A HTD Data Structure for the Analysis of Structured Markov Chains

Peter Buchholz
Informatik IV, TU Dortmund
D-44221 Dortmund, Germany
peter.buchholz@cs.tu-dortmund.de

Tuğrul Dayar
Department of Computer Engineering, Bilkent University,
TR-06800 Bilkent, Ankara, Turkey
tugrul@cs.bilkent.edu.tr

July 12, 2017

Abstract

This short report describes the basic C data structure to represent HTD in the Nsolve package. Furthermore it explains by means of a short example the data structure.

1 Theoretical Basic

The Hierarchical Tucker Decomposition (HTD) [3] became recently popular for the space efficient analysis of certain classes of partial differential equations with a multi-dimensional tensor structure. Basic algorithms for the data structure are available in Matlab [4, 5].

More recently the data structure has also been applied for the numerical analysis of Markov chains with a hierarchical Kronecker structure [1]. In this work, the HTD data structure has been integrated in numerical algorithms from the Nsolve package [2]. Since the algorithm in this package are written in C, appropriate C data structures have to available for the HTD format. The data structure is described here and explained by means of a short example.

In the HTD structure a vector $\boldsymbol{x}$ is represented as

$$\boldsymbol{x} = (\boldsymbol{U}_1 \otimes \cdots \otimes \boldsymbol{U}_d))\,(\mathbf{B}_{1,2} \otimes \cdots \otimes \mathbf{B}_{d-1,d}) \cdots (\mathbf{B}_{1,\ldots,d/2} \otimes \mathbf{B}_{d/2+1,\ldots,d})\mathbf{B}_{1,\ldots,d}$$

This representation defines a hierarchical structure as shown in Fig. 1 taken from [1]. The dimensions of the involved matrices are written near the matrices. Overall the data structure defines a vector of length $\prod_{h=1}^{d} m_d$ and requires only memory in $O(dm_{\max}r_{\max})$ where $m_{\max} = \max_{h=1,\dots,d} m_h$ and $r_{\max} = \max_t r_t$ where t ranges over all nodes in the tree.

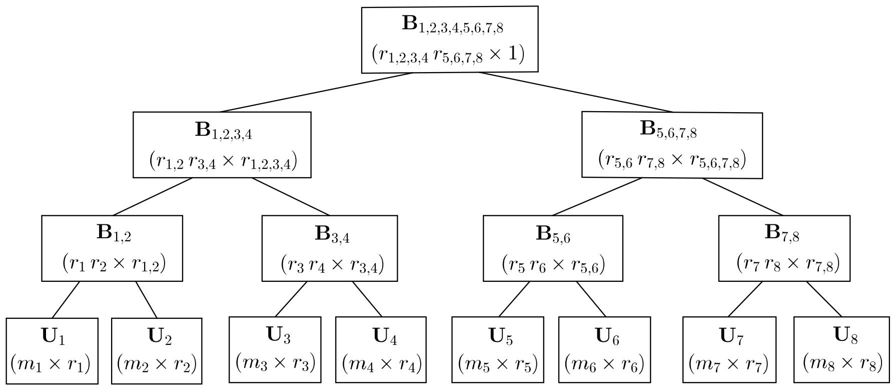
$\mathbf{B}_{1,2,3,4,5,6,7,8}$
$(r_{1,2,3,4}\, r_{5,6,7,8} \times 1)$
$\mathbf{B}_{1,2,3,4}$
$(r_{1,2}\, r_{3,4} \times r_{1,2,3,4})$
$\mathbf{B}_{5,6,7,8}$
$(r_{5,6}\, r_{7,8} \times r_{5,6,7,8})$
$\mathbf{B}_{1,2}$
$(r_1\, r_2 \times r_{1,2})$
$\mathbf{B}_{3,4}$
$(r_3\, r_4 \times r_{3,4})$
$\mathbf{B}_{5,6}$
$(r_5\, r_6 \times r_{5,6})$
$\mathbf{B}_{7,8}$
$(r_7\, r_8 \times r_{7,8})$
$\mathbf{U}_1$ $(m_1 \times r_1)$
$\mathbf{U}_2$ $(m_2 \times r_2)$
$\mathbf{U}_3$ $(m_3 \times r_3)$
$\mathbf{U}_4$ $(m_4 \times r_4)$
$\mathbf{U}_5$ $(m_5 \times r_5)$
$\mathbf{U}_6$ $(m_6 \times r_6)$
$\mathbf{U}_7$ $(m_7 \times r_7)$
$\mathbf{U}_8$ $(m_8 \times r_8)$

Figure 1: Matrices forming $\mathbf{x}$ in HTD format for $d = 8$.

2 The C Data Structure

For a compact representation it is preferable to use arrays rather than pointers. Furthermore, one has to decide whether to use sparse or dense format for the different matrices. Since our first experience with several examples was that matrices to represent iteration vectors are not really sparse, we decided to use dense matrices and to store the matrices in a vector columnwise which allows us to call relevant LAPACK routines without copying or transposing matrices.

The basic tree structure, which is valid for all vectors used for a concrete model, and the structure containing the matrices are kept in two different data structures. We begin with the tree structure which is stored in the following data structure.

```
typedef struct tree_structure_t
{     int       node_count ;
      short     *is_leaf ;
      int       *parent_index ;
      int       *left_child_index ;
      int       *right_child_index ;
      int       *dim_start ;
      int       *dim_end ;
      int       *dim_index_mapping ;
} tree_structure ;
```

The first variable `node_counts` describes the number of nodes in the tree and the dimension of the following arrays, apart from the last one which has a dimension equal to the number of components in the model. Nodes of the tree are stored in breadth first order. The array `is_leaf` contains Boolean values, where a 1 indicates a leave node. According to the ordering of nodes, the leaves of the tree are the final elements in the array. The arrays ending with `index` include the indices of the parent and left and right child of a node. If a node has no parent or child, this is indicated by -1. The arrays `dim_start` and `dim_stop` show which dimensions are described by the corresponding node. A leaf node describes exactly one dimension, namely the component it belongs to. On the other hand, the root node describes all dimensions. The last array `dim_index_mapping` has a dimension equal to the number of components and contains for each component the index of the corresponding leaf node.

The second data structure is used to store the matrices in the tree. Thus the tree structure is required to interpret the data structure.

```
typedef struct compact_vector_t
{
    int       *rank ;
    int       *ni ;
    double    **U_mat ;
    double    **B_mat ;
} compact_vector ;
```

The arrays in the data structure are all of length `node_count` from the tree structure. Array `rank` contains the ranks (i.e. number of columns) of the different matrices. Array `ni` includes the dimensions (i.e., number of rows) of the different nodes. `U_mat` and `B_mat` include the corresponding matrices, one of both array is zero in each node because a node contains a matrix $\boldsymbol{B}$ or $\boldsymbol{U}$ but not both. As already explained matrices are stored as vectors and the vector consists of the columns of the corresponding matrix put together.

3 A Simple Example

We consider a very simple SPN example to clarify the HTD representation. The SPN consists of 3 components and is shown in Figure 2. A component is characterized by the places $pi1$ and $pi2$ ($i = 1, 2, 3$). Place $pi1$ contains two tokens such that each component has 3 states. Transitions $ti1$ have a rate $\lambda_i = 2i$ and transitions $ti2$ rate $\mu_i = i$. The components synchronize via transition $t0$ with rate $\omega = 1$.

For this simple example all states in the product space are reachable and the generator matrix can be represented as a sum of Kronecker products which implies that the hierarchical representation has only one macro state which contains 27 states.

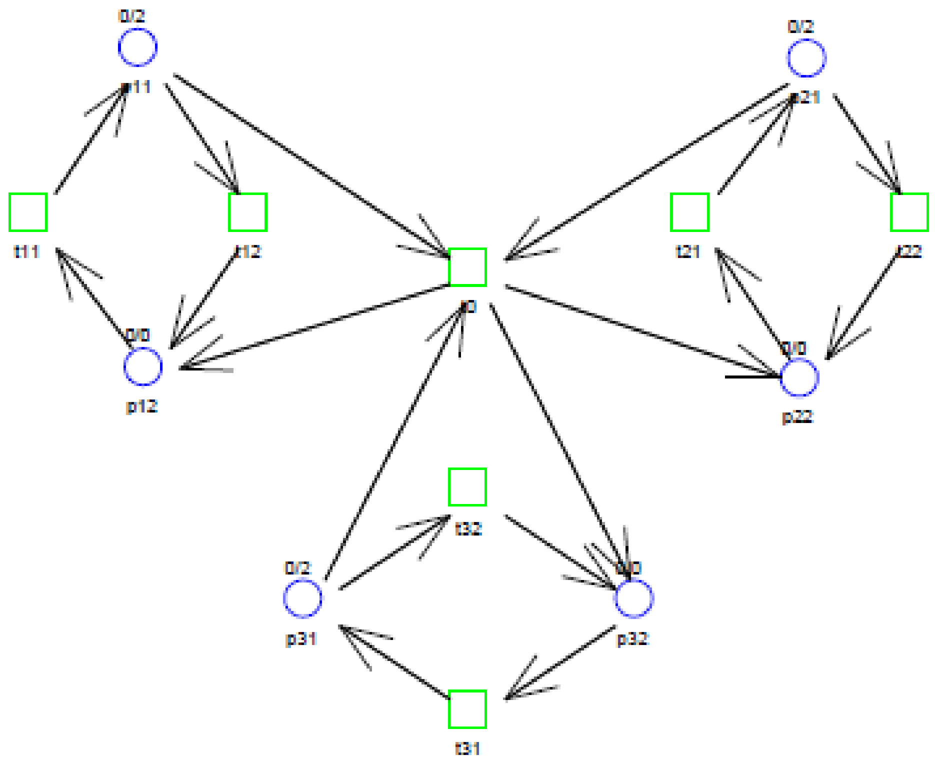

Figure 2: Simple example SPN

Using the notation from [1] we have for each transition of the net a single matrix. However, local transitions (i.e., the transition $ti1$ and $ti2$) can be combined in a single matrix $\boldsymbol{Q}_{i,j}$ where j is the number of the component. We omit the upper macro state index used in [1] because we have only a single macro states. For $i \neq j$ $\boldsymbol{Q}_{i,j} = \boldsymbol{I}_3$, the identity matrix of order 3. For $i = j$ we have

$$\boldsymbol{Q}_{i,i} = \left(\begin{array}{ccc} 0 & \lambda_i & 0 \\ \mu_i & 0 & \lambda_i \\ 0 & \mu_i & 0 \end{array} \right)$$

and $\alpha_i = 1$. The synchronized transition is described by matrices

$$\boldsymbol{Q}_{4,j} = \left(\begin{array}{ccc} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{array} \right)$$

with $\alpha_4 = \omega = 1$. Then the generator matrix of the Markov chain is given by

$$\boldsymbol{Q} = \sum_{k=1}^{4} \alpha_k \bigotimes_{h=1}^{3} \boldsymbol{Q}_{k,h} - \sum_{k=1}^{4} \alpha_k \bigotimes_{h=1}^{3} diag\left(\boldsymbol{Q}_{k,h}\mathbb{1}\right).$$

$\boldsymbol{Q}$ is a 27×27 matrix represented by the sum of Kronecker products of 3×3 matrices. We examine the Power method, i.e., iterations of the form

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \frac{1}{\beta}\boldsymbol{x}^{(k)}\boldsymbol{Q}$$

where $\beta = \max_i |\boldsymbol{Q}(i,i)|/0.999$. The iteration starts with $x^{(0)} = \frac{1}{27}\mathbb{1}^T$ and results in the following vectors after 1 , 10 and 310 iterations.

$$\boldsymbol{x}^{(1)} = \begin{pmatrix} 4.68e-2 \\ 4.09e-2 \\ 3.70e-2 \\ 4.29e-2 \\ 3.70e-2 \\ 3.31e-2 \\ 4.09e-2 \\ 3.51e-2 \\ 2.92e-2 \\ 4.48e-2 \\ 3.90e-2 \\ 3.51e-2 \\ 4.09e-2 \\ 3.70e-2 \\ 3.31e-2 \\ 3.90e-2 \\ 3.51e-2 \\ 2.92e-2 \\ 4.48e-2 \\ 3.90e-2 \\ 3.31e-2 \\ 4.09e-2 \\ 3.70e-2 \\ 3.12e-2 \\ 3.70e-2 \\ 3.31e-2 \\ 2.73e-2 \end{pmatrix} , \ \boldsymbol{x}^{(10)} = \begin{pmatrix} 9.81e-2 \\ 5.47e-2 \\ 3.37e-2 \\ 6.08e-2 \\ 3.44e-2 \\ 2.10e-2 \\ 4.30e-2 \\ 2.40e-2 \\ 1.36e-2 \\ 7.65e-2 \\ 4.43e-2 \\ 2.75e-2 \\ 5.04e-2 \\ 3.51e-2 \\ 2.23e-2 \\ 3.57e-2 \\ 2.48e-2 \\ 1.51e-2 \\ 6.95e-2 \\ 3.92e-2 \\ 2.22e-2 \\ 4.45e-2 \\ 3.04e-2 \\ 1.83e-2 \\ 2.83e-2 \\ 2.01e-2 \\ 1.26e-2 \end{pmatrix} , \ \boldsymbol{x}^{(310)} = \begin{pmatrix} 1.09e-1 \\ 5.90e-2 \\ 3.49e-2 \\ 6.28e-2 \\ 3.48e-2 \\ 2.07e-2 \\ 4.09e-2 \\ 2.25e-2 \\ 1.25e-2 \\ 7.97e-2 \\ 4.53e-2 \\ 2.74e-2 \\ 5.02e-2 \\ 3.53e-2 \\ 2.22e-2 \\ 3.34e-2 \\ 2.37e-2 \\ 1.43e-2 \\ 6.78e-2 \\ 3.78e-2 \\ 2.11e-2 \\ 4.21e-2 \\ 2.92e-2 \\ 1.74e-2 \\ 2.54e-2 \\ 1.86e-2 \\ 1.17e-2 \end{pmatrix}$$

The two norm of the residual after 310 iterations equals $1.65853e-15$ which is in the range of the smallest value that can be reached using double precision in C.

We now consider the iteration with the HTD. Figure 3 shows the basic tree structure. We add an upper index to indicate the iteration number as done for vector $\boldsymbol{x}^{(k)}$. The initial vector is given by

$$\boldsymbol{U}_1^{(0)} = \boldsymbol{U}_2^{(0)} = \boldsymbol{U}_3^{(0)} = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \boldsymbol{B}_{12}^{(0)} = (1), \boldsymbol{B}_{1,2,3}^{(0)} = \begin{pmatrix} \frac{3\sqrt{3}}{27} \end{pmatrix}$$

which corresponds to the flat initial vector. For the solver using the HTD data structure for the solution vector, the truncation is done by setting a truncation bound [1] or by defining a maximal rank [4]. We consider first the truncation due to a truncation bound which determines the singular values

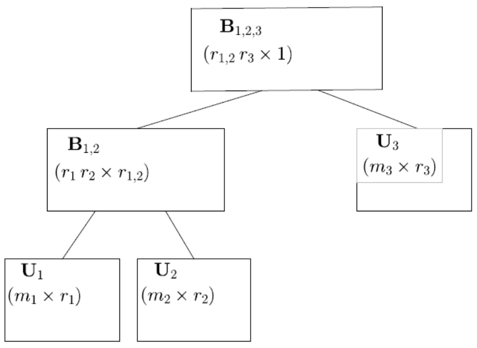

Figure 3: Matrices forming $\mathbf{x}$ in HTD format for $d = 3$.

that are neglected during the addition of vectors in HTD format. Usually, the truncation bound is selected according to the required stopping tolerance and may be changed adaptively during the iteration. For comparison we consider here a fixed truncation bound of $1.0e - 2$ which is a fairly large bound. In this case, the vector after one iteration is represented by

$$\boldsymbol{U}_1^{(1)} = \begin{pmatrix} 6.50e-1 \\ 5.71e-1 \\ 5.02e-1 \end{pmatrix}, \; \boldsymbol{U}_2^{(1)} = \begin{pmatrix} 5.78e-1 \\ 5.77e-1 \\ 5.77e-1 \end{pmatrix}, \; \boldsymbol{U}_3^{(1)} = \begin{pmatrix} 6.16e-1 \\ 5.77e-1 \\ 5.36e-1 \end{pmatrix},$$
$$\boldsymbol{B}_{12}^{(1)} = \begin{pmatrix} 1 \end{pmatrix}, \; \boldsymbol{B}_{12}^{(1)} = \begin{pmatrix} 1.94e-1 \end{pmatrix}$$

Observe that for the iteration only the matricx products $\boldsymbol{Q}_{ij}^T \boldsymbol{U}_j^{(0)}$ have to be computed and the vectors resulting from each Kronecker product are then added and truncated. After 10 iterations we obtain

$$\boldsymbol{U}_1^{(10)} = \begin{pmatrix} 7.88e-1 \\ 5.19e-1 \\ 3.33e-1 \end{pmatrix}, \; \boldsymbol{U}_2^{(10)} = \begin{pmatrix} 5.82e-1 \\ 5.54e-1 \\ 5.96e-1 \end{pmatrix}, \; \boldsymbol{U}_3^{(10)} = \begin{pmatrix} 7.08e-1 \\ 5.49e-1 \\ 4.44e-1 \end{pmatrix},$$
$$\boldsymbol{B}_{12}^{(10)} = \begin{pmatrix} 1 \end{pmatrix}, \; \boldsymbol{B}_{123}^{(10)} = \begin{pmatrix} 2.07e-1 \end{pmatrix}.$$

After 310 we have

$$\boldsymbol{U}_1^{(310)} = \begin{pmatrix} 8.18e-1 \\ 4.92e-1 \\ 3.00e-1 \end{pmatrix}, \; \boldsymbol{U}_2^{(310)} = \begin{pmatrix} 6.44e-1 \\ 5.48e-1 \\ 5.33e-1 \end{pmatrix}, \; \boldsymbol{U}_3^{(310)} = \begin{pmatrix} 7.72e-1 \\ 5.19e-1 \\ 3.66e-1 \end{pmatrix},$$
$$\boldsymbol{B}_{12}^{(310)} = \begin{pmatrix} 1 \end{pmatrix}, \; \boldsymbol{B}_{123}^{(310)} = \begin{pmatrix} 2.17e-2 \end{pmatrix}.$$

This vector has a residual norm of $1.541233e - 1$. Thus, it is only a rough approximation which is very compact because it can be described by a Kronecker product of 3 vectors of dimension 3 each. The complete vector is denoted as $\boldsymbol{y}_{1e-2}^{(310)}$ and is shown below. If we restrict the rank at each level to 2, we obtain the following representation of the iteration vector after 310 iterations.

$$\boldsymbol{U}_1^{(310)} = \begin{pmatrix} 8.26e-1 & -5.58e-1 \\ 4.79e-1 & 7.68e-1 \\ 2.96e-1 & 3.14e-1 \end{pmatrix}, \; \boldsymbol{U}_2^{(310)} = \begin{pmatrix} 7.09e-1 & -6.66e-1 \\ 5.31e-1 & 7.21e-1 \\ 4.64e-1 & 1.92e-1 \end{pmatrix},$$

$$\boldsymbol{U}_3^{(310)} = \begin{pmatrix} 8.03e-1 & -4.79e-1 \\ 4.93e-1 & 8.68e-1 \\ 3.34e-1 & -1.29e-1 \end{pmatrix},$$

$$\boldsymbol{B}_{12}^{(310)} = \begin{pmatrix} 9.99e-1 & 6.05e-3 \\ -2.93e-4 & 7.06e-1 \\ 4.44e-4 & 4.76e-1 \\ 1.23e-2 & 5.25e-1 \end{pmatrix}, \; \boldsymbol{B}_{123}^{(310)} = \begin{pmatrix} 2.24e-1 \\ -1.12e-6 \\ 3.90e-8 \\ 7.82e-3 \end{pmatrix}.$$

The residual norm of the last vector is $1.059233e-01$. The resulting flat vector is show below as $\boldsymbol{y}_{r2}^{(310)}$. The vector is a better approximation of the exact vector but it is for this small example not space efficient However, if we increase the number of components and the size of components, the representation remains space efficient even for larger ranks of the submatrices. Consequently, the HTD is recommended only for very large models where the detailed solution vectors reaches the limit of main memory.

$$\boldsymbol{y}_{1e-2}^{(310)} = \begin{pmatrix} 8.84e-2 \\ 5.32e-2 \\ 3.24e-2 \\ 5.94e-2 \\ 3.57e-2 \\ 2.18e-2 \\ 4.20e-2 \\ 2.52e-2 \\ 1.54e-2 \\ 7.52e-2 \\ 4.52e-2 \\ 2.75e-2 \\ 5.06e-2 \\ 3.04e-2 \\ 1.85e-2 \\ 3.57e-2 \\ 2.15e-2 \\ 1.31e-2 \\ 7.31e-2 \\ 4.40e-2 \\ 2.68e-2 \\ 4.92e-2 \\ 2.96e-2 \\ 1.80e-2 \\ 3.47e-2 \\ 2.09e-2 \\ 1.27e-2 \end{pmatrix}, \ \boldsymbol{y}_{r2}^{(310)} = \begin{pmatrix} 1.08e-1 \\ 5.97e-2 \\ 3.72e-2 \\ 6.35e-2 \\ 3.66e-2 \\ 2.26e-2 \\ 4.46e-2 \\ 2.48e-2 \\ 1.54e-2 \\ 7.87e-2 \\ 4.49e-2 \\ 2.78e-2 \\ 4.63e-2 \\ 3.39e-2 \\ 1.99e-2 \\ 3.26e-2 \\ 1.90e-2 \\ 1.17e-2 \\ 6.95e-2 \\ 3.92e-2 \\ 2.43e-2 \\ 4.09e-2 \\ 2.73e-2 \\ 1.63e-2 \\ 2.88e-2 \\ 1.65e-2 \\ 1.02e-2 \end{pmatrix}$$

If the representation has more than one macro state, a separate HTD is used for every macro state.

References

[1] P. Buchholz, T. Dayar, J. Kriege, and M. C. Orhan. Compact representation of solution vectors in Kronecker-based Markovian analysis. In G. Agha and B. V. Houdt, editors, *Proceedings of the 13th International Conference on Quantitative Evaluation of Systems*, volume 9826 of *Lecture Notes in Computer Science*, pages 260–276. Springer, Heidelberg, 2016.

[2] Peter Buchholz. The nsolve program. Internal report, TU Dortmund, 2010. available under `http://ls4-www.cs.tu-dortmund.de/download/buchholz/nsolve_doc.pdf`.

[3] W. Hackbusch. *Tensor Spaces and Numerical Tensor Calculus.* Springer, Heidelberg, 2012.

[4] D. Kressner and C. Tobler. `htucker` — A Matlab toolbox for tensors in hierarchical Tucker format. Technical Report 2012-02, Mathematics Institute of Computational Science and Engineering, Lausanne, August 2012.

[5] D. Kressner and C. Tobler. Algorithm 941: htucker—A Matlab toolbox for tensors in hierarchical Tucker format. *ACM Trans. Math. Softw.*, 40(3):Article 22, 2014.