A factorization method for completely positive matrices

Patrick Groetzner∗ Mirjam Dür†

March 7, 2018

Abstract
A matrix $A$ is called completely positive, if there exists an entrywise nonnegative matrix $B$ such that $A = BB^T$. These matrices play a major role in combinatorial and quadratic optimization. In this paper, we develop a factorization algorithm which, for a given completely positive matrix $A$, computes the nonnegative factorization $BB^T$. We formulate this factorization problem as a nonconvex feasibility problem and develop a solution method based on alternating projections. A local convergence result can be shown for this algorithm. We also provide a heuristic extension which improves the numerical performance of the algorithm. Extensive numerical tests show that the factorization method is very fast in most instances.

Keywords: completely positive matrices, matrix factorization, semidefinite programming

Mathematical Subject Classification 2010: 90C22, 15A23, 15B48

1 Introduction
A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is called completely positive, if there exists an entrywise nonnegative matrix $B \in \mathbb{R}^{n \times r}$ such that $A = BB^T$. The set of all such matrices,

$$\mathcal{CP}_n := \{A \in \mathbb{R}^{n \times n} \mid A = BB^T \text{ where } B \in \mathbb{R}^{n \times r}, B \geq 0\} = \text{conv}\{xx^T \mid x \in \mathbb{R}_{+}^n\},$$

is a proper cone whose extreme rays are the rank-one matrices $xx^T$ with $x \in \mathbb{R}_{+}^n$, cf. [3].

Completely positive matrices have received a lot of attention in the area of quadratic and binary optimization, as it has been shown that many combinatorial and nonconvex quadratic problems can be formulated as linear problems over $\mathcal{CP}_n$. For surveys on this area we refer to [7, 10, 19]. As an example, consider the problem of computing the stability number $\alpha$ of a graph $G$ on $n$ nodes. De Klerk and Pasechnik [12] showed that $\alpha$ is the solution of a maximization problem over $\mathcal{CP}_n$:

$$\alpha = \max\{\langle E, X \rangle \mid \langle A + I, X \rangle = 1, X \in \mathcal{CP}_n\},$$

(1)

where $A$ is the adjacency matrix of $G$ and $E$ is the all-ones matrix.

∗University of Trier, 54290 Trier, Germany. Email: groetzner@uni-trier.de
†University of Augsburg, 86135 Augsburg, Germany. Email: mirjam.duer@math.uni-augsburg.de
An optimal solution $X^*$ of (1) also contains information about the maximal stable set: if $X^*$ is rank 1, i.e., $X^* = x^* (x^*)^T$, then $\text{supp}(x^*)$ is the unique stable set in $G$. If rank $X^* > 1$, then $X^*$ can be factorized as $X^* = \sum_{i=1}^r x_i x_i^T$, and $\text{supp}(x_i)$ is a maximal stable set for each $i$.

This motivates the question discussed in this paper: given a completely positive matrix $A$, how can we find a factorization $A = BB^T$? Finding such a factorization would not only help to recover optimal solutions in combinatorial optimization problems, it also provides a certificate that $A \in \text{CP}_n$, a membership question which is known to be NP-hard, see [15]. Finding a factorization $A = BB^T$ where $B$ is entrywise strictly positive would even prove that $A \in \text{int CP}_n$ (cf. [13]), a property that may be useful to illustrate strict feasibility of an optimization problem.

Factorizing completely positive matrices can also be seen as the symmetric case of nonnegative matrix factorization, albeit without the low rank constraint. This is related to data clustering, particularly Kernel K-means clustering and Laplacian-based spectral clustering, as discussed in [16].

Moreover, the need to factorize completely positive matrices arises in statistics in the area of multivariate extremes, cf. [11] where it is shown how tail dependence of a multivariate regularly-varying random vector can be summarized in a so called tail pairwise dependence matrix $\Sigma$ of pairwise dependence metrics. This matrix $\Sigma$ can be shown to be completely positive, and a nonnegative factorization of it can be used to estimate probabilities of extreme events or to simulate realizations with pairwise dependence summarized by $\Sigma$.

Other authors have studied the completely positive factorization problem before: factorization of matrices with special structures has been studied for a few decades – we refer to the references given in [3] and [14]. Dickinson and Dür [14] extend this work and give a factorization algorithm for acyclic matrices which works in linear time. Also Bomze [6] deals with special structures and shows how a factorization of an $n \times n$ matrix can be constructed if a factorization of an $(n - 1) \times (n - 1)$ principal submatrix is known. For a general input matrix $A$, Jarre and Schmallowsky [21] use a quadratic factorization heuristic to generate a sequence of matrices $BB^T$ that eventually converges to $A$. Their algorithm works well for matrices of up to order $200 \times 200$. Nie [22] treats the completely positive factorization problem as a special case of a $A$-truncated $K$-moment problem. For this more general problem, Nie develops an algorithm based on solving a sequence of (numerically expensive) semidefinite optimization problems. Nie reports numerical experiments for the factorization of completely positive matrices up to order $8 \times 8$. Sponsel and Dür [26] develop an algorithm for the projection of a matrix onto the dual of $\text{CP}_n$, which can also be used to compute completely positive factorizations. However, for reasonably big input matrices, the algorithm runs into memory problems. Finally, Anstreicher, Burer, and Dickinson [2] are developing a factorization algorithm based on the ellipsoid method.

In this paper, we develop a factorization algorithm which, for a given completely positive matrix $A$, computes the nonnegative factorization $BB^T$. We formulate this factorization problem as a nonconvex feasibility problem and develop a solution method based on alternating projections. In Theorem 4.2 we provide a local convergence result for this algorithm. We also propose a heuristic extension which improves the numerical performance of the algorithm. Extensive numerical tests show
that the factorization method is very fast in most instances.

2 Preliminaries and notation

Properties of factorizations

First, recall the following result by Dickinson [13, Theorem 3.8] who showed that the interior of $\mathcal{CP}_n$ can be characterized as follows:

$$\text{int}(\mathcal{CP}_n) = \{ A \in \mathbb{R}^{n \times n} \mid \text{rank}(A) = n, A = BB^T, B \in \mathbb{R}^{n \times r}, B > 0 \}$$

$$= \{ A \in \mathbb{R}^{n \times n} \mid \text{rank}(A) = n, A = BB^T, B \in \mathbb{R}^{n \times r}, B \geq 0, B_j > 0 \text{ for at least one column } B_j \}.$$

The factorization of a completely positive matrix $A \neq 0$ is never unique. We illustrate this with an example by Dickinson [13].

Example 2.1. Consider the matrix

$$A := \begin{pmatrix} 18 & 9 & 9 \\ 9 & 18 & 9 \\ 9 & 9 & 18 \end{pmatrix}.$$

Then $A = B_iB_i^T$ for each of the following matrices:

$$B_1 := \begin{pmatrix} 4 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 4 \end{pmatrix}, \quad B_2 := \begin{pmatrix} 3 & 3 & 0 & 0 \\ 0 & 3 & 0 & 3 \\ 3 & 0 & 3 & 0 \end{pmatrix}, \quad B_3 := \begin{pmatrix} 3 & 3 & 0 \\ 3 & 0 & 3 \\ 0 & 3 & 3 \end{pmatrix}.$$

It will also be useful to consider the eigenvalue factorization, i.e., the square root of $A$: $A = A^{\frac{1}{2}}(A^{\frac{1}{2}})^T$ with

$$A^{\frac{1}{2}} = B_4 := \begin{pmatrix} -1.2030 & 2.1337 & 3.4641 \\ 2.4494 & -0.0250 & 3.4641 \\ -1.2463 & -2.1087 & 3.4641 \end{pmatrix}.$$ 

Observe that the factorizations $A = B_iB_i^T$ with $i = 1, 2$ prove that $A \in \text{int}(\mathcal{CP}_n)$ whereas the factorizations $A = B_iB_i^T$ with $i = 3, 4$ do not. Moreover, observe that the number of columns of the factors $B_i$ varies. This gives rise to the following definitions.

Definition 2.2. Let $A \in \mathbb{R}^{n \times n}$. The cp-rank of $A$ is defined as

$$\text{cpr}(A) := \inf \{ r \in \mathbb{N} \mid \exists B \in \mathbb{R}^{n \times r}, B \geq 0, A = BB^T \}.$$ 

The $\text{cpr}^+$-rank of $A$ is

$$\text{cpr}^+(A) := \inf \{ r \in \mathbb{N} \mid \exists B \in \mathbb{R}^{n \times r}, B > 0, A = BB^T \}.$$
It is an open problem to compute the cp-rank of a matrix, cf. [4]. However, there are bounds on both the cp-rank and the cp\textsuperscript{+}-rank, as shown in the following lemma.

**Lemma 2.3.** (Bomze, Dickinson, Still [8, Theorem 4.1]) We have:

- There exist matrices \( A \in \text{int}(\mathcal{CP}_n) \) for which \( \text{cpr}(A) \neq \text{cpr}^+(A) \).
- For all \( A \in \mathcal{CP}_n \) we have:
  \[
  \text{cpr}(A) \leq \text{cp}_n := \begin{cases} 
  n & \text{for } n \in \{2, 3, 4\} \\
  \frac{1}{2}n(n+1) - 4 & \text{for } n \geq 5.
  \end{cases}
  \]
- For all \( A \in \text{int}(\mathcal{CP}_n) \) we have:
  \[
  \text{cpr}^+(A) \leq \text{cp}_n^+ := \begin{cases} 
  n + 1 & \text{for } n \in \{2, 3, 4\} \\
  \frac{1}{2}n(n+1) - 3 & \text{for } n \geq 5.
  \end{cases}
  \]

**The role of orthogonal matrices**

Recall that a matrix \( Q \in \mathbb{R}^{r \times r} \) is called orthogonal, if \( QQ^T = I_r \). The set of orthogonal matrices is a smooth manifold, sometimes called the Stiefel manifold, see for example [1]. It plays a crucial role in our approach. We therefore define:

**Definition 2.4.** We denote by \( O_r \) the set of \( r \times r \) orthogonal matrices, and we introduce

\[
O_r^+ := \{ Q \in O_r \mid \det Q = 1 \}, \quad O_r^- := \{ Q \in O_r \mid \det Q = -1 \}.
\]

The first set is the set of rotation matrices, the latter is the set of reflection matrices.

Clearly, \( O_r = O_r^+ \cup O_r^- \) and hence \( O_r \) is nonconnected. It is well known that \( O_r \) is compact but not convex. However, using the Schur complement lemma it is easy to see that

\[
\text{conv} O_r = \{ Q \in \mathbb{R}^{r \times r} \mid QQ^T \preceq I \} = \{ Q \in \mathbb{R}^{r \times r} \mid \begin{pmatrix} I & Q^T \\ Q & I \end{pmatrix} \succeq 0 \}.
\]

(As usual, we take \( A \succeq 0 \) to mean that \( A \) is positive semidefinite.) The convex hull of the set of rotation matrices, \( \text{conv} O_r^+ \), can also be described by semidefiniteness constraints, cf. [23, Theorem 1.3]. This result easily extends to \( \text{conv} O_r^- \).

Expanding the condition \( QQ^T = I_r \) into \( r^2 \) quadratic equations shows that \( O_r \) is a semialgebraic set. We will use this fact in Theorem 4.2 below.

Our factorization method is based on the following lemma which illustrates how different factorizations of a matrix are related. The lemma is well known and can for instance be found in [27, Lemma 1]. To keep the paper self-contained, we provide a proof nevertheless.

**Lemma 2.5.** Let \( B, C \in \mathbb{R}^{n \times r} \). Then \( BB^T = CC^T \) if and only if there exists \( Q \in O_r \) with \( BQ = C \).
In many cases, \( \phi \) so the Cholesky decomposition is positive. Now we decompose \( B \) where either 
\[
\hat{A} = \begin{bmatrix} B_1, \ldots, B_{n-1}, \frac{1}{\sqrt{m}} B_n, \frac{1}{\sqrt{m}} B_n, \ldots, \frac{1}{\sqrt{m}} B_n \end{bmatrix}
\]
so \( \phi \) is an isometry. Extending \( \phi \) from \( R(B) \) to an isometry on \( \mathbb{R}^r \) gives the desired matrix \( Q \).

\( \square \)

3 The factorization problem as a nonconvex feasibility problem

Inspired by Lemma 2.5 the idea of our method to factorize a given matrix \( A \in \mathbb{R}^{n \times n} \) is to start with an arbitrary initial factorization \( A = BB^T \) where \( B \in \mathbb{R}^{n \times r} \) is not necessarily nonnegative, and to construct from this a matrix \( Q \in \mathbb{R}^{r \times r} \) such that \( BQ \geq 0 \). This gives the completely positive factorization \( A = (BQ)(BQ)^T \). We point out that this idea was already observed by [20] where it was used to obtain an upper bound for the cp-rank of \( A \).

As an initial factorization of \( A \), one can for instance use the Cholesky decomposition \( A = LL^T \) where \( L \) is a lower triangular matrix, or the eigenvalue decomposition \( A = V \Sigma V^T \) by setting \( B := V \Sigma^{\frac{1}{2}} \). Note, however, that for \( A \in \mathbb{R}^{n \times n} \) the Cholesky- or eigenvalue decompositions will usually yield a factor \( B \in \mathbb{R}^{n \times n} \), whereas in a nonnegative factorization we will have \( A = BB^T \) with \( B \in \mathbb{R}^{n \times r} \) for some \( r \geq \text{cpr}(A) \). It is known that \( \text{cpr}(A) \) can be considerably larger than \( n \), however in general it is not possible to compute \( \text{cpr}(A) \). Therefore, we use \( r = \text{cp}_n \) or \( r = \text{cp}^*_n \), the bounds from Lemma 2.3.

Consider an initial factorization \( A = BB^T \) with \( B \in \mathbb{R}^{n \times n} \) and assume that \( \text{cpr}(A) \) is unknown, but \( \text{cpr}(A) \leq \text{cp}_n \). One way to construct an \( n \times \text{cp}_n \)-matrix \( \tilde{B} \) with \( A = \tilde{B} \tilde{B}^T \) is to append \( k := \text{cp}_n - n \) zero columns to \( B \), i.e., \( \tilde{B} := [B, 0_{n \times k}] \). A different option is the following: let \( B_j \) denote the \( j \)-th column of \( B \), and assume w.l.o.g. that \( B_n \) is the column with the least number of negative entries. In many cases, \( B_n \) is strictly positive, for example if \( A \) is positive definite, then the first column of the Cholesky decomposition is positive. Now we decompose \( B_n \) into \( m := \text{cp}_n - n + 1 \) columns to obtain

\[
\tilde{B} := \begin{bmatrix} B_1, \ldots, B_{n-1}, \frac{1}{\sqrt{m}} B_n, \frac{1}{\sqrt{m}} B_n, \ldots, \frac{1}{\sqrt{m}} B_n \end{bmatrix} \in \mathbb{R}^{n \times \text{cp}_n}
\]

with \( \tilde{B} \tilde{B}^T = A \). It is clear from Lemmas 2.3 and 2.5 that \( A \in \mathbb{CP}_n \) if and only if there exists \( Q \in \mathcal{O}_r \) such that \( \tilde{B}Q \geq 0 \) (resp. \( \tilde{B}Q \geq 0 \)). Numerically, it turns out that using \( \tilde{B} \) is more promising than using \( \hat{B} \), see Section 7.5 below.

From now on, we will assume that we are given an initial factorization \( A = BB^T \) with \( B \in \mathbb{R}^{n \times r} \), where either \( r = \text{cp}(A) \) if this quantity happens to be known, or otherwise we use the bound from
Lemma 2.3 and set \( r = cp_n \). The problem of finding a completely positive factorization of \( A \) can then be formulated as the following feasibility problem:

\[
\begin{align*}
\text{find} & \quad Q \\
\text{s. t.} & \quad BQ \geq 0 \\
& \quad Q \in O_r.
\end{align*}
\] (3)

This problem is feasible if and only if \( A \in CP_n \). Observe that if (3) has a solution, then it also has a solution in \( O_r^+ \), the set of rotation matrices: indeed, a solution \( Q^- \in O_r^- \) of (3) can be multiplied with a permutation matrix \( P \in O_r^- \) which permutes two columns. Then the matrix \( Q^- P =: Q \in O_r^+ \) is also a solution of (3). Therefore, (3) is equivalent to the problem

\[
\begin{align*}
\text{find} & \quad Q \\
\text{s. t.} & \quad BQ \geq 0 \\
& \quad Q \in O_r^+.
\end{align*}
\] (4)

Unfortunately, neither \( O_r \) nor \( O_r^+ \) are convex sets, so a first idea would be to consider their convex hulls. Convexifying (3) yields the semidefinite feasibility problem

\[
\begin{align*}
\text{find} & \quad Q \\
\text{s. t.} & \quad BQ \geq 0 \\
& \quad \left(
\begin{array}{cc}
I & Q^T \\
Q & I
\end{array}
\right) \succeq 0.
\end{align*}
\] (5)

Problem (4) can be convexified by using the technique from Saunderson et al. [23, Theorem 1.3]. This also leads to a semidefinite feasibility problem, however one of considerably larger size.

It should be noted that (5) is always feasible (take \( Q = 0 \)). If the convexified version of (4) is infeasible, then this certifies that the input matrix \( A \) is not completely positive.

If solving either of the convexified problems happens to provide a \( Q \in O_r \) (resp. \( Q \in O_r^+ \)), then we have derived a completely positive factorization \( A = (BQ)(BQ)^T \). Unfortunately, in numerical tests with randomly generated completely positive input matrices \( A \), we always observed the third case: the convexified problems were feasible, but the resulting matrix \( Q \) we obtained was not orthogonal. In this case, nothing can be inferred about \( A \).

Consequently, convexifying problems (3) or (4) does not produce any useful insight, so we need to pursue a different approach.

4 Alternating projections

Let us go back to the nonconvex feasibility problem (3). Introducing the polyhedral cone

\[
P := \{ Q \in \mathbb{R}^{r \times r} \mid BQ \geq 0 \},
\]
we can write (3) as

\begin{align*}
\text{find} \quad Q \\
\text{s.t.} \quad Q \in \mathcal{P} \cap \mathcal{O}_r.
\end{align*}

A well studied algorithm to find a point in the intersection \( \mathcal{P} \cap \mathcal{O}_r \) is the method of alternating projections: denote by \( \text{proj}_S(x) \) the (not necessarily unique) projection of the point \( x \) onto a set \( S \), i.e., an element of the set of points in \( S \) at minimum distance from \( x \). Then the method of alternating projections to solve (6) can be stated as: choose a starting point \( Q_0 \in \mathcal{O}_r \), compute \( P_0 = \text{proj}_\mathcal{P}(Q_0) \), compute \( Q_1 = \text{proj}_\mathcal{O}_r(P_1) \), and iterate this process.

In order to apply this method, we need to be able to project onto the sets \( \mathcal{P} \) and \( \mathcal{O}_r \). For \( \mathcal{P} \) this is easy: since \( \mathcal{P} \) is a polyhedral cone and hence convex, the projection of a matrix \( M \) onto \( \mathcal{P} \) is unique, and computing it amounts to solving a second order cone problem (SOCP):

\[
\begin{align*}
\min_{X} & \quad \| X - M \| \\
\text{s.t.} & \quad BX \geq 0
\end{align*} \quad \Leftrightarrow \quad \begin{align*}
\min_{t} & \quad t \\
\text{s.t.} & \quad BY \geq -BM \\
(t, \text{vec}(Y)) & \in \text{SOC}.
\end{align*}
\]

Note that SOCPs can be solved in polynomial time by using interior point methods.

The projection of a matrix \( M \) onto \( \mathcal{O}_r \) always exists since \( \mathcal{O}_r \) is compact, however it may not be unique due to the nonconvexity of \( \mathcal{O}_r \). Computing an element of \( \text{proj}_{\mathcal{O}_r}(M) \) can be done through the polar decomposition of \( M \) according to the following lemma, a proof of which can be found in [5, Corollary 5.6.4 and Fact 9.9.42].

**Lemma 4.1.** Let \( M \in \mathbb{R}^{r \times r} \). Then there exists the so called polar decomposition of \( M \), i.e., there exist a positive semidefinite matrix \( T \in \mathbb{R}^{r \times r} \) and an orthogonal matrix \( Q \in \mathbb{R}^{r \times r} \) such that

\[ M = TQ. \]

For any unitarily invariant norm \( \| \cdot \| \), we have

\[ \| M - Q \| \leq \| M - U \| \quad \text{for all} \quad U \in \mathcal{O}_r. \]

We take this \( Q \) as \( \text{proj}_{\mathcal{O}_r}(M) \). Starting from the singular value decomposition \( M = U \Sigma V^T \), we can compute \( Q \) as \( Q := UV^T \). This can be done in \( O(r^3) \) steps.

Hence, we can efficiently compute the projections \( \text{proj}_{\mathcal{P}}(M) \) and \( \text{proj}_{\mathcal{O}_r}(M) \) of a matrix \( M \) onto \( \mathcal{P} \) and \( \mathcal{O}_r \), respectively. The alternating projection method to compute a factorization of a completely positive matrix \( A \) now reads as follows:
Algorithm 1

Input: \(A = BB^T\) with \(B \in \mathbb{R}^{n \times r}\) and \(r \geq \text{cpr}(A)\); initial matrix \(Q_0 \in \mathcal{O}_r\)

1: \(k \leftarrow 0\)
2: while \(BQ_k \not\succeq 0\) do
3: \(P_k \leftarrow \text{proj}_\mathcal{P}(Q_k)\)
4: \(Q_{k+1} \leftarrow \text{proj}_{\mathcal{O}_r}(P_k)\)
5: \(k \leftarrow k + 1\)
6: end while

Output: \(Q_k \in \mathcal{O}_r\) and a completely positive factorization \(A = (BQ_k)(BQ_k)^T\)

Local convergence is ensured by the following theorem.

Theorem 4.2. Let \(A \in \mathcal{CP}_n\). Let \(A = BB^T\) be any initial factorization with \(B \in \mathbb{R}^{n \times r}\) and \(r \geq \text{cpr}(A)\). Define \(\mathcal{P} := \{Q \in \mathbb{R}^{r \times r} \mid BQ \succeq 0\}\). Then we have:

(a) \(\mathcal{P} \cap \mathcal{O}_r \neq \emptyset\),

(b) if started at a point \(Q_0\) close to \(\mathcal{P} \cap \mathcal{O}_r\), then Algorithm 1 converges to a point \(Q^* \in \mathcal{P} \cap \mathcal{O}_r\). In this case, \(A = (BQ^*)(BQ^*)^T\) is a completely positive factorization of \(A\).

Proof. (a): It follows from \(A \in \mathcal{CP}_n\) and \(r \geq \text{cpr}(A)\) that there exists \(C \in \mathbb{R}^{n \times r}\), \(C \succeq 0\) with \(A = CC^T\). Since \(A = BB^T = CC^T\), Lemma 2.5 implies that there exists \(Q \in \mathcal{O}_r\) such that \(BQ = C \succeq 0\), i.e., \(Q \in \mathcal{P} \cap \mathcal{O}_r\).

(b): Both \(\mathcal{P}\) and \(\mathcal{O}_r\) are closed semialgebraic sets. Moreover, \(\mathcal{O}_r\) is bounded. The convergence result now follows by applying [17, Theorem 7.3].

5 Modifying the alternating projection method

Note that in Step 3 of Algorithm 1 we have to solve an SOCP in every iteration in order to compute \(\text{proj}_\mathcal{P}(Q_k)\). While this can be done in polynomial time, it is still very costly, and it turns out that a slight modification of this step provides a much better numerical performance, although we loose the local convergence theory. We proceed as follows:

Instead of computing the projection \(\text{proj}_\mathcal{P}(Q)\) of \(Q\) onto \(\mathcal{P} := \{Q \in \mathbb{R}^{r \times r} \mid BQ \succeq 0\}\), we rather project \(BQ\) onto the nonnegative orthant by computing a matrix \(D \in \mathbb{R}^{n \times r}\) through

\[
D_{ij} := \max \{(BQ)_{ij}, 0\} \quad \text{for all} \ i = 1, \ldots, n \text{ and } j = 1, \ldots, r.
\]

Note that \(D \in \mathbb{R}^{n \times r}\), so in order to get an approximation of \(\text{proj}_\mathcal{P}(Q)\), we need to lift \(D\) into the space \(\mathbb{R}^{r \times r}\). Now, if \(D = BQ\), then \(BQ \succeq 0\), i.e., \(Q \in \mathcal{P}\), so \(Q = \text{proj}_\mathcal{P}(Q)\), and we are done.

If \(D \neq BQ\), then we let \(B^+\) denote the Moore-Penrose-Inverse of \(B\) and define

\[
\widehat{P} := B^+D + (I - B^+B)Q \in \mathbb{R}^{r \times r}
\]
and take \( \hat{P} \) as an approximation of \( \text{proj}_P(Q) \). The motivation for this is as follows: the assumption that \( D \neq BQ \) means that \( Q \) does not solve the equation \( BX = D \). If this equation is solvable, then \( \hat{P} \) is the unique solution which minimizes the distance to \( Q \), i.e., \( \hat{P} \) is the projection of \( Q \) onto the set \( \{X \in \mathbb{R}^{r \times r} \mid BX = D \} \). This set is a subset of \( \mathcal{P} \) because \( D \geq 0 \), so in this case \( \hat{P} \in \mathcal{P} \).

If the equation \( BX = D \) does not have a solution, then \( X = \hat{P} \) minimizes the residual \( \|BX - D\| \) and among all minimizers it is the one closest to \( Q \). In this case, we get from the properties of \( B^+ \) that

\[
B\hat{P} = BB^+D + (B - BB^+)Q = BB^+D,
\]

and if in addition the rows of \( B \) are linearly independent (which is true if \( A \) has full rank), then \( BB^+ = I_n \), which implies that \( B\hat{P} = D \geq 0 \), and hence again \( \hat{P} \in \mathcal{P} \). If the rows of \( B \) are linearly dependent, then it may happen that \( \hat{P} \notin \mathcal{P} \), however this does not seem to impair the good numerical performance.

This reasoning leads to the following modification of Algorithm 1:

Algorithm 2

**Input:** \( A = BB^T \) with \( B \in \mathbb{R}^{n \times r} \) and \( r \geq \text{cpr}(A) \); initial matrix \( Q_0 \in \mathcal{O}_r \)

1. \( k \leftarrow 0 \)
2. **while** \( BQ_k \geq 0 \) **do**
3. \( D \leftarrow \max \{BQ_k, 0\} \) entrywise
4. \( \hat{P}_k \leftarrow B^+D + (I - B^+B)Q_k \)
5. \( Q_{k+1} \leftarrow \text{proj}_{\mathcal{O}_r} (\hat{P}_k) \)
6. \( k \leftarrow k + 1 \)
7. **end while**

**Output:** \( Q_k \in \mathcal{O}_r \) and a completely positive factorization \( A = (BQ_k)(BQ_k)^T \)

Clearly, if Algorithm 2 terminates, then it yields a completely positive factorization of \( A \). However, since Algorithm 2 is not a pure alternating projection method, we do not get a local convergence result like Theorem 4.2 above. Nevertheless, numerical experiments show that this algorithm is highly efficient.

6 Matrices in the interior of the completely positive cone

According to \([13, \text{Theorem 3.8}]\), a matrix \( A \in \text{int}(\mathcal{CP}_n) \) always has a factorization \( A = BB^T \) with \( B > 0 \). Note that both Algorithm 1 and Algorithm 2 can find such a factorization by a slight modification of the stopping criterion: instead of \( BQ_k \geq 0 \), one can use \( BQ_k \geq \epsilon E \) as a stopping criterion, where \( E \) denotes the all ones matrix and \( \epsilon > 0 \) is a small threshold value. Theorem 4.2 holds analogously if \( r \geq \text{cpr}^+(A) \) and if \( \mathcal{P} \) is redefined as \( \mathcal{P} := \{Q \in \mathbb{R}^{r \times r} \mid BQ \geq \epsilon E \} \). If \( r \) is unknown, then the bound \( r = \text{cp}_{n^+} \) from Lemma 2.3 may be used.
7 Numerical results

The numerical results were carried out on a computer with 88 Intel Xenon ES-2699 cores (2.2 Ghz each) and a total of 0.792 TB Ram. The algorithms were implemented in MatlabR2017a, the SOCPs in Algorithm\[1\] were solved using Yalmip R20170626 and SDPT3 4.0.

The experiments were carried out as follows: If $A$ is of full rank, then we use the Cholesky factorization as the initial factorization $\tilde{B}$; otherwise, we use $\tilde{B} = A^{\frac{1}{2}}$. For the given value $r \geq \text{cpr}(A)$, we generate from $\tilde{B}$ a matrix $B \in \mathbb{R}^{n \times r}$ with $A = BB^T$ by column replication as described in [2]. We use column replication throughout, only in Section 7.5 we also use appending zero columns as described in Section [3] for comparison.

We produce the random starting point $Q_0$ by generating a random $r \times r$-matrix $M$ using the Matlab command \texttt{randn} and then setting $Q_0 \leftarrow \text{proj}_O(M)$. Algorithm [1] (resp. Algorithm [2]) terminates successfully at iteration $k$ if $BQ_k \geq -10^{-15}$, it terminates unsuccessfully if a maximum number of iterations (usually 5000) is reached. Since not all starting points lead to successful termination, we repeat the experiment for a number of different starting points (usually 100 starting points).

7.1 A specifically structured example in different dimensions

Example 7.1. Consider the following matrix from [24]: Let $e_n$ denote the all-ones-vector in $\mathbb{R}^n$ and consider

$$A_n := \begin{pmatrix} 0 & e_{n-1}^T \\ e_{n-1} & I_{n-1} \end{pmatrix}^T \begin{pmatrix} 0 & e_{n-1}^T \\ e_{n-1} & I_{n-1} \end{pmatrix} \in \mathbb{R}^{n \times n}$$

It has been shown in [24, Example 7.4] that $A_n \in \text{int CP}_n$ for every $n \geq 2$. By construction, it is clear that $\text{cpr}(A_n) = n$.

We tried to factorize $A_n$ for the values $n \in \{10, 20, 50, 75, 100, 150\}$. It can be seen from Figure[1] that Algorithm[2] succeeds in factorizing $A_n$ in all cases. However, the percentage of successful starting points decreases for increasing $n$. 


Figure 1: Performance of Algorithm 2 for $A_n$ from Example 7.1 with different values of $n$, using $r = \text{cpr}(A_n) = n$. For each $A_n$, we used 100 starting points and a maximum number of 5000 iterations per starting point.

7.2 The influence of the parameter $r$

Our algorithm needs an input parameter $r \geq \text{cpr}(A)$, however for a general input matrix $A \in \mathbb{R}^{n \times n}$, it is usually impossible to compute $\text{cpr}(A)$. Therefore, it becomes necessary to use the bounds $c_p(n)$ or $c_p^+(n)$ from Lemma 2.3, i.e., to use $r = c_p(n) \geq \text{cpr}(A)$. Figure 2 shows the performance of Algorithm 2 when using different values of $r$.

Figure 2 shows that the algorithm produces a cp-factorization of $A_6$ for each value of $r$. Note, however, that the percentage of successful starting points increases for increasing $r$, such that for $r \geq 15$ nearly every starting point gives a cp-factorization of $A_6$ in less than 750 iterations. In addition, Figure 2 indicates that an increasing value of $r$ leads to a decreasing average number of iterations until termination. A similar picture can be obtained for the matrix $A_n$ from Example 7.1 for other values of $n$.

As a recipe for the parameter $r$, we recommend to start the algorithm with $r = n$ and to increase $r$ gradually up to $c_p(n)$, stopping whenever the algorithm successfully outputs a factorization. The reason behind this recommendation is that obviously iterations are faster for smaller values of $r$ (recall that the algorithm works with matrices in $\mathbb{R}^{r \times r}$), cf. also Table 2 in Section 7.8 for this observation. Moreover,
Figure 2: Performance of Algorithm 2 for the matrix $A_6$ from Example 7.1 using different values of $r \geq \text{cpr}(A_6) = 6$. We used 1000 starting points and a maximum number of 5000 iterations per starting point.

for small $n$, the bound $\text{cp}_n \approx n$, whereas for large $n$, we have $\text{cp}_n \gg n$. Numerical evidence shows that for most matrices $\text{cpr}(A) \ll \text{cp}_n$, so setting $r = \text{cp}_n$ results in unnecessarily long computation times.

7.3 A low cp-rank matrix without known factorization

Hitherto we considered matrices for which a cp-factorization is already known. In this section, we consider a matrix which is known to be completely positive, but for which no factorization is known.

Example 7.2. Consider the following matrix from [25 Example 2.7]:

$$A = \begin{pmatrix}
41 & 43 & 80 & 56 & 50 \\
43 & 62 & 89 & 78 & 51 \\
80 & 89 & 162 & 120 & 93 \\
56 & 78 & 120 & 104 & 62 \\
50 & 51 & 93 & 62 & 65
\end{pmatrix}$$
According to a sufficient condition from [25, Theorem 2.5], this matrix is completely positive with \( \text{cpr}(A) = \text{rank}(A) = 3 \).

We tested this matrix, using Algorithm 2 with \( r = 5 \). We use a \( 5 \times 5 \) eigenvalue decomposition of \( A \) which gives the initial factorization matrix

\[
B = \begin{pmatrix}
0.0000 & 0.0000 & 0.1341 & -1.5233 & 6.2178 \\
0.0000 & 0.0000 & -1.6716 & 1.9775 & 7.4361 \\
0.0000 & 0.0000 & 1.5900 & -0.9900 & 12.5893 \\
-0.0000 & -0.0000 & 0.2307 & 3.0140 & 9.7398 \\
0.0000 & -0.0000 & -1.4547 & -3.0168 & 7.3337
\end{pmatrix}
\]

where the absolute values of all the \( \pm 0.0000 \) entries are less or equal to \( 10^{-7} \). Due to the fact that \( \text{rank}(A) = 3 \), we have only three relevant columns. Using the randomly generated starting matrix

\[
Q_0 = \begin{pmatrix}
0.6901 & 0.0000 & -0.0000 & -0.0000 & -0.7237 \\
-0.7237 & -0.0000 & 0.0000 & 0.0000 & -0.6901 \\
0.0000 & -0.1728 & 0.6936 & -0.6993 & -0.0000 \\
0.0000 & -0.9357 & 0.1061 & 0.3365 & 0.0000 \\
0.0000 & -0.3076 & 0.7125 & 0.6307 & 0.0000
\end{pmatrix}
\]

Algorithm 2 provides the following cp-factorization in 0.018 seconds and after 23 iterations, again with a precision of \( 10^{-7} \):

\[
A = \tilde{B}\tilde{B}^T, \text{ with } \tilde{B} = \begin{pmatrix}
0.0000 & 3.3148 & 4.3615 & 3.3150 & 0.0000 \\
0.0000 & 0.7261 & 4.3485 & 6.5241 & 0.0000 \\
0.0000 & 4.5242 & 9.9675 & 6.4947 & 0.0000 \\
0.0000 & 0.8361 & 7.4192 & 6.9955 & 0.0000 \\
0.0000 & 5.3301 & 3.8960 & 4.6272 & 0.0000
\end{pmatrix}
\]

This confirms that \( \text{cpr}(A) = 3 \) and gives an explicit cp-factorization of \( A \) for the first time.

Since \( A \) is not of full rank, \( A \notin \text{int}(\mathcal{CP}_5) \). So this example shows that Algorithm 2 can also factorize matrices on the boundary of the completely positive cone. We compare the performance of Algorithm 2 in the interior and on the boundary of \( \mathcal{CP}_n \) in Section 7.6.

### 7.4 Algorithms 1 and 2 in comparison

The next experiment is based on Example 7.2 and compares the performance of Algorithms 1 and 2.

Figure 3 shows that Algorithms 1 and 2 only have local convergence. Algorithm 1 provides a completely positive factorization for 61% of the starting points, Algorithm 2 for 78%. In addition, we see that the number of iterations necessary to compute a factorization is lower for Algorithm 1. However, the iterations of Algorithm 1 are much more expensive: running all 100 starting points in Algorithm 2 takes 4.5 seconds, but 5656 seconds for Algorithm 1.

This shows that Algorithm 2 is much faster in total: although it may need more iterations than Algorithms 1, the numerical cost of a single iteration is much smaller. Moreover, the percentage of successfully terminating starting points is higher for Algorithm 2.
Figure 3: Success rates of Algorithms 1 and 2 applied to the matrix of Example 7.2 with \( r = \text{cp}^+_5 = 12 \). We used 100 starting points and a maximum number of 500 iterations per starting point.

7.5 Column replication versus appending zero columns

In Section 3, we mentioned two possible ways of expanding an initial factorization matrix \( B \in \mathbb{R}^{n \times n} \) into a matrix with \( r \geq \text{cpr}(A) \) columns: either by column replication as described in (2), or by appending zero columns. To see which approach performs numerically better, we use Example 7.1 for \( n = 5 \).

Figure 4 shows that when appending zero columns, roughly 55% of the starting points yield a cp-factorization, as opposed to 95% if we use column replication. The mean number of iterations needed with successful starting points is 212 for the column replication approach and 346 when appending zero columns. For those cases where both approaches were successful, column replication was always faster, i.e., terminated after fewer iterations. These results suggest that the column replication approach is numerically more efficient than appending zero columns.
Figure 4: Success rate of Algorithm 2 for column replication versus appending zero columns for Example 7.1. We used the same 100 starting points for both approaches and a maximum of 500 iterations per starting point.

7.6 Performance of Algorithm 2 on the boundary and in the interior of \( \mathcal{CP}_n \)

In this section, we illustrate how Algorithm 2 behaves for matrices on the boundary of \( \mathcal{CP}_n \). We start with an instance where the algorithm fails:

**Example 7.3.** Consider the following matrix taken from [18]:

\[
A = \begin{pmatrix}
8 & 5 & 1 & 1 & 5 \\
5 & 8 & 5 & 1 & 1 \\
1 & 5 & 8 & 5 & 1 \\
1 & 1 & 5 & 8 & 5 \\
5 & 1 & 1 & 5 & 8 \\
\end{pmatrix} \in \mathcal{CP}_5 \setminus \text{int}(\mathcal{CP}_5).
\]

Neither Algorithm 1 nor Algorithm 2 succeed in factorizing this matrix. In view of Theorem 4.2, this may seem surprising, however we suspect that the region of local convergence of Algorithm 1 is so small that numerical precision prevents us from finding a starting point there.

In the following we show that for slight perturbations of this matrix, Algorithm 2 becomes successful in finding a factorization. To this end, we investigate convex combinations of \( A \) and the following
matrix \( C \in \text{int}(CP_5) \):

\[
C = MM^T, \quad \text{where } M = \begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}.
\]

Table 1 shows the performance of Algorithm 2 for \( A_\lambda := \lambda A + (1 - \lambda)C \) for different values of \( \lambda \in [0, 1] \).

<table>
<thead>
<tr>
<th>( \lambda )</th>
<th>time (sec.)</th>
<th>success rate (%)</th>
<th>( \lambda )</th>
<th>time (sec.)</th>
<th>success rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>100</td>
<td>0.8</td>
<td>51</td>
<td>50</td>
</tr>
<tr>
<td>0.1</td>
<td>5</td>
<td>99</td>
<td>0.9</td>
<td>72</td>
<td>23</td>
</tr>
<tr>
<td>0.2</td>
<td>9</td>
<td>95</td>
<td>0.95</td>
<td>69</td>
<td>26</td>
</tr>
<tr>
<td>0.3</td>
<td>15</td>
<td>93</td>
<td>0.97</td>
<td>69</td>
<td>36</td>
</tr>
<tr>
<td>0.4</td>
<td>28</td>
<td>82</td>
<td>0.976</td>
<td>71</td>
<td>24</td>
</tr>
<tr>
<td>0.5</td>
<td>32</td>
<td>73</td>
<td>0.977</td>
<td>79</td>
<td>5</td>
</tr>
<tr>
<td>0.6</td>
<td>34</td>
<td>72</td>
<td>0.9774</td>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>0.7</td>
<td>46</td>
<td>51</td>
<td>0.9775</td>
<td>83</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Performance of Algorithm 2 for the matrix \( A_\lambda \) for different values of \( \lambda \in [0, 1] \) and \( r = \text{cp}_5^+ = 12 \). For each \( A_\lambda \), we run Algorithm 2 with 100 starting points and a maximum of 5000 iterations per starting point. Column 2 shows the total computation time of Algorithm 2 for all 100 starting points.

Note that as we approach the boundary of \( CP_n \), the success rate decreases and the computing time increases. The latter is because the closer we get to the boundary, the higher the number of starting points for which the algorithms runs for a high number of iterations. Also note that only very close to the matrix \( A \) does the success rate decrease rapidly, and we even find a factorization of the matrix \( A_\lambda \) with \( \lambda = 0.97749 \).

### 7.7 Other difficult instances

Whereas Algorithm succeeded in factorizing the matrix of low cp-rank from Example 7.2, it failed for several matrices of high cp-rank taken from [9]. These are specifically constructed matrices for which \( \text{cpr}(A) \gg \text{rank}(A) \). Algorithm failed for these instances, even by gradually increasing \( r \) up to two or three times \( \text{cpr}(A) \). So for these artificially generated matrices, Algorithm seems to struggle finding a suitable initial orthogonal matrix, even for higher values of \( r \).

In addition, we tried to apply Algorithm to recover the stable set of a graph from the solution of problem (1). Unfortunately, this was unsuccessful. The reason is that typically the solution of (1)
is a matrix that contains a number of zero entries, which imposes a certain sparsity pattern for the factorization matrix $B$ as well: whenever $A_{ij} = 0$ and $A = BB^T$, then the columns $B_i$ and $B_j$ necessarily have to have disjoint support. Generating an orthogonal matrix that provides this is extremely unlikely, so without further adjustments, Algorithm 2 will typically fail for such instances.

### 7.8 Randomly generated examples of higher order

Next, we investigate randomly generated matrices of higher order to see how the algorithm depends on the order of the input matrix. The instances were generated as follows: First, we generate a random $n \times k$ matrix $B$ using the Matlab command `randn`. Next, we compute $C$ by setting $C_{ij} := |B_{ij}|$ for all $i, j$, and finally we take $A = CCT$ as the matrix to be factorized by our algorithm. By construction, we have $A \in CP_n$ with $\text{cpr}(A) \leq k$. Table 2 illustrates the results.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$cp_n^+$</th>
<th>$r$</th>
<th># of initial $Q_0$</th>
<th># of iterations</th>
<th>time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>627</td>
<td>36</td>
<td>58.6</td>
<td>1937</td>
<td>192</td>
</tr>
<tr>
<td>50</td>
<td>1272</td>
<td>51</td>
<td>92</td>
<td>1765</td>
<td>504</td>
</tr>
<tr>
<td>50</td>
<td>1272</td>
<td>151</td>
<td>1</td>
<td>177</td>
<td>0.9</td>
</tr>
<tr>
<td>100</td>
<td>5047</td>
<td>151</td>
<td>1</td>
<td>1365</td>
<td>9.8</td>
</tr>
<tr>
<td>100</td>
<td>5047</td>
<td>301</td>
<td>1</td>
<td>183</td>
<td>4</td>
</tr>
<tr>
<td>150</td>
<td>11322</td>
<td>201</td>
<td>1.2</td>
<td>2384</td>
<td>42</td>
</tr>
<tr>
<td>200</td>
<td>20097</td>
<td>301</td>
<td>1.1</td>
<td>1547</td>
<td>47</td>
</tr>
<tr>
<td>1000</td>
<td>500497</td>
<td>1500</td>
<td>1.1</td>
<td>2454</td>
<td>1092</td>
</tr>
<tr>
<td>2000</td>
<td>2000997</td>
<td>3000</td>
<td>1.2</td>
<td>2993</td>
<td>9675</td>
</tr>
</tbody>
</table>

Table 2: Performance of Algorithm 2 for randomly generated matrices $A$ for several values of $n$ and $k = 2n$. For $n \leq 50$, we used 100 starting points, for $n > 50$, we used 10 starting points. In each case, we used a maximum of 5000 iterations per starting point. The numbers in columns 4–6 represent the average of 100 randomly generated instances.

Table 2 shows that for every value of $n$ and $r > k$, Algorithm 2 is successful for all randomly generated instances.

Looking in more detail at the case $n = 50$, we see that with $r = n + 1$, on average 92 starting points were necessary, each taking about 500 seconds, whereas for $r = 3n + 1$ all starting points were successful, each taking only one second on average. Table 2 shows similar results for $n = 100$. Therefore, we conclude that using a higher value of $r$ can lead to immense time savings, although a single iteration will take longer.

In addition, we also tested matrices of high order. As can be seen from Table 2, time consumption increases exponentially for increasing $n$, but even for $n = 2000$, Algorithm 2 provides a factorization in 160 minutes on average.
7.9 Comparision with an algorithm by Ding et al.

In [16], Ding et al. proposed a simple algorithm for symmetric nonnegative matrix factorization. Their algorithm works by updating a randomly chosen initial factorization. For the reader’s convenience, we state this method as Algorithm 3.

Algorithm 3  The algorithm by Ding et al. [16]

Input: matrix $A \in \mathbb{R}^{n \times n}$, $r \in \mathbb{N}$, $q_{\text{max}} \in \mathbb{N}$, $\text{MaxIter} \in \mathbb{N}$, $\beta \in (0, 1]$

1: for $q = 1 : q_{\text{max}}$ do
2: $B \leftarrow \text{randn}(n, r)$
3: optional: $B \leftarrow \max(AB(B^TB)^{-1}, 0)$
4: while ($\Vert A - BB^T \Vert_2 \geq 10^{-12}$ and $B_{ik} \geq -10^{-15}$) or $k < \text{MaxIter}$ do
5: for $i = 1 : n$ do
6: for $j = 1 : r$ do
7: $B_{ij} \leftarrow B_{ij} \left(1 - \beta + \beta \frac{(AB)_{ij}}{(BB^TB)_{ij}}\right)$
8: end for
9: end for
10: end while
11: end for

Table 3 shows the performance of this method for Example 2.1. Note that this matrix has cp-rank 3, so setting $r = 3$ in Algorithm 3 would be optimal.

<table>
<thead>
<tr>
<th>$r$</th>
<th>av. computation time (sec.)</th>
<th>av. success rate (%)</th>
<th>av. number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4.8</td>
<td>18.5</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>5.5</td>
<td>13.5</td>
<td>51</td>
</tr>
<tr>
<td>5</td>
<td>5.5</td>
<td>15.6</td>
<td>51</td>
</tr>
<tr>
<td>10</td>
<td>5.1</td>
<td>4.5</td>
<td>52</td>
</tr>
<tr>
<td>15</td>
<td>5.9</td>
<td>2.4</td>
<td>51</td>
</tr>
</tbody>
</table>

Table 3: Performance of Algorithm 3 applied to Example 2.1. We used $\beta = 0.5$ as suggested in [16], $q_{\text{max}} = 100$ and $\text{MaxIter} = 5000$. For each $r$, we perform 100 runs of Algorithm 3. The numbers in the table represent averages over the 100 runs: average computation time, average percentage of successful starting points, and the average number of iterations if a successful initial matrix $B$ is chosen.

Table 3 indicates that both the computation time and the average number of iterations are independent of $r$, however the success rate is very sensitive to this parameter. This observation is even more striking if we apply Algorithm 3 to Example 7.2, a $5 \times 5$ matrix $A$ with $\text{cp-r}(A) = \text{rank}(A) = 3$, as can be seen in Table 4 in that example, the algorithm terminates only if $r = \text{cp-r}(A)$ is used, otherwise it fails completely.
Table 4: Performance of Algorithm\(^3\) by Ding et al. with \(\beta = 0.5\), qmax = 100 and MaxIter = 5000, applied to Example\(^7,2\) and using different values of \(r\). The numbers are averages of 100 runs per value for \(r\).

<table>
<thead>
<tr>
<th>(r)</th>
<th>av. computation time (sec.)</th>
<th>av. success rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.5</td>
<td>4.5</td>
</tr>
<tr>
<td>5</td>
<td>9.7</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>9.3</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>9.2</td>
<td>0</td>
</tr>
</tbody>
</table>

More experiments with randomly generated matrices also showed that the algorithm of Ding et al. is highly sensitive with respect to the parameter \(r\) and works badly if an inappropriate \(r\) is chosen. Since in general the cp-rank is not known a priori, this strong sensitivity with respect to \(r\) must be considered a huge drawback. This is a disadvantage of Algorithm\(^3\) which our Algorithm\(^2\) does not exhibit, and hence we consider our algorithm more stable and robust with respect to the algorithm parameters.

Acknowledgements

This work was done while the second author (M.D.) was affiliated with the University of Trier. It was supported partially by the German-Israeli Foundation for Scientific Research and Development (GIF) through grant no. G-18-304.2/2011, and partially by the German Research Foundation (DFG) through the Research Training Group 2126 “Algorithmic Optimization”. The authors would like to thank Abraham Berman and Naomi Shaked-Monderer for fruitful discussions.

References


