aggregates for performance models. It provides a snapshot of activities for building aggregates for logistics networks.

The experiments show that infeasible answers of the IOHMM might lead to bad aggregation results and that this problem can not be solved easily. In our approach the problem is finally solved by integrating information on the internal state of the sub-model into the IOHMM.

The positive results of Sect. 8 give some hope to determine accurate aggregates also in other environments and for more complex sub-models if employing an appropriate macro state definition. As indicated,

Δt	N	T	population	throughput	sojourn
EXACT			**3.6322**	**1.2429**	**2.9224**
0.1	10	1504	3.388	1.247	2.717
0.2	10	752	3.473	1.281	2.709
0.2	10	1514	3.694	1.273	2.902
0.3	6	755	3.520	1.268	2.776
0.3	6	1503	3.533	1.256	2.813
0.3	10	502	3.311	1.266	2.616
0.3	10	755	3.543	1.283	2.762
0.3	15	502	3.317	1.247	2.661
0.3	20	502	3.247	1.231	2.638
0.8	10	181	2.954	1.228	2.405
0.8	15	181	2.935	1.210	2.426
0.8	20	181	2.940	1.201	2.448
2.0	10	76	2.534	1.117	2.268
3.0	10	51	2.029	0.983	2.064
3.0	15	51	2.372	1.043	2.275
3.0	20	51	2.132	1.011	2.107

Table 6. Results (mean values) for queue 2 now with $\mu_1 = 2.0, \mu_2 = 1.3$ (adding internal information)

the expected effects of Δt and T are now identifiable. Naturally, a proper choice of these parameters needs more investigation.

A further improvement might be obtained by supporting the optimisation process. The optimisation procedure used here has no specific knowledge on the state space of the original sub-model, it merely starts with a complete graph and uniform distributions. In most cases we have a clue on the essential structure of the sub-model's state space, e.g., a quasi birth-death structure, and can exploit this knowledge for the construction of the IOHMM leading to a more adequate representation of the observed behaviour (cf. [17]).

Another idea we are going to follow in the future is to use the IOHMM output for the definition of speed values of a load dependent server. The idea is to set these speed values for each Δt period. This implies that departures may not only happen at the end of a time interval, but are now allowed to take place at any point in time. Hopefully this will increase accuracy and additionally allows for larger Δt values reducing the effort of the overall analysis process.

References

[1] J. Banks (eds.). Handbook of simulation. Principles, Methodology, Advances, Applications, and Practice. John Wiley & Sons, 1998.

[2] B. Baynat, J. Campos. Approximate Methods Based on Net-Driven Decompositions. Tutorials of the 7th International Workshop on Petri Nets and Performance Models, Saint Malo, France, June, 1997.

[3] F. Bause. Input-Output Hidden Markov Models for the Aggregation of Performance Models. Technical Report Sonderforschungsbereich 559 "Modellierung großer Netze in der Logistik", No. 03010, ISSN 1612-1376, 2003.
see also: *http://ls4-www.informatik.uni-dortmund.de/QM/MA/fb/publication_ps_files/Bericht03010neu.pdf*

[4] F. Bause, H. Beilner, M. Fischer, P. Kemper, M. Völker. The Proc/B Toolset for the Modelling and Analysis of Process Chains. in: T. Field, P.G. Harrison, J. Bradley, U. Harder (eds): Computer Performance Evaluation, Modelling Techniques and Tools, Lecture Notes in Computer Science, No 2324, Springer, pp. 51-70, 2002.

[5] F. Bause, R. Boucherie, P. Buchholz. Norton's theorem for batch routing queueing networks. Communications in Statistics - Stochastic Models, Marcel Dekker New York, 17(1), pp. 39-60, 2001.

[6] S. Balsamo, G. Iazeolla. An Extension of Norton's Theorem for Queueing Networks. IEEE Transactions on Software Engineering, 8(4), pp. 298-305, 1982.

[7] H. Beilner, J. Mäter, C. Wysocki. The Hierarchical Evaluation Tool HIT. In: Short Papers and Tool Descriptions of the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, 1994. For more information see http://ls4-www.cs.uni-dortmund.de/HIT/

[8] Y. Bengio. Markovian Models for Sequential Data. Neural Computing Sur-

veys, 2, pp. 129-162, 1999. see also http://www.icsi.berkeley.edu/˜jagota/NCS and ftp://ftp.icsi.berkeley.edu/pub/ai/jagota/vol2_5.pdf

[9] G. Bolch, S. Greiner, H. de Meer, K.S. Trivedi. Queueing Networks and Markov Chains. J. Wiley & Sons, 1998.

[10] R.J. Boucherie. Norton's Equivalent for Queueing Networks Comprised of Quasireversible Components Linked by State-Dependent Routing. Performance Evaluation, 32(2), pp. 83-99, 1998.

[11] K.M. Chandy, U. Herzog, L.S. Woo. Parametric Analysis of Queuing Networks. IBM Journal of Research and Development, 19(1), pp. 36-42, 1975.

[12] F.D. Duarte, E. de Souza e Silva, D. Towsley. An adaptive FEC algorithm using hidden Markov chains. ACM SIG Performance Evaluation Review, 32(2), pp. 11-13, 2003.

[13] K. Kant. Introduction to computer system performance evaluation. Mc Graw Hill, (1992).

[14] B. Knab. Erweiterungen von Hidden-Markov-Modellen zur Analyse ökonomischer Zeitreihen. Dissertation, Mathematisch-Naturwissenschaftliche Fakultät Köln, 2000, see also http://www.zaik.uni-koeln.de/˜ftp/paper/zaik2000-390.ps.gz

[15] A. Law, W. Kelton. Simulation modeling and analysis. 3rd ed., McGraw Hill, 2000.

[16] Z. Michalewicz, D.B. Fogel. How to solve it: Modern Heuristics. Springer, 2000.

[17] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE, 77(2), pp. 257-286, 1989.

[18] K. Salamatian, S. Vaton. Hidden Markov modelling for network communication channels. ACM SIG Performance Evaluation Review, 29(1), pp. 92-101, 2001.

[19] W. J. Stewart. Introduction to the numerical solution of Markov chains. Princeton University Press, 1994.

[20] W. Wei, B. Wang, D. Towsley. Continuous-time hidden Markov models for network performance evaluation. Performance Evaluation, 49(1-4), pp. 129-146, 2003.