ETH Zürich
Seminar for Applied Mathematics

Master Thesis

Parallel Tensor-Formatted Numerics for the Chemical Master Equation

Simon Etter

25th February, 2015

Supervised by Robert Gantner
and Prof. Dr. Christoph Schwab

Contents

1. Introduction

1.1. The Chemical Master Equation

Chemical reaction networks are becoming an increasingly important tool to study the functioning of cells at the molecular level. If such a network involves d chemical species $X_1, \dots X_d$, its dynamics are typically modelled by a set of d continuous-valued functions $c(t, X_i)$ describing the concentration of species X_i at time t, and the evolution of these concentrations is assumed to be governed by nonlinear ordinary differential equations (ODEs) of the form

$$\frac{\mathrm{d}c}{\mathrm{d}t}(t, X_i) = f(c(t, X_1), \dots, c(t, X_d)).$$

The key assumption underlying these models is that each species is present in such abundance that the in principle discrete and stochastic nature of the modelled system can be ignored.

In many biologically relevant cases, the assumption of large copy numbers and averaged fluctuations is not satisfied. In fact, many key constituents of a cell, e.g. proteins or genes, are present only in very small numbers and leverage the noise inherent in all biological system, leading to very different behaviour from the one predicted by deterministic models even on the macroscopic level. Under such circumstances, we must give up on the idea that we could predict the evolution of the studied system with perfect certainty and pursue the humbler goal of estimating the *probability* of finding a certain system state instead. The dynamics of this probability is again described by an ordinary differential equation, the so-called *chemical master equation* (CME), but instead of the concentrations it is formulated in terms of the probabilities $p(t, x(X_1), \dots, x(X_d))$ to count exactly $x(X_i)$ copies of species X_i.

Assume we want to compute the probabilities up to a largest copy number n for each of the d species. Then, the number of unknowns $p(t, x(X_1), \dots, x(X_d))$ is n^d and solving an ODE of this size is effectively impossible using a standard numerical integrator for biologically interesting values of n and d. A large number of alternative methods have been proposed instead, a survey of which is given in [1]. The oldest and most well-known among them is the *Gillespie* or *stochastic simulation algorithm* (SSA) [2], a Monte Carlo method based on generating a large number of sample trajectories from which the quantities of interest are estimated. This algorithm is conceptually simple and has a favourable scaling with respect to the number of species, thereby allowing to study fairly large and complicated systems. On the other hand, achieving sufficient accuracy, in particular when studying rarely occurring events, may require an excessively large number of realisations leading to very long computation times.

(a) TT format (b) HTR

Figure 1.1.: A six-dimensional tensor in the TT format and the HTR.

In [1], a novel approach for this problem was presented which directly tackles the CME. Computational feasibility is achieved by not storing and updating each element of the high-dimensional probability density function $p(t, \ldots)$ individually but rather exploiting its overall structure by using a compression scheme known as the *tensor-train* (TT) format [3, 4]. In the numerical examples presented in [1], already a serial implementation of this ansatz proved to be highly effective and outperformed the Gillespie algorithm running on 1500 cores in terms of wall-clock time. This efficiency is brought about by a much more complicated algorithm, however, with the important consequence that while the SSA is straightforward to parallelize, no truly scalable parallelization scheme has been proposed for computations in the TT format to date. In our time of massively parallel supercomputers, this is a heavy limitation, and it is the purpose of the present thesis to investigate how this limitation may be overcome.

1.2. Tensor Networks

The TT format compresses a *tensor* - a numerical array $a(i_1, \ldots, i_d)$ indexed by d integers - by separating its indices one after the other. That is, in a first step it determines two tensors $u_1(i_1, \alpha_1)$ and $v_1(\alpha_1, i_2, \ldots, i_d)$ such that

$$a(i_1, \ldots, i_d) \approx \sum_{\alpha_1=1}^{r_1} u_1(i_1, \alpha_1)\, v_1(\alpha_1, i_2, \ldots, i_d).$$

The integer r_1 is called rank and the $\approx$ indicates that this separation may be either exact or involve an explicitly controllable approximation error. Next, we similarly separate index i_2 and continue iteratively until we have a final representation of the form (see [4] for details)

$$a(i_1, \ldots, i_d) \approx \sum_{\alpha_1=1}^{r_1} \ldots \sum_{\alpha_{d-1}=1}^{r_{d-1}} u_1(i_1, \alpha_1)\, u_2(\alpha_1, i_2, \alpha_2)\, \ldots\, u_{d-1}(\alpha_{d-1}, i_d). \tag{1.1}$$

This sequential separation leads to the linear structure depicted in Figure 1.1a.

Alternatively, we may separate the dimensions recursively as follows. Assume we have a family of tensors $a(\alpha, i_1, \ldots, i_d)$ parametrized by $\alpha \in 1, \ldots, r$ (with $r = 1$, initially). We pick some dimension $1 < k \leq d$ and separate, in a first step, according to

$$a(\alpha, i_1, \ldots, i_d) \approx \sum_{\alpha_L=1}^{r_L} \sum_{\alpha_R=1}^{r_R} c(\alpha, \alpha_L, \alpha_R)\, u_L(\alpha_L, i_1, \ldots, i_{k-1})\, u_R(\alpha_R, i_k, \ldots, i_d).$$

Then, we recurse for both u_L and u_R and finally obtain the tree-structured representation shown in Figure 1.1b, known as the *hierarchical Tucker representation* (HTR) [5, 6]. Because of representations like the ones in Figure 1.1, we summarize both formats under the name *tensor networks* [7].

1.3. Solvers for Tensor-Network Structured Linear Systems

Following [1], we will use an implicit time-stepping scheme to solve the CME. The key algorithmic step will therefore be to solve linear systems, and for computational feasibility it must be carried out directly in the compressed tensor network form. The *density matrix renormalization group* (DMRG) [8] is an algorithm towards this end which has been known in computational quantum physics for more than 20 years and which has recently been introduced to the numerical linear algebra community in [9, 10]. We briefly outline its key steps assuming the notation for TT network from (1.1). Instead of determining all vertex tensors u_k at once, the DMRG algorithms iteratively updates them by repeatedly carrying out the following three steps:

- Pick two neighbouring vertex tensors u_k, u_{k+1} and combine them to a *supercore*

$$w(\alpha_{k-1}, i_k, i_{k+1}, \alpha_{k+1}) := \sum_{\alpha_k=1}^{r_k} u(\alpha_{k-1}, i_k, \alpha_k)\, u(\alpha_k, i_{k+1}, \alpha_{k+1}). \tag{1.2}$$

- Solve a *local problem* which delivers an updated supercore $w^\star$.

- Separate the new supercore into the two factors $u_k^\star$, $u_{k+1}^\star$ as in (1.2) and let them replace the old vertex tensors, $u_k := u_k^\star$, $u_{k+1} := u_{k+1}^\star$. This step allows to adaptively choose a new rank $r_k^\star$.

The idea is that after sufficiently many of these steps, the tensor a represented by the u_k through (1.1) will eventually converge to the exact solution of the linear system. The DMRG algorithm turned out to be highly effective in practice, yet it has the drawback of being targeted mainly at the TT format which, as we motivate in the next paragraph in a fairly general manner, severely limits its parallelizability.

Any non-trivial algorithm, in particular the solution of linear systems, requires gathering some information from all vertices of the network, and it turns out that this step can only be carried out efficiently if the information is passed on from one vertex to its neighbour like the baton in a relay race. Examples for such information gathering steps are the orthogonalization and, in case of the HTR, computation of the Gramians for truncation [4, 6], and the computation of the projected operators and right-hand sides for the DMRG algorithm [9, 10]. In the TT case, the longest distance between two vertices is $\mathcal{O}(d)$ and it is therefore not possible to reduce the runtime of the information gathering step below $\mathcal{O}(d)$ through parallelization. In contrast, the longest distance in the HTR is only $\mathcal{O}(\log(d))$ (assuming a balanced representation) and all basic algorithms (addition, dot product, orthogonalization and truncation) achieve the resulting optimal

parallel runtime of $\mathcal{O}(\log(d))$ out of the box. The HTR is therefore intrinsically better suited for parallelization than the TT format, see also [6, 11] where similar arguments are given, and any truly parallel tensor network framework must be based on the former.

Nevertheless, it is possible to parallelize the TT DMRG algorithm to some extent. In [12, 13, 14], different schemes were proposed to parallelize the local problems of the DMRG algorithm. This approach faces the problem that the parallel fraction does not scale with the overall workload and is therefore not suitable for massively parallel hardware. Quite a different road, inspiring to a large extent the algorithm we present here, has been taken in [15]. In the mental picture given above, the parallelization scheme proposed there exploits that information needs to be exchanged repeatedly in the DMRG algorithm, and by pipelining up to $\mathcal{O}(d)$ such exchange rounds one can obtain an algorithm scaling up to $\mathcal{O}(d)$ processors after some initial phase. The fundamental problem of the TT format remains, however, and expresses itself in that the first exchange round, requiring $\mathcal{O}(d)$ computational effort, cannot be parallelized beyond two processors (further details will be given in Section 4.4. The parallelization scheme from [15] can therefore be expected to perform well for hard problems requiring many DMRG iterations, but will perform rather poorly in an implicit time-stepping scheme where we need to solve many but fairly easy systems of equations. To illustrate this point, we remark that the maximal iteration count for the DMRG algorithm was set to at most 5 in [1]. We may therefore expect that at least a fifth of the computational effort was spent in a section which would be hardly parallelized by this scheme.

In principle, it is possible to overcome the aforementioned problem by reformulating the DMRG algorithm for HTR networks as has been done in [16] for the case of eigenvalue problems. Such an HTR DMRG algorithm has a number of disadvantages, however, the most important being that the supercore resulting from two interior vertices is $\mathcal{O}(r^4)$ in size which is typically much larger than the $\mathcal{O}(n^2r^2)$ encountered with the TT format. Here, $r \in \mathbb{N}$ denotes some bound for the local ranks and $n \in \mathbb{N}$ denotes the range of a free index i_k which can often be chosen fairly small by means of quantization [17, 18, 19, 20]. Also, the fact that its local problems involve two vertices at the time complicates the implementation in a parallel framework where data for different vertices may be stored on different processes.

The main reason for combining neighbouring vertices in the DMRG algorithm is to allow for adapting the ranks during the course of the computations. In [21], a variant of the DMRG called the ALS(SD) algorithm has been proposed which, despite being rank-adaptive, only updates one vertex at a time and therefore avoids the aforementioned problems. Rank-adaptivity is achieved by combining the one-site DMRG algorithm (also known as the *alternating least squares* (ALS) algorithm [10, 9]) with a steepest descent (SD) and a truncation step. Both of the latter steps are both efficient as well as parallelizable in the HTR.

Based on the above considerations, we conclude that an HTR version of the ALS(SD) algorithm is the most promising candidate for a parallel, tensor-network-based solver for the CME. Developing such an algorithm and analysing its properties is therefore the key topic of this thesis.

2. Tensors

We introduce the tensor-related notation required in later chapters, and recall the concepts of quantization and low-rank tensor representation.

2.1. Cartesian Product and Tuples

Given sets A, B, C, the Cartesian product $A \times B \times C$ is commonly defined as the set of all tuples (a, b, c) such that $a \in A$, $b \in B$, $c \in C$. An important point in this definition is that the Cartesian product and the tuples are ordered: $A \times B \times C$ is not the same set as $C \times A \times B$, and so are (a, b, c) and (c, a, b) not the same tuples. Considering a tuple $t := (a, b, c) \in A \times B \times C$, the orderedness allows to refer to its elements more easily through $t_1 := a$, $t_2 := b$, $t_3 := c$. Effectively, we thus define t to be a function $t : \{1, 2, 3\} \to A \cup B \cup C, i \mapsto t_i$ such that $t_1 \in A$, $t_2 \in B$ and $t_3 \in C$.

In this document, we often want to define a set as the "combination" of some other sets, but any ordering on these sets would be arbitrary. An example for such a situation are the French playing cards. Each card is the combination of a rank $R := \{2, \dots, 10, \mathrm{J}, \mathrm{Q}, \mathrm{K}, \mathrm{A}\}$ and a suit $S := \{\text{hearts, diamonds, spades, clubs}\}$. It is therefore natural to define the set of cards as either $R \times S$ or $S \times R$, but there is no objective reason to prefer one definition over the other. In this simple example involving only two sets, we could introduce a convention on what order should be used, but this approach becomes inconvenient once the number of involved sets becomes variable. Instead, we prefer to generalize the concept of the Cartesian product as follows.

Definition 2.1.1 (Cartesian Product and Tuples)**.** Let D be some finite set and $(S_k)_{k \in D}$ a family of sets parametrized by D. The *Cartesian product* of $(S_k)_{k \in D}$, denoted by $\bigtimes_{k \in D} S_k$, is defined as the set of all functions $D \to \bigcup_{k \in D} S_k$, $k \mapsto s_k$ such that $s_k \in S_k$. Such a function is called a *tuple* and is denoted by $(s_k)_{k \in D}$.

Since the parameter k is usually not of interest, we define the abbreviations $S_D := (S_k)_{k \in D}$ and $s_D := (s_k)_{k \in D}$. We use the shorthand notation $S^D := \bigtimes_{k \in D} S$ in case $S_k = S$ for all $k \in D$. If $D = \{k\}$ is a singleton, we do not distinguish between the set S_k and the trivial Cartesian product $\bigtimes_{k \in D} S_k$ nor between the element s_k and the 1-tuple s_D as long as it is clear what the index k is.

For the playing card example, we may now define $D := \{R, S\}$ such that the set of all cards becomes $C := \bigtimes_{k \in \{R,S\}} k$ and a specific card $c \in C$ is given by e.g. $c_R = \mathrm{A}$, $c_S = \text{spades}$. The above definition is a proper generalization of the standard Cartesian product since the ordered tuples are retained as the special case $D := \{1, \dots, d\}$.

Having to write $\bigtimes_{k\in\{R,S\}} k$ for the Cartesian product of only two sets is clumsy, and so far we have no notational means to construct a $c \in C$ from its constituting elements c_R, c_S. These problems are tackled next.

Definition 2.1.2 (Concatenation)**.** Let $D^{(1)}$, $D^{(2)}$ be two disjoint finite sets, and $S^{(1)}_{D^{(1)}}$, $S^{(2)}_{D^{(2)}}$ two corresponding families of sets. The *concatenation* of the cartesian products $\bigtimes_{k\in D^{(1)}} S^{(1)}_k$ and $\bigtimes_{k\in D^{(2)}} S^{(2)}_k$, denoted by $\left(\bigtimes_{k\in D^{(1)}} S^{(1)}_k\right) \times \left(\bigtimes_{k\in D^{(2)}} S^{(2)}_k\right)$, is defined as

$$\left(\bigtimes_{k\in D^{(1)}} S^{(1)}_k\right) \times \left(\bigtimes_{k\in D^{(2)}} S^{(2)}_k\right) := \bigtimes_{k\in D^{(1)}\cup D^{(2)}} S_k, \quad S_k := \begin{cases} S^{(1)}_k & \text{if } k \in D^{(1)} \\ S^{(2)}_k & \text{if } k \in D^{(2)} \end{cases}.$$

Similarly, we define the *concatenation* of $s^{(1)}_{D^{(1)}} \in \bigtimes_{k\in D^{(1)}} S^{(1)}_k$, $s^{(2)}_{D^{(2)}} \in \bigtimes_{k\in D^{(2)}} S^{(2)}_k$ denoted by $s^{(1)}_{D^{(1)}} \times s^{(2)}_{D^{(2)}}$ as

$$s^{(1)}_{D^{(1)}} \times s^{(2)}_{D^{(2)}} := s_{D^{(1)}\cup D^{(2)}} \in \left(\bigtimes_{k\in D^{(1)}} S^{(1)}_k\right) \times \left(\bigtimes_{k\in D^{(2)}} S^{(2)}_k\right),$$
$$s_k := \begin{cases} s^{(1)}_k & \text{if } k \in D^{(1)} \\ s^{(2)}_k & \text{if } k \in D^{(2)} \end{cases}.$$

The set of all cards can thus be more conveniently defined as $C := R \times S$, and an element in this set may be specified as e.g. A $\times$ spades. In both notations, we use the rule from Definition 2.1.1 of not distinguishing between families / tuples of length 1 and their single element.

Remark 2.1.3. Concatenation is both associative and commutative.

2.2. Tensors and Matrices

Tensors are arrays with $d \in \mathbb{N}$ indices and elements from $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. They thus straightforwardly generalize vectors (columns of numbers, therefore $d = 1$ tensors) and matrices (tables of numbers or $d = 2$ tensors).

Definition 2.2.1 (Tensor)**.** Let D be some finite set and I_D a family of finite sets parametrized by D. An element $x \in \mathbb{K}^{\bigtimes_{k\in D} I_k}$ is called a *tensor*. The elements of D are called *modes* and its size $d := \#D$ *dimension*. For notational convenience, we write $x\,(i_D)$ instead of x_{i_D} to denote the evaluation of x at $i_D \in \bigtimes_{k\in D} I_k$. The I_k are called *index sets*, their size $n_k := \#I_k$ *mode size* and their elements i_k *mode indices*. If $D := \{\}$ is the empty set, we define $\mathbb{K}^{\bigtimes_{k\in D} I_k} := \mathbb{K}$ and formally set $z(i_D) := z$ for $z \in \mathbb{K}$ and $i_D \in \bigtimes_{k\in D} I_k$.

We define the element-wise addition and scalar multiplication of tensors,

$$(x + y)\,(i_D) := x\,(i_D) + y\,(i_D),$$

$$(\alpha x)\,(i_D) := \alpha\, x\,(i_D)$$

for all $x, y \in \mathbb{K}^{\bigtimes_{k\in D} I_k}$ and $\alpha \in \mathbb{K}$. The inner product and norm on $\mathbb{K}^{\bigtimes_{k\in D} I_k}$ are the standard Euclidean inner product and norm [22, Example 4.126],

$$(x, y) := \sum_{i_D \in \bigtimes_{k\in D} I_k} \overline{x(i_D)}\, y(i_D), \quad \|x\| := \sqrt{(x, x)},$$

where $\overline{z}$ denotes the complex conjugate if $z \in \mathbb{C}$ and is to be ignored for $z \in \mathbb{R}$.

The tensors defined above differ from objects of the same name found in the literature (e.g. [22, §1.1.1]) in that we use the unordered tuples from Section 2.1 to index a tensor. This allows using any symbols k and i_k to reference a mode and a specific entry along that mode, in contrast to the usual definitions which enforce the integers $1, \ldots, d$ and $1, \ldots, n_k$, respectively. The new freedom has a number of advantages which will become clear as we proceed.

We will generalize a large part of the well-known matrix terminology to tensors. Before we can do so, we first have to clarify the definition of "matrix" being used, however.

Definition 2.2.2 (Matrix)**.** A two-dimensional tensor $x \in \mathbb{K}^{I_R \times I_C}$ is called a *matrix* if one of its modes is qualified as *row mode* and the other as *column mode*. In an abuse of notation, we define $\mathbb{K}^{I_R \times I_C}$ to imply that R (i.e., the first mode) is the row mode and C (the second mode) the column mode. We emphasize the special roles of the modes by writing $x(i_R, i_C)$ instead of $x(i_R \times i_C)$.

Also, it will be useful to have some map which bridges between tensors and matrices. We establish it in two steps.

Definition 2.2.3 (Long Mode [3])**.** Let D be some finite mode set and I_D the corresponding index sets. The symbol $\overline{D}$ defines a new mode with associated index set $I_{\overline{D}} := \bigtimes_{k\in D} I_k$. We call $\overline{D}$ a *long mode*. If $k_1, \ldots, k_d$ are some modes, we write $\overline{k_1, \ldots, k_d}$ instead of $\overline{\{k_1, \ldots, k_d\}}$.

Definition 2.2.4 (Matricization)**.** Let R, C be two disjoint finite sets and $x \in \mathbb{K}^{\bigtimes_{k\in R\cup C} I_k}$ a tensor. The symbol $\mathcal{M}_{R,C}(x)$ refers to the matrix in $\mathbb{K}^{I_{\overline{R}} \times I_{\overline{C}}}$ whose entries are given by

$$\mathcal{M}_{R,C}(x)(i_{\overline{R}}, i_{\overline{C}}) := x(i_{\overline{R}} \times i_{\overline{C}}).$$

$\mathcal{M}_{R,C}(x)$ is called a *matricization* of x. If either R or C is a singleton $\{k\}$, we also allow using k directly as a subscript to $\mathcal{M}$.

Alternative names for matricization [6] are *matrix unfolding* [3] or *flattening*. See also [22, §5.2].

2.3. Mode Multiplication and Squared Modes

The most important operation for vectors and matrices is matrix multiplication. We now define its appropriate generalization for tensors.

Definition 2.3.1 (Mode Multiplication[1])**.** Let $x \in \mathbb{K}^{\times_{k\in M\cup K} I_k}$, $y \in \mathbb{K}^{\times_{k\in K\cup N} I_k}$ be two tensors such that M, K and N are pairwise disjoint. The expression xy, called the *mode product* of x and y, defines a new tensor $z \in \mathbb{K}^{\times_{k\in M\cup N} I_k}$ whose entries are given by

$$z\,(i_M \times i_N) := \sum_{i_K \in \times_{k\in K} I_k} x\,(i_M \times i_K)\; y\,(i_K \times i_N)\,.$$

K may also be empty. In that case, the mode product is also called *tensor product* [22, §1.1.1].

If M, K and N are singletons, the above expression is exactly the matrix product. Not surprisingly, mode multiplication inherits many properties of the matrix product.

Remark 2.3.2. The mode product $z = xy$ with x, y and z as above is equivalent to the matrix product $\mathcal{M}_{M,N}(z) = \mathcal{M}_{M,K}(x)\,\mathcal{M}_{K,N}(y)$. It therefore costs $\mathcal{O}\left(\prod_{k\in M\cup K\cup N} \#I_k\right)$.

Remark 2.3.3. Mode multiplication is associative, i.e. for any three tensors x, y and z we have $(xy)z = x(yz)$. We may thus drop the parentheses and simply write xyz.

Remark 2.3.4. Mode multiplication is distributive over tensor addition, i.e. we have $x\,(y+z) = xy + xz$ and $(x+y)\,z = xz + yz$ for any three tensors x, y and z where x and y are taken from the same tensor space.

Remark 2.3.5. The inner product (x, y) of two tensors $x, y \in \mathbb{K}^{\times_{k\in D} I_k}$ is given by the mode product $\overline{x}y$, where $\overline{x}$ denotes the conjugate tensor defined in Definition 2.4.4.

As stated earlier, we may use any symbol to reference a mode. Each symbol may reference at most one mode per tensor, however, since there would be no way of distinguishing between multiple modes (i.e., indices) of the same mode symbol. Yet there are cases where we would like some modes to appear "twice" in a tensor in some suitable sense, the most prominent example being the linear operators from a tensor space $\mathbb{K}^{\times_{k\in E} I_k}$ to another tensor space $\mathbb{K}^{\times_{k\in F} I_k}$. Linear operators on vectors are usually identified with matrices. Likewise, we would like to identify the aforementioned mappings with tensors from a third tensor space of dimension $d := \#E + \#F$. If E and F are disjoint, this is straightforward: the operator space in question can be identified with $\mathbb{K}^{\times_{k\in E\cup F} I_k}$ such that the application of an operator $A \in \mathbb{K}^{\times_{k\in E\cup F} I_k}$ to a tensor $x \in \mathbb{K}^{\times_{k\in E} I_k}$ is given by the mode product Ax. However, if E and F overlap, this identification no longer works since in $E \cup F$ we "lose" some modes in the sense that $\#(E \cup F) < \#E + \#F$.

We resolve the problem in two steps. First, we introduce a new notation which allows us to easily generate two new mode symbols out of one.

[1] An operation of the same name has been defined in [23] which, in our terminology, corresponds to the special case when one of the multiplied tensors is a matrix.

Definition 2.3.6 (Tag)**.** Let k be a mode. A *tag* T introduces a new mode $T(k)$ with the same index set as k.

Definition 2.3.7 (Squared Mode)**.** Let D be a finite mode set and I_D the corresponding index sets. We introduce the *row tag* R and the *column tag* C and we define $D^2 := R(D) \cup C(D)$. If both $R(k)$ and $C(k)$ appear in the mode set of a tensor, k is called a *squared mode*. $R(k)$ is called k's *row mode* and $C(k)$ k's *column mode*.

We define $I_k^2 := I_{R(k)} \times I_{C(k)}$. If a tensor $x \in \mathbb{K}^{\times_{k \in D^2} I_k}$ has only squared modes, we further define $\mathcal{M}(x) := \mathcal{M}_{R(D),C(D)}(x)$.

In our example, we would like the linear operators to have two modes for each $k \in E \cap F$, namely one for the mode in the range tensor space and one for the mode in the domain tensor space. We can achieve this by setting $O_2 := E \cap F$, $O_1 := (E \cup F) \setminus O_2$ and identifying the operator space with $\mathbb{K}^{\times_{k \in O_1 \cup O_2^2} I_k}$ (note how O_2 contains the squared Operator modes, and O_1 the non-squared ones). The notation for applying A to x does not yet work out, however. The common modes of A and x are only $E \setminus F$, thus Ax results in a tensor in $\mathbb{K}^{\times_{k \in F \cup O_2^2 \cup O_2} I_k}$ instead of the desired $\mathbb{K}^{\times_{k \in F} I_k}$. What is still needed are special rules for mode products involving row and column modes such that Ax indeed corresponds to the application of A to x. This is tackled next.

Definition 2.3.8 (Mode Multiplication and Squared Modes)**.** Let $x \in \mathbb{K}^{\times_{k \in D^{(x)}} I_k}$, $y \in \mathbb{K}^{\times_{k \in D^{(y)}} I_k}$ be two tensors. We define:

- Let k be a mode such that $C(k) \in D^{(x)}$ and $k \in D^{(y)}$. In the expression xy, the $C(k)$-mode of x is multiplied with the k-mode of y. The $R(k)$-mode of the result is set to k.
- Let k be a mode such that $k \in D^{(x)}$ and $R(k) \in D^{(y)}$. In the expression xy, the k-mode of x is multiplied with the $R(k)$-mode of y. The $C(k)$-mode of the result is set to k.
- Let k be a mode such that $C(k) \in D^{(x)}$ and $R(k) \in D^{(y)}$. In the expression xy, the $C(k)$-mode of x is multiplied with the $R(k)$-mode of y.

In this new notation, the expression Ax implies mode multiplication of all modes of A in $(E \setminus F) \cup C(O_2)$ with all the modes of x. We thus get an intermediate result in $\mathbb{K}^{\times_{k \in (F \setminus E) \cup R(O_2)} I_k}$, in which the $R(O_2)$-modes are then relabelled to $O_2 = F \cap E$. In conclusion, we have $Ax \in \mathbb{K}^{\times_{k \in F} I_k}$ as desired. Note how the above notation generalizes the common notation for matrix products: the column modes of a tensor x are multiplied with the corresponding untagged or row modes of another tensor y if x stands to the left of y. This is exactly the same rule as the one used for the matrix product except that for matrices we may drop the plural as we multiply over only one mode.

Remark 2.3.9. Unlike the matrix product, mode multiplication is also commutative unless one of the multiplied modes is a row or column mode.

Mode multiplication always multiplies over all matching modes. Sometimes, we would like to explicitly exclude some modes from being multiplied, however. We introduce a new notation to enable this.

Definition 2.3.10 (Exclusive Mode Multiplication)**.** Let $x \in \mathbb{K}^{\times_{k \in D^{(x)}} I_k}$, $y \in \mathbb{K}^{\times_{k \in D^{(y)}} I_k}$ be two tensors and M some mode set such that $M \subseteq D^{(x)} \cap D^{(y)}$. The expression $x \,\langle M \rangle\, y$ excludes the modes in M from mode multiplication. Instead, the M-modes of x become $R(M)$-modes and the M-modes of y become $C(M)$-modes in the result. If $M = \{k\}$ is a singleton, we also write $x \,\langle k \rangle\, y$ instead of $x \,\langle \{k\} \rangle\, y$. If there is a third tensor $z \in \mathbb{K}^{\times_{k \in D^{(z)}} I_k}$ with $D^{(z)} \cap M = \{\}$ between x and y, we split $\langle M \rangle$ into two parts as in $x_{\langle M \, z \, M \rangle} y$.

The picture to be associated with expressions of the form $x_{\langle M}$ is that the angle bracket $\langle$ pushes the M-modes out of x towards the left.

2.4. Further Tensor Notation

Definition 2.4.1. Let $n \in \mathbb{N}$. We define $[n] := \{0, \ldots, n-1\}$.

Definition 2.4.2 (Slice)**.** Let $x \in \mathbb{K}^{\times_{k \in D} I_k}$ be a tensor, $M \subset D$ a mode set and $i_M \in \bigtimes_{k \in M} I_k$ a tuple of indices. The symbol $x\,(i_M)$, called an *M-slice* of x, denotes the tensor in $\mathbb{K}^{\times_{k \in D \setminus M} I_k}$ whose entries are given by

$$x\,(i_M)\left(i_{D \setminus M}\right) = x\left(i_M \times i_{D \setminus M}\right).$$

Definition 2.4.3 (Induced Subspace)**.** Let R, C be two disjoint mode sets. A tensor $x \in \mathbb{K}^{\times_{k \in R \cup C} I_k}$ induces the subspace

$$\mathbb{S}_R(x) := \operatorname{span}\left\{ x\,(i_C) \mid i_C \in \bigtimes_{k \in C} I_k \right\} \subseteq \mathbb{K}^{\times_{k \in R} I_k}.$$

If $R = \{k\}$ is a singleton, we also write $\mathbb{S}_k(x)$ instead of $\mathbb{S}_{\{k\}}(x)$.

Definition 2.4.4 (Conjugate Tensor)**.** Let $x \in \mathbb{C}^{\times_{k \in D} I_k}$ be a tensor. The symbol $\overline{x}$ denotes the element-wise conjugate of x, i.e.

$$\overline{x}\,(i_D) := \overline{x\,(i_D)}.$$

For a tensor $x \in \mathbb{R}^{\times_{k \in D} I_k}$, $\overline{x}$ is equal to x.

Definition 2.4.5 (Transposed Tensor)**.** Let $A \in \mathbb{K}^{(\times_{k \in D} I_k) \times (\times_{k \in S^2} I_k)}$ be a tensor with some squared modes S. The symbol A^T denotes the tensor in the same space $\mathbb{K}^{(\times_{k \in D} I_k) \times (\times_{k \in S} I_k^2)}$ where the $R(k)$- and $C(k)$-modes are interchanged for each $k \in S$. The non-squared modes D are not modified.

Definition 2.4.6 (Diagonal Tensor)**.** Let $x \in \mathbb{K}^{\times_{k \in D^2} I_k}$ be a tensor. x is called *diagonal* if its matricization $\mathcal{M}(x)$ is diagonal.

Definition 2.4.7 (Identity Tensor)**.** Let D be some finite set. The *identity tensor*, denoted by $\mathbb{I}_D$, is the tensor in $\mathbb{K}^{\times_{k\in D^2} I_k}$ whose entries are given by

$$\mathbb{I}_D \left(i_{R(D)} \times i_{C(D)}\right) := \prod_{k\in D} \delta_{i_{R(k)}, i_{C(k)}}$$

where the $\delta_{i,j}$ on the right denotes the standard Kronecker symbol. If $D = \{k\}$ is a singleton, we also write $\mathbb{I}_k$ instead of $\mathbb{I}_{\{k\}}$.

Definition 2.4.8 (Inverse Tensor)**.** Let $x \in \mathbb{K}^{\times_{k\in D^2} I_k}$ be a tensor such that its matricization $\mathcal{M}(x)$ is invertible. The symbol x^{-1} refers to the unique tensor in $\mathbb{K}^{\times_{k\in D^2} I_k}$ such that $xx^{-1} = x^{-1}x = \mathbb{I}_D$.

Definition 2.4.9 (Orthogonal Tensor)**.** Let $x \in \mathbb{K}^{\times_{k\in D} I_k}$ be a tensor and $M \subset D$ some mode set. x is called *M-orthogonal* if its matricization $\mathcal{M}_{D\setminus M,M}(x)$ has orthonormal columns. If $M = \{k\}$ is a singleton, we also write k-orthogonality instead of $\{k\}$-orthogonality.

Theorem 2.4.10. *Let $x \in \mathbb{K}^{\times_{k\in D} I_k}$ be a tensor and $M \subset D$ some mode set. M-orthogonality of x is equivalent to $\overline{x} \,\langle M\rangle\, x = \mathbb{I}_M$.*

Proof. The claim follows trivially from $\mathcal{M}(\overline{x} \,\langle M\rangle\, x) = \mathcal{M}_{D\setminus M,M}(x)^* \,\mathcal{M}_{D\setminus M,M}(x)$. □

Definition 2.4.11 (Tensor Orthogonalization)**.** Let $x \in \mathbb{K}^{\times_{k\in D} I_k}$ be a tensor and $k \in D$ some mode such that $\prod_{\ell\in D\setminus\{k\}} \#I_\ell \geq \#I_k$. The symbol $\mathcal{Q}_k(x)$ denotes a pair $(q \in \mathbb{K}^{(\times_{\ell\in D\setminus\{k\}} I_\ell)\times[r_k]}, r \in \mathbb{K}^{[r_k]\times I_k})$ of tensors with $r_k := \#I_k$. We treat the mode pair $[r_k] \times I_k$ of r as a squared mode with $R(k)$ referring to $[r_k]$, and $C(k)$ referring to I_k. The tensors q, r are such that $x = qr$ and q is k-orthogonal.

Remark 2.4.12. Tensor orthogonalization can be implemented by computing the QR decomposition $QR := \mathcal{M}_{D\setminus\{k\},\{k\}}(x)$ and setting $\mathcal{M}_{D\setminus\{k\},\{k\}}(q) := Q$ and $\mathcal{M}(r) := R$. This procedure costs $\mathcal{O}\left((\#I_k)^2 \prod_{\ell\in D\setminus\{k\}} \#I_\ell\right)$ floating-point operations [24, §5.2.1].

2.5. Low-Rank Tensor Representation

Let $z \in \mathbb{K}^{\times_{k\in D} I_k}$ be a tensor and assume some modes $M \subset D$ can be *separated* from the others, i.e. there exist tensors $x \in \mathbb{K}^{[r_M]\times(\times_{k\in M} I_k)}$, $y \in \mathbb{K}^{[r_M]\times(\times_{k\in D\setminus M} I_k)}$ such that $z = xy$. The common mode of x and y, denoted by the symbol M, is called a *rank mode*, and its mode size $r_M \in \mathbb{N}$ is called *rank*. While storing z requires storing $\prod_{k\in D} \#I_k$ floating-point numbers, storing the two tensors x, y containing the same information requires

$$r_M \prod_{k\in M} \#I_k + r_M \prod_{k\in D\setminus M} \#I_k$$

floating-point numbers, which is much less if the rank r_M is small. Assuming that also the factors x and y are separable with low ranks, we can further increase the compression

rate by recursively writing these tensors as the mode product of a number of other tensors of even smaller dimensions. Eventually, we obtain a family of tensors x_V indexed by some set V such that $z = \prod_{v \in V} x_v$. Identities of this type are called *low-rank tensor representations*, and tensor families x_V are called *tensor networks*.

We give a precise definition of tensor networks and introduce the associated notation and data structures in Chapter 3. Here, we gather some basic results regarding low-rank representability of tensors. We start with an auxiliary result concerning the separability of tensors from so-called *tensor product spaces*.

Definition 2.5.1 (Tensor Product Space)**.** Let $D^{(x)}$, $D^{(y)}$ be two disjoint mode sets and $\mathbb{S}^{(x)} \subseteq \mathbb{K}^{\times_{k \in D^{(x)}} I_k}$, $\mathbb{S}^{(y)} \subseteq \mathbb{K}^{\times_{k \in D^{(y)}} I_k}$ two subspaces. The *tensor product space*

$$\mathbb{S}^{(x)} \otimes \mathbb{S}^{(y)} \subseteq \mathbb{K}^{\times_{k \in D^{(x)} \cup D^{(y)}} I_k}$$

is defined as

$$\mathbb{S}^{(x)} \otimes \mathbb{S}^{(y)} := \operatorname{span}\{xy \mid x \in \mathbb{S}^{(x)}, y \in \mathbb{S}^{(y)}\}.$$

Lemma 2.5.2. *Let $D^{(x)}$, $D^{(y)}$ be two disjoint mode sets, $\mathbb{S}^{(x)} \subseteq \mathbb{K}^{\times_{k \in D^{(x)}} I_k}$, $\mathbb{S}^{(y)} \subseteq \mathbb{K}^{\times_{k \in D^{(y)}} I_k}$ two subspaces and $z \in \mathbb{S}^{(x)} \otimes \mathbb{S}^{(y)}$ a tensor. There exist tensors*

$$x \in \mathbb{K}^{[r] \times \left(\times_{k \in D^{(x)}} I_k\right)}, \quad y \in \mathbb{K}^{[r] \times \left(\times_{k \in D^{(y)}} I_k\right)}$$

with rank $r \leq \min\left\{\dim \mathbb{S}^{(x)}, \dim \mathbb{S}^{(y)}\right\}$ such that $z = x\, y$.

Proof. Let $\hat{x} \in \mathbb{K}^{[r_x] \times (\times_{k \in M} I_k)}$ with $r_x := \dim \mathbb{S}^{(x)}$ be a tensor such that

$$\{\hat{x}(i_x) \mid i_x \in [r_x]\}$$

provides a basis for $\mathbb{S}^{(x)}$, and $\hat{y} \in \mathbb{K}^{[r_y] \times (\times_{k \in D \setminus M} I_k)}$ with $r_y := \dim \mathbb{S}^{(y)}$a tensor providing a basis for $\mathbb{S}^{(y)}$ in the same manner. There exists a tensor $\hat{w} \in \mathbb{K}^{[r_x] \times [r_y]}$ gathering the coefficients of the linear combination implied by

$$z \in \operatorname{span}\{\hat{x}(i_x)\, \hat{y}(i_y) \mid i_x \in [r_x], i_y \in [r_y]\}$$

such that $z = \hat{w}\hat{x}\hat{y}$. Setting

$$x = \begin{cases} \hat{x} & \text{if } r_x \leq r_y \\ \hat{w}\hat{x} & \text{otherwise} \end{cases}, \qquad y = \begin{cases} \hat{w}\hat{y} & \text{if } r_x \leq r_y \\ \hat{y} & \text{otherwise} \end{cases}$$

proves the result. □

Our aim is to prove separability of any tensor $z \in \mathbb{K}^{\times_{k \in D} I_k}$ with respect to any mode set $M \subset D$. Thanks to the above result, it only remains to prove tensor-product structure of $\mathbb{K}^{\times_{k \in D} I_k}$.

Theorem 2.5.3. *Let $\mathbb{K}^{\times_{k \in D} I_k}$ be a tensor space and $M \subset D$ a mode set. We have*

$$\mathbb{K}^{\times_{k \in M} I_k} \otimes \mathbb{K}^{\times_{k \in D \setminus M} I_k} = \mathbb{K}^{\times_{k \in D} I_k}.$$

Proof. The inclusion $\mathbb{K}^{\times_{k\in M} I_k} \otimes \mathbb{K}^{\times_{k\in D\setminus M} I_k} \subseteq \mathbb{K}^{\times_{k\in D} I_k}$ is obvious. By [22, Lemma 3.11(b)], the two spaces have the same dimension. □

Corollary 2.5.4. *Let $z \in \mathbb{K}^{\times_{k\in D} I_k}$ be a tensor and $M \subset D$ a mode set. There exist tensors $x \in \mathbb{K}^{[r_M]\times(\times_{k\in M} I_k)}$, $y \in \mathbb{K}^{[r_M]\times(\times_{k\in D\setminus M} I_k)}$ with rank*

$$r_M \leq \min\left\{\prod_{k\in M} \#I_k, \prod_{k\in D\setminus M} \#I_k\right\}$$

such that $z = xy$.

The separations asserted by the above corollary are not very useful because in the worst case $r_M = \min\left\{\prod_{k\in M} \#I_k, \prod_{k\in D\setminus M} \#I_k\right\}$, either x or y has as many entries as z. The following theorem indicates how we may find separations of lower rank.

Theorem 2.5.5. *Let $z \in \mathbb{K}^{\times_{k\in D} I_k}$ be a tensor, $M \subset D$ a mode set and $x \in \mathbb{K}^{[r_M]\times(\times_{k\in M} I_k)}$ a tensor with some rank $r_M \in \mathbb{N}$. There exists a tensor $y \in \mathbb{K}^{[r_M]\times(\times_{k\in D\setminus M} I_k)}$ such that $z = xy$ if and only if $\mathbb{S}_M(z) \subseteq \mathbb{S}_M(x)$[2].*

Proof. ($\Rightarrow$): $\mathbb{S}_M(z) = \operatorname{span}\left\{x\, y(i_{D\setminus M}) \mid i_{D\setminus M} \in \bigtimes_{k\in D\setminus M} I_k\right\}$

$$= \operatorname{span}\left\{\sum_{i_{\{M\}}\in[r_M]} x(i_{\{M\}})\, y(i_{\{M\}} \times i_{D\setminus M}) \mid i_{D\setminus M} \in \bigtimes_{k\in D\setminus M} I_k\right\}$$

$$\subseteq \operatorname{span}\left\{x(i_{\{M\}}) \mid i_{\{M\}} \in [r_M]\right\} = \mathbb{S}_M(x).$$

($\Leftarrow$): Since $z(i_{D\setminus M}) \in \mathbb{S}_M(x)$, there exists a $y(i_{D\setminus M}) \in \mathbb{K}^{[r_M]}$ such that $z(i_{D\setminus M}) = x\, y(i_{D\setminus M})$ for every $i_{D\setminus M} \in \bigtimes_{k\in D\setminus M} I_k$. Grouping the $y(i_{D\setminus M})$ into a tensor $y \in \mathbb{K}^{[r_M]\times(\times_{k\in D\setminus M} I_k)}$ yields the result. □

The first conclusion from Theorem 2.5.5 is that only the subspace $\mathbb{S}_M(x)$ induced by x is relevant, not x itself. To minimize the ranks, we should therefore choose the $x(i_M)$ such that they provide a basis for $\mathbb{S}_M(x)$. Additionally, we may optimize $\mathbb{S}_M(x)$ to be of smallest possible dimension, which by the above result means choosing $\mathbb{S}_M(x) = \mathbb{S}_M(z)$. This justifies the following definition.

Definition 2.5.6 (Optimal Rank)**.** Let $z \in \mathbb{K}^{\times_{k\in D} I_k}$ be a tensor and $M \subset D$ a mode set. The *optimal rank* $\operatorname{rank}_M(z)$ is defined as $\operatorname{rank}_M(z) := \dim \mathbb{S}_M(z)$. If $M = \{k\}$ is a singleton, we also write $\operatorname{rank}_k(x)$ instead of $\operatorname{rank}_{\{k\}}(x)$.

[2] In situations where a set of modes $M \subset D$ is also used as a label for a single mode (typically a rank mode) and it is not clear which interpretation is meant, we give preference to the set interpretation. Here, $\mathbb{S}_M(x)$ is thus a subspace of $\mathbb{K}^{\times_{k\in M} I_k}$, and the subspace of $\mathbb{K}^{[r_M]}$ induced by x would have to be denoted by $\mathbb{S}_{\{M\}}(x)$.

In more mathematically oriented texts (e.g. [22, Definition 3.32]), the term *rank* always denotes the optimal rank defined above. We therefore emphasize that we use *rank* in a much more relaxed manner to denote any mode size related in some sense to a separation of dimensions, irrespective of whether it is the optimal (i.e. smallest possible) rank or not.

We will repeatedly face the situation where we are given a separation $z = xy$ and want to check its optimality. One way to proceed would be to check whether the $\{M\}$-slices of x are a basis of $\mathbb{S}_M(z)$, but the following theorem provides an easier and more elegant alternative.

Theorem 2.5.7. *Let $z \in \mathbb{K}^{\times_{k\in D} I_k}$ be a tensor, $M \subset D$ a mode set and $x \in \mathbb{K}^{[r_M]\times(\times_{k\in M} I_k)}$, $y \in \mathbb{K}^{[r_M]\times(\times_{k\in D\setminus M} I_k)}$ tensors with rank $r_M \in \mathbb{N}$ such that $z = xy$. Linear independence of the sets*

$$\{x(i_{\{M\}}) \mid i_{\{M\}} \in [r_M]\}, \quad \{y(i_{\{M\}}) \mid i_{\{M\}} \in [r_M]\}$$

implies $\mathbb{S}_M(z) = \mathbb{S}_M(x)$ and $r_M = \operatorname{rank}_M(z)$.

Proof. The first implication is proven in [22, Lemma 6.5]. Since the $x(i_{\{M\}})$ are linearly independent, they are a basis of $\mathbb{S}_M(x) = \mathbb{S}_M(z)$ which proves the second implication. □

We now focus on splitting a tensor $z \in \mathbb{K}^{\times_{k\in D} I_k}$ into more factors than the binary case $z = xy$ considered above. For illustration, assume we partition D into three sets $M^{(0)}$, $M^{(1)}$, $M^{(2)}$ and want to determine three tensors

$$x^{(0)} \in \mathbb{K}^{[r_1]\times\left(\times_{k\in M^{(0)}} I_k\right)}, \quad x^{(1)} \in \mathbb{K}^{[r_1]\times[r_2]\times\left(\times_{k\in M^{(1)}} I_k\right)}, \quad x^{(2)} \in \mathbb{K}^{[r_2]\times\left(\times_{k\in M^{(2)}} I_k\right)}$$

such that $z = x^{(0)}x^{(1)}x^{(2)}$. We can obtain such a ternary splitting using two binary splittings as in $z = x^{(0)}(x^{(1)}x^{(2)})$ or $z = (x^{(0)}x^{(1)})x^{(2)}$. In the first case, the optimal rank r_1 is $\operatorname{rank}_{M^{(0)}}(z)$ whereas in the second case it is $\operatorname{rank}_{M^{(0)}}(x^{(0)}x^{(1)})$. At this point, it is therefore not clear whether the order in which we separate the modes influences the final rank structure we obtain. This question is settled in the following theorem by showing that the order does not matter.

Theorem 2.5.8. *Let $z \in \mathbb{K}^{\times_{k\in D} I_k}$ be a tensor, $M^{(1)} \subset D$, $M^{(2)} \subset M^{(1)}$ two mode sets and $x^{(1)} \in \mathbb{K}^{[r_1]\times\left(\times_{k\in M^{(1)}} I_k\right)}$ a tensor with rank $r_1 \in \mathbb{N}$ such that $\mathbb{S}_{M^{(1)}}(x^{(1)}) = \mathbb{S}_{M^{(1)}}(z)$. Then, $\mathbb{S}_{M^{(2)}}(x^{(1)}) = \mathbb{S}_{M^{(2)}}(z)$.*

Proof. [22, Remark 11.14]. □

Finally, we discuss a special case related to the HTR networks introduced in Section 3.2. We want to split a mode set D into three parts, namely a single mode k and two disjoint mode sets $M^{(x)}, M^{(y)} \subset D \setminus \{k\}$ such that $D = \{k\} \cup M^{(x)} \cup M^{(y)}$. We propose the following scheme to find ranks $r_x, r_y \in \mathbb{N}$ and tensors

$$w \in \mathbb{K}^{[I_k]\times[r_x]\times[r_y]}, \quad x \in \mathbb{K}^{[r_x]\times\left(\times_{k\in M^{(x)}} I_k\right)}, \quad y \in \mathbb{K}^{[r_y]\times\left(\times_{k\in M^{(y)}} I_k\right)}$$

such that $z = wxy$, with $z \in \mathbb{K}^{\times_{k\in D} I_k}$.

- For each $i_k \in I_k$, determine optimal separations $z(i_k) = \hat{x}(i_k)\,\hat{y}(i_k)$ with

$$\hat{x}(i_k) \in \mathbb{K}^{[r_x(i_k)]\times\left(\times_{k\in M^{(x)}} I_k\right)}, \quad \hat{y}(i_k) \in \mathbb{K}^{[r_y(i_k)]\times\left(\times_{k\in M^{(y)}} I_k\right)}$$

  and $r_x(i_k) := \operatorname{rank}_{M^{(x)}}(z(i_k))$, $r_y(i_k) := \operatorname{rank}_{M^{(y)}}(z(i_k))$.

- Determine two bases

$$\{x(i_x) \mid i_x \in [r_x]\} \subset \operatorname{span}\{\hat{x}(i_k)(i_x) \mid i_k \in I_k, i_x \in [r_x(i_k)]\},$$
$$\{y(i_y) \mid i_y \in [r_y]\} \subset \operatorname{span}\{\hat{y}(i_k)(i_y) \mid i_k \in I_k, i_y \in [r_y(i_k)]\}$$

  of size $r_x, r_y \in \mathbb{N}$ for the right-hand side spaces, and group them into tensors $x \in \mathbb{K}^{[r_x]\times\left(\times_{k\in M^{(x)}} I_k\right)}$, $y \in \mathbb{K}^{[r_y]\times\left(\times_{k\in M^{(y)}} I_k\right)}$.

- Determine a tensor $w \in \mathbb{K}^{[I_k]\times[r_x]\times[r_y]}$ such that $z = wxy$.

By Theorem 2.5.5 and treating $\overline{r_x r_y}$ as a long mode, the last step is well-posed if and only if

$$\mathbb{S}_{M^{(x)}\cup M^{(y)}}(z) \subseteq \mathbb{S}_{M^{(x)}}(x) \otimes \mathbb{S}_{M^{(y)}}(y). \tag{2.1}$$

We prove this inclusion in two steps.

Proposition 2.5.9. *Let $z \in \mathbb{K}^{\times_{k\in D} I_k}$ be a tensor, $k \in D$ a mode and $M \subset D \setminus \{k\}$ a mode set. We have*

$$\mathbb{S}_M(z) = \operatorname{span} \bigcup_{i_k \in I_k} \mathbb{S}_M(z(i_k)).$$

Proof. [22, Proposition 6.10]. □

Proposition 2.5.10. *Let $z \in \mathbb{K}^{\times_{k\in D} I_k}$ be a tensor, and $M^{(x)}, M^{(y)} \subset D$ two disjoint mode sets. We have*

$$\mathbb{S}_{M^{(x)}\cup M^{(y)}}(z) \subseteq \mathbb{S}_{M^{(x)}}(z) \otimes \mathbb{S}_{M^{(y)}}(z).$$

Proof. [22, Corollary 6.18]. □

Proposition 2.5.9 shows that $\mathbb{S}_{M^{(x)}}(x) = \mathbb{S}_{M^{(x)}}(z)$ and $\mathbb{S}_{M^{(y)}}(y) = \mathbb{S}_{M^{(y)}}(z)$, and thus Proposition 2.5.10 proves (2.1). Furthermore, the first part ascertains that the ranks found in this manner are optimal.

2.6. The Tensor Singular Value Decomposition

The last section introduced and motivated low-rank tensor representations. Computationally, such low-rank representations are almost always determined by the *singular value decomposition* (SVD) defined next.

Definition 2.6.1 (Tensor SVD [23])**.** Let $x \in \mathbb{K}^{\times_{k\in D} I_k}$ be a tensor and $k \in D$ some mode such that $\prod_{\ell\in D\setminus\{k\}} \#I_\ell \geq I_k$. The symbol $\mathcal{S}_k(x)$ denotes a triplet

$$\left(u \in \mathbb{K}^{(\times_{\ell\in D\setminus\{k\}} I_\ell)\times[r_k]}, s \in \mathbb{K}^{[r_k]^2}, v \in \mathbb{K}^{[r_k]\times I_k}\right)$$

with $r_k := \#I_k$. We treat the mode pair $[r_k] \times I_k$ of v as a squared mode with $R(k)$ referring to $[r_k]$, and $C(k)$ referring to I_k. The tensors u, s and v are such that $x = usv$, s is diagonal and u and v are k- and $R(k)$-orthogonal, respectively. The diagonal entries of s are real and non-negative. They are called *singular values* and denoted by s_{i_k}, $i_k \in [r_k]$. We assume the singular values to be sorted in decreasing order, i.e. $s_{i_k} \geq s_{i'_k}$ for $i_k < i'_k$.

Remark 2.6.2. The tensor SVD can be implemented by computing the matrix SVD $U\Sigma V^* := \mathcal{M}_{D\setminus\{k\},k}(x)$ and setting $\mathcal{M}_{D\setminus\{k\},k}(u) := U$, $\mathcal{M}(s) := \Sigma$ and $\mathcal{M}(v) := V^*$ (note the adjoint). This procedure costs $\mathcal{O}\left((\#I_k)^2 \prod_{\ell\in D\setminus\{k\}} \#I_\ell\right)$ floating-point operations [24, §5.4.5].

The above version of the SVD is not very useful because it always returns a representation of the largest possible rank $r_k := \#I_k$. Assuming that some singular values are zero, however, we may define the *reduced rank* r'_k as the smallest integer such that $s_{i_k} = 0$ for $i_k > r'_k$, and the *reduced tensors*

$$u' \in \mathbb{K}^{(\times_{\ell\in D\setminus\{k\}} I_\ell)\times[r'_k]}, \quad s' \in \mathbb{K}^{[r'_k]^2}, \quad v' \in \mathbb{K}^{[r'_k]\times I_k}$$

obtained by removing the last $r_k - r'_k$ slices in the k-mode of u, the k^2-modes of s and the $R(k)$-mode of v. Since

$$usv = \sum_{i_k=0}^{r_k-1} s_{i_k}\, u(i_k)\, v(i_k) = \sum_{i_k=0}^{r'_k-1} s_{i_k}\, u(i_k)\, v(i_k) = u's'v',$$

we have $x = usv = u's'v'$, i.e. the reduced tensors provide a low-rank representation of x with rank $r'_k \leq r_k$. In fact, by the k- and $R(k)$-orthogonality of u and v and Theorem 2.5.5, the reduced rank r'_k is the optimal $\operatorname{rank}_k(x)$.

In cases where even the optimal rank r'_k is too large to store the factors u, s and v, we may further compress x by setting the smallest $r'_k - r''_k$ non-zero singular values to zero, with $r''_k \leq r'_k$ the new rank of the (now approximate) representation. This procedure is called *truncation*, and the error it introduces is estimated by the following theorem.

Theorem 2.6.3 (Truncation Error)**.** *Let $r_k \in \mathbb{N}$ be an integer and*

$$u \in \mathbb{K}^{(\times_{\ell\in D\setminus\{k\}} I_\ell)\times[r_k]}, \quad v \in \mathbb{K}^{[r_k]\times I_k}, \quad s \in \mathbb{K}^{[r_k]^2}, \quad \tilde{s} \in \mathbb{K}^{[r_k]^2}$$

tensors such that u and v are k- and $R(k)$-orthogonal. Then, $\|usv - u\tilde{s}v\| = \|s - \tilde{s}\|$.

Proof. This result is a special case of Theorem 3.1.5 which we will prove later. □

Denoting by u'', s'', v'' the reduced tensors after truncation, the above theorem yields

$$\|x - u''s''v''\| = \sqrt{\sum_{i_k=r''_k+1}^{r'_k} s_{i_k}^2}.$$

2.7. Quantization

Quantization as introduced in [17, 18, 19, 20] denotes the splitting of a single mode k of size n_k into l modes $j \in [l]$ of size n_j such that k may be interpreted as the joint long mode of the $j \in [l]$, i.e. $k = \overline{[l]}$. This condition implies that the $n_{[l]}$ have to be a *factorization* of n_k, i.e. $n_k = \prod_{j=0}^{l-1} n_j$. The original mode k typically has a natural interpretation in the given application and is therefore called *physical*, while the new modes $j \in [l]$ are called *virtual*.

This section introduces a notation to make working with quantized tensors more convenient. We begin by defining generic labels for the virtual modes.

Definition 2.7.1 (Virtual Mode Labels)**.** Let k be a physical mode split into $l_k \in \mathbb{N}$ virtual modes. The jth virtual mode of k is denoted by (k, j) for all $j \in [l_k]$. The set of all virtual modes of k is denoted by $(k, [l_k]) := \{(k, j) \mid j \in [l_k]\}$.

Let D be a set of physical modes where each $k \in D$ is split into $l_k \in \mathbb{N}$ virtual modes. The set of all virtual modes of all physical modes is denoted by $(D, [l]) := \bigcup_{k \in D} (k, [l_k])$.

While we allow a general mode k to be indexed by an arbitrary index set I_k, we restrict this freedom for quantized modes. Specifically, we require the index set of a physical mode k to be the integers $[n_k]$, and its jth virtual mode must have the index set $[n_{(k,j)}]$ where $n_{(k,[l_k])}$ is some factorization of n_k. This structure allows for a generic map between physical and virtual indices.

Definition 2.7.2 (Quantized Index)**.** Let $(k, [l_k])$ be a quantized mode. We define the bijection

$$f : \bigtimes_{(k,j) \in (k,[l_k])} [n_{(k,j)}] \to [n_k], \quad i_{(k,[l_k])} \mapsto \sum_{j=0}^{l_k-1} s_{(k,j)}\, i_{(k,j)}, \quad s_{(k,j)} := \prod_{j'=0}^{j-1} n_{(k,j)}.$$

Typically, the physical modes k are more convenient to work with whereas the virtual modes $(k, [l_k])$ enable us to exploit important additional structure in the tensors. With the intent to make switching between the quantized and non-quantized views as easy as possible, we introduce the following conventions.

Definition 2.7.3. Let $(k, [l_k])$ be a quantized mode. The quantized index $i_{(k,[l_k])} \in \bigtimes_{(k,j) \in (k,[l_k])} [n_{(k,j)}]$ implicitly also defines the non-quantized index $i_k := f\left(i_{(k,[l_k])}\right)$, and vice versa.

Definition 2.7.4. Let $(k, [l_k])$ be a quantized mode and $i_k, i_{(k,[l_k])}$ an index pair from Definition 2.7.3. We define the expression $x\left(i_{(k,[l_k])}\right) := x\left(i_k\right)$ for any tensor $x \in \mathbb{K}^{[n_k]}$, and likewise $x(i_k) := x\left(i_{(k,[l_k])}\right)$ for any tensor $x \in \mathbb{K}^{\times_{(k,j)\in(k,[l_k])}[n_{(k,j)}]}$.

The idea behind quantization is to increase the dimension of a tensor and thereby open up new possibilities for low-rank representations. This is demonstrated in the following example, which at the same time also illustrates the new notation developed in this section.

Example 2.7.5 (Proposition 1.1 in [19])**.** Consider the tensor $x \in \mathbb{K}^{[n_k]}$ given by $x(i_k) := \exp(i_k)$. Assume k is quantized into l virtual modes, i.e. $k = \overline{(k, [l])}$. We have

$$\exp(i_k) = \exp\left(\sum_{j=0}^{l-1} s_{(k,j)}\, i_{(k,j)}\right) = \prod_{j=0}^{l-1} \exp\left(s_{(k,j)}\, i_{(k,j)}\right). \tag{2.2}$$

Therefore, defining $x_j \in \mathbb{K}^{[n_{(k,j)}]}$, $x_j(i_{(k,j)}) := \exp\left(s_{(k,j)}\, i_{(k,j)}\right)$ for all $j \in [l]$, we obtain

$$x = \prod_{j=0}^{l-1} x_j. \tag{2.3}$$

Storing x explicitly amounts to storing $n_k = \prod_{j=0}^{l-1} n_{(k,j)}$ floating-point numbers, whereas storing it as the mode product $\prod_{j=0}^{l-1} x_j$ requires only $\sum_{j=0}^{l-1} n_{(k,j)}$ floating-point numbers.

Note how (2.2) and (2.3) make use of Definitions 2.7.3 and 2.7.4, respectively.

3. Tensor Networks

Matrix decompositions are well-known tools to simplify certain operations. Examples include the LU decomposition for solving linear system, the eigenvalue decomposition for computing functions of matrices or the singular value decomposition for compression. Their higher-dimensional analogues are tensor networks, which represent a single tensor as the mode product of several tensors. This chapter introduces the associated data structures and notation and recalls basic algorithms and results.

3.1. General Networks

Definition 3.1.1 (Tensor Network)**.** Let V be a finite set and E_V, D_V families of finite sets parametrized by V. Define $E := \bigcup_{v\in V} E_v$, $D := \bigcup_{v\in V} D_v$. Assume E and D are disjoint, the D_V are pairwise disjoint and $\#\{v \in V \mid e \in E_v\} = 2$ for each $e \in E$. Additionally, let $r_E \in \mathbb{N}^E$ be a tuple of integers and I_D a family of finite sets.

A *tensor network* is a tuple of tensors $x_V \in \bigtimes_{v\in V} \mathbb{K}^{(\times_{e\in E_v}[r_e])\times(\times_{k\in D_v} I_k)}$. It defines a tensor $x \in \mathbb{K}^{\times_{k\in D} I_k}$ given by $x := \prod_{v\in V} x_v$. The $v \in V$ are called *vertices*, the $e \in E$ *edges* and the $k \in D$ *free modes*. The r_E are called *ranks*.

We use the symbol x to denote both the tensor x defined above and the tuple of tensors x_V. Likewise, we do not distinguish notationally between E and E_V nor between D and D_V, and we define $r := r_E$ and $I := I_D$. We will use the abbreviation

$$\mathrm{TN}(V, E, r, D, I) := \bigtimes_{v\in V} \mathbb{K}^{(\times_{e\in E_v}[r_e])\times(\times_{k\in D_v} I_k)}$$

to denote the space of all tensor networks with network structure V, E and D, ranks r and index sets I.

Free modes may appear in pairs of row and column modes, i.e. $R(k), C(k) \in D$ for some object k. Edge modes $e \in E$ are assumed to be non-squared.

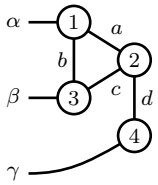
Edge modes of two networks $x \in \mathrm{TN}(V, E, r^{(x)}, D^{(x)}, I^{(x)})$, $y \in \mathrm{TN}(V, E, r^{(y)}, D^{(y)}, I^{(y)})$ are considered distinct. In particular, in expressions of the form $x_v\, y_v$ with $v \in V$, the modes $e \in E_v$ are not multiplied. If it is not clear that a mode e belongs to the network x, we clarify by tagging e as in $x(e)$.

The last rule will become important once we multiply vertex tensors from different networks. This is illustrated in the following example.

Example 3.1.2. Let $V := \{1, 2\}$, $E_1 := \{e\}$, $E_2 := \{e\}$, $D_1 := \{k\}$, $D_2 := \{\}$, and $x, y \in \mathrm{TN}(V, E, r, D, I)$. The expression $x_1\, y_1$ denotes multiplication of x_1, y_1 over mode k but not e. The result is thus in $\mathbb{K}^{[r_{x(e)}]\times[r_{y(e)}]}$, where we clarified that we have one e-mode coming from x and one from y by tagging these modes accordingly.

$$V := \{1, 2, 3, 4\}$$
$$E_1 := \{a, b\}, E_2 := \{a, c, d\}, E_3 := \{b, c\}, E_4 := \{d\}$$
$$D_1 := \{\alpha\}, D_2 := \{\}, D_3 := \{\beta\}, D_4 := \{\gamma\}$$

(a) Tensor network



(b) Diagram

Figure 3.1.: Example tensor network diagram.

As implied by the terminology, tensor networks are closely related to graphs. We can use this relationship to visualize tensor networks as shown in Figure 3.1. Such visualizations are called *tensor network diagrams*.

In matrix decompositions, we are often interested in how errors in one of the factors influence the overall error. For example, it is crucial for SVD-based compression of a matrix A with singular value decomposition $U\Sigma V^* := A$ that

$$\|\tilde{\Sigma} - \Sigma\| < \varepsilon \implies \|A - U\tilde{\Sigma}V^*\| < \varepsilon \tag{3.1}$$

for any matrix $\tilde{\Sigma}$. In order to do a similar analysis for tensor networks, we need some quantity which captures the effect of a single vertex tensor on the tensor represented by the network.

Definition 3.1.3 (Environment Tensor)**.** Let $x \in \mathrm{TN}(V, E, r, D, I)$ be a tensor network and $v \in V$ a vertex. We define the complement $D_v^c := D \setminus D_v$ of D_v. The *environment tensor*

$$U_v(x) \in \mathbb{K}^{(\times_{e \in E_v}[r_e]) \times (\times_{k \in D_v^c} I_k)}$$

is defined as

$$U_v(x) := \prod_{u \in V \setminus \{v\}} x_v.$$

Informally, the environment tensor is the contraction of what is left after removing a single vertex v from the network. As promised, it captures the influence of x_v on x since we have $x = U_v(x)\, x_v$.

The stability property (3.1) of the SVD is due to the fact that U and V are orthogonal. The appropriate generalization of matrix orthogonality is defined next.

Definition 3.1.4 (Orthogonal Tensor Network)**.** Let $x \in \mathrm{TN}(V, E, r, D, I)$ be a tensor network and $v \in V$ a vertex. x is called v-orthogonal if the environment tensor $U_v(x)$ is E_v-orthogonal.

This generalization preserves the stability property of orthogonal matrices:

Theorem 3.1.5. *Let $x \in \mathrm{TN}(V, E, r, D, I)$ be a tensor network and $v \in V$ a vertex such that x is v-orthogonal. Let $\tilde{x} \in \mathrm{TN}(V, E, r, D, I)$ be another tensor network such that $\tilde{x}_u = x_u$ for all $u \in V \setminus \{v\}$. We have*

$$\|\tilde{x} - x\| = \|\tilde{x}_v - x_v\|.$$

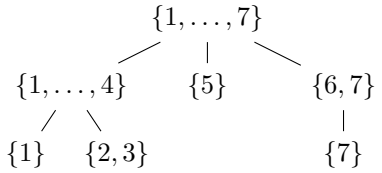


Figure 3.2.: Example dimension partition tree for $D = \{1, \ldots, 7\}$.

Proof. v-orthogonality of x implies $\overline{U_v(x)} \,\langle E_v \rangle\, U_v(x) = \mathbb{I}_{E_v}$. Therefore,

$$\begin{aligned}\|\tilde{x} - x\|^2 &= \overline{(\tilde{x} - x)}\,(\tilde{x} - x) = \overline{(\tilde{x}_v - x_v)}\,\left(\overline{U_v(x)} \,\langle E_v \rangle\, U_v(x)\right)(\tilde{x}_v - x_v) \\ &= \overline{(\tilde{x}_v - x_v)}\,(\tilde{x}_v - x_v) = \|\tilde{x}_v - x_v\|^2.\end{aligned}$$

□

If we use the same symbol to reference a vertex and a set of free modes, Definition 3.1.4 collides with Definition 2.4.9. In such situations, we give preference to the concept defined in this section.

3.2. Hierarchical Tucker Representation

The *Hierarchical Tucker Representation* (HTR) [5] is a specific tensor network based on a hierarchical splitting of the free modes D. To construct such a splitting, we first split D into some number $c \in \{0, \ldots, \#D\}$ of pairwise disjoint, non-empty subsets D_i. We do not require this splitting to be complete, i.e. $\bigcup_{i=1}^{c} D_i = D$ is not required. Next, we split each D_i again into $c_i \in \{0, \ldots, \#D_i\}$ disjoint, non-empty but not necessarily complete subsets D_{ij}, and we continue recursively until we are satisfied with the splitting. The set

$$T_D = \{D,\, D_1, \ldots, D_c,\, D_{11}, \ldots, D_{1c_1},\, \ldots,\, D_{c1}, \ldots, D_{cc_c},\, \ldots\}$$

of all the sets constructed in this manner is called a *dimension partition tree* (see also Figure 3.2).

Definition 3.2.1 (Dimension Partition Tree)**.** Let D be some finite set and $\mathcal{P}(D)$ its power set, i.e. the set of all improper subsets of D. A set $T_D \subset \mathcal{P}(D)$ is called a *dimension partition tree* if it satisfies $D \in T_D$, $\{\} \notin T_D$ and $\alpha \cap \beta \neq \{\} \implies \alpha \subseteq \beta \vee \beta \subseteq \alpha$ for all $\alpha, \beta \in T_D$. The last condition establishes the hierarchical structure of T_D.

D is called the *root* of T_D. We define the following terms for $\alpha \in T_D$ and $\alpha' \in T_D \setminus \{D\}$:

$$\begin{aligned}\text{descendant}(\alpha) &:= \{\beta \in T_D \mid \beta \subseteq \alpha\}, \\ \text{ancestor}(\alpha) &:= \{\beta \in T_D \mid \alpha \subset \beta\}, \\ \text{child}(\alpha) &:= \{\beta \in \text{descendant}(\alpha) \setminus \{\alpha\} \mid \nexists\, \gamma \in T_D : \beta \subset \gamma \subset \alpha\}, \\ \text{parent}(\alpha') &:= \beta \in T_D \text{ such that } \alpha' \in \text{child}(\beta), \\ \text{neighbour}(\alpha) &:= \begin{cases} \text{child}(\alpha) \cup \{\text{parent}(\alpha)\} & \text{if } \alpha \neq D \\ \text{child}(\alpha) & \text{if } \alpha = D \end{cases},\end{aligned}$$

$$\begin{aligned}\operatorname{sibling}(\alpha') &:= \operatorname{child}(\operatorname{parent}(\alpha')) \setminus \{\alpha'\},\\ \operatorname{level}(\alpha) &:= \#\operatorname{ancestor}(\alpha),\\ \operatorname{inv}\operatorname{level}(\alpha) &:= \max\{\operatorname{level}(\beta) \mid \beta \in \operatorname{descendant}(\alpha)\} - \operatorname{level}(\alpha).\end{aligned}$$ [1]

The dimension partition tree T_D on which these terms depend is implicitly given as the tree from which α or α' were taken. Furthermore, we define the sets

$$\begin{aligned}\operatorname{leaf}(T_D) &:= \{\alpha \in T_D \mid \operatorname{child}(\alpha) = \{\}\},\\ \operatorname{interior}(T_D) &:= T_D \setminus (\{D\} \cup \operatorname{leaf}(T_D)),\\ \operatorname{level}_\ell(T_D) &:= \{\alpha \in T_D \mid \operatorname{level}(\alpha) = \ell\}.\end{aligned}$$

Definition 3.2.2 (HTR Network)**.** Let T_D be a dimension partition tree. A tensor network $x \in \mathrm{TN}(T_D, E, r, D, I)$ with $E_\alpha = \{\alpha\} \cup \operatorname{child}(\alpha)$ for all $\alpha \in T_D \setminus \{D\}$ and $E_D = \operatorname{child}(D)$, and $D_\alpha = \alpha \setminus \left(\bigcup_{\beta \in \operatorname{child}(\alpha)} \beta\right)$ is called an *HTR network*. We define the abbreviation

$$\mathrm{HTR}(T_D, r, I) := \mathrm{TN}(T_D, E, r, D, I)$$

to denote the space of all HTR networks with given dimension partition tree T_D, ranks $r = r_{T_D} \in \mathbb{N}^{T_D}$ and index sets $I = I_D$. Note that for notational convenience, we subscript r with T_D instead of the actual domain of definition $T_D \setminus \{D\}$.

In the existing literature (e.g. [22, Definition 11.2]), dimension partition trees are required to satisfy two additional conditions. We call such a dimension partition tree *standard*.

Definition 3.2.3 (Standard Dimension Partition Tree)**.** A dimension partition tree T_D is called *standard* if it satisfies

$$\#\alpha \geq 2 \implies \#\operatorname{child}(\alpha) = 2 \wedge D_\alpha = \{\}$$

for each $\alpha \in T_D$.

In text form, the constraints introduced by standardness are the following:

- T_D has to be a proper binary tree, i.e. $\operatorname{child}(\alpha) = 0 \vee \operatorname{child}(\alpha) = 2$ for all $\alpha \in T_D$.
- Only the leaf vertices $\alpha \in \operatorname{leaf}(T_D)$ have free modes, and each such leaf vertex has exactly one free mode.

We will assume standardness whenever the form of a result depends on the specific shape of the tree. Typically, these results are complexity estimates like the following.

Theorem 3.2.4. *Let T_D be a standard dimension partition tree, $x \in \mathrm{HTR}(T_D, r, I)$ an HTR network and $d := \#D$, $n := \max_{k \in D} \#I_k$, $r := \max_{e \in E} r_e$. The storage cost of x is $dnr + (d-2)r^3 + r^2$.*

[1] Put differently, $\operatorname{inv}\operatorname{level}(\alpha)$ is the longest distance from α to any leaf in $\operatorname{descendant}(\alpha)$. It is "inverse" to $\operatorname{level}(\alpha)$ in that it increases in leaves-to-root direction.

Proof. The following table specifies the storage cost of x_α for $\alpha \in \text{leaf}(T_D)$, $\alpha \in \text{interior}(\alpha)$ and $\alpha = D$, and indicates how many such α exist. The complexity estimate then follows from multiplying and adding its elements.

	Number of vertices	Cost per vertex
leaves	d	nr
interior	$d-2$	r^3
root	1	r^2

□

Another tree property, important for parallelization, is *balancedness*.

Definition 3.2.5 (Balanced Dimension Partition Tree)**.** A dimension partition tree T_D is called *balanced* if it satisfies

$$\max_{\alpha \in \text{leaf}(T_D)} \text{level}(\alpha) \; - \min_{\alpha \in \text{leaf}(T_D)} \text{level}(\alpha) \leq 1.$$

Finally, we introduce an abbreviation to square all modes of a dimension partition tree.

Definition 3.2.6 (Squared Dimension Partition Tree)**.** Let T_D be a dimension partition tree and $\alpha \in T_D$ a vertex. We define $T_D^2 := \{\alpha^2 \mid \alpha \in T_D\}$.

We constructed HTR networks by first constructing the dimension partition tree T_D and then deriving the vertices, edges and free modes from T_D. One can also proceed the other way around: we first specify a tree-structured graph, attach the free modes to its vertices, choose one of its vertices as the root and then construct the vertex labels $\alpha \in T_D$ as the set of all free modes in the subtree of the vertex to be labelled. The latter approach shows more clearly that dimension partition trees are to some extent arbitrary. If we choose two different vertices as the root, we obtain two trees T_D and $\tilde{T}_D$ which are different but in some sense equivalent. The following function maps between such equivalent dimension partition trees.

Definition 3.2.7 (Tree Rerooting)**.** Let T_D be a dimension partition tree and $\alpha \in T_D$ a vertex. We define for all $\beta \in T_D$:

$$\mathcal{R}_\alpha(\beta) := \begin{cases} D \setminus \{\gamma \in \text{child}(\beta) \mid \alpha \subseteq \gamma\} & \text{if } \beta \in \text{ancestor}(\alpha) \\ D & \text{if } \beta = \alpha \\ \beta & \text{otherwise} \end{cases}.$$

We refer to Figure 3.4 for an example of the action of $\mathcal{R}_\alpha$. With the notation in place, we can precisely specify what we mean by "equivalent".

Theorem 3.2.8. *Let T_D be a dimension partition tree and $\alpha, \beta \in T_D$ vertices. We have:*

- $\mathcal{R}_\alpha(\alpha) = D$ *(α becomes the new root.)*
- $D_{\mathcal{R}_\alpha(\beta)} = D_\beta$ *(The free modes are preserved.)*

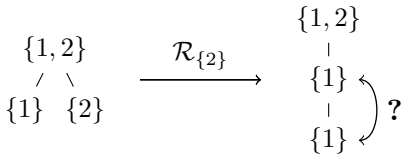


Figure 3.3.: Two different vertices are mapped to the same vertex under tree rerooting.

- neighbour$(\mathcal{R}_\alpha(\beta)) = \mathcal{R}_\alpha(\text{neighbour}(\beta))$ *(The network structure is preserved.)*

The proof of Theorem 3.2.8 is irrelevant to the exposition here. We therefore omit it.

Sometimes we would like to use the terminology for rooted trees defined in Definition 3.2.1 in a rerooted version $\mathcal{R}_\alpha(T_D)$ of T_D. In order to keep the notation concise, we define the following abbreviations.

Definition 3.2.9 (Relative Tree Terms)**.** Let T_D be a dimension partition tree and $\alpha, \beta \in T_D$ two vertices. We define the terms

$$\tau(\beta \mid \alpha) = \mathcal{R}_\alpha^{-1}(\tau(\mathcal{R}_\alpha(\beta)))$$

where τ is a template for any element of the set $\{\text{descendant}, \text{ancestor}, \text{child}, \text{parent}, \text{sibling}\}$.

A technical difficulty of tree rerooting is that different vertices $\alpha, \beta \in T_D$ may be mapped to the same set $\mathcal{R}_\gamma(\alpha) = \mathcal{R}_\gamma(\beta)$, see Figure 3.3. We will not introduce a new notation to resolve this problem, however, because we use $\mathcal{R}$ only as a tool to formulate Definition 3.2.9 whose intended meaning is clear even with this flawed notation.

In Definition 3.2.2, we assigned the same label β both to a vertex in $T_D \setminus \{D\}$ as well as to the edge which connects this vertex to its parent. Given a non-root vertex $\beta \in T_D$, it is therefore very easy to refer to this *parent edge* of β. We would like to make it similarly easy to refer to the parent edge of β *relative to a vertex* $\alpha \in T_D$, i.e. the edge which points from β towards α. This is achieved with the following definition.

Definition 3.2.10 (Relative Parent Edge)**.** Let T_D be a dimension partition tree and $\alpha, \beta \in T_D$ two vertices. We define

$$\uparrow(\beta \mid \alpha) := \begin{cases} \text{parent}(\beta \mid \alpha) & \text{if } \beta \in \text{ancestor}(\alpha) \\ \beta & \text{otherwise} \end{cases}.$$

Example 3.2.11. Consider the left dimension partition tree from Figure 3.4. We have

$$\begin{aligned} \text{descendant}(\{1,\dots,4\} \mid \{2,3\}) &= \{\{1,\dots,4\}, \{1\}, \{1,\dots,7\}, \{5\}, \{6,7\}, \{7\}\}, \\ \text{ancestor}(\{6,7\} \mid \{2,3\}) &= \{\{1,\dots,7\}, \{1,\dots,4\}, \{2,3\}\}, \\ \text{child}(\{1,\dots,4\} \mid \{2,3\}) &= \{\{1,\dots,7\}, \{1\}\}, \end{aligned}$$

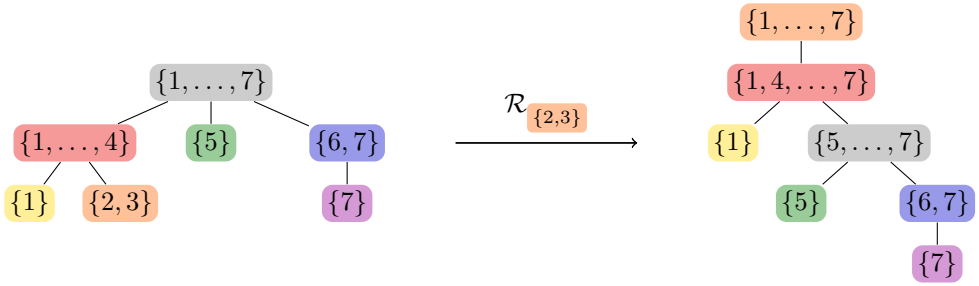


Figure 3.4.: Example of tree rerooting.

$$\text{parent}(\{1,\dots,7\} \mid \{2,3\}) = \{1,\dots,4\},$$
$$\text{sibling}(\{1,\dots,7\} \mid \{2,3\}) = \{\{1\}\}.$$

For the parent edges, we have

$$\uparrow(\{1\} \mid \{2,3\}) = \{1\}, \quad \uparrow(\{1,\dots,7\} \mid \{2,3\}) = \{1,\dots,4\}.$$

We did not color the right-hand sides to emphasize that they are edge labels and not vertex labels.

In the HTR case, the environment tensor (Definition 3.1.3) has a little sibling.

Definition 3.2.12 (Subtree Tensor)**.** Let $x \in \mathrm{HTR}(T_D, r, I)$ be an HTR network and $\alpha, \beta \in T_D$ two vertices. The *subtree tensor*

$$S_{\beta|\alpha}(x) \in \begin{cases} \mathbb{K}^{[r_{\uparrow(\beta|\alpha)}]\times(\times_{k\in\mathcal{R}_\alpha(\beta)} I_k)} & \text{if } \beta \neq \alpha \\ \mathbb{K}^{\times_{k\in D} I_k} & \text{if } \beta = \alpha \end{cases}$$

is defined as

$$S_{\beta|\alpha}(x) := \prod_{\gamma\in\mathrm{descendant}(\beta|\alpha)} x_\gamma.$$

As before, we define $S_\beta(x) := S_{\beta|D}(x)$.

3.3. HTR Orthogonalization

Section 3.1 introduced the meaning of tensor network orthogonality and motivated why it is a useful concept. One advantage of the HTR is that we can *orthogonalize* any $x \in \mathrm{HTR}(T_D, r, I)$ with respect to any $\alpha \in T_D$, meaning that there exists an algorithm which finds a new network $\tilde{x} \in \mathrm{HTR}(T_D, r, I)$ representing the same tensor, $\prod_{\alpha\in T_D} x_\alpha = \prod_{\alpha\in T_D} \tilde{x}_\alpha$, but which has the additional property of being α-orthogonal. This algorithm is based on a related but different concept of orthogonality which is specific to HTR networks.

Definition 3.3.1 (Strongly Orthogonal HTR Network)**.** Let $x \in \mathrm{HTR}(T_D, r, I)$ be an HTR network and $\alpha \in T_D$ a vertex. x is called strongly α-orthogonal if $S_{\beta|\alpha}(x)$ is $\{\uparrow(\beta \mid \alpha)\}$-orthogonal for each $\beta \in T_D \setminus \alpha$.

As suggested by the name, strong α-orthogonality is a stronger form of α-orthogonality.

Theorem 3.3.2. *Let $x \in \mathrm{HTR}(T_D, r, n)$ be an HTR network and $\alpha \in T_D$ a vertex such that x is strongly α-orthogonal. Then, x is α-orthogonal.*

Proof. The environment tensor at α is given by

$$U_\alpha(x) = \prod_{\beta \in \mathrm{neighbour}(\alpha)} S_{\beta|\alpha}(x).$$

Since x is strongly α-orthogonal, we have

$$\overline{S_{\beta|\alpha}(x)} \,\langle\{\uparrow(\beta|\alpha)\}\rangle\, \left(S_{\beta|\alpha}(x)\right) = \mathbb{I}_{\{\beta\}}$$

for each $\beta \in \mathrm{neighbour}(\alpha)$. Therefore,

$$\begin{aligned}
\overline{U_\alpha(x)} \,\langle E_\alpha\rangle\, U_\alpha(x) &= \overline{\left(\prod_{\beta \in \mathrm{neighbour}(\alpha)} S_{\beta|\alpha}(x)\right)} \,\langle E_\alpha\rangle\, \left(\prod_{\beta \in \mathrm{neighbour}(\alpha)} S_{\beta|\alpha}(x)\right) \\
&= \prod_{\beta \in \mathrm{neighbour}(\alpha)} \overline{S_{\beta|\alpha}(x)} \,\langle\{\uparrow(\beta|\alpha)\}\rangle\, S_{\beta|\alpha}(x) \\
&= \prod_{\beta \in \mathrm{neighbour}(\alpha)} \mathbb{I}_{\{\uparrow(\beta|\alpha)\}} = \mathbb{I}_{E_\alpha}
\end{aligned}$$

which by Theorem 2.4.10 implies E_α-orthogonality of $U_\alpha(x)$ and thus α-orthogonality of x. □

We present an algorithm from [6] which establishes strong D-orthogonality. Following the discussion in Section 3.2, this algorithm could be generalized to establish strong α-orthogonality for any $\alpha \in T_D$, but the general algorithm is more difficult to formulate and will not be used in this thesis. Some authors, e.g. [6], call a strongly D-orthogonal HTR network simply *orthogonal*. Following this terminology, we call the algorithm to strongly D-orthogonalize a network *HTR orthogonalization*. Its basic building block is the following vertex-wise operation.

Definition 3.3.3 (Vertex Orthogonalization)**.** Let $x \in \mathrm{TN}(V, E, r, D, I)$ be a tensor network, $e \in E$ an edge and $v, u \in V$ the two vertices such that $e \in E_v \wedge e \in E_u$. *Orthogonalization of x_v with respect to e* is defined as the following operation:

1: $(q, r) := \mathcal{Q}_e(x_v)$
2: $x_v := q$
3: $x_u := r x_u$

Theorem 3.3.4. *Let $x \in \mathrm{TN}(V, E, r, D, I)$ be a tensor network, $e \in E$ an edge and $v, u \in V$ the two vertices such that $e \in E_v \wedge e \in E_u$. Denote by $\tilde{x} \in \mathrm{TN}(V, E, r, D, I)$ the network resulting from orthogonalizing x_v with respect to e. Then, $\prod_{v \in V} x_v = \prod_{v \in V} \tilde{x}_v$.*

Proof. The claim is a direct consequence of $\tilde{x}_v \, \tilde{x}_u = q(r x_u) = x_v \, x_u$. □

The trick to orthogonalize an HTR network is to orthogonalize its vertices in the correct order, which is leaves-to-root.

Definition 3.3.5 (HTR Orthogonalization [6, Algorithm 3])**.** Let $x \in \mathrm{HTR}(T_D, r, I)$ be an HTR network. *Orthogonalization of x* is defined as the following operation:

```
1: RECURSE(D)
2: function RECURSE(α)
3:     for β ∈ child(α) do
4:         RECURSE(β)
5:     end for
6:     Orthogonalize x_α with respect to α
7: end function
```

Let us prove correctness of this algorithm.

Lemma 3.3.6. *Let $x \in \mathrm{HTR}(T_D, r, I)$ be an HTR network and $\alpha \in T_D$ a vertex such that x_α is $\{\alpha\}$-orthogonal and $S_\beta(x)$ is $\{\beta\}$-orthogonal for each $\beta \in \mathrm{child}(\alpha)$. Then, $S_\alpha(x)$ is $\{\alpha\}$-orthogonal.*

Proof.

$$\begin{aligned}\overline{S_\alpha(x)} \, \langle\{\alpha\}\rangle \, S_\alpha(x) &= \overline{x}_{\alpha \langle\{\alpha\}} \left(\prod_{\beta \in \mathrm{child}(\alpha)} \overline{S_\beta(x)} \, \langle\{\beta\}\rangle \, S_\beta(x) \right)_{\{\alpha\}\rangle} x_\alpha \\ &= \overline{x}_\alpha \, \langle\{\alpha\}\rangle \, x_\alpha = \mathbb{I}_{\{\alpha\}}\end{aligned}$$

□

Theorem 3.3.7. *Let $x \in \mathrm{HTR}(T_D, r, I)$ be an HTR network. After running HTR orthogonalization on x, x is strongly D-orthogonal.*

Proof. Apply Lemma 3.3.6 inductively in leaves-to-root direction. □

Theorem 3.3.8. *Let T_D be a standard dimension partition tree, $x \in \mathrm{HTR}(T_D, r, I)$ an HTR network and $d := \#D$, $n := \max_{k \in D} \# I_k$, $r := \max_{e \in E} r_e$. The cost of HTR orthogonalization of x is $\mathcal{O}(dnr^2 + dr^4)$.*

Proof. We have to compute the orthogonalization $(q, r) := \mathcal{Q}_\alpha(x_\alpha)$ and the mode product $r x_{\mathrm{parent}(\alpha)}$ for each vertex $\alpha \in T_D \setminus \{D\}$. At the d leaves, these operations cost $\mathcal{O}(nr^2)$ and $\mathcal{O}(r^4)$, respectively. At $d - 2$ interior vertices, both operations cost $\mathcal{O}(r^4)$.

A slightly more precise result on the cost of HTR orthogonalization is given in [6, Lemma 4.8]. □

One of the reasons why we only need an algorithm to strongly D-orthogonalize an HTR network x is that once x is strongly orthogonal with respect to any vertex $\alpha \in T_D$, we can move this orthogonal centre around using the following theorem.

Theorem 3.3.9. *Let $x \in \mathrm{HTR}(T_D, r, I)$ be an HTR network and $\alpha \in T_D$, $\beta \in \mathrm{neighbour}(\alpha)$ two vertices such that x is strongly α-orthogonal. After orthogonalizing x_α with respect to $\uparrow(\beta \mid \alpha)$, x is strongly β-orthogonal.*

Proof. The proof is very similar to the one of Lemma 3.3.6. We therefore omit it. □

3.4. HTR Expressions

Assume we are given a tensor $x \in \mathbb{K}^{\times_{k\in D} I_k}$ and a dimension partition tree T_D, and we wish to find ranks $r_{T_D} \in \mathbb{N}^{T_D}$ and an HTR network $x_{T_D} \in \mathrm{HTR}(T_D, r, I)$ such that x is the tensor represented by x_{T_D}, i.e. $x = \prod_{\alpha\in T_D} x_\alpha$. Solutions x_{T_D} of this problem are called *HTR expressions* for x.

We present a formalism to easily derive and write down HTR expressions for families of tensors $x_D \in \mathbb{K}^{\times_{k\in D} I_k}$ parametrized by their mode sets D. For ease of exposition, we assume T_D to be a standard dimension partition tree, but the formalism straightforwardly generalizes to more general tree structures.

The essential ingredient is the following definition.

Definition 3.4.1. Let L, R be two disjoint finite sets and $r \in \mathbb{N}$. A *basis family* is a set of tensor families $\{s_D(i_D) \mid i_D \in [r]\} \subset \mathbb{K}^{\times_{k\in D} I_k}$ parametrized by their mode sets D satisfying

$$s_{L\cup R}(i_{L\cup R}) = \sum_{i_L=0}^{r-1} \sum_{i_R=0}^{r-1} c_{L\cup R}\,(i_{L\cup R} \times i_L \times i_R)\, s_L(i_L)\, s_R(i_R) \tag{3.2}$$

for some $c_{L\cup R} \in \mathbb{K}^{[r_{L\cup R}:=r]\times[r_L:=r]\times[r_R:=r]}$ and all $i_{L\cup R} \in [r]$. As implied by the slice-notation, s_D may also be interpreted as a tensor in $\mathbb{K}^{[r_D:=r]\times(\times_{k\in D} I_k)}$ such that the above equation becomes $s_{L\cup R} = c_{L\cup R}\, s_L s_R$. Equations of the form (3.2) are called *splittings*.

Assume $\{s_D(i_D) \mid i_D \in [r]\}$ is a basis family such that $s_D(i_D = 0) = x_D$. We define an HTR network $\hat{x}_{T_D} \in \mathrm{HTR}(T_D, \hat{r}, I)$ with uniform rank $\hat{r}_\alpha = r$ for all $\alpha \in T_D \setminus \{D\}$ by setting

- $\hat{x}_D := c_D(i_D = 0)$ for the root,
- $\hat{x}_\alpha := c_\alpha$ for all interior vertices $\alpha \in \mathrm{interior}(T_D)$, and
- $\hat{x}_{\{k\}} := s_{\{k\}}$ for the leaves $\{k\} \in \mathrm{leaf}(T_D)$.

It is easily verified by induction in leaves-to-root direction that the $\{\alpha\}$-slices $\{S_\alpha(\hat{x})(i_\alpha)\}$ of the subtree tensors $S_\alpha(\hat{x})$ are exactly the basis family $\{s_\alpha(i_\alpha)\}$ for all $\alpha \in T_D \setminus \{D\}$, and that $S_D(\hat{x}) = s_D(i_D = 0)$ at the root. Since $\prod_{\alpha\in T_D} \hat{x}_\alpha = S_D(\hat{x}) = s_D(i_D = 0) = \hat{x}_D$, it follows that $\hat{x}_{T_D}$ is an HTR expression for x_D.

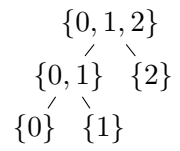
To specify an HTR expression, it is thus sufficient to specify a basis family for x_D. In general, the uniform ranks $\hat{r}_\alpha = r$ are not the smallest possible ranks, however, since not every $s_\alpha(i_\alpha)$ is needed at every vertex α. Once all the splittings of a basis family $S := \{s_D(i_D) \mid i_D \in [r]\}$ have been determined, we will therefore also specify a *bases tree*, which is a map from the vertices $\alpha \in T_D$ to a subset $\{s'_\alpha(i_\alpha) \mid i_\alpha \in [r_\alpha]\} \subseteq S$ of the basis family such that the relation $s'_\alpha = c'_\alpha s'_{\beta_1} s'_{\beta_2}$ is still satisfiable by some tensor $c'_\alpha \in \mathbb{K}^{[r_\alpha]\times[r_{\beta_1}]\times[r_{\beta_2}]}$ for all non-leaf vertices $\alpha \in T_D \setminus \mathrm{leaf}(T_D)$ with children $\mathrm{child}(\alpha) = \{\beta_1, \beta_2\}$. The actual HTR expression may then be derived as explained above.

By Theorems 2.5.5 and 2.5.8, the smallest possible ranks of an HTR expression $x_{T_D} \in \mathrm{HTR}(T_D, r, I)$ for a tensor $x \in \mathbb{K}^{\times_{k\in D} I_k}$ are $r_\alpha = \mathrm{rank}_\alpha(x)$. All bases trees given in this document are optimal in this sense, i.e. they lead to HTR expressions of the aforementioned smallest possible ranks, which may easily be verified using the results from Section 2.5. Since these proofs are fairly technical, we will not carry them out explicitly.

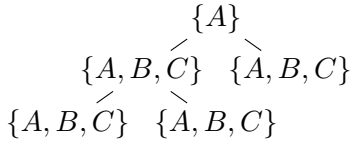
Example 3.4.2. Let $A_D, B_D, C_D \in \mathbb{K}^{\times_{k\in D} I_k}$ be families of tensors parametrized by their mode sets D, and assume they satisfy the splittings

$$A_{L\cup R} = \alpha\, A_L B_R + \beta\, B_L C_R, \quad B_{L\cup R} = B_L B_R \tag{3.3}$$

with $\alpha, \beta \in \mathbb{K} \setminus \{0\}$. Without loss of generality, we assume $\{A_D, B_D, C_D\}$ to be linearly independent for any D. If they were not, we could reformulate (3.3) in terms of only the linearly independent tensors, see also [22, Lemma 3.13]. We would like to find an HTR expression for $A_{\{0,1,2\}}$ based on the dimension partition tree

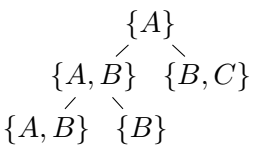


with 0,1,2 arbitrary modes. Clearly, $\{A_D, B_D, C_D\}$ provides a basis family for A_D, thus a possible bases tree would be



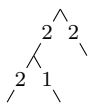
,

where for brevity we have omitted the subscripts of the tensors. This bases tree is not optimal, however, and can be improved as follows. No term in (3.3) involves A_R or C_L, thus we can remove A from the right and C from the left subtree. Similarly, we do not need either A or C at the middle leaf. Summarizing, the optimized bases tree is



.

Since all the involved tensors are linearly independent, it follows from Section 2.5 that this bases tree is optimal in the sense that the resulting ranks



are the smallest possible.

Finally, we mention that the vertex tensor at the root would be given by

$$\begin{array}{c} \\ A \\ B \end{array} \begin{array}{c} \begin{array}{cc} B & C \end{array} \\ \begin{pmatrix} \alpha & 0 \\ 0 & \beta \end{pmatrix} \end{array}$$

where the rows represent the edge mode towards the left child and the columns the edge mode towards the right child.

3.5. Parallelization of HTR Algorithms

To discuss the parallelization of HTR algorithms, it is useful to treat the vertices of a dimension partition tree as independent units and the edges between them as communication channels. Many important HTR algorithms can then be reformulated according to a common pattern.

Definition 3.5.1 (Tree Traversing Algorithm)**.** An HTR algorithm running on a dimension partition tree T_D is called a *tree traversing algorithm* if there exists a partially ordered set S of ordered pairs (α, β) involving neighbouring vertices $\alpha, \beta \in T_D$ such that the algorithm can be formulated as follows.

1: **for** each $(\alpha, \beta) \in S$ **do**
2: On α: Prepare a message m
3: Transfer m to β
4: On β: Consume m
5: **end for**

The for-loop on line 1 traverses through the pairs such that if a pair $p \in S$ is visited before another pair $p' \in S$, then either $p \leq p'$ or p and p' are incomparable in the partial order of S.

The parallelizability of a tree traversing algorithm depends crucially on the set S and the partial order defined thereon. Of particular importance are the following two types of algorithms.

Definition 3.5.2 (Root-to-Leaves Algorithm)**.** A tree traversing algorithm is called *root-to-leaves* if S is the set

$$S := \{(\text{parent}(\alpha), \alpha) \mid \alpha \in T_D \setminus \{D\}\},$$

and the partial order on S is given by

$$(\text{parent}(\alpha), \alpha) \leq (\text{parent}(\beta), \beta) \quad :\Longleftrightarrow \quad \alpha \in \text{ancestor}(\beta) \cup \{\beta\}.$$

Definition 3.5.3 (Leaves-to-Root Algorithm)**.** A tree traversing algorithm is called *leaves-to-root* if S is the set

$$S := \{(\alpha, \text{parent}(\alpha)) \mid \alpha \in T_D \setminus \{D\}\},$$

and the partial order on S is given by

$$(\alpha, \text{parent}(\alpha)) \leq (\beta, \text{parent}(\beta)) \quad :\Longleftrightarrow \quad \alpha \in \text{descendant}(\beta).$$

For illustration, we show how the HTR orthogonalization from Definition 3.3.5 can be formulated as a tree traversing algorithm.

Example 3.5.4. HTR orthogonalization is a leaves-to-root algorithm given by

1: **for** each $(\alpha, \text{parent}(\alpha)) \in S$ **do**
2: On α: Compute $(q, r) = \mathcal{Q}_\alpha(x_\alpha)$, set $x_\alpha := q$
3: Transfer r to parent(α)
4: On parent(α): Set $x_{\text{parent}(\alpha)} := r\, x_{\text{parent}(\alpha)}$
5: **end for**

An algorithm can only be sped up through parallelization if at least some part of its operations can be executed concurrently. The following definition describes the typical setting encountered with HTR algorithms.

Definition 3.5.5 (Tree Parallel Algorithm)**.** An algorithm consisting of one or more root-to-leaves and/or leaves-to-root parts is called *tree parallel* if the loop iterations not ordered by the respective partial order on S can be executed concurrently in each part.

Ideally, an algorithm run on p processors finishes p times faster than the same algorithm run on a single processor. For a tree parallel algorithm, such perfect speedup is not possible, however, because edges lying on the same branch must always be processed consecutively, irrespective of the compute power we invest. To quantify the effect of this serial part on the parallel scaling, we make the following simplifying assumptions.

Assumption 3.5.6. Let `A` be a tree parallel algorithm consisting of a single root-to-leaves/leaves-to-root part running on a balanced standard dimension partition tree T_D. We assume:

- The operations on a vertex $\alpha \in T_D$ can only be run once all incoming messages have been received. These local operations cannot be further parallelized, and the outgoing messages can only be sent once all operations on vertex α have finished.
- It takes `A` one time unit to process an interior vertex $\alpha \in \text{interior}(T_D)$, and no time to process the root D or a leaf $\alpha \in \text{leaf}(T_D)$.

- Transferring messages takes no time.

The first assumption simplifies the model in that it allows to associate all operations with vertices instead of endpoints of edges. The second assumption is derived from the fact that for a standard dimension partition tree, the interior vertex tensors are three-dimensional and therefore typically have many more elements and require much more effort than the root or leaf tensors which are only two-dimensional. Neither of these assumptions are perfectly satisfied in practice, yet they greatly simplify the model analysis and the resulting predictions turn out to be sufficiently accurate for our purposes.

Theorem 3.5.7. *Let* `A` *be an algorithm satisfying Assumption 3.5.6 and set* $d := \#D$ *such that the dimension partition tree* T_D *has* $d-2$ *interior vertices. The optimal runtime of* `A` *on* p *processors is given by*

$$T(p) := \lceil \log_2 p \rceil - 1 + \left\lceil \frac{d - 2^{\lceil \log_2 p \rceil}}{p} \right\rceil = O\left(\log_2 p + \frac{d}{p} \right),$$

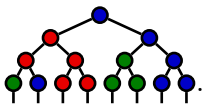
and the optimal parallel speedup is

$$\frac{T(1)}{T(p)} = \frac{d-2}{T(p)} = O\left(\frac{d}{\log_2 p + \frac{d}{p}} \right). \tag{3.4}$$

Proof. [11, Theorem 3]. □

If a tree parallel algorithm consists of more than one root-to-leaves/leaves-to-root part, we assume that the parts must be run one after the other and cannot overlap. Again, this assumption is not necessarily satisfied in practice, but it simplifies the argument and provides a reasonable approximation to reality. The optimal runtime of the algorithm on p processors is then given by $\sum_{i=1}^{n} c_i\, T(p)$, where n denotes the number of such parts and the c_i take into account that each part may require a different unit time per interior vertex. When computing the optimal parallel speedup, the c_i factor out and cancel, therefore formula (3.4) is still valid even in this more general setting.

In our implementation, we assign each vertex of the dimension partition tree to a process and let this process store all the data and execute all the operations associated with the vertex. The resulting map from the vertices to the processes is called a *vertex distribution*, and we depict it graphically in a tree where the color of each vertex denotes the process on which it is placed, e.g.

 . (3.5)

Each process manages a list of ready jobs, i.e. a list of messages which are ready to be prepared or consumed. Once it finishes a job, the process waits until this list becomes non-empty and then chooses the job to work on next according to either of the following rules, depending on the type of the algorithm.

Root-to-leaves: Pick any ready job on one of the vertices $\alpha \in T_D$ which maximize

$$\max\{\text{inv level}(\beta) \mid \beta \in \text{descendant}(\alpha) \wedge \beta \text{ is on a different process than } \alpha\}.$$

Leaves-to-root: Pick any ready job on one of the vertices $\alpha \in T_D$ which maximize

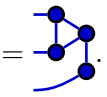
$$\max\{\text{level}(\beta) \mid \beta \in \text{ancestor}(\alpha) \wedge \beta \text{ is on a different process than } \alpha\}.$$

With the vertex distribution from (3.5) and in a root-to-leaves algorithm, the red process thus works first on the left vertex on the penultimate level before descending into the right subtree, and in a leaves-to-root algorithm, the blue process first finishes the single vertex on the left before starting on the right branch.

Consider a leaves-to-root algorithm and assume $\alpha \in T_D \setminus \{D\}$ is a vertex located on process q such that parent(α) is located on a different process q'. We call such a vertex a *local root*. Under Assumption 3.5.6, the algorithm can only finish by time T if q sends the message from α to parent(α) at or before time $T - \text{level}(\alpha)$. Requiring the algorithm to achieve the optimal runtime $T(p)$ from Theorem 3.5.7 thus imposes a number of deadlines by which each process must have finished its local roots, and the above scheduling rule implies that the processes always work towards the local root with the earliest deadline. This *earliest deadline first* (EDF) scheduling algorithm was introduced in [25] and it has been proven in [26] that if any scheduling algorithm meets all deadlines, then EDF is one of them. We therefore conclude that if the vertex distribution is such that the optimal parallel runtime for a leaves-to-root algorithm is attainable, our scheduling algorithm will indeed achieve it. The same statement applies also for root-to-leaves algorithms, and it can be proven with an argument analogous to the above.

4. ALS-Type Algorithms for Linear Systems in HTR

Consider the problem of finding an $x \in \mathbb{K}^{\times_{k\in D} I_k}$ such that $Ax = b$ with $A \in \mathbb{K}^{\times_{k\in D^2} I_k}$ and $b \in \mathbb{K}^{\times_{k\in D} I_k}$. The problem is equivalent to the linear system of equations (LSE) $\mathcal{M}(A)\mathcal{M}_{D,\{\}}(x) = \mathcal{M}_{D,\{\}}(b)$, therefore the "only" new difficulty is that the number of unknowns $n := \prod_{k\in D} \#I_k$ grows exponentially in $\#D$. Already for moderate dimension $\#D$ and mode sizes $\#I_k$, n can become so large that we could not store the solution x even if it were given to us for free. To avoid this *curse of dimensionality*, we represent x as a tensor network x_V which for the moment we assume to have any arbitrary structure, e.g.

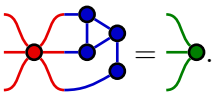
$x =$  .

Of course, if we cannot afford to store the solution x, then the same also applies to the parameters A and b. We postpone defining their representations, however, and assume for the moment that they are given as a black box linear operator and vector

$A =$  , $b =$  ,

respectively.

Substituting the tensor network ansatz for x into $Ax = b$, we get

 (4.1)

Note that this is a nonlinear equation in terms of the x_V, and even though the number of unknowns is greatly reduced, it is still large. Therefore, iterative approaches are needed to solve (4.1).

4.1. The ALS Algorithm

The iterative approach we are aiming for is called the *Alternating Least Squares* (ALS) algorithm [10, 9]. Its key idea is to replace the difficult equation (4.1) with a sequence of much smaller and linear equations which we define next.

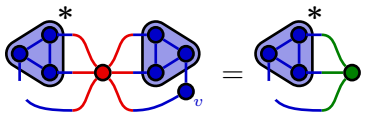
Definition 4.1.1 (Local LSE)**.** Let $x_V \in \mathrm{TN}(V, E, r, D, I)$ be a tensor network and $v \in V$ one of its vertices. Additionally, let $A \in \mathbb{K}^{\times_{k\in D^2} I_k}$ and $b \in \mathbb{K}^{\times_{k\in D} I_k}$ be two

tensors. The problem of determining $x_v \in \mathbb{K}^{(\times_{e\in E_v}[r_e])\times(\times_{k\in D_v} I_k)}$ such that

$$\overline{U_v(x)}_{\langle E_v} A_{E_v\rangle} U_v(x)\, x_v = \overline{U_v(x)}\, b \tag{4.2}$$

is called the *local LSE* at v. As indicated by the notation, the local LSE is solved in-place, i.e. x_v will be updated to the solution of (4.2).

The equivalent formulation of (4.2) in terms of a tensor network diagram is



where the grouped vertices denote $U_v(x)$ and the stars denote complex conjugation of the left-sided $U_v(x)$.

The ALS algorithm consists in repeatedly solving the local LSE, see Algorithm 1. Here, we assume for simplicity that the number of iterations n_{iter} is specified a priori, but adaptive choices are also possible and will be discussed in Section 5.6.

If $\mathcal{M}(A)$ is symmetric and positive definite (spd), the ALS algorithm can be interpreted as an iterative scheme for minimizing the energy norm of the error $\|x-x^\star\|_A^2 = (x, Ax) - 2\,\mathrm{Re}(x, b) + (x^\star, b)$ with $x^\star := A^{-1}b$ the exact solution [9, 10]. In this case, it is also easily seen that $\|x-x^\star\|_A$ decreases monotonically each time we solve the local problem. If A is not spd, however, we no longer have an associated energy norm and it is not clear whether ALS is still an optimization method. In [9], it is proposed to interpret the non-spd ALS algorithm as a projection method on the subspaces $\mathbb{S}_{F_v^c}(U_v(x))$ instead.

Algorithm 1 ALS Algorithm

```
for i = 1, ..., n_iter do
    for v ∈ V do
        Solve the local LSE                    // Equation (4.2)
    end for
end for
```

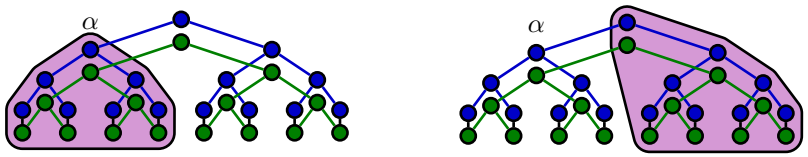
4.2. The HTR ALS Algorithm

The ALS algorithm as presented in the last subsection only requires x to be a tensor network, but it makes no assumptions on either the network structure of x or the representations of A and b. This is because we left out two important aspects of the ALS algorithm:

- The local LSE (4.2) can only be solved numerically if the condition number

$$\kappa\left(\mathcal{M}\left(\overline{U_v(x)}_{\langle E_v} A_{E_v\rangle} U_v(x)\right)\right)$$

  of the local matrix is reasonably small.



(a) $(x|y)(\alpha)$ (b) $(x|y)^c(\alpha)$

Figure 4.1.: Contracted subtrees.

- The ALS algorithm is only computationally feasible if the local matrix and right-hand side can be assembled efficiently.

It turns out that both of these problems can be solved if we require A, x and b to be in HTR.

Regarding the first point, we have the useful result that

$$\kappa\left(\mathcal{M}\left(\overline{U_v(x)}_{\langle E_v}\, A\, {}_{E_v\rangle}U_v(x)\right)\right) \leq \kappa\left(\mathcal{M}\left(A\right)\right)$$

if $U_v(x)$ is E_v-orthogonal [10, Theorem 4.1] or equivalently that x is v-orthogonal. If x is an HTR network, α-orthonormality can be established for each $\alpha \in T_D$ using the HTR orthogonalization procedure (Definition 3.3.5). Furthermore, if we traverse through the network such that consecutive vertices are neighbours, we can use Theorem 3.3.9 to move the orthogonal centre from one vertex to the next. Ignoring the initial orthogonalization, we therefore need to orthogonalize only one vertex per local LSE.

The key to the second point are the *contracted subtrees* defined next. We remark already here that these quantities will be the HTR analogues of the tensors Ψ_k, Φ_k in [9] and G_i, H_i in [10].

Definition 4.2.1 (Contracted Subtree $(x|y)(\alpha)$)**.** Let

$$x \in \mathrm{HTR}(T_D, r^{(x)}, n), \quad y \in \mathrm{HTR}(T_D, r^{(y)}, n)$$

be two HTR networks and $\alpha \in T_D$ a vertex. The *contracted subtree*

$$(x|y)(\alpha) \in \mathbb{K}^{[r^{(x)}_{x(\alpha)}]\times[r^{(y)}_{y(\alpha)}]}$$

is defined as

$$(x|y)(\alpha) := \overline{S_\alpha(x)}\, S_\alpha(y) = \prod_{\beta\in\mathrm{descendant}(\alpha)} \overline{x_\beta}\; y_\beta.$$

Remark 4.2.2. The contracted subtrees $(x|y)(\alpha)$ can be computed recursively in leaves-to-root order as follows:

$$(x|y)(\alpha) := \overline{x_\alpha}\; y_\alpha \left(\prod_{\beta\in\mathrm{child}(\alpha)} (x|y)(\beta)\right). \tag{4.3}$$

Definition 4.2.3 (Contracted Subtree $(x|y)^c(\alpha)$)**.** Let

$$x \in \mathrm{HTR}(T_D, r^{(x)}, n), \quad y \in \mathrm{HTR}(T_D, r^{(y)}, n)$$

be two HTR networks and $\alpha \in T_D \setminus \{D\}$ a vertex. The *contracted subtree*

$$(x|y)^c(\alpha) \in \mathbb{K}^{[r^{(x)}_{x(\alpha)}]\times[r^{(y)}_{y(\alpha)}]}$$

is defined as

$$(x|y)^c(\alpha) := \overline{S_{\mathrm{parent}(\alpha)|\alpha}(x)}\, S_{\mathrm{parent}(\alpha)|\alpha}(y) = \prod_{\beta\in\mathrm{descendant}(\mathrm{parent}(\alpha)|\alpha)} \overline{x_\beta}\; y_\beta.$$

Remark 4.2.4. The contracted subtrees $(x|y)^c(\alpha)$ can be computed recursively in root-to-leaves order as follows:

$$(x|y)^c(\alpha) := \overline{x_{\mathrm{parent}(\alpha)}}\, y_{\mathrm{parent}(\alpha)}\, (x|y)^c(\mathrm{parent}(\alpha)) \left(\prod_{\beta\in\mathrm{sibling}(\alpha)} (x|y)(\beta) \right). \tag{4.4}$$

Definition 4.2.5 (Contracted Subtree $(x|A|y)(\alpha)$)**.** Let

$$x \in \mathrm{HTR}(T_D, r^{(x)}, n), \quad A \in \mathrm{HTR}(T_D^2, r^{(A)}, n), \quad y \in \mathrm{HTR}(T_D, r^{(y)}, n)$$

be HTR networks and $\alpha \in T_D$ a vertex. The *contracted subtree*

$$(x|A|y)(\alpha) \in \mathbb{K}^{[r^{(x)}_{x(\alpha)}]\times[r^{(A)}_{A(\alpha)}]\times[r^{(y)}_{y(\alpha)}]}$$

is defined as

$$(x|A|y)(\alpha) := \overline{S_\alpha(x)}\, S_\alpha(A)\, S_\alpha(y) = \prod_{\beta\in\mathrm{descendant}(\alpha)} \overline{x_\beta}\, A_\beta\, y_\beta.$$

Definition 4.2.6 (Contracted Subtree $(x|A|y)^c(\alpha)$)**.** Let

$$x \in \mathrm{HTR}(T_D, r^{(x)}, n), \quad A \in \mathrm{HTR}(T_D^2, r^{(A)}, n), \quad y \in \mathrm{HTR}(T_D, r^{(y)}, n)$$

be HTR networks and $\alpha \in T_D \setminus \{D\}$ a vertex. The *contracted subtree*

$$(x|A|y)^c(\alpha) \in \mathbb{K}^{[r^{(x)}_{x(\alpha)}]\times[r^{(A)}_{A(\alpha)}]\times[r^{(y)}_{y(\alpha)}]}$$

is defined as

$$(x|A|y)^c(\alpha) := \overline{S_{\mathrm{parent}(\alpha)|\alpha}(x)}\, S_{\mathrm{parent}(\alpha)|\alpha}(A)\, S_{\mathrm{parent}(\alpha)|\alpha}(y) = \prod_{\beta\in\mathrm{descendant}(\mathrm{parent}(\alpha)|\alpha)} \overline{x_\beta} A_\beta y_\beta$$

Remark 4.2.7. The contracted subtrees $(x|A|y)(\alpha)$ and $(x|A|y)^c(\alpha)$ can be computed recursively using recursion formulas very similar to (4.3) and (4.4).

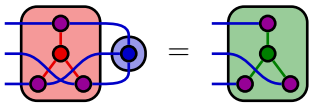


Figure 4.2.: The local LSE if all involved tensors are given in HTR. The vertices denote A_α, x_α, b_α, $(x, A, x)^c(\alpha)$, $(x, A, x)(\beta)$ for each $\beta \in \mathrm{child}(\alpha)$, and the analogous contracted subtrees for (x, b), respectively. The grouped vertices denote the local matrix and right-hand side, respectively.

Using the contracted subtrees, the local matrix and the local right-hand side can be expressed as

$$\begin{aligned} \overline{U_v(x)}_{\langle E_v} A_{E_v\rangle} U_v(x) &:= A_\alpha \ (x|A|x)^c(\alpha) \prod_{\beta \in \mathrm{child}(\alpha)} (x|A|x)(\beta), \\ \overline{U_v(x)}\, b &:= b_\alpha \ (x|b)^c(\alpha) \prod_{\beta \in \mathrm{child}(\alpha)} (x|b)(\beta), \end{aligned} \tag{4.5}$$

see Figure 4.2. Thus, if the involved contracted subtrees are available, we can assemble the local LSE using only one tensor per neighbouring vertex. The trick to make the HTR ALS algorithm computationally feasible is to precompute and cache these contracted subtrees, and to update them on the fly whenever the constituting vertex tensors change. This procedure is outlined in Algorithm 2.

Algorithm 2 HTR ALS Algorithm

```
Orthogonalize x                                              // Definition 3.3.5
Compute (x|A|x)(α) and (x|b)(α) for all α ∈ T_D \ D          // Remarks 4.2.2 and 4.2.7
for i = 1, ..., n_iter do
    RECURSE(D)
end for
function RECURSE(α)
    for β ∈ child(α) do
        Solve the local LSE                                  // Equations (4.2) and (4.5)
        Orthogonalize x_α with respect to β                  // Definition 3.3.3
        Compute (x|A|x)^c(β) and (x|b)^c(β)                  // Remarks 4.2.4 and 4.2.7
        RECURSE(β)
    end for
    Solve the local LSE                                      // Equations (4.2) and (4.5)
    if α ≠ D then
        Orthogonalize x_α with respect to α                  // Definition 3.3.3
        Compute (x|A|x)(α) and (x|b)(α)                      // Remarks 4.2.2 and 4.2.7
    end if
end function
```

4.3. Cost of the HTR ALS Algorithm

In this subsection, we assume T_D to be a standard dimension partition tree and $x \in \mathrm{HTR}(T_D, r^{(x)}, [n])$, $A \in \mathrm{HTR}(T_D^2, r^{(A)}, [n])$, $b \in \mathrm{HTR}(T_D, r^{(b)}, [n])$ with $n_D \in \mathbb{N}^D$ and $[n] := ([n_k])_{k \in D}$. We define $d := \#D$, $n := \max_{k \in D} n_k$, $r := \max_{\alpha \in T_D \setminus \{D\}} r_\alpha^{(x)}$, $R := \max_{\alpha \in T_D \setminus \{D\}} r_\alpha^{(A)}$ and $R_b := \max_{\alpha \in T_D \setminus \{D\}} r_\alpha^{(b)}$.

The ranks of x are usually larger than the ranks of A or b. In the following, we will therefore assume $R^a r^b < R^{a'} r^{b'}$ if $a+b = a'+b'$ and $b < b'$, and the analogous inequality for R_b. At the leaves, we redefine the rank quantity r to refer to the maximum rank of x at the leaves, which guarantees that in this case $r < n$. Since in the below terms involving both n as well as r the rank symbol r refers to the ranks at the leaves, we can again order such terms according to

$$n^a r^b < n^{a'} r^{b'} \quad \text{if } a + b = a' + b' \wedge a < a'.$$

The HTR ALS algorithm consists of three types of operations: orthogonalization, contracted subtree computation and solution of local LSE. In the following, we analyze the computational cost per iteration for each type. The initial orthogonalization and contracted subtree computation (lines 1 and 2) scale like the corresponding operations in the iteration. The costs at the root will always be negligible, therefore we do not discuss this special case.

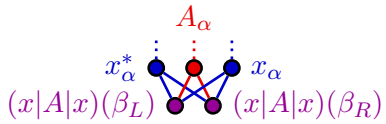
The cost of the ALS algorithm in the TT case has already been analyzed in [9] and [10], and the discussion below largely follows the expositions given there.

4.3.1. Orthogonalization

Orthogonalization requires some constant number of tensor orthogonalizations and mode multiplications per vertex, both of which cost $\mathcal{O}(nr^2)$ for leaves and $\mathcal{O}(r^4)$ for interior vertices. We thus spend $\mathcal{O}(dr^4 + dnr^2)$ floating-point operations on orthogonalization. See also Theorem 3.3.8 and [6, Lemma 4.8].

4.3.2. Contracted Subtrees

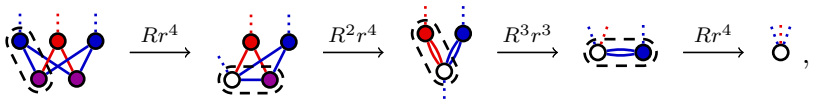
The recursive computation of $(x|A|x)(\alpha)$ with $\alpha \in \mathrm{interior}(T_D)$ and $\mathrm{child}(\alpha) = \{\beta_L, \beta_R\}$ requires evaluating



. (4.6)

The computational cost of this step depends on the order in which the mode multiplications are carried out, which is called a *contraction sequence*. We propose the contraction

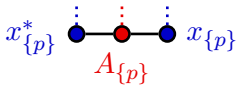
sequence[1]



which costs $\mathcal{O}(R^2r^4)$ operations.

If we replace $(x|A|x)(\beta_L)$ with $(x|A|x)^c(\alpha)$ in (4.6), we obtain the formula for computing $(x|A|x)^c(\beta_L)$. Since both $(x|A|x)(\alpha)$ and $(x|A|x)^c(\alpha)$ are elements of $\mathbb{K}^{[r]\times[R]\times[r]}$, we conclude that the recursion step for $(x|A|x)^c(\alpha)$ can be executed with the same cost.

At the leaf vertices, the formula for computing $(x|A|x)(\alpha)$ is much simpler:



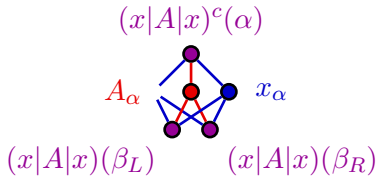
This contraction takes $\mathcal{O}(n^2Rr)$ floating-point operations. In total, computing the contracted subtrees $(x|A|x)(\alpha)$ and $(x|A|x)^c(\alpha)$ thus costs $\mathcal{O}\left(dR^2r^4 + dn^2Rr\right)$.

The costs for computing $(x|b)(\alpha)$ and $(x|b)^c(\alpha)$ are obtained similarly. We only state the final result, which is $\mathcal{O}\left(dR_br^3 + dnR_br\right)$.

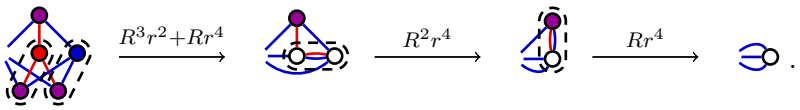
4.3.3. Local LSE

The number of unknowns in the local LSE at an interior vertex is r^3, thus solving the local LSE using Gaussian elimination has a cost of $\mathcal{O}(r^9)$. With r ranging up to $\mathcal{O}(100)$, such an algorithm is not feasible.

If we can start the ALS iteration with a good initial guess x (e.g. once the algorithm has almost converged), we may expect that x_α is already a good initial guess for the local LSE. The vertex tensor x_α then only needs a small correction which can be obtained cheaply through a few steps of an iterative method like the conjugate gradient or GMRES algorithm [28]. The performance bottleneck of these algorithms is the matrix-vector product which takes $O(r^6)$ floating-point operations for a generic LSE in $O(r^3)$ unknowns. However, at an interior vertex the matrix-vector product has the form



see also Figure 4.2. This allows to reduce its cost to $\mathcal{O}(R^2r^4)$ by evaluating it in the following order:



[1]The Matlab script from [27] was of great use to determine such contraction sequences.

The cost of the matrix-vector product at the leaves is $\mathcal{O}(n^2Rr)$. The right hand side can be computed in $\mathcal{O}(R_b r^3)$ (interior vertex) and $\mathcal{O}(nR_b r)$ (leaf), respectively. The total cost of solving the local LSE is thus

$$\mathcal{O}\left(d\,\rho\left(R^2r^4 + n^2Rr\right) + d\left(R_b r^3 + nR_b r\right)\right),$$

where ρ denotes the number of steps required by the iterative local solver.

4.3.4. Summary

Collecting all the results, we find the overall complexity of a single ALS iteration to be

$$\mathcal{O}\left(d\,(\rho+1)\left(R^2r^4 + n^2Rr\right) + d\left(R_b r^3 + nR_b r\right)\right).$$

In practice, the terms involving ρ describing the effort spent on solving the local LSEs dominate. We therefore compute precisely how many such LSEs we solve.

Theorem 4.3.1. *Let T_D be a standard dimension partition tree, $d := \#D$ and $x \in \mathrm{HTR}(T_D, r, I)$. The HTR ALS Algorithm 2 solves $4\,(d-1)$ local LSEs per iteration.*

Proof. We count for $\alpha \in \mathrm{leaf}(T_D)$, $\alpha \in \mathrm{interior}(\alpha)$ and $\alpha = D$ how many such α exist and how often we solve a local LSE on vertex α.

	Number of vertices	# local LSE solves per vertex
leaves	d	1
interior	$d-2$	3
root	1	2

□

4.4. The Parallel HTR ALS Algorithm

An important feature of the local LSE (4.2) is that its matrix and right-hand side depend on all vertex tensors of x. We therefore must not modify x while one local solve process is running, and in particular we cannot solve several local LSEs concurrently. In Algorithm 2, this is expressed by the fact that we must run the loop over the children (line 7) sequentially, because the computations for the local LSE and the contracted subtrees on lines 8 and 10 depend on the contracted subtrees computed on line 16 in the subordinate calls to RECURSE.

If we insist on parallelizing the ALS algorithm, we must get rid of this dependency between loop iterations. Since we have cached values for all $(x|A|x)(\beta)$, $(x|b)(\beta)$, $\beta \in \mathrm{child}(\alpha)$, we may use those instead of waiting for the updated values. Put differently, we may compute all contracted subtrees $(x|A|x)^c(\beta)$ and $(x|b)^c(\beta)$ before we recurse into any subtree. When solving local LSEs or recursively computing further contracted subtrees in the calls to RECURSE(β), we then use the precomputed values for $(x|A|x)^c(\alpha)$ and $(x|b)^c(\alpha)$. This implies that once the first child tensor x_{β_1} has been updated, the contracted subtrees actually used in the local LSE at later children $\beta_i \in \mathrm{child}(\alpha) \setminus \{\beta_1\}$

or any of their descendants will differ from the ones we would obtain if we computed them anew. The parallel algorithm thus produces a different result than the serial one. Numerical evidence indicates that their convergence properties are very similar, however, see Section 5.4.

A new technical difficulty introduced in the parallel ALS algorithm is that we have to ensure orthonormality of all environment matrices $U_\beta(x)$, $\beta \in \text{child}(\alpha)$, at the same time. Sequentially orthogonalizing x_α with respect to all $\beta \in \text{child}(\alpha)$ as we did in Algorithm 2 cannot achieve this, because after orthogonalizing x_α with respect to the first child β_1, x is not α-orthogonal anymore. Therefore, orthogonalizing x_α with respect to some other $\beta_2 \in \text{child}(\alpha)$ will not ensure orthonormality of $U_{\beta_2}(x)$. Instead, we propose the following algorithm.

Definition 4.4.1 (Simultaneous Vertex Orthogonalization)**.** Let $x \in \text{HTR}(T_D, r, I)$ be an HTR network and $\alpha \in T_D$ a vertex. *Orthogonalization of x_α with respect to its children* is defined as the following operation:

1: **for** $\beta \in \text{child}(\alpha)$ **do**
2: $\quad (u_\beta, s_\beta, v_\beta) := \mathcal{S}_\beta(x_\alpha)$
3: $\quad x_\alpha := u_\beta \, s_\beta$
4: $\quad x_\beta := v_\beta \, x_\beta$
5: **end for**
6: **for** $\beta \in \text{child}(\alpha)$ **do**
7: $\quad x_\alpha := x_\alpha \, s_\beta^{-1}$
8: $\quad x_\beta := s_\beta \, x_\beta$
9: **end for**

Simultaneous vertex orthogonalization has similar properties as the edge-wise vertex orthogonalization we have introduced in Definition 3.3.3.

Theorem 4.4.2. *Let $x \in \text{HTR}(T_D, r, I)$ be an HTR network and $\alpha \in T_D$ a vertex. Denote by $\tilde{x} \in \text{HTR}(T_D, r, I)$ the network resulting from orthogonalizing x_α with respect to its children. Then, $\prod_{\alpha \in T_D} x_\alpha = \prod_{\alpha \in T_D} \tilde{x}_\alpha$.*

Proof. We prove that the tensor x represented by x_{T_D} is not changed by the line pairs 3,4 and 7,8, respectively. In the first case, the claim follows from $u_\beta s_\beta v_\beta \, x_\beta = x_\alpha x_\beta$, in the second it follows from $x_\alpha s_\beta^{-1} s_\beta x_\beta = x_\alpha x_\beta$. □

Theorem 4.4.3. *Let $x \in \text{HTR}(T_D, r, I)$ be an HTR network and $\alpha \in T_D$ a vertex such that $S_{\beta|\alpha}(x)$ is $\{\uparrow(\beta \mid \alpha)\}$-orthogonal for each $\beta \in \text{neighbour}(\alpha)$. After orthogonalizing x_α with respect to its children, $S_{\alpha|\beta}(x)$ is $\{\beta\}$-orthogonal for all $\beta \in \text{child}(\alpha)$.*

Proof. While the statement holds for any $\alpha \in T_D$, we only prove it for $\alpha \in T_D \setminus \{D\}$ in order to avoid conditioning on the existence of parent(α). The modifications needed for $\alpha = D$ will be obvious.

We denote by x the original network, by x' the network that is obtained after the first loop (lines 1 to 5) has been executed and by x'' the final network. We define

$\beta_1, \dots, \beta_c \in \text{child}(\alpha)$, $c := \#\,\text{child}(\alpha)$, to be the children of α in the order in which they appear in the first loop, and denote by $x^{(i)}$, $i = 1, \dots, c$ the state of the network after the iteration $\beta = \beta_i$ of the first loop has been executed.

The proof splits into two parts.

1. We have
$$S_{\beta_i}(x^{(j)}) = \begin{cases} S_{\beta_i}(x) & \text{if } j < i \\ v_{\beta_i} S_{\beta_i}(x) & \text{otherwise} \end{cases}$$
for $i, j = 1, \dots, c$. Since v_{β_i} and $S_{\beta_i}(x)$ are $\{R(\beta_i)\}$- and β_i-orthogonal, respectively, $S_{\beta_i}(x^{(j)})$ is $\{\beta_i\}$-orthogonal for any $i, j = 1, \dots, c$.

2. Let $\gamma \in \text{child}(\alpha)$. From the proof of Theorem 4.4.2 it follows that $S_{\alpha|\gamma}(x)$ is not modified by lines 3,4 unless γ is equal to the loop variable β. This proves
$$S_{\alpha|\beta_i}(x') = u_{\beta_i} s_{\beta_i} S_{\text{parent}(\alpha)}(x) \prod_{\gamma \in \text{sibling}(\beta_i)} S_\gamma(x^{(i)}) \tag{4.7}$$
for all $i = 1, \dots, c$. Arguing similarly for the second loop, we obtain
$$S_{\alpha|\beta_i}(x'') = u_{\beta_i} S_{\text{parent}(\alpha)}(x) \prod_{\gamma \in \text{sibling}(\beta_i)} S_\gamma(x^{(i)}). \tag{4.8}$$
u_{β_i} is β_i-orthogonal, $S_{\text{parent}(\alpha)}(x)$ is $\{\alpha\}$-orthogonal and the $S_\gamma(x^{(i)})$ are $\{\gamma\}$-orthogonal by part 1. Therefore, $S_{\alpha|\beta_i}(x'')$ is $\{\beta_i\}$-orthogonal. □

So far, we have silently assumed that s_β is invertible, i.e. that no singular value is exactly 0. In our code, we ensure this condition by transforming the singular values $(s_\beta)_{i_\beta}$, $i_\beta \in [r_\beta]$ with
$$(s_\beta)_{i_\beta} := \max\{(s_\beta)_{i_\beta}, \varepsilon\,(s_\beta)_0\} \tag{4.9}$$
where ε denotes the machine precision and $(s_\beta)_0$ the largest singular value. We now analyse how this rounding effects the above results.

Theorem 4.4.2 relies on the identities $u_\beta s_\beta v_\beta = x_\alpha$ and $s_\beta\, s_\beta^{-1} = \mathbb{I}_{\{\beta\}}$. With the singular values from (4.9), both identities are satisfied up to machine precision, thus also the statement in Theorem 4.4.2 is valid up to machine precision.

In Theorem 4.4.3, the rounding in the singular values and numerical noise implies that (4.7) is satisfied up to a relative error of $\mathcal{O}(\varepsilon)$. Multiplication with s_β^{-1} may then blow this error up such that the relative error in (4.8) is $\mathcal{O}(1)$. Therefore, in finite machine precision the $S_{\alpha|\beta}(x)$ may not be $\{\beta\}$-orthogonal at all.

Luckily, we can limit the impact of the rounding errors by interleaving orthogonalization and subtree computation as follows.

Definition 4.4.4 (Combined Orthogonalization and Contracted Subtree Computation)**.** Let $x \in \text{HTR}(T_D, r, I)$ be an HTR network and $\alpha \in T_D$ a vertex. *Combined orthogonalization and contracted subtree computation at* x_α is defined as the following operation:

1: **for** $\beta \in \text{child}(\alpha)$ **do**

2: // *Orthogonalize* x_α *with respect to* β
3: $(u_\beta, s_\beta, v_\beta) := \mathcal{S}_\beta(x_\alpha)$
4: $x_\alpha := u_\beta$
5: $x_\beta := s_\beta \, v_\beta \, x_\beta$
6: // *Compute subtrees*
7: Compute $(x|A|x)^c(\beta)$ and $(x|b)^c(\beta)$ // *Remarks 4.2.4 and 4.2.7*
8: // *Temporarily move the non-orthogonal factor to* x_α
9: $x_\alpha := x_\alpha \, s_\beta$
10: $x_\beta := s_\beta^{-1} \, x_\beta$ // *(*)*
11: // *Update* $(x|A|x)(\beta)$ *and* $(x|b)(\beta)$
12: $(x|A|x)(\beta) := \overline{v_\beta} \, (x|A|x)(\beta) \, v_\beta^T$
13: $(x|b)(\beta) := \overline{v_\beta} \, (x|b)(\beta)$
14: **end for**
15: **for** $\beta \in \mathrm{child}(\alpha)$ **do**
16: // *Move the non-orthogonal factor to* x_β *again*
17: $x_\alpha := x_\alpha \, s_\beta^{-1}$
18: $x_\beta := s_\beta \, x_\beta$ // *(*)*
19: // *Update* $(x|A|x)(\beta)$ *and* $(x|b)(\beta)$
20: $(x|A|x)(\beta) := s_\beta \, (x|A|x)(\beta) \, s_\beta$ // *(+)*
21: $(x|b)(\beta) := s_\beta \, (x|b)(\beta)$ // *(+)*
22: **end for**

The two lines marked with *(*)* may be omitted since the second undoes the effect of the first. The two lines marked with *(+)* may be dropped if $(x|A|x)(\beta)$ and $(x|b)(\beta)$ are updated anyway before being used again, as is the case in Algorithm 3.

It is easily verified that the modifications applied to x_{T_D} in Definition 4.4.4 are equivalent to the ones in Definition 4.4.1, therefore Theorems 4.4.2 and 4.4.3 are also valid for combined orthogonalization and subtree computation. The key point in Definition 4.4.4 is that we compute the contracted subtrees (line 7) at a point where $S_{\alpha|\beta}(x)$ is $\{\beta\}$-orthogonal up to errors of order $\mathcal{O}(\varepsilon)$. Since the local LSE are assembled based on these cached $(x|A|x)^c(\beta)$, $(x|b)^c(\beta)$ capturing accurately $\{\beta\}$-orthogonal $S_{\alpha|\beta}(x)$, it no longer matters that the final $S_{\alpha|\beta}(x)$ are not accurately $\{\beta\}$-orthogonal.

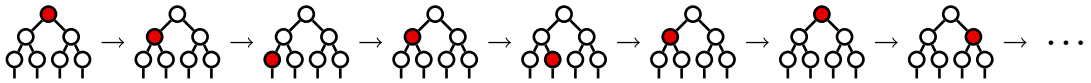
We remark that the updates to the contracted subtrees on lines 12, 13 and 20, 21 are a consequence of the modifications done to x_β on lines 5, 10 and 18. In the serial Algorithm 2, such updates are not necessary because $(x|A|x)^c(\beta)$, $(x|b)^c(\beta)$ are updated anyway on line 16 before being used again.

Compared to the orthogonalization and subtree computation in the serial ALS Algorithm 2, we replace the QR-based orthogonalization with SVD-based orthogonalization and do some additional mode products in Definition 4.4.4. One verifies that neither of these changes increases the asymptotic computational costs, therefore the asymptotic complexity results from Section 4.3 are valid also for the parallel HTR ALS Algorithm 3. However, the prefactors change. For the orthogonalization and subtree computation, they increase since the SVD is more costly than the QR decomposition and we need more mode products than in the serial case. On the other hand, the number of solved

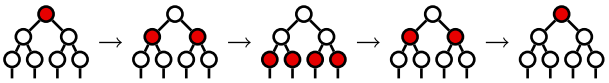
Algorithm 3 Parallel HTR ALS Algorithm

We assume an implicit cache of contracted subtrees. This cache is initialized on line 2 and only updated when we explicitly say so, namely on lines 8 and 15. Each time contracted subtrees are needed (i.e. when computing new contracted subtrees and when solving the local LSE), their values are read from the cache.

1: Orthogonalize x // *Definition 3.3.5*
2: Compute $(x|A|x)(\alpha)$ and $(x|b)(\alpha)$ for all $\alpha \in T_D \backslash \{D\}$ // *Remarks 4.2.2 and 4.2.7*
3: **for** $i = 1, \ldots, n_{iter}$ **do**
4: RECURSE(D)
5: **end for**
6: **function** RECURSE(α)
7: Solve the local LSE // *Equations* (4.2) *and* (4.5)
8: Orthogonalize and compute contracted subtrees at x_α // *Definition 4.4.4*
9: **parallel for** $\beta \in \text{child}(\alpha)$
10: RECURSE(β)
11: **end parallel for**
12: Solve the local LSE // *Equations* (4.2) *and* (4.5)
13: **if** $\alpha \neq D$ **then**
14: Orthogonalize x_α with respect to α // *Definition 3.3.3*
15: Compute $(x|A|x)(\alpha)$ and $(x|b)(\alpha)$ // *Remarks 4.2.2 and 4.2.7*
16: **end if**
17: **end function**



(a) Serial



(b) Parallel

Figure 4.3.: Tree traversal orders of the serial and parallel HTR ALS algorithm. The red vertices denote the ones on which we currently solve local LSEs.

local LSE decreases as shown next.

Theorem 4.4.5. *Let T_D be a standard dimension partition tree, $d := \#D$ and $x \in \mathrm{HTR}(T_D, r, I)$. The parallel HTR ALS Algorithm 3 solves $3d-2$ local LSEs per iteration.*

Proof. We count for $\alpha \in \mathrm{leaf}(T_D)$, $\alpha \in \mathrm{interior}(\alpha)$ and $\alpha = D$ how many such α exist and how often we solve a local LSE on vertex α.

	Number of vertices	# local LSE solves per vertex
leaves	d	1
interior	$d-2$	2
root	1	2

□

The parallelization and the orthogonalization strategies outlined above are based on similar propositions for the TT DMRG algorithm in [15]. The parallelized algorithm developed there requires the initial orthogonalization and computation of contracted subtrees to be carried out on at most two processors (one for each end of the TT chain, assuming we pick the central vertex as root), and then the parallelizability gradually increases during the first DMRG sweep. The serial workload thus scales linearly with the number of dimensions $d := \#D$, which is again proportional to the overall workload. By Amdahl's law [29], this limits the parallel scaling of the TT algorithm to some constant irrespective of d. In contrast, our algorithm falls into the category of tree parallel algorithms and therefore reduces the serial part to being proportional to the depth of the tree. Assuming a balanced standard dimension partition tree, the parallel scaling is thus bounded by only $\mathcal{O}\left(\frac{d}{\log_2 d}\right)$ which is optimal up to the logarithmic factor.

4.5. The HTR ALS(SD) Algorithm

The major drawback of the ALS algorithm is that it cannot adapt the ranks of x. Thus, if the initial ranks are too small, the approximations produced by the ALS algorithm will never approximate the exact solution well because no such approximation exists with

the given ranks. On the other hand, if the initial ranks are too large, the iterations are unnecessarily costly and compute power is wasted.

Decreasing the ranks can be done using the HTR truncation procedure introduced in [6]. It has recently been proposed in [21] to increase the ranks by adding some approximation $z \approx r := b - Ax$ of the current residual to x. If we assume $\mathcal{M}(A)$ to be spd and treat the original LSE as an optimization problem, this corresponds to a steepest descent (SD) step, and the resulting Algorithm 4 is therefore called the ALS(SD) algorithm. We call it serial if the algorithm used on line 4 is the serial Algorithm 2, and parallel if we use Algorithm 3.

We settle with only an approximation z of the exact residual r because the ranks of r are in general only bounded by $R_r := R_b + R\,r$ [22, §13.1.4 & §13.5.3], which is too large to be handled efficiently (see Section 4.3 regarding the notation). The approximation z can be obtained in two ways:

1. We form r explicitly and apply the HTR truncation procedure to it.
2. We employ the fixed rank ALS procedure to approximately solve the trivial LSE $\mathbb{I}z = b - Ax$, starting from a random tensor in the first and the previous z in all subsequent iterations.

The latter method has the advantage that the residual need not be formed explicitly (only the contracted subtrees $(z|b)$ and $(z|A|x)$ are needed) and thus tensors of large ranks can be avoided. Under the assumptions from Section 4.3, this reduces the computational cost from $\mathcal{O}\left(dR_r^4 + dnR_r^2\right)$ for the truncation based residual approximation to

$$\mathcal{O}\left(d\left(RR_z r^3 + R^2 R_z^2 r^2 + n^2 R R_z + R_b R_z^3 + n R_b R_z\right)\right)$$

for the ALS based method where R_z denotes the rank of z which is chosen by the user and for which we assume $R_z < r$.

In the numerical experiments reported in [21] as well as in our own experiments, we find that the somewhat cruder approximation of the residual delivered by the ALS based method has no negative impact on the overall convergence of the ALS(SD) algorithm. In the experiments reported below, we therefore always use the asymptotically faster ALS based residual approximation, and the ALS algorithm employed therein is always the parallel one (Algorithm 3) even when the actual ALS step on line 4 is performed using the serial algorithm. The truncation step on line 5 is performed by choosing the ranks adaptively such that the truncated network $\tilde{x}$ satisfies $\|\tilde{x} - x\| \leq \varepsilon\|x\|$ with x the network before truncation and $\varepsilon \in \mathbb{R}_{>0}$ a user-specified parameter.

The parallel ALS algorithm and the truncation are both tree parallel, and the addition is *perfectly parallel*, i.e. it can process all vertices simultaneously when given enough compute power. Since a perfectly parallel algorithm is in particular tree parallel, the parallel ALS(SD) is tree parallel as well.

Algorithm 4 ALS(SD) Algorithm

1: **for** $i = 1, \ldots, n_{iter}$ **do**
2: Compute residual approximation $z \approx b - Ax$
3: Update $x := x + z$
4: Run a single ALS iteration (Algorithm 2 or 3)
5: Truncate x
6: **end for**

5. Case Study: The Poisson Equation

We apply the algorithms from Chapter 4 to the well-known linear system of equations arising from the finite difference discretization of the Poisson equation. This allows to compare the performance of the newly proposed parallel algorithms with respect to their serial counterparts, and investigate the parallel scaling.

5.1. Problem Statement

We consider the Poisson equation on the d-dimensional hypercube $[0,1]^d$ with homogeneous Dirichlet boundary conditions,

$$\begin{aligned} -\Delta u &= f \quad \text{on } \Omega := [0,1]^{[d]}, \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned} \tag{5.1}$$

Following the usual finite difference scheme, we replace continuous functions $f : [0,1]^{[d]} \to \mathbb{R}$ with tensors $g \in \mathbb{R}^{\times_{k\in[d]}[n_k]}$ such that

$$g\left(i_{[d]}\right) := f\left(\left(\frac{i_k}{n_k+1}\right)_{k\in[d]}\right),$$

and we approximate the continuous Laplacian with the following operator.

Definition 5.1.1 (Discrete Laplace Operator)**.** Let D be some finite set and $n_D \in \mathbb{N}^D$. We define the *discrete Laplace operator* $\Delta_D \in \mathbb{R}^{\times_{k\in D^2}[n_k]}$ through its action on a tensor $f \in \mathbb{R}^{\times_{k\in D}[n_k]}$ which is given by

$$\begin{aligned} &\left(\Delta_D f\right)\left(i_D\right) = \ldots \\ &\quad = \sum_{k\in D} (n_k+1)^2 \left(-f\left(i_{D\setminus\{k\}} \times (i_k-1)\right) + f\left(i_D\right) - f\left(i_{D\setminus\{k\}} \times (i_k+1)\right)\right). \end{aligned} \tag{5.2}$$

In (5.2) as well as in the remainder of this chapter, we assume the convention that tensors are padded with zeros outside their actual domain of definition, see Definition A.0.1.

The finite difference approach thus reduces the partial differential equation (5.1) to the *discrete Poisson equation* $\Delta_{[d]}\, u = f$ with $u, f \in \mathbb{R}^{\times_{k\in[d]}[n_k]}$. In the examples considered below, we choose the dimension $d = 16$ and use $n_k = 64$ spatial grid points in all directions $k \in [d]$. We quantize each mode k into $l_k = 6$ virtual modes with mode sizes $n_{(k,j)} = 2$ for all $j \in [l_k]$, and we use the dimension partition tree obtained by creating a balanced binary tree for all physical dimensions $[d]$, and then replacing each leaf $\{k\}$ of

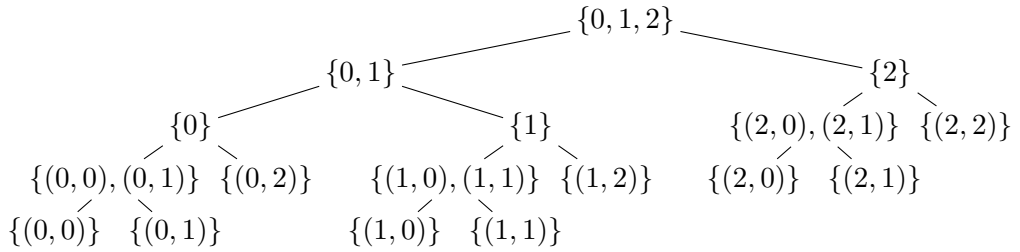


Figure 5.1.: Dimension partition tree for $d = 3$ and $l_k = 3$, $k \in [3]$.

this tree with a balanced binary tree for the virtual modes $(k, [l_k])$. See Figure 5.1 for an example of such a tree with smaller d and l_k. f will always be the all-ones tensor $1_{[d]}$ defined in Definition A.1.1.

5.2. HTR Expression for the Discrete Laplace Operator

The ALS-type algorithms from Chapter 4 require the operator and right-hand side to be given in HTR. We therefore derive HTR expressions for Δ_D and 1_D.

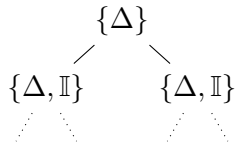
The expression for 1_D follows trivially from Theorem A.1.2. For Δ_D, we note that the multidimensional Laplacian can be written as the direct sum of one-dimensional Laplacians,

$$\Delta_D = \sum_{k \in D} \Delta_k \, \mathbb{I}_{D \setminus \{k\}},$$

which proves the splitting

$$\Delta_{L \cup R} = \Delta_L \, \mathbb{I}_R + \mathbb{I}_L \, \Delta_R.$$

for a partition L, R of D. For the non-quantized Laplacian, we therefore obtain the bases tree



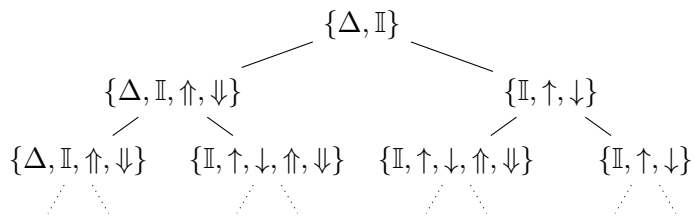
. (5.3)

Here and in the following, the dotted edges indicate that the rest of the bases tree is easily derived from the given part.

To account for quantization, we derive a quantized bases tree for each leaf $\{\Delta_k, \mathbb{I}_k\}$ of (5.3). Using the operators from Appendix A.3, we obtain the quantization splitting

$$\begin{aligned}
\Delta_k &= -\downarrow_k + 2\,\mathbb{I}_k - \uparrow_k \\
&= -(\downarrow_{(k,0)} \mathbb{I}_{(k,1)} + \Uparrow_{(k,0)} \downarrow_{(k,1)}) + 2\,\mathbb{I}_{(k,0)} \mathbb{I}_{(k,1)} - (\uparrow_{(k,0)} \mathbb{I}_{(k,1)} + \Downarrow_{(k,0)} \uparrow_{(k,1)}) \\
&= (-\downarrow_{(k,0)} + 2\,\mathbb{I}_{(k,0)} - \uparrow_{(k,0)}) \, \mathbb{I}_{(k,1)} - \Uparrow_{(k,0)} \downarrow_{(k,1)} - \Downarrow_{(k,0)} \uparrow_{(k,1)} \\
&= \Delta_{(k,0)} \mathbb{I}_{(k,1)} - \Uparrow_{(k,0)} \downarrow_{(k,1)} - \Downarrow_{(k,0)} \uparrow_{(k,1)} \, .
\end{aligned}$$

A bases tree for $\{\Delta_k, \mathbb{I}_k\}$ is therefore



(5.4)

A quantized bases tree for the multi-dimensional Laplacian is obtained by attaching a copy of this tree at every leaf of (5.3).

In the one-dimensional case $D = \{k\}$, the optimal bases tree for the Laplace operator Δ_k is given by a modification of (5.4) where the identity is removed from every vertex on the branch from the root to the leftmost leaf. Since the largest ranks appear in the inner part (as opposed to the left-most and right-most branch) of Equation (5.4), it follows that the ranks of both the one-dimensional as well as the multi-dimensional Laplace operator range up to five. This result is to be compared with [30] where it was found that the ranks of the discrete Laplacian in the QTT format are bounded by three ($\#D = 1$) and four ($\#D > 1$), respectively.

5.3. Common Details for Numerical Experiments

All benchmarks were run on four Quad-Core AMD Opteron$^{\text{TM}}$ 8356 processors (2.3 GHz), of which only a single core was used in Sections 5.4 and 5.5. Floating-point numbers are represented in the `long double` type of the C++ programming language, which is 16 bytes long and delivers the machine precision `eps` $\approx 1.1 \times 10^{-19}$ on the compiler and architecture we use.

Unless explicitly stated otherwise, we use the following parameters. We solve the local LSEs using the conjugate gradient (CG) algorithm, terminating the iterations once the relative local residual drops below 10^{-10} or the iteration count reaches the dimension of the LSE. In the ALS(SD) algorithms, the rank of the residual approximation z is chosen as $R_z = 4$ and the truncation step is carried out using $\varepsilon = 10^{-8}$. The initial guess is a random tensor of the indicated ranks for the ALS method, and the rank-one right-hand side $1_{[d]}$ for the ALS(SD) algorithm.

In the following plots, "Time per local LSE" denotes the total time of the iteration divided by the number of local LSEs solved, i.e. it includes the time spent on orthogonalization and subtree computation, and for the ALS(SD) method also the time spent on residual approximation, summation and truncation. "Error" refers to the relative error in the energy norm and "Residual" to the relative residual in the Euclidean norm, i.e.

$$\text{Error} := \frac{\|u - u^\star\|_{\Delta_{[d]}}}{\|u^\star\|_{\Delta_{[d]}}}, \quad \text{Residual} := \frac{\|f - \Delta_{[d]} u\|}{\|f\|},$$

where u denotes the current approximation and $u^\star$ the exact solution of the discrete Poisson equation.

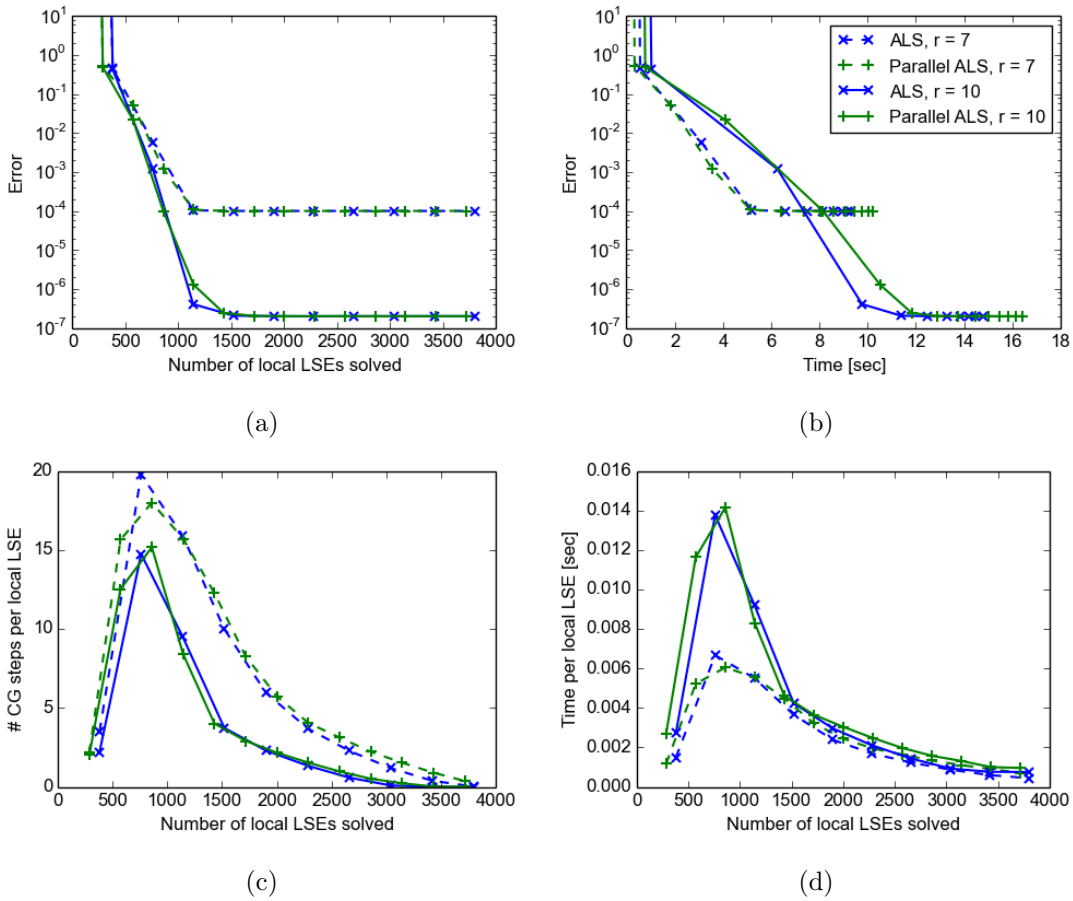


Figure 5.2.: Convergence of the serial and parallel ALS algorithm applied to the 16-dimensional discrete Poisson equation. r denotes the rank of the random initial tensor.

5.4. Comparison of Serial and Parallel ALS Algorithm

In Figure 5.2, we compare the convergence of the serial and parallel ALS Algorithms 2 and 3 starting from random HTR tensors of rank $r = 7$ and $r = 10$. We note that the convergence measured as a function of solved local LSEs (Figure 5.2a) is almost indistinguishable for the two methods, and the same applies to the convergence measured as a function of compute time (Figure 5.2b). We also note that the increased prefactors for orthogonalization and subtree computation in the parallel algorithm are barely visible in Figure 5.2d. Finally, we remark that both methods show monotone convergence, even though such convergence is only theoretically guaranteed for the serial ALS method.

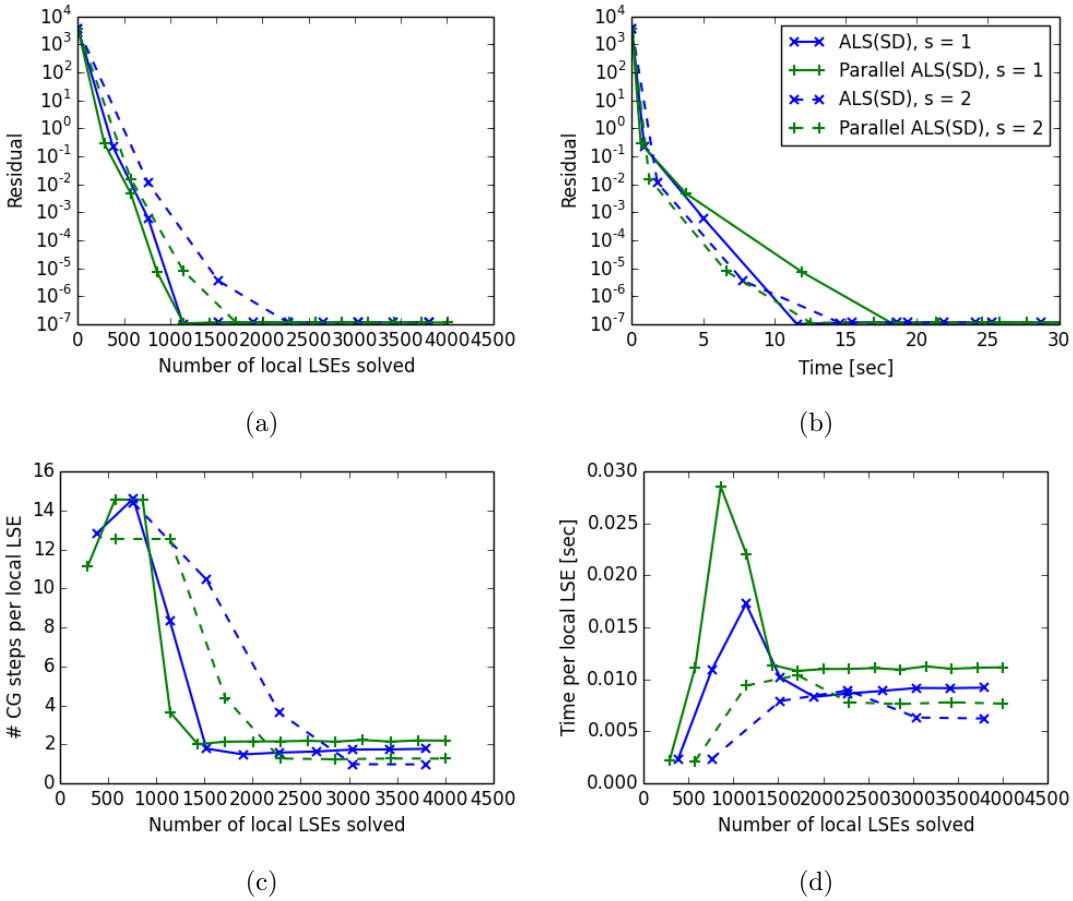


Figure 5.3.: Convergence of the serial and parallel ALS(SD) algorithm applied to the 16-dimensional discrete Poisson equation. s denotes the number of ALS steps (line 4 in Algorithm 4) per ALS(SD) step. All methods return results of the same ranks r_{T_D} with $\max_{\alpha \in T_D} r_\alpha = 13$.

5.5. Comparison of Serial and Parallel ALS(SD) Algorithm

We repeat the above convergence comparison, this time for the serial and parallel version of the ALS(SD) algorithm. Focussing on Figure 5.3a and ignoring the dashed lines for the moment, we conclude as before that the convergence as a function of solved local LSEs is the same for both the serial as well as the parallel algorithms. With respect to compute time, however, the parallel algorithm is significantly slower (Figure 5.3b). Since both methods require comparable numbers of CG steps (Figure 5.3c), we conclude that this slow-down must come from the increased cost per CG iteration due to the faster rank growth in the parallel algorithm (recall from Theorems 4.3.1 and 4.4.5 that the serial algorithms increases the ranks once every $4\,(d-2)$ local LSE solves compared to once every $3d-2$ solves in the parallel case).

To study the effect of the ratio between the number of local LSE solves and rank updates, we rerun the experiment doing $s = 2$ ALS iterations in each ALS(SD) step instead of $s = 1$ as in the original method. The quantities measured in this way are shown by the dashed lines in Figure 5.3. We note that while the serial ALS(SD) algorithm with $s = 1$ still reaches the final accuracy the fastest in terms of compute time, the parallel algorithm with $s = 2$ is almost as fast.

5.6. Convergence Criterion

Iterative solvers for linear systems commonly use a termination criterion based on the residual norm because it is asymptotically proportional to the error in the solution and thus the best accuracy measure available. Furthermore, the residual is computed anyway by many methods and checking this convergence criterion is therefore very cheap.

These statements apply to the ALS(SD) algorithm as well, however with the small but important caveat that the residual employed by the algorithm is an approximate one obtained from fixed-rank ALS iterations, cf. Section 4.5. In Figure 5.4, we compare the norm of this approximate residual with the exact norm in the case of the discrete Poisson equation. We note that the approximate norm slightly underestimates the actual error but should be reasonably accurate for most practical purposes.

5.7. Parallel Scaling of the ALS(SD) Algorithm

We analyze the scaling of the parallel ALS(SD) Algorithm 4 using the following two vertex distributions:

- **Optimized vertex distribution.** We distribute the vertices such that as many neighboring vertices as possible are located on the same process under the constraint that the optimal runtime from Theorem 3.5.7 must be achievable.[1]

[1] Our algorithm to produce such distributions is heuristic and may not find the optimal solution in every case. We expect the remaining difference to be irrelevant for the purposes of this section, however, and we checked that each solution used here permits optimal scaling under Assumption 3.5.6.

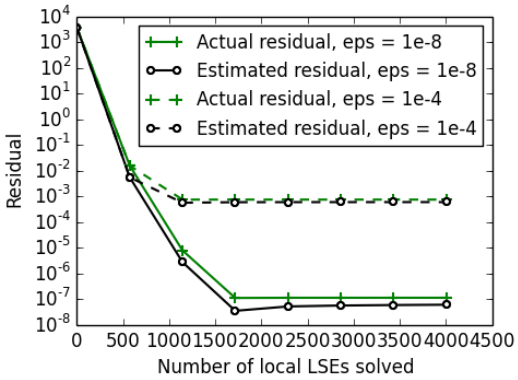


Figure 5.4.: Comparison of actual and estimated residual norm. `eps` denotes the accuracy ε in the truncation step. Both runs use $s = 2$, cf. Section 5.5.

- **Round-robin vertex distribution.** We enumerate the vertices in breadth-first order starting at the root and assign the ith vertex in this order to process $i \bmod p$, assuming the set of processes is $\{0, \dots, p-1\}$.

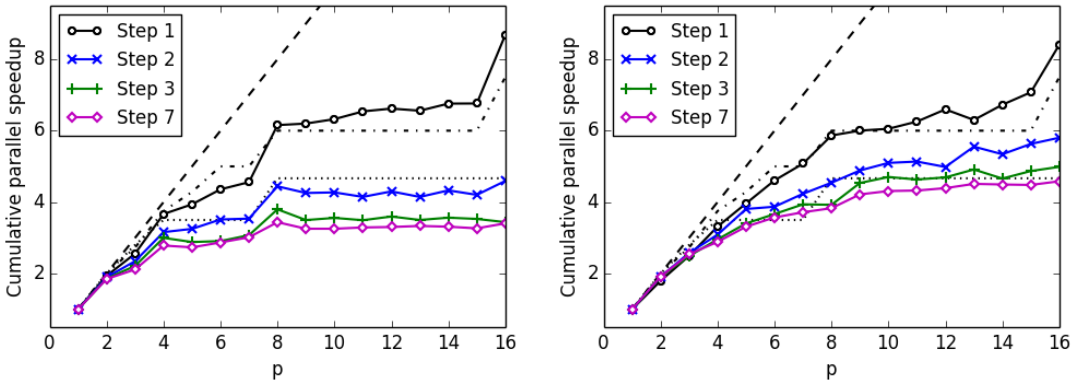
Examples of the resulting vertex distributions are given in Figure 5.6. Both distributions balance the number of vertices per process level-wise, i.e. they satisfy

$$\max_{q \in P} c(\ell, q) - \min_{q \in P} c(\ell, q) \leq 1$$

for each level ℓ with P denoting the set of processes and $c(\ell, q)$ the number of vertices on level ℓ assigned to process $q \in P$. The round-robin distribution also balances the overall number of vertices per process, whereas in the optimized vertex distribution the process with the heaviest load accommodates $\mathcal{O}(\log_2 p)$ more vertices than the process with the lightest load.

The parallel speedup of the ALS(SD) algorithm is shown in Figure 5.5. In the first iteration, the rank of the iterand u at an edge $\alpha \in T_D \setminus \{D\}$ is given by $\min\{2^{\#\alpha}, 5\}$, therefore all vertex tensors u_α with $\alpha \in \text{interior}(T_D)$ and $\text{level}(\alpha) \leq 4$ contain $5^3 = 125$ elements whereas the remaining vertices have ≤ 40 entries each. In the light of Section 3.5, we therefore expect the speedup to follow roughly the optimal speedup stated in Theorem 3.5.7 with $d = 32$. In the final iteration, the ranks connecting the first four levels are relatively uniform (between 11 and 13) and significantly larger than the remaining ranks (smaller or equal to 8), thus the empirical speedup should follow the optimal speedup for $d = 16$, and for the intermediate iterations the speedup should lie in between those two extremes. As can be seen in Figure 5.5b, these theoretical predictions are met with reasonable accuracy.

The rationale behind the optimized vertex distribution is that inter-process communication is more costly than intra-process communication, thus we distribute the vertices such that the largest possible fraction of exchanged messages falls into the latter category. One therefore expects the optimized distribution to perform at least as good as

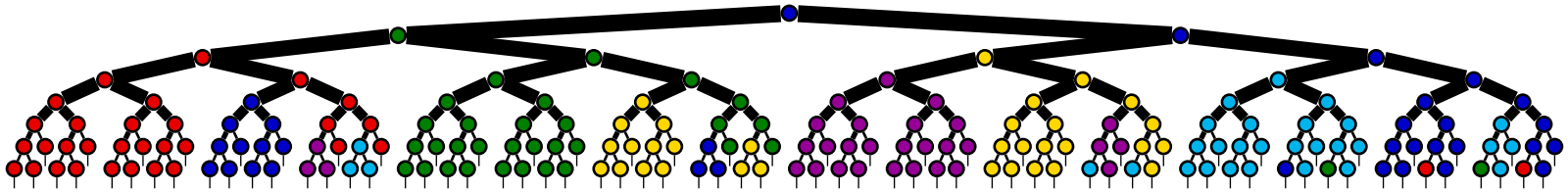


(a) Optimized vertex distribution.

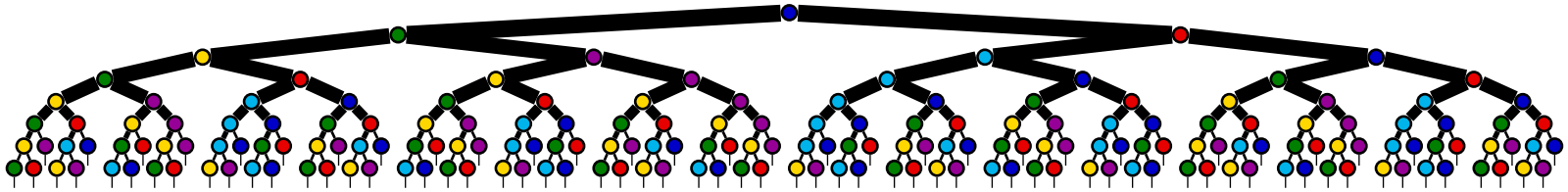
(b) Round-robin vertex distribution.

Figure 5.5.: Parallel scaling of the ALS(SD) solver with $s = 2$ applied to the 16-dimensional discrete Poisson equation. Cumulative parallel speedup is computed as $\frac{T(1)}{T(p)}$, where $T(p)$ is the wall clock time up to the indicated iteration step on p processors. The dashed lines denote perfect speedup, the dash-dotted lines the optimal speedup for $d = 32$ and the dotted lines the optimal speedup for $d = 16$, cf. Theorem 3.5.7.

the round-robin one, in contrast to the empirical findings shown in Figure 5.5. The reasons for this discrepancy remain unclear.



(a) Optimized.



(b) Round-robin.

Figure 5.6.: Illustration of the two vertex distributions for $p = 6$ processes. The edge widths are proportional to the final ranks of the ALS(SD) solution for the 16-dimensional discrete Poisson equation, cf. Section 5.5.

6. The Chemical Master Equation

6.1. Introduction

Consider an experiment involving a set of chemical species D subject to $R \in \mathbb{N}$ reactions of the form

$$\sum_{k \in D} e_k^s \, k \longrightarrow \sum_{k \in D} p_k^s \, k, \quad s = 0, \ldots, R-1.$$

Here, $e_k^s \in \mathbb{N}$ denotes the number of copies of species k consumed in a single occurrence of the sth reaction (the *educts*), and $p_k^s \in \mathbb{N}$ denotes the number of copies produced thereby (the *products*). We would like to know how many copies $i_k(t) \in \mathbb{N}$ of species $k \in D$ are present in the experiment at any moment $t \in \mathbb{R}$. Clearly, if we knew the initial copy numbers $i_k(0)$ and could count the number of occurrences $z_s(t) \in \mathbb{N}$ of reaction s in the interval $[0, t]$, we could compute the copy numbers at any time $t \in \mathbb{R}$ by

$$i_k(t) = i_k(0) + \sum_{s=0}^{R-1} \eta_k^s \, z_s(t), \quad \eta_k^s := p_k^s - e_k^s. \tag{6.1}$$

The problem is that we typically cannot predict the occurrence of reactions with absolute confidence. We must therefore treat the $z_s(t)$ and as a consequence of (6.1) also the $i_k(t)$ as time-dependent random variables $Z_s(t)$ and $I_k(t)$, respectively. These random variables are assumed to satisfy the following properties.

Definition 6.1.1 (Markov Process [31, §IV.1])**.** A time-dependent random variable $X(t) \in \Omega$ is called a *Markov process* if for any distinct time points $t_{[n+1]} \in \mathbb{R}^{[n+1]}$, $n \in \mathbb{N}$, sorted in ascending order, i.e. $t_0 < t_1 < \ldots < t_n$, it holds

$$\Pr[X(t_n) = x_n \mid X(t_{n-1}) = x_{n-1}, \ldots, X(t_0) = x_0] = \Pr[X(t_n) = x_n \mid X(t_{n-1}) = x_{n-1}]$$

for all $x_0, \ldots, x_n \in \Omega$.

Definition 6.1.2 (Homogeneity [31, §IV.4])**.** A time-dependent random variable $X(t) \in \Omega$ is called *homogeneous* if for any two time points $t_1, t_2 \in \mathbb{R}$ and offset $\Delta t \in \mathbb{R}$ it holds

$$\Pr[X(t_1 + \Delta t) = x' \mid X(t_1) = x] = \Pr[X(t_2 + \Delta t) = x' \mid X(t_2) = x]$$

for all $x, x' \in \Omega$.

Assuming $Z_s(t)$ is a homogeneous Markov process, one can conclude (see [31, 32] for detailed derivations) that there exist functions $\omega_s : \mathbb{N}^D \to \mathbb{R}_{\geq 0}$ such that the probability

for $z \in \mathbb{N}$ occurrences of reaction $s \in [R]$ in the interval $[0, \Delta t]$ is given by

$$\Pr[Z_s(\Delta t) = z \mid I_D(0) = i_D] = o(\Delta t) + \begin{cases} 1 - \omega_s(i_D)\Delta t & z = 0 \\ \omega_s(i_D)\Delta t & z = 1 \\ 0 & z > 1 \end{cases}$$

with $\Delta t \in \mathbb{R}_{\geq 0}$ the (small) length of the interval and $i_D \in \mathbb{N}^D$ the known initial state of the system. The function $\omega_s(i_D)$ is called the *propensity* of the sth reaction and can be interpreted as the probability density for the occurrence of a single reaction of this type. Note that in this model, all information about the current system state is contained in the copy number vector i_D. We will therefore use the two terms "state" and "copy numbers" interchangeably, and similarly we will sometimes call the set of all possible copy numbers $\mathbb{N}^D$ or its finite replacement introduced in Section 6.2 the *state space*.

Since we cannot obtain the exact copy numbers $i_k(t)$, we instead would like to compute the probability density function $p(t, i_D) := \Pr[I_D(t) = i_D]$ specifying the likelihood to count $i_D \in \mathbb{N}^D$ copies at time $t \in \mathbb{R}$. Clearly, the copy numbers remain unchanged until some reaction $s \in [R]$ occurs and changes the copy number vector from i_D to $i_D + \eta_D^s$. The probability for this event to occur in the interval $[t, t + \mathrm{d}t]$ is given by the product of the probability $p(t, i_D)$ to be in state i_D at time t and the conditional probability $\omega_s(i_D)\,\mathrm{d}t$ of moving from i_D to $i_D + \eta_D^s$ given that we are in state i_D. We therefore expect our quantity of interest $p(t, i_D)$ to satisfy the *chemical master equation* (CME)

$$\frac{\mathrm{d}p}{\mathrm{d}t}(t, i_D) = \sum_{s=1}^{R} \Big(\underbrace{\omega_s(i_D - \eta_D^s)\, p(t, i_D - \eta_D^s)}_{(A)} - \underbrace{\omega_s(i_D)\, p(t, i_D)}_{(B)} \Big) \tag{6.2}$$

for all $i_D \in \mathbb{N}^D$. In this equation, term (A) denotes the probability that we enter state i_D over reaction s and term (B) is the probability that reaction s moves us out of state i_D. Note that for some values of i_D and s, the shifted copy number vector $i_D - \eta_D^s$ may not lie in the physically feasible region, i.e. there may be a species k whose copy number $i_k - \eta_k^s$ is negative. For such i_D and s, we assume $p(t, i_D - \eta_D^s) = 0$ (the probability to be in an infeasible state is zero) and $\omega_s(i_D) = 0$ (the probability to move into an infeasible state is 0) such that the corresponding terms in (6.2) vanish. More rigorous derivations of the CME can be found in either [31] or [32].

From a mathematical point of view, equation (6.2) is a system of linear ordinary differential equations (ODE) in infinitely many unknowns $p(t, i_D)$. In order to make this problem amenable to numerical simulation, we must reduce the system to a finite number of unknowns, and we must discretize the time dimension. These two tasks are tackled next.

6.2. Finite State Projection

In most applications, the probability $p(t, i_D)$ decays to zero as one or more copy numbers i_D tend to infinity, which can be physically explained by noting that this extreme case

would correspond to a blow-up of the chemical system. This property forms the basis for the *finite state projection* (FSP) ansatz [33], which in our case reads: replace the function $p(t, i_D)$ of infinite support with a finitely sized, time dependent tensor $\hat{p}(t) \in \mathbb{R}^{\times_{k\in D}[n_k]}$, $n_D \in \mathbb{N}^D$, such that $\hat{p}(t, i_D)$ satisfies (6.2) for all $i_D \in \bigtimes_{k\in D}[n_k]$ under the zero-padding convention from Definition A.0.1. The FSP ansatz thus replaces $p(t, i_D)$ with an approximation

$$\hat{p}(t, i_D) := \begin{cases} p(t, i_D) + \varepsilon(t, i_D) & \text{if } i_D \in \bigtimes_{k\in D}[n_k] \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

where $\varepsilon(t, i_D) \in \mathbb{R}$ denotes the error in the tracked part $\bigtimes_{k\in D}[n_k]$ of the state space and is hoped to be small (a more precise statement will be given below).

An important consistency condition for the CME is that it preserves the total probability mass, i.e.

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\sum_{i_D\in\mathbb{N}^D} p(t, i_D)\right) = \sum_{i_D\in\mathbb{N}^D} \frac{\mathrm{d}p}{\mathrm{d}t}(t, i_D) = 0 \tag{6.4}$$

must hold for any probability density function $p(t, i_D)$. Inserting (6.2) into the above equation, this is easily seen to be satisfied: there is a one-to-one correspondence mapping each (B)-term with $i_D = i_D^\star$ in (6.2) to an (A)-term with $i_D = i_D^\star + \eta_D^s$ such that these two terms cancel. Let us check whether this condition is still satisfied in the finite state projection, i.e. if we replace p with $\hat{p}$ and $\mathbb{N}^D$ with $\bigtimes_{k\in D}[n_k]$. We distinguish four cases:

- The reaction causes a transition within the finite state space, i.e.

  $$i_D \in \bigtimes_{k\in D}[n_k], \quad i_D + \eta_D^s \in \bigtimes_{k\in D}[n_k].$$

  The same arguments as in the untruncated case apply.

- The reaction moves us out of the finite state space, i.e.

  $$i_D \in \bigtimes_{k\in D}[n_k], \quad i_D + \eta_D^s \notin \bigtimes_{k\in D}[n_k].$$

  In general, the corresponding (B)-term is nonzero whereas the (A)-term is not present in (6.4).

- The reaction moves us into the finite state space, i.e.

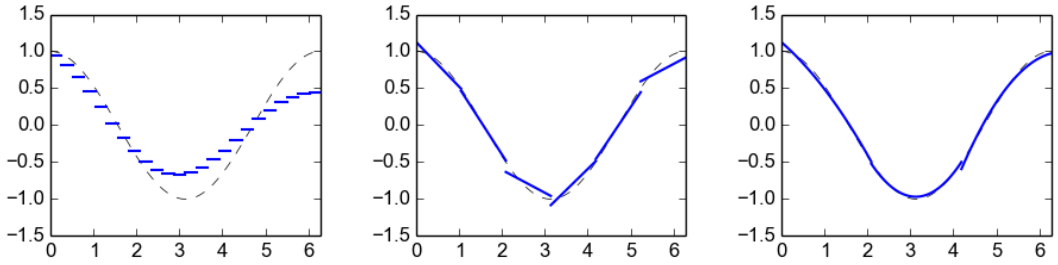
  $$i_D \notin \bigtimes_{k\in D}[n_k], \quad i_D + \eta_D^s \in \bigtimes_{k\in D}[n_k].$$

  The (B)-term is not present in (6.4) and the (A)-term is zero.

- The reaction causes a transition completely outside of the finite state space, i.e.

  $$i_D \notin \bigtimes_{k\in D}[n_k], \quad i_D + \eta_D^s \notin \bigtimes_{k\in D}[n_k].$$

  Neither term is present in (6.4).



(a) $n_t = 1$, $M = 24$ (b) $n_t = 2$, $M = 6$ (c) $n_t = 3$, $M = 3$

Figure 6.1.: Illustration of the discontinuous Galerkin scheme applied to $\ddot{f}(t) = -f(t)$.

In conclusion, we get that $\frac{\mathrm{d}}{\mathrm{d}t}\left(\sum_{i_D \in \times_{k\in D}[n_k]} \hat{p}(t, i_D)\right)$ may be negative, i.e. probability mass may flow out of the monitored domain. It turns out that this is precisely the error $\varepsilon(t, i_D)$ from (6.3).

Theorem 6.2.1. *Let $p(t, i_D)$ be the exact solution of the CME and $\hat{p}(t, i_D)$ the FSP approximation as defined above. We define the* probability deficiency $\varepsilon \in [0, 1]$ *as*

$$\varepsilon := 1 - \sum_{i'_D \in \times_{k\in D}[n_k]} \hat{p}(t, i'_D).$$

The probability deficiency provides a hard error bound, namely it holds

$$\hat{p}(t, i_D) \leq p(t, i_D) \leq \hat{p}(t, i_D) + \varepsilon$$

for all $i_D \in \times_{k\in D}[n_k]$.

Proof. [33, Theorem 2.2]. □

The above theorem allows to numerically bound the error incurred by the FSP, and in the experiments reported below we made sure that this bound is not larger than the errors stemming from other sources. In the remainder of this chapter, we will therefore no longer distinguish between the exact solution $p(t)$ and its FSP approximation $\hat{p}(t)$.

6.3. Discontinuous Galerkin Time-Stepping Scheme

The last section reduced the CME (6.2) to a finite system of linear ODEs

$$\frac{\mathrm{d}p}{\mathrm{d}t} = A_D\, p \quad \text{on } (0, T), \quad p(0) = p_0, \tag{6.5}$$

with unknown $p(t) \in \mathbb{R}^{\times_{k\in D}[n_k]}$, prescribed final time $T \in \mathbb{R}_{\geq 0}$ and initial state $p_0 \in \mathbb{R}^{\times_{k\in D}[n_k]}$, and the linear operator $A_D \in \mathbb{R}^{\times_{k\in D^2}[n_k]}$ defined by the right-hand side of (6.2) under the zero-padding convention from Definition A.0.1. We discretize these

equations in time using the *discontinuous Galerkin* scheme proposed in [34], which reads as follows.

Let $\mathcal{P}_{n_t}(t_0 \times t_1)$ be the space of all polynomials on the interval (t_0, t_1), $t_0, t_1 \in \mathbb{R}$, $t_0 < t_1$, up to some degree $n_t - 1 \in \mathbb{N}$ with coefficients from $\mathbb{R}^{\times_{k\in D}[n_k]}$, i.e.

$$\mathcal{P}_{n_t}(t_0 \times t_1) := \left\{ p : (t_0, t_1) \to \mathbb{R}^{\times_{k\in D}[n_k]}, t \mapsto \sum_{i_t \in [n_t]} x(i_t)\, t^{i_t} \mid x \in \mathbb{R}^{[n_t]\times(\times_{k\in D}[n_k])} \right\},$$

and let $\mathcal{P}_{n_t}\left(t_{[M+1]}\right)$ be the space of piecewise polynomials on the interval $(0, T)$,

$$\mathcal{P}_{n_t}\left(t_{[M+1]}\right) := \left\{ p : (0,T) \to \mathbb{R}^{\times_{k\in D}[n_k]} \mid \forall m = 1, \ldots, M : p|_{(t_{m-1}, t_m)} \in \mathcal{P}_{n_t}(t_{m-1} \times t_m) \right\}$$

with $t_{[M+1]} \in (0,T)^{[M+1]}$ such that $0 = t_0 < t_1 < \ldots < t_M = T$. The discontinuous Galerkin scheme approximates the solution $p(t)$ of (6.5) with the element $\hat{p}(t) \in \mathcal{P}_{n_t}\left(t_{[M+1]}\right)$ satisfying

$$\int_{t_{m-1}}^{t_m} \left(q(t),\, \frac{\mathrm{d}\hat{p}}{\mathrm{d}t}(t) - A\,\hat{p}(t) \right) \mathrm{d}t + \left(q(t_{m-1}),\, \Delta\hat{p}(t_{m-1}) \right) = 0 \tag{6.6}$$

for all $q(t) \in \mathcal{P}_{n_t}(t_{m-1} \times t_m)$ and $m = 1, \ldots, M$. Here, $\Delta\hat{p}(t_{m-1})$ denotes the jump of $\hat{p}(t)$ at an interior interval boundary t_{m-1} with $m = 2, \ldots M$,

$$\Delta\hat{p}(t_{m-1}) := \lim_{\Delta t \downarrow 0} \hat{p}(t_{m-1} + \Delta t) - \hat{p}(t_{m-1} - \Delta t),$$

and we set $\Delta\hat{p}(t_0) := \lim_{\Delta t \downarrow 0} \hat{p}(\Delta t) - p_0$ for the initial jump. Since (6.6) depends on the solution $\hat{p}(t)$ in the prior interval (t_{m-2}, t_{m-1}) or the initial conditions p_0, respectively, $\hat{p}(t)$ is most easily determined interval-wise starting on (t_0, t_1) and proceeding iteratively towards larger times.

We briefly recapitulate the key properties of the discontinuous Galerkin method from [34]. For this purpose, let $\Delta t := \max\{t_m - t_{m-1} \mid m = 1, \ldots M\}$ denote the maximal step size, and

$$\varepsilon := \sup_{t \in (0,T)} \|p(t) - \hat{p}(t)\|$$

be a bound for the error. Then:

- The discontinuous Galerkin solution is well-defined if $\|A\|\,\Delta t < 1$ [34, Theorem 2.6].
- The method converges with order n_t in the step size [34, Corollary 3.15] and exponentially in the polynomial degree [34, Theorem 3.18], i.e. $\varepsilon \in \mathcal{O}\left(\Delta t^{n_t}\right)$ for $\Delta t \to 0$, and $\varepsilon \in \mathcal{O}\left(\exp(-b\, n_t)\right)$ for $n_t \to \infty$, $b \in \mathbb{R}_{>0}$.

We next reformulate (6.6) as a tensor-structured linear system of equations amenable to the ALS-type solvers from Chapter 4. The same endeavour has already been undertaken in [35], and our exposition proceeds along the same lines.

Let $\{\phi_{i_t}^{(m)}(t) \mid i_t \in [n_t]\}$ be a basis for $\mathcal{P}_{n_t}(t_{m-1} \times t_m)$, $m = 1, \ldots, M$. Expanding $\hat{p}|_{[t_{m-1},t_m]}$ according to

$$\hat{p}(t) = \sum_{i_t=[n_t]} p^{(m)}(i_t)\ \phi_{i_t}^{(m)}(t), \quad t \in (t_{m-1}, t_m), \tag{6.7}$$

with some tensor $p^{(m)} \in \mathbb{R}^{[n_t] \times (\times_{k \in D}[n_k])}$, (6.6) can be recast as the tensor-formatted linear system

$$\Big(D_t\, \mathbb{I}_D - M_t\, A_D\Big)\, p^{(m)} = F_t\, p^{(m-1)} \tag{6.8}$$

with operators $D_t, M_t, F_t \in \mathbb{R}^{[n_t]^2}$ given by

$$\begin{aligned}
D_t(i_{R(t)} \times i_{C(t)}) &:= \int_{t_{m-1}}^{t_m} \phi_{i_{R(t)}}^{(m)}(t)\ \frac{\mathrm{d}\phi_{i_{C(t)}}^{(m)}}{\mathrm{d}t}(t)\,\mathrm{d}t + \phi_{i_{R(t)}}^{(m)}(t_{m-1})\,\phi_{i_{C(t)}}^{(m)}(t_{m-1}),\\
M_t(i_{R(t)} \times i_{C(t)}) &:= \int_{t_{m-1}}^{t_m} \phi_{i_{R(t)}}^{(m)}(t)\ \phi_{i_{C(t)}}^{(m)}(t)\,\mathrm{d}t,\\
F_t(i_{R(t)} \times i_{C(t)}) &:= \phi_{i_{R(t)}}^{(m)}(t_{m-1})\,\phi_{i_{C(t)}}^{(m-1)}(t_{m-1}),
\end{aligned}$$

The basis functions are chosen as $\phi_{i_t}^{(m)}(t) := L_{i_t}\left(f_m^{-1}(t)\right)$, where $L_{i_t}(t) : (-1,1) \to \mathbb{R}$ are the Legendre polynomials normalized such that $L_{i_t}(1) = 1$, and $f : (-1,1) \to (t_{m-1}, t_m)$ is the affine linear map

$$f_m\left(\hat{t}\right) = \frac{1}{2}(t_m + t_{m-1}) + \frac{1}{2}(t_m - t_{m-1})\,\hat{t}.$$

Note that in contrast to [35] we do not normalize the basis functions, because we empirically found that the GMRES solver used in the local problems delivers better performance for the unnormalized basis.

Below, we will use a slightly different notation in that we let $p(t)$ denote the discontinuous Galerkin solution (the $\hat{p}(t)$ from above) and denote the exact solution by $p^\star(t)$ instead. $p^{(m)}$ will always denote the coefficient tensor from (6.7) on the mth interval.

6.4. Chemical Notation and the CME Operator

Following the usual chemical notation, we denote a reaction consuming $e_k \in \mathbb{N}_{>0}$ copies of the educt species $E \subseteq D$ and producing $p_k \in \mathbb{N}_{>0}$ copies of the product species $P \subseteq D$ by

$$\sum_{k \in E} e_k\, k \xrightarrow{a(D)} \sum_{k \in P} p_k\, k. \tag{6.9}$$

The formal function $a : D \to \mathbb{R}_{\geq 0}$ is called the *stochastic reaction rate constant* [32] and is related to the propensity $\omega(i_D)$ of this reaction by

$$\omega(i_D) = a(i_D) \prod_{k \in E} \binom{i_k}{e_k}. \tag{6.10}$$

The intended interpretation of $a(i_D)$ is that each occurrence of the symbol $k \in D$ in the $a(D)$ from (6.9) is to be replaced with the corresponding copy number i_k in the $a(i_D)$ from (6.10). Physically, the reaction rate constant is the probability density of a specific combination of copies of the educt species to undergo this reaction. If a reaction involves no educts, $E = \{\}$, or delivers no products, $P = \{\}$, we denote this by putting the *null species* $\varnothing$ on the respective side of the reaction arrow [32, §2.1].

Using the shift and counting operators $S_k^{(s)}$ and C_k from Appendix A, the term in the CME operator A_D resulting from reaction (6.9) can be written as

$$\left(\prod_{k\in P} S_k^{(-p_k)}\right)\left(\prod_{k\in E} S_k^{(e_k)}\binom{C_k}{e_k}\right)\operatorname{diag}(a) - \left(\prod_{k\in E}\binom{C_k}{e_k}\right)\operatorname{diag}(a), \tag{6.11}$$

or, by factoring out the propensity part,

$$\left(\left(\prod_{k\in E} S_k^{(e_k)}\right)\left(\prod_{k\in P} S_k^{(-p_k)}\right) - \mathbb{I}_{E\cup P}\right)\left(\prod_{k\in E}\binom{C_k}{e_k}\right)\operatorname{diag}(a). \tag{6.12}$$

Here, $\binom{C_k}{e_k}$ denotes the binomial coefficient

$$\binom{C_k}{e_k} := \frac{1}{e_k!}\prod_{n=0}^{e_k-1}\left(C_k - n\mathbb{I}_k\right)$$

and a is the tensor in $\mathbb{R}^{\times_{k\in D}[n_k]}$ containing the values of the reaction rate constant. Starting from either (6.11) or (6.12), the formulas for concrete CME operators given below are easily derived.

6.5. Common Details for the Numerical Experiments

In the remainder of this chapter, we reproduce the numerical experiments from [1], using the more general and parallelizable HTR instead of the TT format which was used there. Besides investigating the parallel potential of these problems, we will also be interested in whether the more general dimension partition trees supported by the HTR allow to achieve smaller ranks and thereby possibly shorter execution times. We remark, however, that besides different formats we also use different hardware (2.7 GHz vs. 2.3 GHz) and different programming environments (MATLAB vs. C++) compared to [1], therefore we will base our comparison mostly on the ranks rather than the runtime.

All benchmarks were run on four Quad-Core AMD Opteron$^{\text{TM}}$ 8356 processors (2.3 GHz). Floating-point numbers are represented in the `double` type of the C++ programming language, which is 8 bytes long and delivers the machine precision $\texttt{eps} \approx 2.2\times10^{-16}$ on the compiler and architecture we use.

The tensor-formatted LSE (6.8) arising in each time step is solved using the parallel ALS(SD) solver from Section 4.5. In the first time interval, we take $p^{(1)} = \delta(i_t = 0)\, p_0$ corresponding to $p(t) = p_0$ as initial guess for the solver, in later intervals we take the

previous solution $p^{(m)} = p^{(m-1)}$. The ALS(SD) iterations are terminated once the estimated relative residual (c.f. Section 5.6) drops below 10^{-5} or the iteration count reaches 3. Except for the $\Delta t = 10^{-2}$ run for the toggle switch problem, the accuracy criterion was already met in the first iteration, typically with at least an order of magnitude of slackness.

The local problems are solved using the GMRES algorithm with restarts every 60 iterations for the birth-death problem and every 120 iterations for the other two problems. The iterations are terminated once the iteration count reaches the dimension of the LSE or the relative residual drops below $\theta \cdot 10^{-7}$ with $\theta = \sqrt{2d-3}$. Here, d denotes the total number of free modes (including the time dimension), and the prefactor θ makes sure that the accuracy in the local LSEs scales with the tolerance used in the subsequent truncation step, see [6, Theorem 3.18]. The rank of the residual approximation z is chosen as $R_z = 4$ and the truncation step is carried out using $\varepsilon = 10^{-6}$. Only one ALS iteration is performed per ALS(SD) step, i.e. $s = 1$ in the notation from Section 5.5.

We construct the temporal mesh as in [1]. Given the final time $T \in \mathbb{R}_{>0}$ and user-specified parameters $\Delta t, T_1 \in \mathbb{R}_{>0}$, we define

$$\sigma := \frac{1}{1 - \frac{\Delta t}{T_1}}, \quad M_1 := \left\lceil \frac{T_1}{\Delta t} \right\rceil, \quad M_2 := \left\lfloor \log_\sigma \left(\frac{T}{T_1} \right) \right\rfloor, \quad M := 11 + M_1 + M_2 + 1$$

and choose the interval boundaries $t_{[M]} \in \mathbb{R}^{[M]}$ as

$$t_m := \begin{cases} 0 & \text{if } m = 0 \\ \Delta t \, 2^{11-m} & \text{if } 1 \leq m < 11 \\ \Delta t \, (m - 10) & \text{if } 11 \leq m < 11 + M_1 \\ T_1 \, \sigma^{(m - M_1 - 10)} & \text{if } 11 + M_1 \leq m < 11 + M_1 + M_2 \\ T & \text{if } m = 11 + M_1 + M_2 \end{cases}.$$

The rationale of this mesh is to gradually build up $p^{(m)}$ on $(0, \Delta t)$ $(m = 1, \ldots, 11)$ by starting with very small but geometrically increasing step sizes. Then, on $(\Delta t, T_1)$ $(m = 12, \ldots, 10 + M_1)$ where there are strong fluctuations in $p(t)$ we use an equidistant mesh and finally let the solution converge to the steady state on (T_1, T) $(m = 11 + M_1, \ldots, M-1)$ using again a geometrically graded mesh but with the fairly small grading factor σ.

6.6. Independent Birth-Death Processes

Consider $d \in \mathbb{N}$ chemical species $D := \{X_0, \ldots, X_{d-1}\}$ each of which is produced and destroyed at constant rates $b_k, d_k \in \mathbb{R}_{>0}$, i.e. we have $2d$ reactions of the form

$$\varnothing \underset{d_k}{\overset{b_k}{\rightleftharpoons}} X_k, \quad k \in D. \tag{6.13}$$

The simple structure of (6.13) as well as the availability of an analytical solution of the corresponding CME, see [36], allow us to investigate the scaling of our ansatz with respect to d.

6.6.1. Dimension Partition Tree

We use a dimension partition tree obtained as follows. First, construct a balanced binary tree for the set of species D, then replace each leaf $\{k\}$ of this tree with a balanced binary tree for the virtual modes $(k, [l_k])$ (c.f. Figure 5.1 for the construction so far), and finally attach an additional vertex for the (non-quantized) time dimension at the root.

6.6.2. HTR Expression for the CME Operator

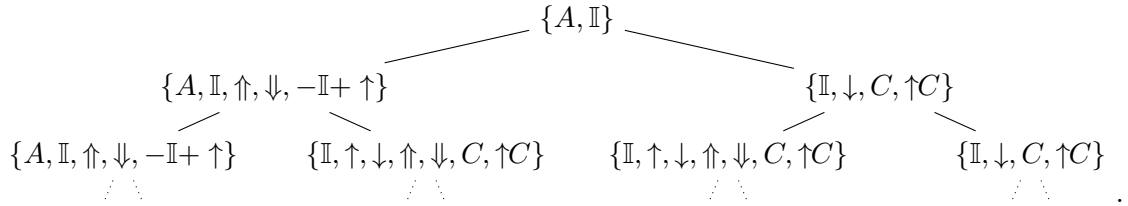
Since each reaction involves only a single copy of a single species, the CME operator in this case is very similar to the discrete Laplace operator from Chapter 5, namely

$$A_D = \sum_{k \in D} A_k \, \mathbb{I}_{D \setminus \{k\}} \tag{6.14}$$

with one-dimensional operators

$$A_k = b_k \downarrow_k -(b_k + d_k C_k) \, \mathbb{I}_k + d_k \uparrow_k C_k.$$

The upper, inter-species part of the bases tree is therefore as in (5.3) with Δ replaced by A, and with the splitting relations given in the appendix, the intra-species part is straightforwardly found to be



6.6.3. Parameters

Time-stepping:	$\Delta t = 10^{-3}$, $T_1 = 10^{-1}$, $T = 10$ ($M = 569$ time steps), $n_t = 4$ or $n_t = 20$, as indicated
FSP:	$l_k = 12$, $n_{(k,j)} = 2$ for all $k \in D$, $(k, j) \in (k, [l_k])$
Reaction:	$b_k = 1000$, $d_k = 1$ for all $k \in D$
Initial conditions:	$p_0 = \delta\left(\bigtimes_{k \in D}(i_k = 0)\right)$

6.6.4. Results

In the first batch of runs, we use the same time stepping parameters as in [1], i.e. $n_t = 4$. As shown in Table 6.1, the HTR scales more favourably for this particular problem than the TT format, allowing us to more easily investigate the scaling for larger d than what has been shown in [1]. Before discussing these empirical findings, however, we would first like to highlight an important theoretical property of this problem.

	TT		HTR	
d	T	r_{max}	T	r_{max}
1	87	11	29	10
2	704	21	70	10
3	1548	21	112	11
4	2516	21	158	11
5	3544	21	204	14

Table 6.1.: Independent birth-death processes. Comparison of runtimes T (in seconds) and maximal ranks r_{max} for the TT-based approach from [1] and the HTR-based approach (with $n_t = 4$) considered here. The maximum in r_{max} is taken over all edges of $p^{(m)}$ for all $m = 1, \dots, M$.

The initial conditions $p_0 = \delta\left(\bigtimes_{k\in D}(i_k = 0)\right) = \prod_{k\in D}\delta(i_k = 0)$ imply statistical independence of the initial copy numbers and since the species do not interact, we expect their copy numbers at later times to be statistically independent as well, i.e. $p(t, i_D) = \prod_{k\in D} p(t, i_k)$ should hold for $\#D$ univariate probability density functions $p : (0, T) \times [n_k] \to [0, 1]$ which satisfy the one-dimensional CME

$$\frac{\mathrm{d}p}{\mathrm{d}t}(t, i_k) = A_k\, p(t, i_k), \quad p(0) = \delta(i_k = 0).$$

Given the structure (6.14) of the CME operator, this is straightforward to verify:

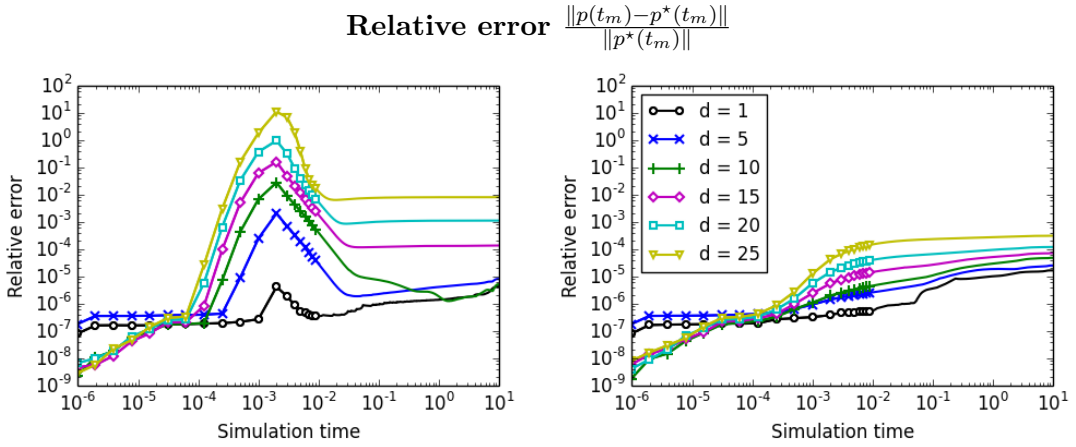
$$\begin{aligned}\frac{\mathrm{d}p}{\mathrm{d}t}(t, i_D) &= \sum_{k\in D}\left(\prod_{\ell\in D\setminus\{k\}} p(t, i_\ell)\right)\frac{\mathrm{d}p}{\mathrm{d}t}(t, i_k)\\ &= \sum_{k\in D}\left(\prod_{\ell\in D\setminus\{k\}} p(t, i_\ell)\right) A_k\, p(t, i_k) = A_D\, p(t, i_D).\end{aligned}$$

In the terminology of linear algebra, $p(t, i_D) = \prod_{k\in D} p(t, i_k)$ implies that $p(t, i_D)$ is rank-one separable with respect to any $M \subset D$. At any *single* time point $t \in (0, T)$, $p(t, i_D)$ can thus be represented in an HTR network with all physical ranks[1] equal to one, and even though the virtual ranks may be larger they should be constant with respect to d.

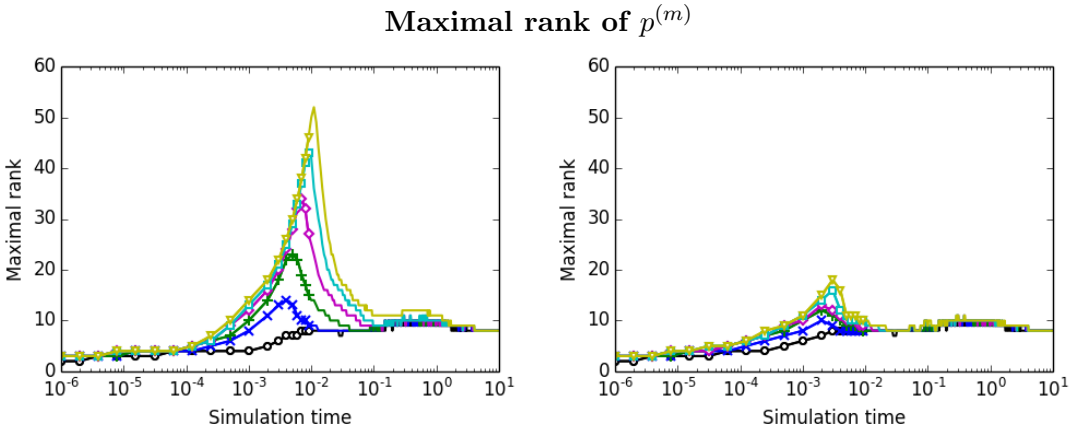
As is shown in Figure 6.2c, the same does not hold for the ranks of $p^{(m)}$ which may be explained as follows. Assume the one-dimensional $p(t, i_k)$ can be accurately represented on the interval (t_{m-1}, t_m) by

$$p(t, i_k) = \sum_{i_t\in[n_t]} p^{(m)}(i_t \times i_k)\,\phi^{(m)}_{i_t}(t).$$

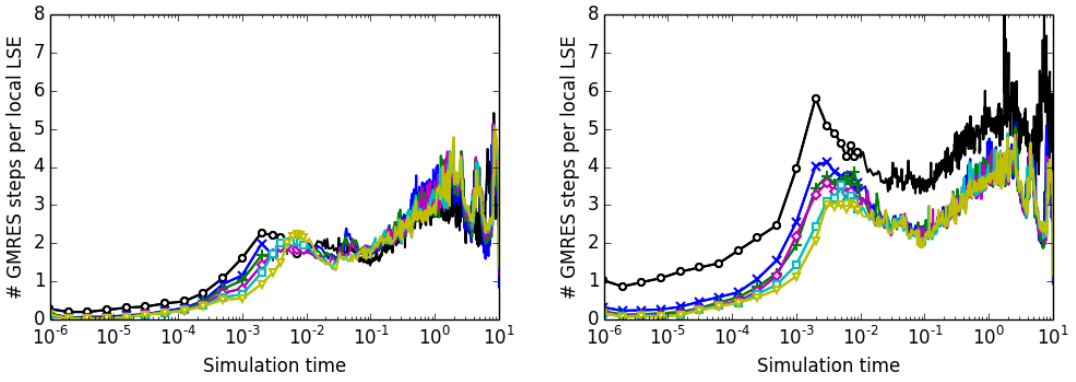
[1] A rank r_α with $\alpha \in T_D$ is called *physical* if it only contains complete physical modes, i.e. $(k, j) \in \alpha \implies (k, [l_k]) \subseteq \alpha$, and *virtual* otherwise.



(a) $n_t = 4$ (b) $n_t = 20$



(c) $n_t = 4$ (d) $n_t = 20$



(e) $n_t = 4$ (f) $n_t = 20$

Figure 6.2.: Independent birth-death processes. Various quantities as a function of the simulation time t. Each time step required only a single ALS(SD) iteration, thus "number of GMRES step per local LSE" is directly proportional to the effort in that time step.

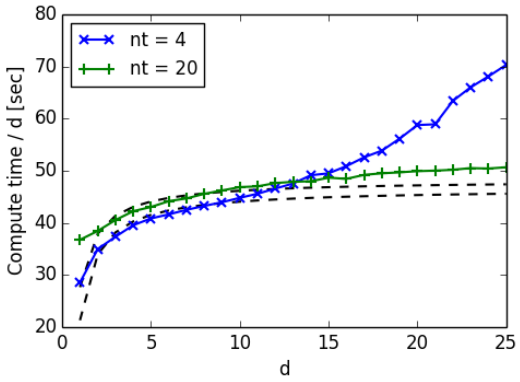


Figure 6.3.: Independent birth-death processes. Compute time $T(d)$ divided by the dimension d. The dashed lines show linear scaling fitted to $T(d)$ on $d = 1, \ldots, 10$.

Then, the corresponding outer-product representation for the bivariate function reads

$$\begin{aligned} p(t, i_0 \times i_1) &= \left(\sum_{i_t \in [n_t]} p^{(m)}(i_t \times i_0)\, \phi^{(m)}_{i_t}(t) \right) \left(\sum_{i_t \in [n_t]} p^{(m)}(i_t \times i_1)\, \phi^{(m)}_{i_t}(t) \right) \\ &= \sum_{i_t \in [n_t]} \sum_{i'_t \in [n_t]} p^{(m)}(i_t \times i_0)\, p^{(m)}(i'_t \times i_1)\, \phi^{(m)}_{i_t}(t)\, \phi^{(m)}_{i'_t}(t) \end{aligned}$$

and involves the product $\phi^{(m)}_{i_t}(t)\, \phi^{(m)}_{i'_t}(t)$ of temporal basis functions. To get back to the discontinuous Galerkin ansatz (6.7), we have to approximate these products in $\operatorname{span}\{\phi^{(m)}_{i_t}(t) \mid i_t \in [n_t]\}$ which is in general not possible with reasonable accuracy as is shown in Figure 6.2a. It seems plausible that once we deviate from the exact solution of known low rank by more than the truncation tolerance, we will not find another low-rank tensor within this (small) tolerance, which is exactly what we observe in Figure 6.2c. This finding is a particular example of the general rule that truncation procedures are only effective when carried out with a tolerance ε which lies above the noise in the tensor stemming from other sources (the time-stepping, in this case). A similar empirical observation has already been made in [37, §5.1].

Clearly, the aforementioned problem can be overcome by increasing the temporal approximation space $\mathcal{P}_{n_t}(t_{[M+1]})$. It is a particular merit of the tensor network approach that overestimating the dimension of this space is very cheap, because basis functions $\phi^{(m)}_{i_t}(t)$ whose associated coefficients $p^{(m)}(i_t)$ are small will be removed from the representation in the sense that the *time rank* r_t - the rank of the edge which connects the time to the remaining modes - can be smaller than n_t. We therefore present a second batch of runs in which we increase the number of basis functions per interval from $n_t = 4$ to $n_t = 20$. The error and correspondingly also the ranks remain much smaller in this case (Figures 6.2b and 6.2d), and the method scales more favourably with respect to d

(Figure 6.3). The somewhat higher computational costs for small d can be explained by noting that the runs with $n_t = 20$ need more GMRES iterations than the $n_t = 4$ runs (Figures 6.2e and 6.2f), which may be because of the somewhat worse condition number due to the increased problem size.

As a corollary of the above discussion, we note that we can use the time rank r_t as a (heuristic) indicator for the suitability of the step sizes: if r_t is smaller than n_t, we may assume that the solution is sufficiently smooth to allow for a low-order expansion of $p(t)$ on the current interval. In the remaining runs, we therefore always use an overestimated n_t to make sure that we do not commit substantial time-stepping errors.

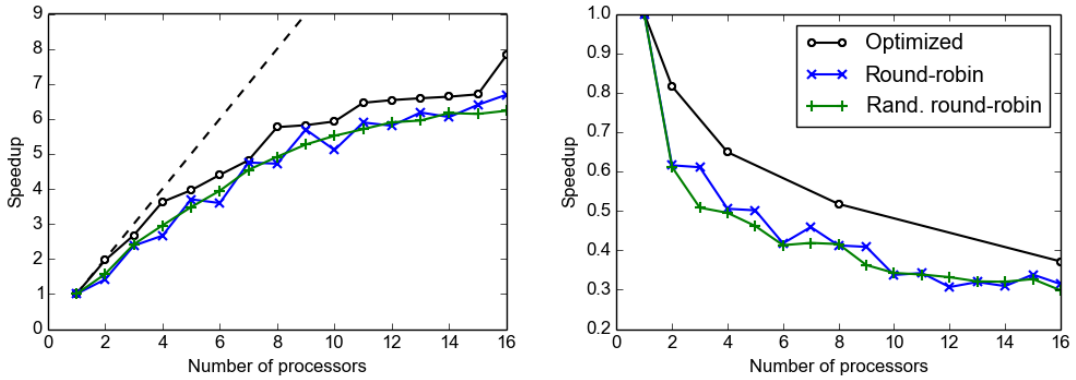
6.6.5. Parallel Scaling

In [36] it is shown that the exact univariate probability density function $p(t, i_k)$ is a Poisson distribution with parameter $\lambda(t) := \frac{b_k}{d_k}\left(1 - e^{-d_k t}\right)$. Given the above values for b_k and d_k, the standard deviation of $p(t, i_k)$ is thus bounded by $\sqrt{1000} \approx 32$ meaning that $p(t, i_D)$ is fairly localized for all times t. In a quantized representation, locality implies that the ranks associated with the small virtual indices (the ones which capture the small-scale variations) are much larger than the ranks of the large virtual indices, see Figure 6.5. This special structure of the ranks must be taken into account when distributing the vertices as we will see next.

The standard round-robin vertex distribution from Section 5.7 performs worse for even numbers of processors than for odd ones, see Figure 6.4. The reason for this is illustrated in Figure 6.6: for an odd number of processes, the "heavy" vertices, say the ones with more than 50 elements in this example, are distributed evenly among the processes (red has three such vertices, green and blue have two each). For an even number, however, it is possible that some processes host a larger fraction of heavy vertices than others (yellow has three, green two and blue and red one), which undermines the purpose of parallelization. In an attempt to overcome this problem, we introduce the *randomized round-robin distribution* which deals the ith vertex in breadth-first order to process $i + c \bmod p$ where $c \in [p]$ is a random integer chosen anew after every p dealt processes. As we can see in Figure 6.4, this modification indeed smoothes out the irregular scaling of the round-robin distribution, but it does not improve the overall scaling. The best distribution for this particular problem is the optimized one because it assigns entire physical modes to the same process whenever possible. Unfortunately, our algorithm to produce optimized distributions works only for balanced dimension partition trees, which prohibits its application if d is not a power of two.

6.7. Toggle Switch

In the context of biological reaction networks, a *toggle switch* is a network which possesses two stable steady states. A synthetic construction of such a network inside a living organism has been presented in [38] together with a deterministic, concentration-based mathematical model to describe its behaviour. This model involves two chemical species



(a) Strong scaling, $d = 16$. (b) Weak scaling, $d = p$.

Figure 6.4.: Independent birth-death processes. Parallel scaling on p processors ($n_t = 20$).

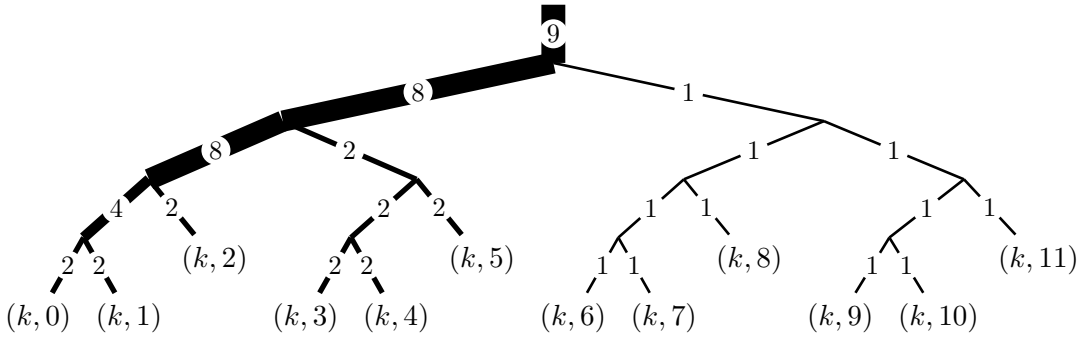
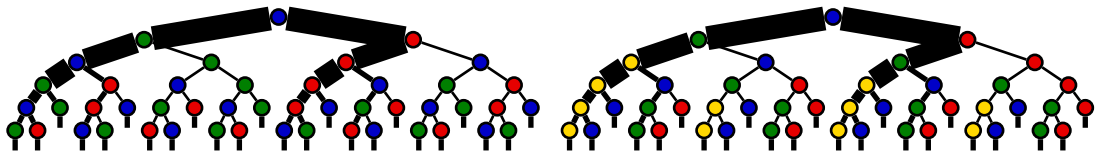


Figure 6.5.: Independent birth-death processes. Virtual ranks of $p^{(m)}$ for $d = 16$, $n_t = 20$, $t_m = 0.003$ and species $k = X_{13}$ (for other species, the ranks may be one less).



(a) $p = 3$ (b) $p = 4$

Figure 6.6.: Independent birth-death processes. Round-robin vertex distribution for $d = 2$ and number of processes p as indicated. The edge weights are obtained by duplicating the tree from Figure 6.5 and connecting the dangling edges from the two old roots to the new root. The time vertex, which would be attached to the root, is not shown.

$D := \{U, V\}$ each of which gets produced at a rate inversely proportional to the copy number of the other and degrades at a constant rate, i.e. we have the four reactions

$$\varnothing \underset{d_U}{\overset{a_U(V)}{\rightleftharpoons}} U, \quad \varnothing \underset{d_V}{\overset{a_V(U)}{\rightleftharpoons}} V, \quad a_k(\ell) := \frac{b_k}{1 + (\ell)^{e_k}}, \tag{6.15}$$

with parameters $b_k, e_k, d_k \in \mathbb{R}_{>0}$ for $k, \ell \in D$.

This system is a toggle switch with the two steady states informally described by "high copy number of U, low copy number of V" and "low copy number of U, high copy number of V". In a deterministic setting, knowing the initial state of a trajectory allows to predict with perfect certainty the steady state it will eventually end up in, and once this steady state has been reached the trajectory will remain there for all times. The system thus provides an elementary biological memory unit (a biological bit) which could e.g. be used as a basic building block of a biocomputer [38]. Biological systems are intrinsically noisy, however, which causes a number of complications. Not only can trajectories from the same initial state end up in different steady states, the system can also toggle repeatedly from one steady state to the other. While such behaviour is clearly not desirable for a memory unit, it is encountered in nature as a strategy used by pathogens to evade the defense mechanisms of the host [39, 40].

6.7.1. Dimension Partition Tree

We use a dimension partition tree constructed as described in Section 6.6.1.

6.7.2. Obtaining the CME Operator

The CME operator for the reactions (6.15) is

$$A_D = (\downarrow_U - \mathbb{I}_U)\,\mathrm{diag}(a_U) + (\downarrow_V - \mathbb{I}_V)\,\mathrm{diag}(a_V) + d_U(\uparrow_U - \mathbb{I}_U)\,C_U + d_V(\uparrow_V - \mathbb{I}_V)\,C_V.$$

HTR expressions for the last two operators describing the destruction of species can be derived straightforwardly from the splittings given in Appendix A. The splitting at the root for the first term is given by

$$\begin{array}{ccc} & \{A_D\} & \\ \swarrow & & \searrow \\ \{\downarrow_U - \mathbb{I}_U\} & & \{\mathrm{diag}(a_V)\}\ . \end{array}$$

The left part of this bases tree can again be filled in straightforwardly. For the right part, we assemble a_U as a full tensor and truncate it to HTR using [6, Algorithm 2] with a relative truncation tolerance $\varepsilon = 10^{-14}$. The second term is obtained analogously. Finally, we assemble A_D by summing all terms and truncating once more with a relative tolerance of 10^{-14}. The resulting ranks of the time-stepping operator are shown in Figure 6.10a.

	Δt=10^{-3}	TT	SSA
Δt=10^{-2}	$1.64 \cdot 10^{-4}$	$2.22 \cdot 10^{-5}$	$8.35 \cdot 10^{-4}$
Δt=10^{-3}		$1.56 \cdot 10^{-4}$	$8.62 \cdot 10^{-4}$
TT			$8.34 \cdot 10^{-4}$

Table 6.2.: Toggle switch. Differences in the ℓ_1-norm at the final time $t = 100$. "TT" and "SSA" denote the respective solutions from [1].

6.7.3. Parameters

Time-stepping: $\Delta t = 10^{-2}$ or $\Delta t = 10^{-3}$, $T_1 = 1$, $T = 100$, $n_t = 20$
($M = 576$ or $M = 5620$ time steps)

FSP: $l_U = 13$, $l_V = 12$,
$n_{(U,j)} = n_{(V,j)} = 2$ for all $j \in [l_U]$ and $j \in [l_V]$, respectively

Reaction: $b_U = 5000$, $b_V = 1600$, $e_U = 2.5$, $e_V = 1.5$, $d_U = d_V = 1$

Initial conditions: $p_0 = \delta\big((i_U = 0) \times (i_V = 0)\big)$

6.7.4. Results

Figure 6.8 presents numerical data for two runs, one carried out with $\Delta t = 10^{-2}$ and the other with $\Delta t = 10^{-3}$. We observe that the smaller time step leads to smaller ranks, and the reduced cost of solving an LSE at smaller ranks makes more than up for the increased number of LSEs that we have to solve in this case. In numbers, running the benchmark with $\Delta t = 10^{-3}$ takes 7.5h compared to 14h for $\Delta t = 10^{-2}$.

In [1], it was found that the largest ranks encountered with the TT format range up to over 90 and are thus significantly higher than what we report here. Nevertheless, the runtime observed there is lower, namely 2.8h. A possible explanation for this phenomenon is the different scaling of the TT-DMRG ($\mathcal{O}(r^3)$) and the HTR ALS(SD) ($\mathcal{O}(r^4)$) solver with respect to the rank r, but major contributions from other sources cannot be rule out such that this finding should not be overrated.

6.7.5. Parallel Scaling

For this particular problem, the ALS(SD) algorithm exhibits very poor parallel scaling as shown in Table 6.3. To some extent, this can be explained by noting that the ranks are heavily skewed towards the root (see Figure 6.10b) where the potential for parallelization is low. The by far heaviest vertices, however, appear on level $\ell = 2$ with $\#\,\mathrm{level}_\ell(T_D) = 2$ such that the algorithm should scale fairly reasonably at least for $p = 2$. The reason why this is not the case is shown in Figure 6.7: the local LSEs in the left subtree require many more GMRES iterations than the ones in the right such that the main computational burden is assigned to very few vertices all located on the same branch.

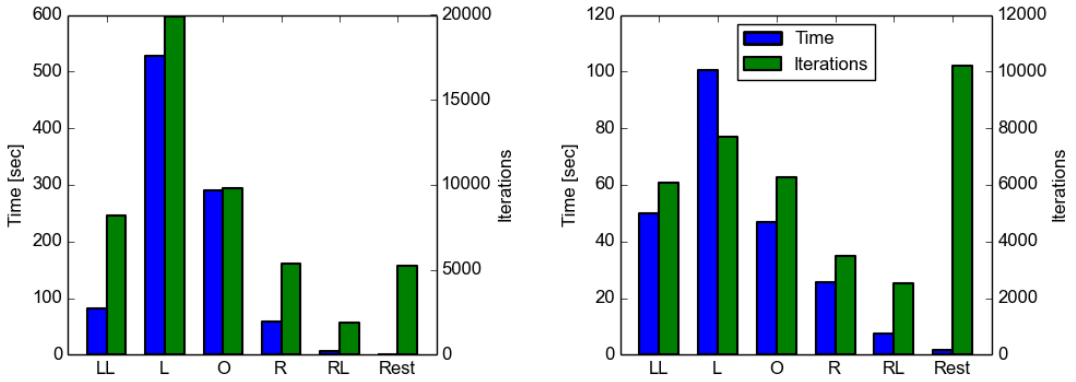
	$p = 1$	$p = 2$	$p = 4$
Default	982 sec.	1.09x	0.89x
Round-robin		1.12x	1.09x

(a) $\Delta t = 10^{-2}$ ($M = 20$).

	$p = 1$	$p = 2$	$p = 4$
Default	320 sec.	1.16x	0.97x
Round-robin		1.16x	1.25x

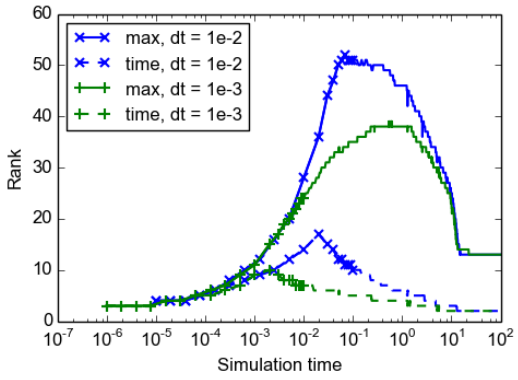
(b) $\Delta t = 10^{-3}$ ($M = 110$).

Table 6.3.: Toggle switch. Serial runtime up to $t = 10^{-1}$ for $p = 1$ processor and parallel speedup for $p > 1$ processors.

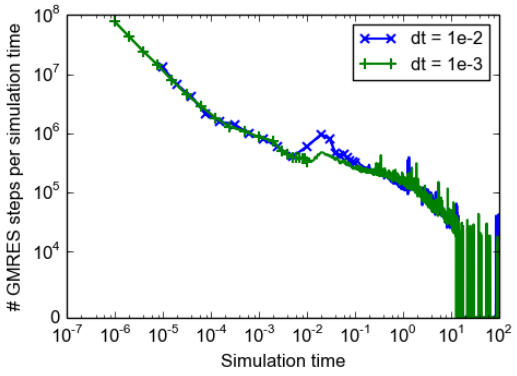


(a) $\Delta t = 10^{-2}$ ($M = 20$ time steps). (b) $\Delta t = 10^{-3}$ ($M = 110$ time steps).

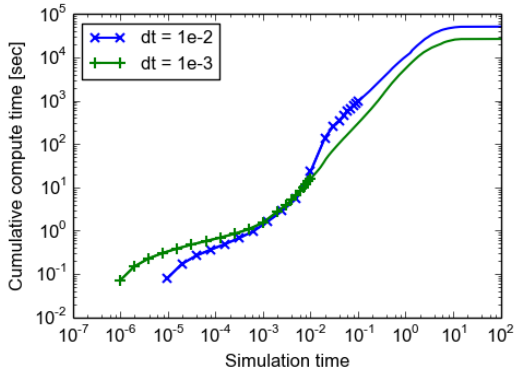
Figure 6.7.: Toggle switch. Runtime and number of iterations of the GMRES algorithm up to $t = 10^{-1}$ separated by the vertex on which they are spent. "O" denotes the root vertex of the physical dimensions, and each "L"/"R" encodes making one step to the Left/Right starting from "O".



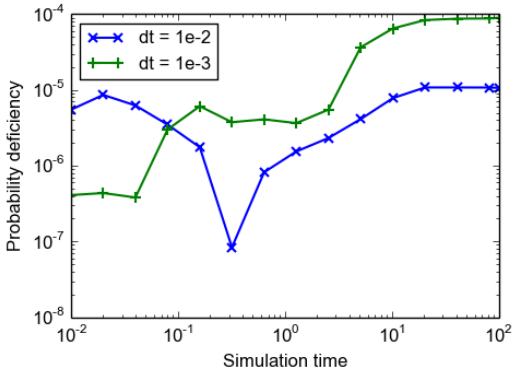
(a) Ranks of $p^{(m)}$



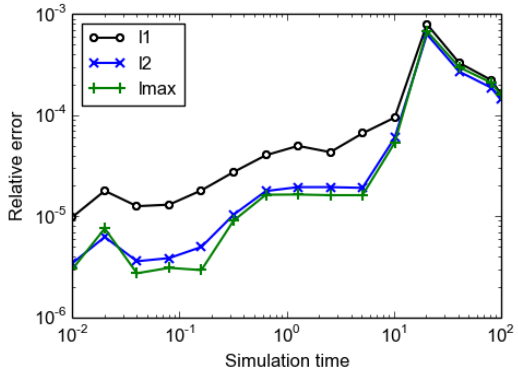
(b) GMRES steps per simulation time



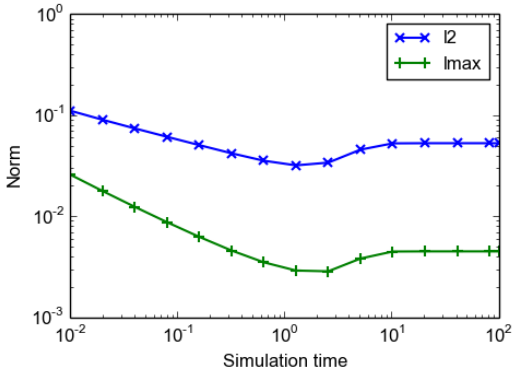
(c) Cumulative compute time



(d) Probability deficiency

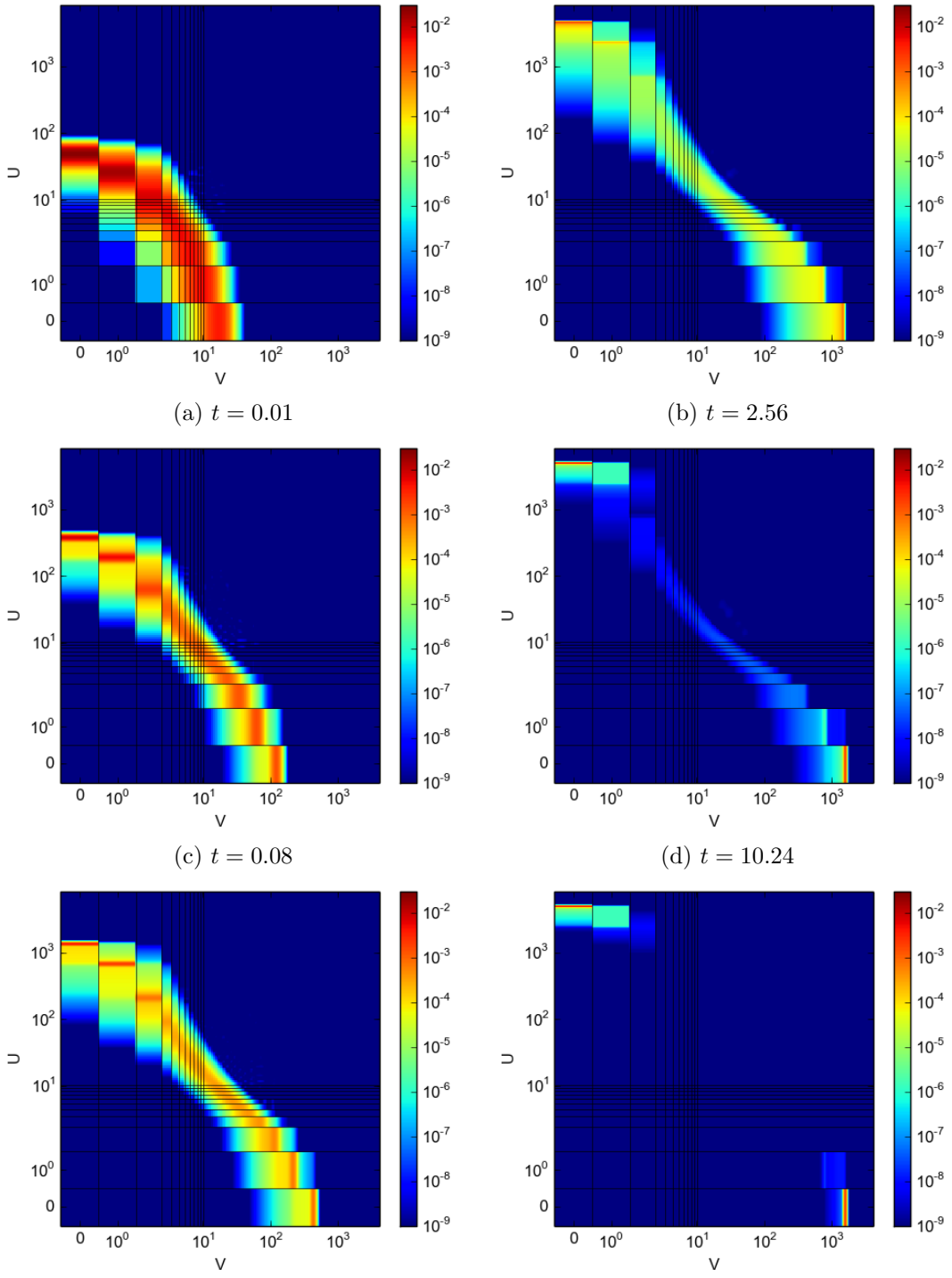


(e) Relative error



(f) Norm of $p(t)$

Figure 6.8.: Toggle switch. Various quantities as a function of the simulation time. In (a), "time rank" denotes the rank on the edge separating the time from the other dimensions. In (b), we show $\frac{n(m)}{\Delta t(m)}$ where $n(m)$ denotes the number of GMRES iterations carried out and $\Delta t(m) := t_m - t_{m-1}$ the interval length in the mth time step. In (e), the error is estimated as the difference between the solutions for $\Delta t = 10^{-2}$ and $\Delta t = 10^{-3}$.



(a) $t = 0.01$

(b) $t = 2.56$

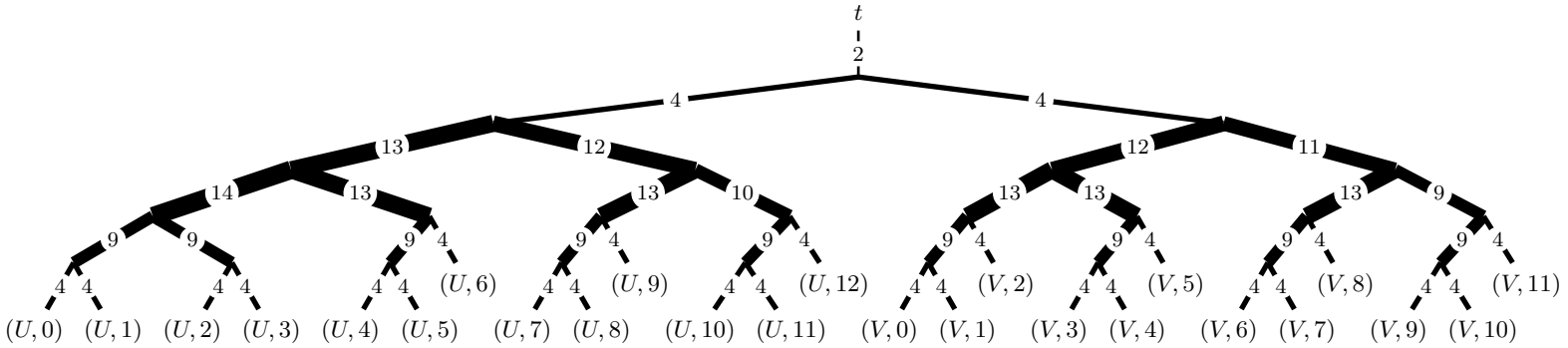
(c) $t = 0.08$

(d) $t = 10.24$

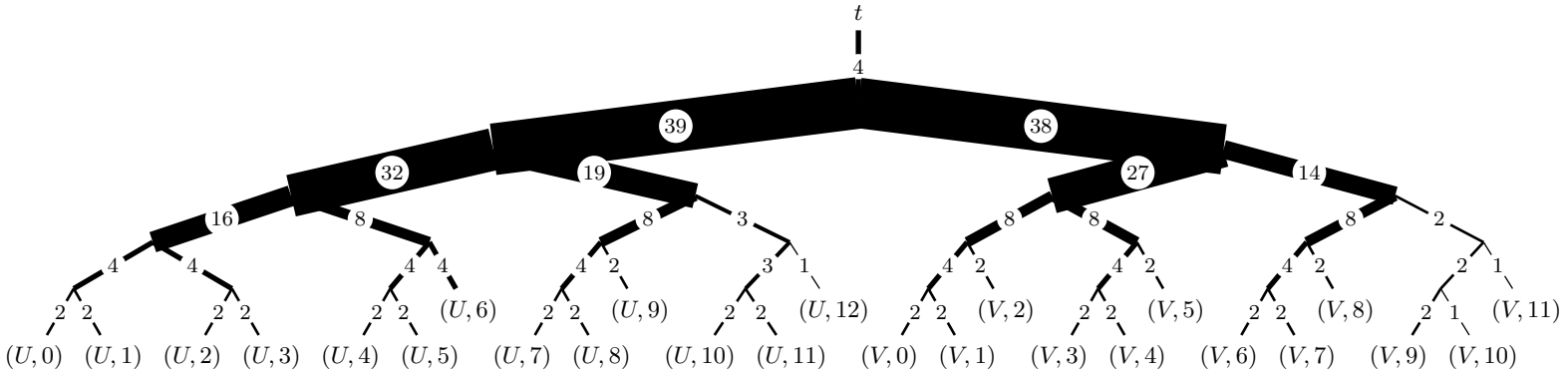
(e) $t = 0.32$

(f) $t = 100$

Figure 6.9.: Toggle switch. Evolution of the probability density function. Grid lines are omitted for copy numbers larger 10. We show $\max\{p(t), 10^{-9}\}$, where $p(t)$ is the numerical solution obtained with $\Delta t = 10^{-3}$.



(a) Time-stepping operator $D_t \mathbb{I}_D - M_t A_D$.



(b) Coefficient tensor $p^{(m)}$.

Figure 6.10.: Toggle switch. Ranks of the time-stepping operator $D_t \mathbb{I}_D - M_t A_D$ (top) and of the coefficient tensor $p^{(m)}$ for $\Delta t = 10^{-3}$ and $m = 610$ ($t_{610} = 0.6$).

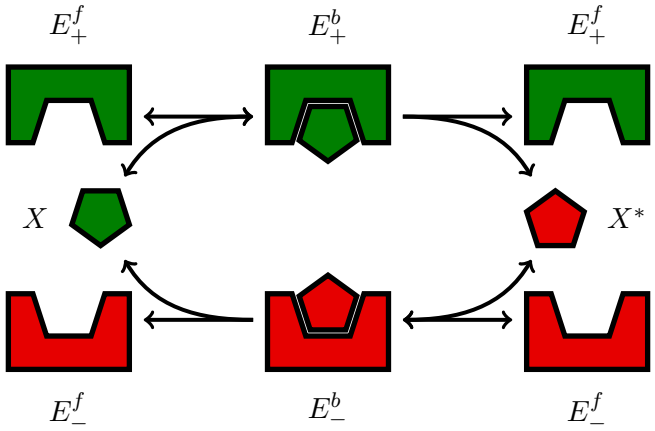


Figure 6.11.: Enzymatic futile cycle. Figure taken from [1].

6.8. Enzymatic Futile Cycle

Futile cycles (also known as substrate cycles) are reaction networks resulting only in the dissipation of heat but no net mass flux [41]. Reaction networks of this type are a common motif in nature [42], and despite the pejorative name it is assumed that this seemingly wasteful mechanism can serve a number of different purposes, see [41] and the references therein. To illustrate one such purpose, we consider an abstract futile cycle network taken from [43]. This model involves two *substrate species* X, X^* which may be converted between each other with the help of *forward* and *backwards enzymes* E_+ and E_-, respectively. We thus have the six species $D = \{X, X^*, E_+^f, E_+^b, E_-^f, E_-^b\}$ (the superscript f and b distinguish between *free* and *bound* enzymes) and the six reactions

$$\begin{aligned} X + E_+^f &\underset{k_2^+}{\overset{k_1^+}{\rightleftharpoons}} E_+^b, \qquad E_+^b \xrightarrow{k_3^+} X^* + E_+^f, \\ X^* + E_-^f &\underset{k_2^-}{\overset{k_1^-}{\rightleftharpoons}} E_-^b, \qquad E_-^b \xrightarrow{k_3^-} X + E_-^f. \end{aligned} \tag{6.16}$$

A schematic view of the model is given in Figure 6.11.

Interpreting the total forward enzyme count $i_{E_+^{tot}} := i_{E_+^f} + i_{E_+^b}$ as input and the equilibrium copy number i_X of one of the substrate species as output, it has been shown in [43] that this system maps a continuously varying input signal to an almost binary output, assuming suitably chosen reaction parameters. This step-function-like behaviour is further enhanced if the input $i_{E_+^{tot}}$ is subject to noise, and one can even tune the system to become bistable with a characteristic frequency of oscillation between the two steady states by choosing the probability distribution of $i_{E_+^{tot}}$ appropriately. The binarity of the output is proposed as an effective control mechanism for cell functions [44], and the bistability could be used as an additional communication channel for signaling [43].

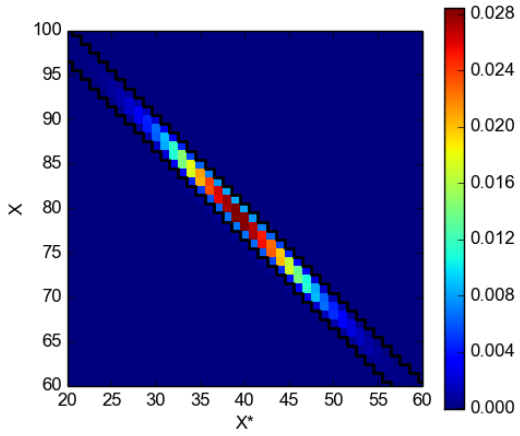


Figure 6.12.: Futile cycle. Marginal probability density (summed over the enzyme modes) at $t = 10^{-2}$. The black lines delimit the states reachable from the initial conditions.

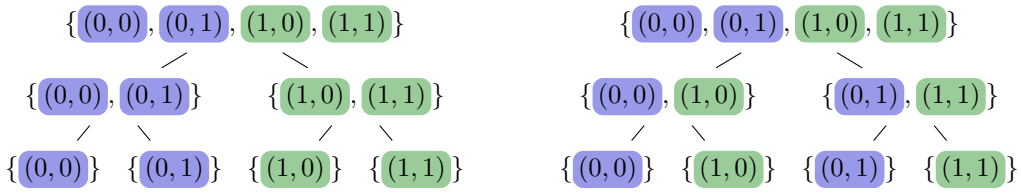
6.8.1. Dimension Partition Tree

Our aim here is to show that the tensor-network-based approach allows for the efficient simulation of futile cycle networks, and investigating the stochastic effects mentioned in the previous paragraph lies beyond the scope of this thesis. We therefore consider a model where the total copy numbers of both enzymes as well as the substrate are constant such that we have the three conservation relations

$$\begin{aligned} i_{E_+^{tot}} &:= i_{E_+^f} + i_{E_+^b} = const, \\ i_{E_-^{tot}} &:= i_{E_-^f} + i_{E_-^b} = const, \\ i_{X_{tot}} &:= i_X + i_{X^*} + i_{E_+^b} + i_{E_-^b} = const. \end{aligned} \tag{6.17}$$

We use the first two relations to reduce the numbers of unknowns from six to four by keeping track of only the free enzyme counts $i_{E_\pm^f}$ and obtaining the bound enzyme counts through $i_{E_\pm^b} = i_{E_\pm^{tot}} - i_{E_\pm^f}$ when needed. We emphasize, however, that we only do so because this simplifies the following discussion as well as the implementation of the CME operator. For a suitable dimension partition tree, tensor network ansätze readily exploit such structure without additional input from the user, as we show next by means of the last conservation relation.

An immediate consequence of the conservation of substrate species (6.17) is that any slice $p(t, i_{E_+^f} \times i_{E_-^f})$ of the probability density function has at most $i_{X_{tot}} - i_{E_+^f} - i_{E_-^f}$ nonzero entries which are located along one of the anti-(off-)diagonals. See also Figure 6.12, where we delimit these anti-diagonals by black lines for the parameter values specified below. Due to this diagonal structure, the optimal rank of separating the substrate modes, $r := \mathrm{rank}_X \left(p(t, i_{E_+^f} \times i_{E_-^f}) \right)$, is exactly equal to the number of nonzero



(a) Natural. (b) Transposed.

Figure 6.13.: Natural versus transposed dimension partition tree.

entries in this slice, and clearly this number provides a lower bound on $\operatorname{rank}_X(p(t))$. Physically, r measures the spread of the probability density function (we have e.g. $r \approx 40$ in Figure 6.12) and in general we cannot expect this quantity to be small. We therefore conclude that the X-mode (or, likewise, the X^*-mode) is not well separable from the remaining modes, and the natural dimension partition tree from Section 6.6.1 will not lead to small ranks. This has been demonstrated experimentally in [1].

The authors of [1] proposed to *transpose* the quantized modes instead, which means to separate the virtual before the physical modes as illustrated in Figure 6.13. When applied to $d = 2$ physical modes, the resulting representation is almost equivalent to the HTR for linear operators, the only difference being that we do not separate the squared modes of operators while we do separate the modes (k, j) with the same virtual index j under transposition. From Appendices A.3 to A.5 it follows that such a transposed representation leads to almost optimal compression in the sense that the ranks of the diagonal tensor are at most twice the ranks of the tensor generating it, which has been demonstrated in [1, S1.2] by means of an example. We thus use the transposed dimension partition tree shown in Figure 6.16 for this problem.

6.8.2. Obtaining the CME Operator

The CME operator for (6.16) reads

$$\begin{aligned} A_D = {} & k_1^+ \left(\uparrow_X \uparrow_{E_+^f} - \mathbb{I}\right) C_X C_{E_+^f} + k_2^+ \left(\downarrow_X \downarrow_{E_+^f} - \mathbb{I}\right) C'_{E_+^f} + k_3^+ \left(\downarrow_{X^*} \downarrow_{E_+^f} - \mathbb{I}\right) C'_{E_+^f} + \dots \\ & k_1^- \left(\uparrow_{X^*} \uparrow_{E_-^f} - \mathbb{I}\right) C_{X^*} C_{E_-^f} + k_2^- \left(\downarrow_{X^*} \downarrow_{E_-^f} - \mathbb{I}\right) C'_{E_-^f} + k_3^- \left(\downarrow_X \downarrow_{E_-^f} - \mathbb{I}\right) C'_{E_-^f} \end{aligned}$$

where for brevity we left out the subscripts of the identity operator $\mathbb{I}_M$, $M \subseteq D$. We assemble this operator in three steps:

- Assemble each factor $O_k \mathbb{I}_{D\setminus\{k\}}$ with $k \in D$ and $O \in \{\uparrow, \Downarrow, C, C'\}$ separately based on the dimension partition tree from Figure 6.16.
- Evaluate the above expression as the product and sum of operators in the HTR.
- Truncate A_D to remove excess ranks.

Since A_D involves only tensors of finite ranks, the last step can be carried out without incurring any error (apart from numerical noise). The final ranks of the time-stepping operator are shown in Figure 6.16a.

	$p = 1$	$p = 2$	$p = 3$	$p = 4$
Default	270 sec.	1.04x	0.93x	1.13x
Round-robin		1.03x	1.17x	1.16x

Table 6.4.: Futile cycle. Serial runtime for $p = 1$ processor and parallel speedup for $p > 1$ processors.

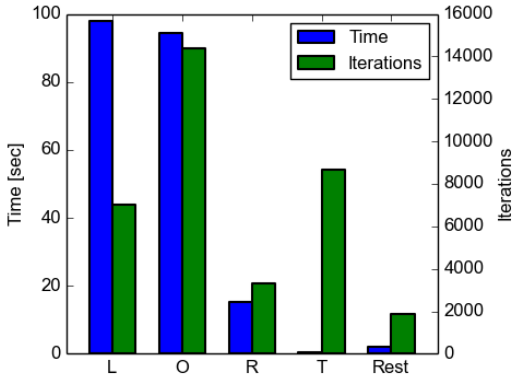


Figure 6.14.: Futile cycle. Runtime and number of iterations of the GMRES algorithm separated by the vertex on which they are spent. "O" denotes the root vertex of the physical dimensions, and "L", "R" its Left and Right child, respectively. "T" denotes the time vertex.

6.8.3. Parameters

Time-stepping: $\Delta t = 5 \cdot 10^{-4}$, $T_1 = 0.3$, $T = 1$ ($M = 1332$ time steps), $n_t = 10$

FSP: $l_k = 7$, $n_{(k,j)} = 2$ for $k \in \{X, X^*\}$, $(k, j) \in (k, [l_k])$,
$n_{E_+^f} = n_{E_-^f} = 3$ (not quantized)

Reaction: $k_1^+ = 40$, $k_2^+ = 10'000$, $k_3^+ = 10'000$,
$k_1^- = 200$, $k_2^- = 100$, $k_3^- = 5000$

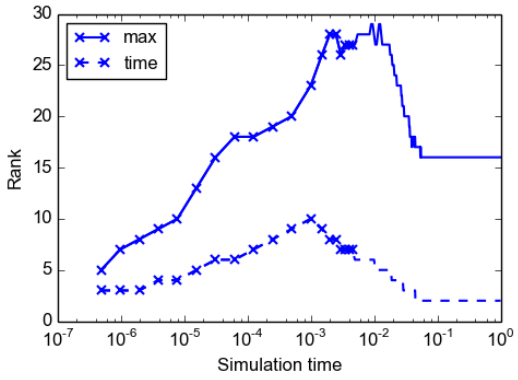
Initial conditions: $p_0 = \delta\left((i_X = 30) \times (i_{X^*} = 90) \times (i_{E_+^f} = 2) \times (i_{E_-^f} = 2)\right)$

6.8.4. Results

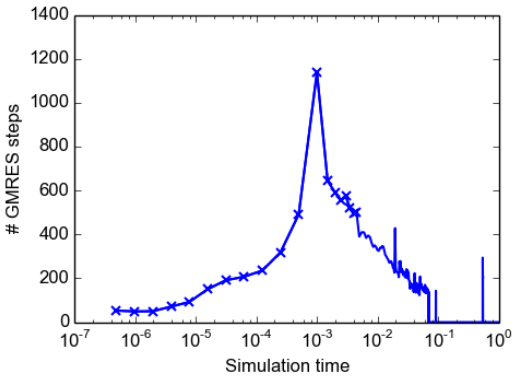
The numerical results are presented in Figure 6.15. We again note that the HTR delivers smaller ranks than the TT format (maximal rank of ~ 30 compared to ~ 50), and this time also the runtime of the HTR-based approach is smaller (5 minutes vs. 62 minutes).

6.8.5. Parallel Scaling

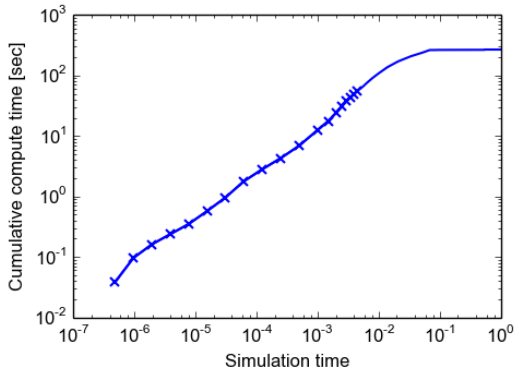
As shown in Table 6.4 and Figure 6.14, the parallel scaling is again poor for this example due to the same reasons as in Section 6.7.5.



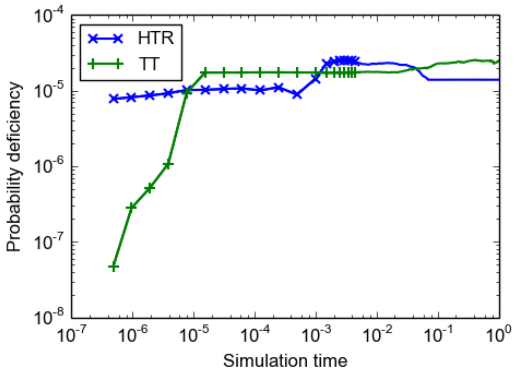
(a) Ranks of $p^{(m)}$



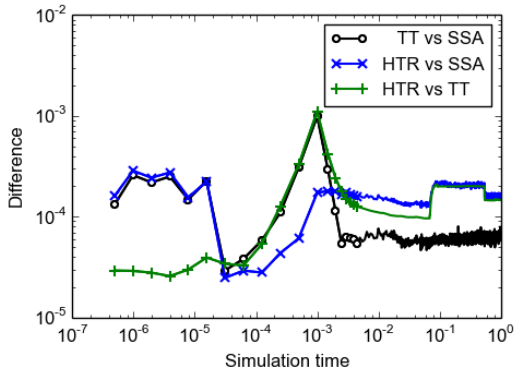
(b) Number of GMRES steps



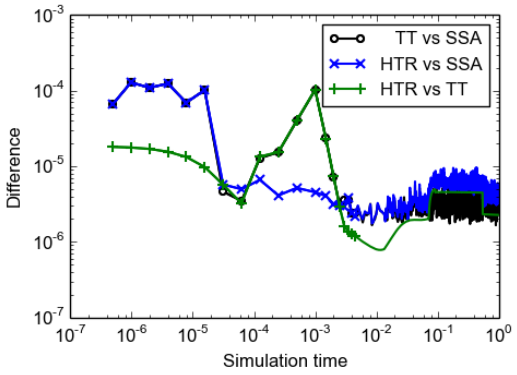
(c) Cumulative compute time



(d) Probability deficiency

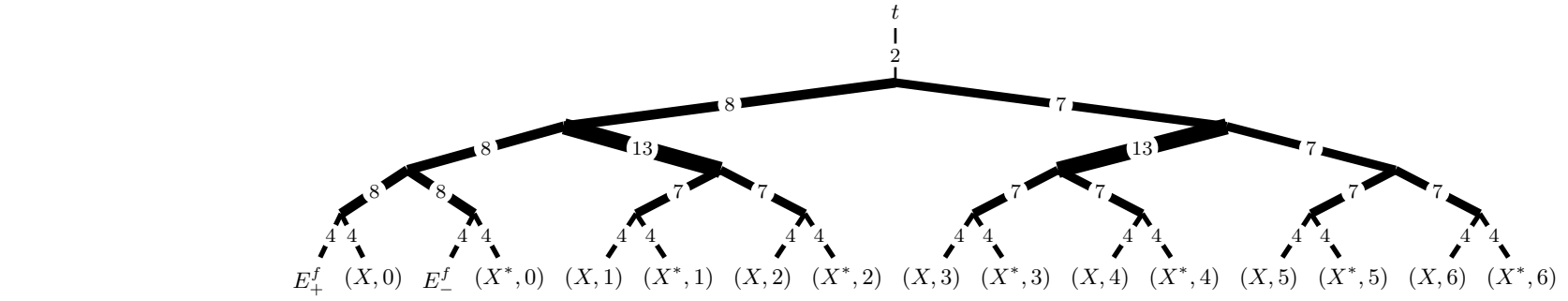


(e) Difference in ℓ_1-norm.

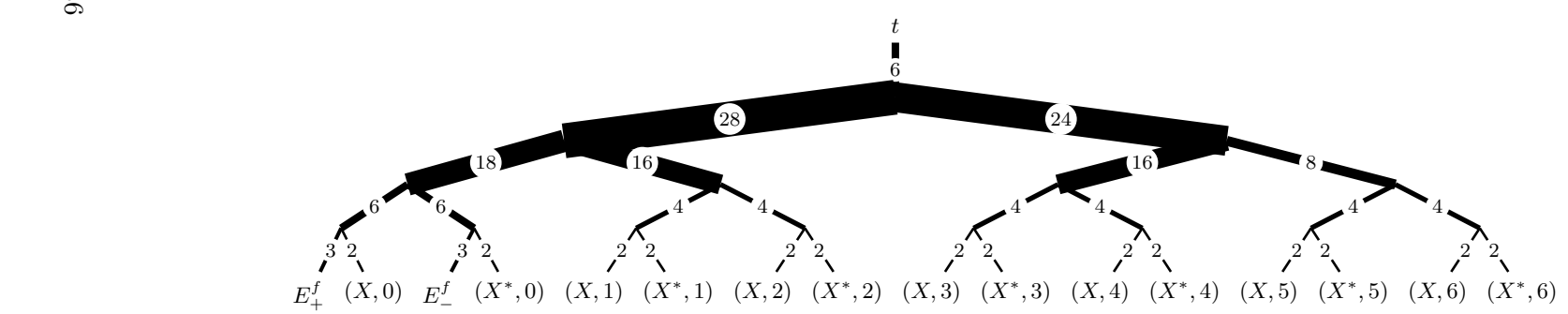


(f) Difference in $\ell_{\max}$-norm

Figure 6.15.: Futile cycle. Various quantities as a function of the simulation time. In (a), "time rank" denotes the rank of the edge separating the time from the other dimensions. In (d-f), "HTR" denotes the solution presented here and "TT" and "SSA" the respective solutions from [1].



(a) Time-stepping operator $D_t\,\mathbb{I}_D - M_t\,A_D$.



(b) Coefficient tensor $p^{(m)}$.

Figure 6.16.: Futile cycle. Ranks of the time-stepping operator $D_t\,\mathbb{I}_D - M_t\,A_D$ (top) and of the coefficient tensor $p^{(m)}$, $m = 30$ ($t_{30} = 0.1$).

7. Conclusion

We described the HTR ALS algorithm and showed how its complexity can be reduced through clever evaluation of the mode products appearing throughout the algorithm. We then highlighted the modifications required for parallelization and indicated how they must be implemented to obtain a numerically stable algorithm. At the example of the high-dimensional Poisson equation, we demonstrated that these modifications have virtually no impact on the numerical properties of the algorithm but allow for a parallel scaling which we believe to be only limited by the problem size. All of the presented algorithms have been implemented in an MPI-based C++ framework which, to the authors' knowledge, is the first software package for massively parallel computations on tensor networks.

In the second part of this thesis, we applied the new solver to the chemical master equation, a fundamental but very high-dimensional problem from systems biology, by reproducing the numerical experiments from [1]. We found that some care must be taken when scaling this ansatz to large dimensions, i.e. $d \geq 10$, and we pointed out in Sections 6.7 and 6.8 why our parallel algorithm may not scale well in certain circumstances. There is reason to assume, however, that the effects found there are an artefact of the essentially two-dimensional nature of the problems and may become irrelevant for larger d.

Finally, as a side product of this work we presented a formalism to easily derive HTR expressions and prove their optimality.

7.1. Acknowledgements

I would like to thank Vladimir Kazeev for his advice regarding tensor network techniques as well as for providing the reference solutions from [1] (see Tables 6.1 and 6.2 and Figure 6.15). I am further indebted to Robert Gantner for his help with this manuscript and to Prof. Christoph Schwab for guiding my work into the right direction.

I acknowledge the use of the following software packages and would like to express my gratitude towards their respective authors:

- The Eigen C++ template library for linear algebra [45]
- NETCON: MATLAB script for the easy determination of optimal contraction sequences for tensor networks [27]
- ETE: A Python environment for tree exploration [46]
- `tree.hh`: An STL-like C++ tree class [47]

- TT-Toolbox: MATLAB toolbox for the TT format [48]
- `htucker`: MATLAB toolbox for the HTR [49]

The last two toolboxes were not explicitly used but provided an important reference for our own code.

A. Splittings for Common Tensors

We define a number of tensors and derive splitting relations for them. In this chapter as well as in the remainder of this report, we assume the following convention.

Definition A.0.1 (Zero-Padding Convention)**.** Let $x \in \mathbb{K}^{\times_{k\in D}[n_k]}$ be a tensor. We define the expression $x\,(i_D)$ for all $i_D \in \mathbb{Z}^D$ through

$$x\,(i_D) := \begin{cases} x\,(i_D) & \text{if } i_D \in \bigtimes_{k\in D}[n_k] \\ 0 & \text{otherwise} \end{cases}.$$

A.1. All-Ones Tensor

Definition A.1.1 (All-Ones Tensor)**.** Let D be a finite mode set with associated index sets I_D. The *all-ones tensor* 1_D is the tensor in $\mathbb{K}^{\times_{k\in D} I_k}$ whose entries are given by $1_D(i_D) := 1$.

Theorem A.1.2. *Let L, R be two disjoint finite mode sets. We have $1_{L\cup R} = 1_L\, 1_R$.*

Proof. $$1_{L\cup R}\,(i_{L\cup R}) = 1 = 1 \cdot 1 = 1_L(i_L) \cdot 1_R(i_R).$$ □

A.2. Delta Tensor

Definition A.2.1 (Delta Tensor)**.** Let D be a finite mode set with associated index sets I_D, and let $i^\star_D \in \bigtimes_{k\in D} I_k$. The *delta tensor* $\delta\,(i_D = i^\star_D)$ is the tensor in $\mathbb{K}^{\times_{k\in D} I_k}$ whose entries are given by

$$\delta\,(i_D = i^\star_D)\,(i_D) := \begin{cases} 1 & \text{if } i_D = i^\star_D \\ 0 & \text{otherwise} \end{cases}.$$

Theorem A.2.2. *Let L, R be two disjoint finite mode sets and $i^\star_{L\cup R} \in \bigtimes_{k\in L\cup R} I_k$. We have*

$$\delta\,(i_{L\cup R} = i^\star_{L\cup R}) = \delta\,(i_L = i^\star_L)\ \delta\,(i_R = i^\star_R)\,.$$

Proof. The right-hand side is one if and only if $i_L = i^\star_L$ and $i_R = i^\star_R$, which is equivalent to $i_{L\cup R} = i^\star_{L\cup R}$. □

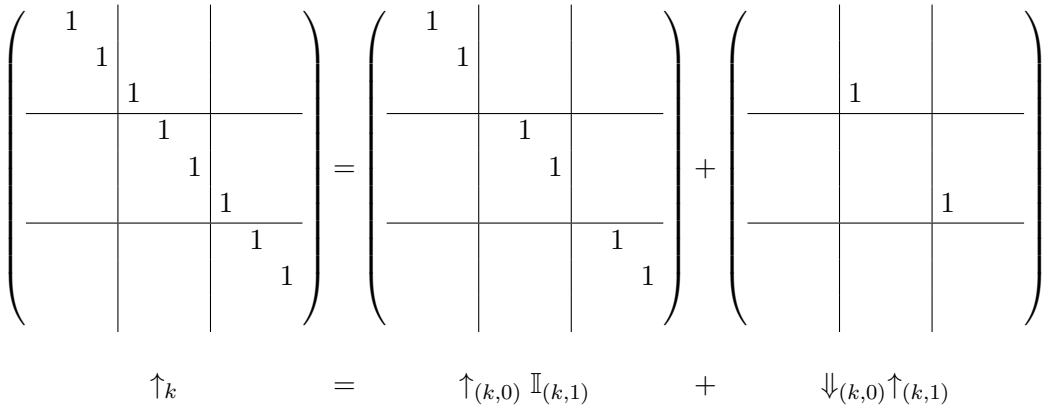


Figure A.1.: Illustration of the splitting $\uparrow_k = \uparrow_{(k,0)} \mathbb{I}_{(k,1)} + \Downarrow_{(k,0)} \uparrow_{(k,1)}$ for $n_k = 9$, $n_{(k,0)} = 3$, $n_{(k,1)} = 3$.

A.3. Shift Operator

Definition A.3.1 (Shift Operator)**.** Let k be a mode, $n_k \in \mathbb{N}$ its mode size and $s \in \mathbb{Z}$. We define the *shift operator* $S_k^{(s)} \in \mathbb{K}^{[n_k^2]}$ through its action on a tensor $x \in \mathbb{K}^{[n_k]}$ which is given by

$$\left(S_k^{(s)} x\right)(i_k) := x(i_k + s).$$

Additionally, we define $\uparrow_k = S_k^{(1)}$, $\downarrow_k = S_k^{(-1)}$, $\Uparrow_k = S_k^{(n-1)}$, $\Downarrow_k = S_k^{(-n+1)}$.

Theorem A.3.2. *Let $(k, [2])$ be a quantized mode, $s_0 \in \pm[n_{(k,0)}]$, $s_1 \in \pm[n_{(k,1)}]$ and $s := s_0 + n_{(k,0)} s_1$. We have*

$$S_k^{(s)} = S_{(k,0)}^{(s_0)} S_{(k,1)}^{(s_1)} + S_{(k,0)}^{(s_0 \mp n_{(k,0)})} S_{(k,1)}^{(s_1 \pm 1)}.$$

Proof. For brevity, we denote (k, j) by only j, $j = 0, 1$. By Definition 2.7.2, we have

$$\begin{aligned} i_k + s &= i_0 + s_0 \qquad + n_0(i_1 + s_1 \qquad) \\ &= i_0 + s_0 \mp n_0 + n_0(i_1 + s_1 \pm 1). \end{aligned}$$

Applying the zero-padding convention, we obtain with $i_0 \in [n_0]$, $i_1 \in [n_1]$

$$x((i_0 + s_0 \qquad) \times (i_1 + s_1 \qquad)) = \begin{cases} x(i_k + s) & \text{if } i_0 + s_0 \in [n_0] \\ 0 & \text{if } i_0 + s_0 \notin [n_0] \end{cases},$$

$$x((i_0 + s_0 \mp n_0) \times (i_1 + s_1 \pm 1)) = \begin{cases} 0 & \text{if } i_0 + s_0 \in [n_0] \\ x(i_k + s) & \text{if } i_0 + s_0 \notin [n_0] \end{cases}.$$

The two cases are complementary and we obtain

$$\begin{aligned} x(i_k + s) = {} & x((i_0 + s_0 \qquad) \times (i_1 + s_1 \qquad)) + \dots \\ & x((i_0 + s_0 \mp n_0) \times (i_1 + s_1 \pm 1)). \end{aligned}$$

□

Corollary A.3.3. *Let $(k, [2])$ be a quantized mode. We have*

$$\uparrow_k = \uparrow_{(k,0)} \mathbb{I}_{(k,1)} + \Downarrow_{(k,0)} \uparrow_{(k,1)}, \qquad \Uparrow_k = \Uparrow_{(k,0)} \Uparrow_{(k,1)},$$
$$\downarrow_k = \downarrow_{(k,0)} \mathbb{I}_{(k,1)} + \Uparrow_{(k,0)} \downarrow_{(k,1)}, \qquad \Downarrow_k = \Downarrow_{(k,0)} \Downarrow_{(k,1)} .$$

Remark A.3.4. Analogous splittings for the shift operator in the QTT format have been derived in [50].

A.4. Diagonalization

Definition A.4.1 (Diagonalization)**.** Let $x \in \mathbb{K}^{\times_{k \in D} I_k}$ be a tensor. We define its *diagonalization* $\operatorname{diag}(x) \in \mathbb{K}^{\times_{k \in D} I_k^2}$ through its action on another tensor $y \in \mathbb{K}^{\times_{k \in D} I_k}$ given by

$$\big(\operatorname{diag}(x)\, y\big)(i_D) = x(i_D)\, y(i_D).$$

Theorem A.4.2 (Remark 13.10 in [22])**.** *Let $z \in \mathbb{K}^{\times_{k \in D} I_k}$, $x \in \mathbb{K}^{[r_M] \times (\times_{k \in M} I_k)}$ and $y \in \mathbb{K}^{[r_M] \times (\times_{k \in D \setminus M} I_k)}$ with $M \subset D$ and $r_M \in \mathbb{N}$ be tensors such that $z = xy$ is a splitting of z. Then, we have*

$$\operatorname{diag}(z) = \sum_{i_{\{M\}} \in [r_M]} \operatorname{diag}\big(x(i_{\{M\}})\big)\, \operatorname{diag}\big(y(i_{\{M\}})\big).$$

Proof. Let $w \in \mathbb{K}^{\times_{k \in D} I_k}$ be arbitrary. Then,

$$\begin{aligned}\big(\operatorname{diag}(z)\, w\big)(i_D) &= \sum_{i_{\{M\}} \in [r_M]} x(i_{\{M\}} \times i_M)\, y(i_{\{M\}} \times i_{D \setminus M})\, w(i_D) \\ &= \left(\sum_{i_{\{M\}} \in [r_M]} \operatorname{diag}\big(x(i_{\{M\}})\big)\, \operatorname{diag}\big(y(i_{\{M\}})\big)\, w \right)(i_D). \qquad \square\end{aligned}$$

A.5. Flipping Operator

Definition A.5.1 (Flipping Operator)**.** Let k be a mode and $n_k \in \mathbb{N}$ its mode size. We define the *flipping operator* $\updownarrow_k \in \mathbb{K}^{[n_k^2]}$ through its action on a tensor $x \in \mathbb{K}^{[n_k]}$ which is given by

$$(\updownarrow_k x)\,(i_k) := x(n_k - i_k - 1).$$

Theorem A.5.2. *Let $(k, [2])$ be a quantized mode with virtual mode sizes $n_{(k,[2])} \in \mathbb{N}^{(k,[2])}$. We have*

$$\updownarrow_k = \updownarrow_{(k,0)} \updownarrow_{(k,1)} .$$

Proof. We abbreviate (k, j) by only j, $j = 0, 1$. By Definition 2.7.2, it holds

$$\begin{aligned}(\updownarrow_k x)\,(i_k) &= x(n_0 n_1 - i_0 - n_0\, i_1 - 1)\\ &= x(n_0 - i_0 - 1 + n_0(n_1 - i_1 - 1))\\ &= (\updownarrow_0 \updownarrow_1 x)\,(i_0 \times i_1).\end{aligned}$$

□

A.6. Counting Tensors

Definition A.6.1 (Counting Tensors)**.** Let k be a mode and $n_k \in \mathbb{N}$ its mode size. The *counting tensor* c_k is the tensor in $\mathbb{K}^{[n_k]}$ whose entries are given by $c_k(i_k) := i_k$. Additionally, we define the *counting operator* $C_k := \mathrm{diag}(c)$ and the inverse counting tensor / operator $c'_k :=\updownarrow_k c_k$ and $C'_k = \mathrm{diag}(c'_k)$, respectively.

Theorem A.6.2. *Let $(k, [2])$ be a quantized mode with virtual mode sizes $n_{(k,[2])} \in \mathbb{N}^{(k,[2])}$. We have*

$$\begin{aligned}c_k &= c_{(k,0)}\, 1_{(k,1)} + n_{(k,0)}\, 1_{(k,0)}\, c_{(k,1)},\\ c'_k &= c'_{(k,0)}\, 1_{(k,1)} + n_{(k,0)}\, 1_{(k,0)}\, c'_{(k,1)}.\end{aligned}$$

Proof. We abbreviate (k, j) by only j, $j = 0, 1$. By Definition 2.7.2, it holds

$$c_k(i_k) = i_0 + n_0\, i_1 = c_0\, 1_1 + n_0\, 1_0\, c_1,$$

which proves the claim for the counting tensor. For the inverse counting tensor, we have

$$c'_k =\updownarrow_0 \updownarrow_1 (c_0 1_1 + n_0 1_0 c_1) =\updownarrow_0 c_0\, 1_1 + n_0\, 1_0 \updownarrow_1 c_1 = c'_0\, 1_1 + n_0\, 1_0\, c'_1.$$

□

Remark A.6.3. Theorem A.6.2 implies that any linear function can be represented in $\mathrm{HTR}(T_D, I, r)$ with $r_\alpha \leq 2$ for all $\alpha \in T_D$. Through recursive application of this theorem one can more generally show that any polynomial of order p has HTR ranks bounded by $p + 1$, which has already been proven in [19] and [51].

Bibliography

[1] Vladimir Kazeev et al. 'Direct Solution of the Chemical Master Equation Using Quantized Tensor Trains'. In: *PLoS Comput Biol* 10.3 (Mar. 2014), e1003359. URL: http://dx.doi.org/10.1371/journal.pcbi.1003359.

[2] Daniel T. Gillespie. 'Exact stochastic simulation of coupled chemical reactions'. In: *The Journal of Physical Chemistry* 81.25 (1977), pp. 2340–2361. URL: http://dx.doi.org/10.1021/j100540a008.

[3] I. V. Oseledets and E. E. Tyrtyshnikov. 'Breaking the curse of dimensionality, or how to use SVD in many dimensions'. In: *SIAM Journal on Scientific Computing* 31.5 (Oct. 2009), pp. 3744–3759. URL: http://epubs.siam.org/sisc/resource/1/sjoce3/v31/i5/p3744_s1.

[4] I. V. Oseledets. 'Tensor-Train Decomposition'. In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2295–2317.

[5] W. Hackbusch and S. Kühn. 'A New Scheme for the Tensor Representation'. In: *Journal of Fourier Analysis and Applications* 15.5 (2009). 10.1007/s00041-009-9094-9, pp. 706–722. URL: http://www.springerlink.com/content/t3747nk47m368g44/.

[6] Lars Grasedyck. 'Hierarchical singular value decomposition of tensors'. In: *SIAM J. Matrix Anal. Appl.* 31.4 (2009/10), pp. 2029–2054. URL: http://dx.doi.org/10.1137/090764189.

[7] R Hübener, V Nebendahl and W Dür. 'Concatenated tensor network states'. In: *New Journal of Physics* 12.2 (2010), p. 025004. URL: http://stacks.iop.org/1367-2630/12/i=2/a=025004.

[8] Steven R. White. 'Density-matrix algorithms for quantum renormalization groups'. In: *Phys. Rev. B* 48 (14 Oct. 1993), pp. 10345–10356. URL: http://link.aps.org/doi/10.1103/PhysRevB.48.10345.

[9] I. Oseledets and S. Dolgov. 'Solution of Linear Systems and Matrix Inversion in the TT-Format'. In: *SIAM Journal on Scientific Computing* 34.5 (2012), A2718–A2739. URL: http://dx.doi.org/10.1137/110833142.

[10] S. Holtz, T. Rohwedder and R. Schneider. 'The Alternating Linear Scheme for Tensor Optimization in the Tensor Train Format'. In: *SIAM Journal on Scientific Computing* 34.2 (2012), A683–A713. URL: http://dx.doi.org/10.1137/100818893.

[11] Simon Etter. 'MPI-Based Tensor Network Library'. Semester Thesis. ETH Zürich, May 2014. URL: www.sam.math.ethz.ch/teaching/termproj/etterSTH.pdf.

[12] Garnet Kin-Lic Chan. 'An algorithm for large scale density matrix renormalization group calculations'. In: *The Journal of Chemical Physics* 120.7 (2004), pp. 3172–3178. URL: http://scitation.aip.org/content/aip/journal/jcp/120/7/10.1063/1.1638734.

[13] G. Hager et al. 'Parallelization strategies for density matrix renormalization group algorithms on shared-memory systems'. In: *Journal of Computational Physics* 194.2 (2004), pp. 795–808. URL: http://www.sciencedirect.com/science/article/pii/S0021999103005084.

[14] Yuki Kurashige and Takeshi Yanai. 'High-performance ab initio density matrix renormalization group method: Applicability to large-scale multireference problems for metal compounds'. In: *The Journal of Chemical Physics* 130.23, 234114 (2009). URL: http://scitation.aip.org/content/aip/journal/jcp/130/23/10.1063/1.3152576.

[15] E. M. Stoudenmire and Steven R. White. 'Real-space parallel density matrix renormalization group'. In: *Phys. Rev. B* 87 (15 Apr. 2013), p. 155137. URL: http://link.aps.org/doi/10.1103/PhysRevB.87.155137.

[16] Daniel Kressner and Christine Tobler. 'Preconditioned low-rank methods for high-dimensional elliptic PDE eigenvalue problems'. In: *Comput. Methods Appl. Math.* 11.3 (2011), pp. 363–381. URL: http://dx.doi.org/10.2478/cmam-2011-0020.

[17] E. E. Tyrtyshnikov. 'Tensor approximations of matrices generated by asymptotically smooth functions'. In: *Sbornik: Mathematics* 194.5 (2003), pp. 941–954. URL: http://iopscience.iop.org/1064-5616/194/6/A09.

[18] I. Oseledets. 'Approximation of matrices with logarithmic number of parameters'. In: *Doklady Mathematics* 80.2 (Apr. 2009), pp. 653–654. URL: http://dx.doi.org/10.1134/S1064562409050056.

[19] Boris N. Khoromskij. '$O(d \log N)$-quantics approximation of N-d tensors in high-dimensional numerical modeling'. In: *Constr. Approx.* 34.2 (2011), pp. 257–280. URL: http://dx.doi.org/10.1007/s00365-011-9131-1.

[20] I. V. Oseledets. 'Approximation of $2^d \times 2^d$ matrices using tensor decomposition'. In: *SIAM Journal on Matrix Analysis and Applications* 31.4 (2010), pp. 2130–2145. URL: http://epubs.siam.org/doi/abs/10.1137/090757861.

[21] S. Dolgov and D. Savostyanov. 'Alternating Minimal Energy Methods for Linear Systems in Higher Dimensions'. In: *SIAM Journal on Scientific Computing* 36.5 (2014), A2248–A2271. URL: http://dx.doi.org/10.1137/140953289.

[22] Wolfgang Hackbusch. *Tensor spaces and numerical tensor calculus*. Vol. 42. Springer Series in Computational Mathematics. Springer, Heidelberg, 2012, pp. xxiv+500. URL: http://dx.doi.org/10.1007/978-3-642-28027-6.

[23] Lieven De Lathauwer, Bart De Moor and Joos Vandewalle. 'A multilinear singular value decomposition'. In: *SIAM J. Matrix Anal. Appl.* 21.4 (2000), 1253–1278 (electronic). URL: http://dx.doi.org/10.1137/S0895479896305696.

[24] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)* Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[25] C. L. Liu and J. W. Layland. 'Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment'. In: *J. ACM* 20.1 (Jan. 1973), pp. 46–61. URL: http://doi.acm.org/10.1145/321738.321743.

[26] Michael L. Dertouzos. 'Control Robotics: The Procedural Control of Physical Processes'. In: *IFIP Congress*. 1974, pp. 807–813.

[27] Robert N. C. Pfeifer, Jutho Haegeman and Frank Verstraete. 'Faster identification of optimal contraction sequences for tensor networks'. In: *Phys. Rev. E* 90 (3 Sept. 2014), p. 033315. URL: http://link.aps.org/doi/10.1103/PhysRevE.90.033315.

[28] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Second. Society for Industrial and Applied Mathematics, 2003. URL: http://epubs.siam.org/doi/abs/10.1137/1.9780898718003.

[29] Gene M. Amdahl. 'Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities'. In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. AFIPS '67 (Spring). Atlantic City, New Jersey: ACM, 1967, pp. 483–485. URL: http://doi.acm.org/10.1145/1465482.1465560.

[30] V. Kazeev and B. Khoromskij. 'Low-Rank Explicit QTT Representation of the Laplace Operator and Its Inverse'. In: *SIAM Journal on Matrix Analysis and Applications* 33.3 (2012), pp. 742–758. URL: http://dx.doi.org/10.1137/100820479.

[31] N.G. Van Kampen. *Stochastic Processes in Physics and Chemistry*. North-Holland Personal Library. Elsevier Science, 2011.

[32] Mukhtar Ullah and Olaf Wolkenhauer. *Stochastic approaches for systems biology*. Springer, New York, 2011, pp. xxxii+290. URL: http://dx.doi.org/10.1007/978-1-4614-0478-1.

[33] Brian Munsky and Mustafa Khammash. 'The finite state projection algorithm for the solution of the chemical master equation'. In: *The Journal of Chemical Physics* 124.4, 044104 (2006).

[34] D. Schötzau and C. Schwab. 'An *hp* a priori error analysis of the DG time-stepping method for initial value problems'. In: *Calcolo* 37.4 (2000), pp. 207–232. URL: http://dx.doi.org/10.1007/s100920070002.

[35] V. Kazeev, O. Reichmann and Ch. Schwab. *hp-DG-QTT solution of high-dimensional degenerate diffusion equations*. Tech. rep. 2012-11. Switzerland: Seminar for Applied Mathematics, ETH Zürich, 2012. URL: /sam_reports/reports_final/reports2012/2012-11.pdf.

[36] Tobias Jahnke and Wilhelm Huisinga. 'Solving the chemical master equation for monomolecular reaction systems analytically'. In: *Journal of Mathematical Biology* 54.1 (2007), pp. 1–26. URL: http://dx.doi.org/10.1007/s00285-006-0034-x.

[37] Ivan Oseledets. 'DMRG approach to fast linear algebra in the TT-format'. In: *Comput. Methods Appl. Math.* 11.3 (2011), pp. 382–393. URL: http://dx.doi.org/10.2478/cmam-2011-0021.

[38] Timothy S Gardner, Charles R Cantor and James J Collins. 'Construction of a genetic toggle switch in Escherichia coli'. In: *Nature* 403.6767 (2000), pp. 339–342.

[39] Brian Munsky et al. 'Stochastic modeling of the pap-pili epigenetic switch'. In: *Proc. FOSBE* (2005), pp. 145–148.

[40] Adam Arkin, John Ross and Harley H. McAdams. 'Stochastic Kinetic Analysis of Developmental Pathway Bifurcation in Phage λ-Infected Escherichia coli Cells'. In: *Genetics* 149.4 (1998), pp. 1633–1648. URL: http://www.genetics.org/content/149/4/1633.abstract.

[41] Jean-Charles Portais and Anne-Marie Delort. 'Carbohydrate cycling in micro-organisms: what can 13C-NMR tell us?' In: *FEMS Microbiology Reviews* 26.4 (2002), pp. 375–402. URL: http://dx.doi.org/10.1111/j.1574-6976.2002.tb00621.x.

[42] Jörg Schwender, John Ohlrogge and Yair Shachar-Hill. 'Understanding flux in plant metabolic networks'. In: *Current Opinion in Plant Biology* 7.3 (2004), pp. 309–317. URL: http://www.sciencedirect.com/science/article/pii/S1369526604000482.

[43] Michael Samoilov, Sergey Plyasunov and Adam P. Arkin. 'Stochastic amplification and signaling in enzymatic futile cycles through noise-induced bistability with oscillations'. In: *Proceedings of the National Academy of Sciences of the United States of America* 102.7 (2005), pp. 2310–2315. URL: http://www.pnas.org/content/102/7/2310.abstract.

[44] E.A. Newsholme, R.A.J. Challiss and B. Crabtree. 'Substrate cycles: their role in improving sensitivity in metabolic control'. In: *Trends in Biochemical Sciences* 9.6 (1984), pp. 277–280. URL: http://www.sciencedirect.com/science/article/pii/0968000484901658.

[45] Gaël Guennebaud, Benoît Jacob et al. *Eigen v3*. 2010. URL: http://eigen.tuxfamily.org.

[46] Jaime Huerta-Cepas, Joaquin Dopazo and Toni Gabaldon. 'ETE: a python Environment for Tree Exploration'. In: *BMC Bioinformatics* 11.1 (2010), p. 24. URL: http://www.biomedcentral.com/1471-2105/11/24.

[47] Kaspar Peeters. *tree.hh: an STL-like C++ tree class*. URL: http://tree.phi-sci.com.

[48] *TT-Toolbox*. URL: http://github.com/oseledets/TT-Toolbox.

[49] Daniel Kressner and Christine Tobler. *htucker - A MATLAB toolbox for tensors in hierarchical Tucker format*. Tech. rep. EPF Lausanne, Chair of Numerical Algorithms and High-Performance Computing, 2012. URL: http://anchp.epfl.ch/htucker.

[50] V. Kazeev, B. Khoromskij and E. Tyrtyshnikov. 'Multilevel Toeplitz Matrices Generated by Tensor-Structured Vectors and Convolution with Logarithmic Complexity'. In: *SIAM Journal on Scientific Computing* 35.3 (2013), A1511–A1536. URL: http://dx.doi.org/10.1137/110844830.

[51] L. Grasedyck. *Polynomial Approximation in Hierarchical Tucker Format by Vector–Tensorization*. Tech. rep. 308. RWTH Aachen, Institut für Geometrie und Praktische Mathematik, Apr. 2010. URL: http://www.igpm.rwth-aachen.de/forschung/preprints/308.