Copositive Programming by Simplicial Partition

Julius ŽILINSKAS
Vilnius University, Institute of Mathematics and Informatics
Akademijos 4, LT-08663 Vilnius, Lithuania
e-mail: julius.zilinskas@mii.vu.lt

Received: August 2011; accepted: November 2011

Abstract. Copositivity plays an important role in combinatorial and quadratic optimization since
setting up a linear optimization problem over the copositive cone leads to exact reformulations
of combinatorial and quadratic programming problems. A copositive programming problem may
be approached checking copositivity of several matrices built with different values of the variable
and the solution is the extreme value for which the matrix is copositive. However, this approach
has some shortcomings. In this paper, we develop a simplicial partition algorithm for copositive
programming to overcome the shortcomings. The algorithm has been investigated experimentally
on a number of problems.

Keywords: simplicial partition, copositive programming, copositive matrices.

1. Introduction

Copositivity plays an important role in combinatorial and quadratic optimization. Setting up a linear optimization problem over the copositive cone leads to exact reformulations of combinatorial and quadratic problems, for example, maximum clique (Bomze et al., 2000), stable set (de Klerk and Pasechnik, 2002; Dukanovic and Rendl, 2010), quadratic assignment (Povh and Rendl, 2009) and graph-partitioning (Povh and Rendl, 2007). Any quadratic problem with linear constraints and binary variables can be equivalently formulated as a linear optimization problem over the cone of copositive matrices (Burer, 2009). Optimization of linear function over copositive cone is called copositive programming, which is a relatively young field in mathematical optimization (Dür, 2010).

The problems listed above are not polynomially solvable. This property is preserved when the problem is formulated as a copositive program. The complexity is moved to the cone constraint: checking whether a given matrix is copositive, is a co-NP-complete problem (Murty and Kabadi, 1987). Most of proposed methods to check copositivity are linear algebraic and rely on checking properties of exponentially many principal submatrices (Väliaho, 1986; Andersson et al., 1995; Kaplan 2000, 2001; Yang and Li, 2009).

An algorithm for copositivity detection by simplicial partition (Bundfuss and Dür, 2008) reduces the problem to that of verifying non-negativity of a quadratic form over the standard simplex and iteratively scans finer and finer simplicial partitions. The features of this partitioning algorithm is similar to a branch and bound algorithm with simplicial partitions for global optimization (Žilinskas, 2008). It is well known that branch and
bound algorithms require much less computations practically than in the worst case. Similarly, copositivity detection by simplicial partition is much faster than traditional linear algebraic algorithm (Žilinskas and Dür, 2011).

A copositive programming problem may be approached checking copositivity of several matrices built with different values of the variable and the solution is the extreme value for which the matrix is copositive. This was illustrated for the maximum clique problem (Bundfuss and Dür, 2008; Žilinskas and Dür, 2011). However, this approach has some shortcomings. It is necessary to check copositivity of several matrices. Although \( \varepsilon \)-copositivity may be checked, the tolerance is not directly related to the accuracy of solution of a copositive problem. Such an approach does not provide extremizer to the underlying problem, for example, it only provides the clique number, not the optimal clique. In this paper we propose the way to overcome these shortcomings.

2. Copositive Programming and Conditions for Copositivity

An \( n \times n \) real symmetric matrix \( A \) is called copositive if \( x^T Ax \geq 0 \) for all \( x \in \mathbb{R}^n_+ \), where \( \mathbb{R}^n_+ := \{ x \in \mathbb{R}^n : x_i \geq 0 \text{ for all } i \} \) denotes the non-negative orthant.

We consider copositive programming problem formulated as

\[
\max \{ y : Q - yD \in \mathcal{C} \},
\]

where \( y \in \mathbb{R} \) is a variable, \( Q \) and \( D \) are \( n \times n \) real symmetric matrices, and \( \mathcal{C} \) is the cone of copositive matrices. This problem is dual of a quadratic programming problem

\[
\min \langle Q, X \rangle \quad \text{s.t.} \quad \langle D, X \rangle = b, \quad X = xx^T, \quad x \geq 0.
\]

In the case \( D = J \), where \( J \) is the \( n \times n \) matrix with all ones, the problem may be formulated as

\[
\max \{ y : Q - yJ \in \mathcal{C} \},
\]

which is dual of the standard quadratic programming problem – optimization of a quadratic function over the standard simplex (Bomze, 2009).

The maximum clique problem may be formulated as the copositive programming problem (Bomze, 2009)

\[
\omega(G) = \min \{ t : tQ - J \in \mathcal{C} \},
\]

where \( \omega(G) \) is the clique number of a graph \( G \), \( t \in \mathbb{N} \) is a variable, and \( Q = J - A_G \) is a matrix derived from the adjacency matrix \( A_G \) of the graph \( G \). The goal of this copositive programming problem is to find the smallest value of \( t \) that \( tQ - J \) is copositive.

As it is shown by Bundfuss and Dür (2008)

\[ A \text{ is copositive} \iff x^T Ax \geq 0 \quad \text{for all } x \in \mathbb{R}^n_+ \text{ with } \|x\|_1 = 1. \]
This follows from the definition by using appropriate scaling. The set \( \Delta^S := \{ x \in \mathbb{R}_+^n : \|x\|_1 = 1 \} \) is called the standard simplex whose vertices are the unit vectors \( e_1, \ldots, e_n \).

The approach relies on the observation that \( A \) is copositive iff the quadratic form \( x^T Ax \geq 0 \) on the standard simplex. If \( v_1, \ldots, v_n \) denote the vertices of a simplex, we can write a point \( x \) in barycentric coordinates as \( x = \sum_{i=1}^n \lambda_i v_i \) with \( \sum_{i=1}^n \lambda_i = 1 \). This gives

\[
x^T Ax = \left( \sum_{i=1}^n \lambda_i v_i \right)^T A \left( \sum_{j=1}^n \lambda_j v_j \right) = \sum_{i,j=1}^n v_i^T A v_j \lambda_i \lambda_j.
\]

Hence, a necessary condition for \( x^T Ax \) to be nonnegative on the simplex is that

\[
v_i^T A v_j \geq 0 \quad \text{for all } i, j = 1, \ldots, n.
\] (4)

This condition can be refined by studying simplicial partitions of the standard simplex. As the partition gets finer, stronger and stronger necessary conditions are derived, which, in the limit, capture all strictly copositive matrices.

Therefore the following algorithm for copositivity detection can be formulated: start with \( P = \{ \Delta^S \} \). Check if \( v_i^T A v_j \geq 0 \) for all vertices \( v_i, v_j \) of all simplices in the partition \( P \). If yes, stop: the matrix is copositive. If no, select a simplex from \( \Delta \in P \) and subdivide it into \( \Delta = \Delta_1 \cup \Delta_2 \). Iterate this process.

The algorithm starts with the standard simplex whose vertices are the unit vectors \( e_1, \ldots, e_n \). Simplices are subdivided until either the candidate list is empty (i.e., \( A \) is copositive), or \( v_i^T A v < 0 \) for one vertex \( v \) of one of the simplices which means that \( A \) is not copositive.

Observe that simplices for which \( v_i^T A v_j \geq 0 \) for all pairs of vertices \( v_i, v_j \) can be discarded: it is clear that \( x^T Ax \geq 0 \) on those simplices. Therefore, we can prune the search tree at the nodes corresponding to these simplices.

For numerical reasons the notion of \( \varepsilon \)-copositivity was introduced by Bundfuss and Dür (2008): a matrix is called \( A \varepsilon \)-copositive, if \( x^T Ax \geq -\varepsilon \) for all \( x \in \Delta^S \).

It is possible to search for the solution of a copositive programming problem by checking copositivity of several matrices built with different values of the variable \( t \) and the solution is the extreme value for which the matrix is copositive. In the case of the maximum clique problem, start with \( t = 2 \), check copositivity of \( tQ - J \), and proceed with \( t = 3, 4, \ldots \) while \( tQ - J \) is not copositive. Unfortunately, copositivity of several matrices should be checked. Although this is not too bad when the variable is discrete (\( t \in \mathbb{N} \)), it may come to problems when the variable is continuous (\( y \in \mathbb{R} \)). Moreover, such an approach does not provide extremizer to the underlying problem, for example, it only provides the clique number, not the optimal clique. To overcome these shortcomings an algorithm for copositive programming is developed in the next section.

3. Algorithm for Copositive Programming by Simplicial Partition

Observe that for the copositive programming problem (1), the matrix to be copositive is \( A = Q - yD \). Let us assume that the matrix \( D \) is copositive. In this case the condition (4)
can be rewritten as

\[ v_i^T (Q - yJ)v_j = v_i^T Qv_j - yv_i^T Jv_j \geq 0, \]

and

\[ y \leq \frac{v_i^T Qv_j}{v_i^T Jv_j}. \]

Therefore, the matrix \( A \) is not copositive, if

\[ y > \frac{v_i^T Qv}{v_i^T Jv} \]

for any \( v \) of any simplex in the partition. Moreover, the matrix \( Q - (y - \varepsilon)J \) is copositive if

\[ y - \varepsilon \leq \frac{v_i^T Qv_j}{v_i^T Jv_j} \]

for all vertices \( v_i, v_j \) of all simplices in the partition \( \mathcal{P} \).

In the case of the standard quadratic programming (2), \( A = Q - yJ \) and the condition (4) can be rewritten as

\[ y \leq v_i^T Qv, \]

since \( v_i^T Jv_j = 1 \) for \( v_i, v_j \in \Delta^S \). The matrix \( A \) is not copositive, if

\[ y > v_i^T Qv \]

for one vertex \( v \) of one of the simplices. The matrix \( Q - (y - \varepsilon)J \) is copositive if

\[ y - \varepsilon \leq v_i^T Qv_j \]

for all vertices \( v_i, v_j \) of all simplices in the partition \( \mathcal{P} \).

In the case of the maximum clique problem (3), \( A = tQ - J \) and the condition (4) can be rewritten as

\[ v_i^T (tQ - J)v_j = tv_i^T Qv_j - v_i^T Jv_j = tv_i^T Qv_j - 1 \geq 0, \]

and

\[ t \geq \frac{1}{v_i^T Qv_j}. \]
$Q = J - A_G$ and $v$ do not have negative entries. Therefore, the matrix $A$ is not copositive, if

$$t < \frac{1}{v^T Qv}$$

for one vertex $v$ of one of the simplices. The matrix $(t + \varepsilon)Q - J$ is copositive if

$$t + \varepsilon \geq \frac{1}{v_i^T Qv_j}$$

for all vertices $v_i, v_j$ of all simplices in the partition $\mathcal{P}$. For the maximum clique problem $\varepsilon = 1$ can be set, as this is exactly the tolerance needed for the integer clique number. The last observation: if the previous condition holds, $\arg\max v / (v^T Qv)$ defines the solution of the underlying problem (non zero values of $v$ mean the node is in the clique), where $v$ is a vertex of a simplex.

These rewritten conditions give rise to the following algorithm for copositive programming (alternatively the standard quadratic programming and maximum clique formulations):

1. Start with $\mathcal{P} = \{\Delta^S\}$.
2. $y = \min(v^T Qv) / (v^T Dv)$ (alternatively $y = \min v^T Qv$ or $t = \max 1 / (v^T Qv)$), where $v$ is a vertex of a simplex in the partition.
3. Check, if $y - \varepsilon \leq (v_i^T Qv_j) / (v_i^T Dv_j)$ (alternatively $y - \varepsilon \leq v_i^T Qv_j$ or $t + \varepsilon \geq 1 / (v_i^T Qv_j)$) for all vertices $v_i, v_j$ of all simplices in the partition $\mathcal{P}$.
4. If yes, stop: $y$ (alternatively $t$) is the solution.
5. If no, select a simplex from $\Delta \in \mathcal{P}$ and subdivide it into $\Delta = \Delta^1 \cup \Delta^2$. Iterate the process.

Since we optimize $y = \min v^T Qv$ and $t = 1 / \min(v^T Qv)$ for the standard quadratic programming and maximum clique formulations, the algorithm is optimization of a quadratic function over the standard simplex in these cases what is not surprising. In a general formulation $y = \min(v^T Qv) / (v^T Dv)$ what is related to quadratic fractional programming (Yamamoto and Konno, 2007; Phillips, 2009).

There is some freedom in selecting the next simplex from the current partition $\mathcal{P}$. One option is to do it using a depth-first strategy, which leads to an algorithm displayed in Algorithm 1. As shown for example by Žilinskas and Žilinskas (2009), it is possible to develop a branch and bound algorithm with depth-first strategy which avoids storing of unbranched nodes of the search tree. Similarly, this idea is applied by Žilinskas and Dür (2009) and enables solution of much larger problems due to significantly smaller memory requirements.

The algorithm starts with the standard simplex whose vertices are the unit vectors $e_1, \ldots, e_n$. Simplices are subdivided until the candidate list is empty. Subdivision is performed similarly as by Bundfuss and Dür (2008), just the current extreme value of $y$ (alternatively $t$) is used to build $Q - yD$ (alternatively $Q - yJ$ or $tQ - J$) instead of pre-defined. Simplices for which $y - \varepsilon \leq (v_i^T Qv_j) / (v_i^T Dv_j)$ (alternatively $y - \varepsilon \leq v_i^T Qv_j$ for all vertices $v_i, v_j$ of all simplices in the partition $\mathcal{P}$.

1. Start with $\mathcal{P} = \{\Delta^S\}$.
2. $y = \min(v^T Qv) / (v^T Dv)$ (alternatively $y = \min v^T Qv$ or $t = \max 1 / (v^T Qv)$), where $v$ is a vertex of a simplex in the partition.
3. Check, if $y - \varepsilon \leq (v_i^T Qv_j) / (v_i^T Dv_j)$ (alternatively $y - \varepsilon \leq v_i^T Qv_j$ or $t + \varepsilon \geq 1 / (v_i^T Qv_j)$) for all vertices $v_i, v_j$ of all simplices in the partition $\mathcal{P}$.
4. If yes, stop: $y$ (alternatively $t$) is the solution.
5. If no, select a simplex from $\Delta \in \mathcal{P}$ and subdivide it into $\Delta = \Delta^1 \cup \Delta^2$. Iterate the process.
Algorithm 1: \( \max \{ y; Q - yD \in C \}, \max \{ y; Q - yJ \in C \}, \text{or} \min \{ t; tQ - J \in C \} \)

**Input:** \( n; Q \in \mathcal{S}; D \in \mathcal{S}; \varepsilon > 0 \)

1. \( V \leftarrow (e_1, \ldots, e_n); QS \leftarrow Q; DS \leftarrow D; l \leftarrow 0; y \leftarrow \infty \) (or \( t \leftarrow 0 \))
2. \( \text{while } l \geq 0 \) do
3. \( \text{for } i \in \{1, \ldots, n\} \) do
4. \( \text{if } y > QS_{ii}/DS_{ii} \) (alt. \( y > QS_{ii} \) or \( t < 1/QS_{ii} \)) then
5. \( y \leftarrow QS_{ii}/DS_{ii} \) (alt. \( y \leftarrow QS_{ii} \) or \( t \leftarrow 1/QS_{ii} \))
6. \( s \leftarrow V_{\{1, \ldots, n\}} \) (or \( \{ j \in \{1, \ldots, n\} \}; V_{ji} > 0 \})
7. \( \text{end if} \)
8. \( \text{end for} \)
9. \( \text{if } \forall i, j: y - \varepsilon \leq QS_{ij}/DS_{ij} \) (alt. \( y - \varepsilon \leq QS_{ij} \) or \( t + \varepsilon \geq 1/QS_{ij} \)) then
10. \( l \leftarrow l - 1 \)
11. \( \text{while } l \geq 0 \) and \( i_l^* = -1 \) do
12. \( V_{\{1, \ldots, n\}}^* \leftarrow V_{\{1, \ldots, n\}}^*; V_{\{1, \ldots, n\}}^* \leftarrow v_l^* \)
13. \( QS_{\{1, \ldots, n\}}^* \leftarrow q_l^*; QS_{\{1, \ldots, n\}}^* \leftarrow q_l^*; QS_{\{1, \ldots, n\}}^* \leftarrow q_l^* \)
14. \( DS_{\{1, \ldots, n\}}^* \leftarrow d_l^*; DS_{\{1, \ldots, n\}}^* \leftarrow d_l^*; l \leftarrow l - 1 \)
15. \( \text{end while} \)
16. \( \text{if } l \geq 0 \) then
17. \( V_{\{1, \ldots, n\}}^* \leftarrow V_{\{1, \ldots, n\}}^*; V_{\{1, \ldots, n\}}^* \leftarrow v_l^* \)
18. \( QS_{\{1, \ldots, n\}}^* \leftarrow q_l^*; QS_{\{1, \ldots, n\}}^* \leftarrow q_l^*; QS_{\{1, \ldots, n\}}^* \leftarrow q_l^* \)
19. \( DS_{\{1, \ldots, n\}}^* \leftarrow d_l^*; V_{\{1, \ldots, n\}}^* \leftarrow d_l^*; \)
20. \( DS_{\{1, \ldots, n\}}^* \leftarrow V_{\{1, \ldots, n\}}^*; DS_{\{1, \ldots, n\}}^* \leftarrow V_{\{1, \ldots, n\}}^*; \)
21. \( i_l^* \leftarrow -1; l \leftarrow l + 1 \)
22. \( \text{end if} \)
23. \( \text{else} \)
24. \( \text{choose } i, j \in \{1, \ldots, n\} \) so that \( \min QS_{ij}/DS_{ij} \) (min \( QS_{ij} \))
25. \( \alpha \leftarrow QS_{ii} - yDS_{ii} \) (alt. \( QS_{ii} - y \) or \( tQS_{ii} - 1 \));
26. \( \beta \leftarrow QS_{jj} - yDS_{jj} \) (alt. \( QS_{jj} - y \) or \( tQS_{jj} - 1 \));
27. \( \gamma \leftarrow QS_{ij} - yDS_{ij} \) (alt. \( QS_{ij} - y \) or \( tQS_{ij} - 1 \));
28. \( \text{if } \frac{\gamma}{\gamma - \alpha} \leq \frac{\beta}{\beta - \gamma} \) then
29. \( \lambda \leftarrow \max \left\{ \frac{\gamma}{\gamma - \alpha}, \min \left\{ \frac{\beta - \gamma}{\alpha - 2\gamma + \beta}, \frac{\beta}{\beta - \gamma} \right\} \right\} \)
30. \( \text{else} \)
31. \( \lambda \leftarrow \frac{\beta - \gamma}{\alpha - 2\gamma + \beta} \)
32. \( \text{end if} \)
33. \( \sigma \leftarrow \lambda V_{\{1, \ldots, n\}} + (1 - \lambda)V_{\{1, \ldots, n\}}; i_l^* \leftarrow i; j_l^* \leftarrow j \)
34. \( v_l^* \leftarrow V_{\{1, \ldots, n\}}; V_{\{1, \ldots, n\}} \leftarrow \sigma \)
35. \( q_l^* \leftarrow QS_{\{1, \ldots, n\}}; QS_{\{1, \ldots, n\}} \leftarrow V_{\{1, \ldots, n\}}; V_{\{1, \ldots, n\}} \leftarrow \sigma \)
36. \( d_l^* \leftarrow DS_{\{1, \ldots, n\}}; DS_{\{1, \ldots, n\}} \leftarrow V_{\{1, \ldots, n\}}; DS_{\{1, \ldots, n\}} \leftarrow \sigma \)
37. \( \text{end if} \)
38. \( \text{end while} \)

**Output:** \( y \) (or \( t \)), \( s \)
or \( t + \varepsilon \geq 1/(v_i^TQv_j) \) for all pairs of vertices \( v_i \) and \( v_j \) can be discarded. Therefore, we can prune our tree at the nodes corresponding to those simplices.

The matrix \( QS \) (and \( DS \) if needed) in Algorithm 1 stores the values \( v_i^TQv_j \) (and \( v_i^TDv_j \) if needed) so not to compute them for each simplex as only one row and one column is changed when subdividing a simplex. In the beginning the matrix \( QS \) is initialized by the matrix \( Q \) (and \( DS \) by \( D \) if needed) as the vertices of the initial simplex are the unit vectors. The matrix \( V \) is used to store the vertices of a simplex.

Storing the matrices \( V \) and \( QS \) (and \( DS \) if needed) of all candidate simplices would require a large amount of memory. We apply a depth-first selection strategy without storing the whole set of simplices, thus we store only the information required to restore \( V \) and \( QS \) (and \( DS \) if needed) when returning to the level \( l \). By the “level of a simplex” we mean the following: the initial simplex is of level 0, and if we perform a subdivision step, the level of the sub-simplices is by 1 greater than the level of the parent simplex.

\( i^*_l \) and \( j^*_l \) are the row and column numbers of the smallest negative element of \( QS \) and the numbers of vertices which are used in subdivision of a simplex. The vector \( v^*_l \) is used to restore the vertex of the simplex, whereas the vector \( q^*_l \) is used to restore row and column of the matrix \( QS \), and \( d^*_l \) is used to restore these of the matrix \( DS \).

### 4. Experimental Results

In this section we present the results of our experiments with the proposed algorithm for copositive programming. As test instances we use example problems, the maximum clique instances from several generators available through the web, as well as from the collection of benchmark problems of the Second DIMACS Challenge.

#### 4.1. Example Problems

Let us consider a copositive programming problem

\[
\max \left\{ \left. y : \begin{pmatrix} -3 & 0.5 \\ 0.5 & -4 \end{pmatrix} - y \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right| \in C \right\}.
\]

It can be solved by using our algorithm finding the optimum \( y = -4 \) at \( x = (0, 1) \). The problem is similar to an example problem of fractional programming from Yamamoto and Konno (2007):

\[
\max f(x) = \frac{3x_1^2 - x_1x_2 + 4x_2^2}{x_1^2 + x_2^2} \quad \text{s.t.} \quad x_1 + x_2 = 1, \ x_1 \geq 0, \ x_2 \geq 0,
\]
formulated as a minimization problem

$$\min_{x} \frac{x^T \begin{pmatrix} -3 & 0.5 \\ 0.5 & -4 \end{pmatrix} x}{x^T \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} x} \text{ s.t. } x \in \Delta^S, \ x \geq 0.$$ 

We consider some example problems from Bomze and de Klerk (2002). These are copositive programming problems $\max \{y: Q - yJ \in \mathcal{C}\}$ with various matrices:

$$Q_1 = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix},$$

$$Q_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$Q_3 = \begin{pmatrix} -14 & -15 & -16 & 0 & 0 \\ -15 & -14 & -12.5 & -22.5 & -15 \\ -16 & -12.5 & -10 & -26.5 & -16 \\ 0 & -22.5 & -26.5 & 0 & 0 \\ 0 & -15 & -16 & 0 & -14 \end{pmatrix},$$

$$Q_4 = \begin{pmatrix} 0.9044 & 0.1054 & 0.5140 & 0.3322 & 0 \\ 0.1054 & 0.8715 & 0.7385 & 0.5866 & 0.9751 \\ 0.5140 & 0.7385 & 0.6936 & 0.5368 & 0.8086 \\ 0.3322 & 0.5866 & 0.5368 & 0.5633 & 0.7478 \\ 0 & 0.9751 & 0.8086 & 0.7478 & 1.2932 \end{pmatrix}.$$ 

Several of the problems have maximum clique formulation therefore we can solve them using all versions of the algorithm. Results of experimental investigation are given in Table 1. All problems are solved in a fraction of a second. These are relatively easy problems for the algorithm. The tolerance $\varepsilon = 1$ enables faster solution of one of the
maximum clique problem. The solution times are lower than ones reported by Bundfuss and Dür (2009).

4.2. Maximum Clique Problems from Problem Generators

We used maximum clique instances to explore the size of problems that can be solved by our approach. To this purpose, we generated problems of various sizes using the graph generators available at FTP site ftp://dimacs.rutgers.edu/pub/challenge/graph/contributed/ of the Second DIMACS Challenge (Johnson and Trick, 1996).

The graphs named Brock$n$ are graphs with hidden cliques of different sizes (Brockington and Culberson, 1996). Jagota$n$ are graphs with five random cliques and are described by Jagota (1992). The Sanchis$n$ graphs are described by Sanchis and Jagota (1996).

We used the graph generators described by Hasselberg et al. (1993) to generate various other graphs: c-fat$n$-1 graphs with $n$ nodes are based on fault diagnosis; Hamming$m$-$d$ graphs have $2^m$ nodes where two nodes are adjacent if the Hamming distance between their binary codes is at least $d$; Johnson$m$-$w$-$d$ graphs have $\binom{m}{w}$ nodes; Keller2 graph is based on Keller’s cube-tillings (Lagarias and Shor, 1992).

The results of experimental investigation are displayed in Table 2. The column $t$ gives the optimal value of the variable returned by the algorithm, $n_{simp}$ gives the number of analyzed simplices, $\max l$ states the maximal level of simplices, time in seconds taken by the algorithm is shown in the last column.

Differently from Žilinskas and Dür (2009) one run of the algorithm is required to solve a copositive programming problem. Moreover, the algorithm for copositive programming requires less computations than the algorithm to check copositivity of $\omega(G)Q - J$, because of the tolerance $\epsilon = 1$. This allows us to solve larger problems: some problems have been solved while copositivity check of $\omega(G)Q - J$ has not finished during allowed time. The largest problems solved with the current algorithm are of size 24.
The results show that the clique number has been identified in all problem instances. Even with $\varepsilon = 1$ the estimates of $t$ are equal to the clique numbers. This is due to improved subdivision – the algorithm based on bisection through the longest edge requires more computations and provides solutions not equal to the maximum clique number although within the given tolerance $\varepsilon = 1$. Last, but not least, the algorithm identifies not only the maximum clique number, but the maximum clique as well. It has been checked, that all identified maximum cliques exist in the graphs.

4.3. Benchmark Problems from the Second DIMACS Challenge

In this section we report results for various benchmark problems from the Second DIMACS Challenge which are available through 
ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique/. In these instances, neither of
The copositive programming problems has been solved to optimality within the maximum allowed time. This shows that these benchmark problems are very hard.

However, the proposed algorithm provides lower bound of the solution and the corresponding clique even if there is no guarantee that the maximum clique number is found. A summary of the results for the DIMACSs benchmark problems is given in Table 3. Similarly as in the previous table, the column $t$ gives the estimate of the optimal value of variable returned by the algorithm, $n_{simp}$ gives the number of analyzed simplices, $\text{max } l$ states the maximal level of simplices, allowed time (in seconds) is indicated in the last column.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Nodes</th>
<th>Edges</th>
<th>Clique number</th>
<th>$t$</th>
<th>$n_{simp}$</th>
<th>$\text{max } l$</th>
<th>Allowed time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brock200_1</td>
<td>200</td>
<td>14834</td>
<td>21</td>
<td>16</td>
<td>708261</td>
<td>1989</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Brock200_2</td>
<td>200</td>
<td>9876</td>
<td>12</td>
<td>10</td>
<td>710547</td>
<td>980</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Brock200_3</td>
<td>200</td>
<td>12048</td>
<td>15</td>
<td>11</td>
<td>699520</td>
<td>1330</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Brock200_4</td>
<td>200</td>
<td>13089</td>
<td>17</td>
<td>14</td>
<td>708598</td>
<td>1504</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Brock400_1</td>
<td>400</td>
<td>59723</td>
<td>27</td>
<td>20</td>
<td>18115</td>
<td>4984</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Brock400_2</td>
<td>400</td>
<td>59786</td>
<td>29</td>
<td>20</td>
<td>17955</td>
<td>4946</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Brock400_3</td>
<td>400</td>
<td>59681</td>
<td>31</td>
<td>20</td>
<td>18027</td>
<td>4920</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Brock400_4</td>
<td>400</td>
<td>59765</td>
<td>33</td>
<td>20</td>
<td>18162</td>
<td>4978</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Brock800_1</td>
<td>800</td>
<td>207505</td>
<td>23</td>
<td>17</td>
<td>19290</td>
<td>8376</td>
<td>$10^5$</td>
</tr>
<tr>
<td>Brock800_2</td>
<td>800</td>
<td>208166</td>
<td>24</td>
<td>17</td>
<td>19317</td>
<td>8482</td>
<td>$10^5$</td>
</tr>
<tr>
<td>Brock800_3</td>
<td>800</td>
<td>207333</td>
<td>25</td>
<td>17</td>
<td>19275</td>
<td>8385</td>
<td>$10^5$</td>
</tr>
<tr>
<td>Brock800_4</td>
<td>800</td>
<td>207643</td>
<td>26</td>
<td>16</td>
<td>19379</td>
<td>8389</td>
<td>$10^5$</td>
</tr>
<tr>
<td>Hamming6-2</td>
<td>64</td>
<td>1824</td>
<td>32</td>
<td>32</td>
<td>15805366</td>
<td>883</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Hamming6-4</td>
<td>64</td>
<td>704</td>
<td>4</td>
<td>4</td>
<td>16612788</td>
<td>112</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Hamming8-2</td>
<td>256</td>
<td>31616</td>
<td>128</td>
<td>128</td>
<td>128307</td>
<td>14441</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Hamming8-4</td>
<td>256</td>
<td>20864</td>
<td>16</td>
<td>16</td>
<td>128632</td>
<td>1628</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Hamming10-2</td>
<td>1024</td>
<td>518656</td>
<td>512</td>
<td>512</td>
<td>3889</td>
<td>3899</td>
<td>$10^5$</td>
</tr>
<tr>
<td>Hamming10-4</td>
<td>1024</td>
<td>434176</td>
<td>40</td>
<td>33</td>
<td>3857</td>
<td>3857</td>
<td>$10^5$</td>
</tr>
<tr>
<td>Johnson8-2-4</td>
<td>28</td>
<td>210</td>
<td>4</td>
<td>4</td>
<td>230691654</td>
<td>56</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Johnson8-4-4</td>
<td>70</td>
<td>1855</td>
<td>14</td>
<td>14</td>
<td>14850322</td>
<td>405</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Johnson16-2-4</td>
<td>120</td>
<td>5460</td>
<td>8</td>
<td>8</td>
<td>3202090</td>
<td>488</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Johnson32-2-4</td>
<td>496</td>
<td>107880</td>
<td>16</td>
<td>16</td>
<td>7818</td>
<td>4308</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Keller4</td>
<td>171</td>
<td>9435</td>
<td>11</td>
<td>8</td>
<td>696414</td>
<td>794</td>
<td>$10^4$</td>
</tr>
<tr>
<td>Keller5</td>
<td>776</td>
<td>225990</td>
<td>27</td>
<td>20</td>
<td>19377</td>
<td>8608</td>
<td>$10^5$</td>
</tr>
<tr>
<td>Keller6</td>
<td>3361</td>
<td>4619898</td>
<td>$\geq$ 59</td>
<td>37</td>
<td>1493</td>
<td>1493</td>
<td>$10^6$</td>
</tr>
<tr>
<td>MANN_a9</td>
<td>45</td>
<td>918</td>
<td>16</td>
<td>16</td>
<td>57018692</td>
<td>320</td>
<td>$10^4$</td>
</tr>
<tr>
<td>MANN_a27</td>
<td>378</td>
<td>70551</td>
<td>126</td>
<td>121</td>
<td>242995</td>
<td>23055</td>
<td>$10^5$</td>
</tr>
<tr>
<td>MANN_a45</td>
<td>1035</td>
<td>533115</td>
<td>345</td>
<td>336</td>
<td>74763</td>
<td>74763</td>
<td>$10^5$</td>
</tr>
<tr>
<td>MANN_a81</td>
<td>3321</td>
<td>5506380</td>
<td>$\geq$ 1100</td>
<td>302</td>
<td>1550</td>
<td>1550</td>
<td>$10^6$</td>
</tr>
</tbody>
</table>
Although the algorithm provides a lower bound for the maximum clique number, in several cases it coincides with the maximum clique number. Some lower bounds are better than those found using copositivity checks by simplicial partition (Bundfuss and Dür, 2008; Žilinskas and Dür, 2011). It has been checked, that all identified cliques exist in the graphs.

5. Conclusions

We proposed a simplicial partition algorithm for copositive programming. The algorithm solves copositive programming problems of size up to 24 in a reasonable time. For larger problems, the algorithm does not finish with guaranteed optimal solution during allowed time, but it gives bounds for solutions of problems of size up to several thousands. The approach is faster than checking copositivity of several matrices built with various values of variable.

Acknowledgments. This research was funded by a grant (No. MIP-108/2010) from the Research Council of Lithuania.

References

Copositive Programming by Simplicial Partition


J. Žilinskas is a principal researcher in Systems Analysis Department at the Institute of Mathematics and Informatics, Vilnius University, Lithuania. He is a member of editorial boards of Central European Journal of Computer Science, Central European Journal of Engineering, Computational Management Science, Informatica, Mathematical Modelling and Analysis, and Optimization Letters. His research interests include global optimization, parallel computing, data analysis and visualization.
Kopozityvusis programavimas simpleksu skaidymu

Julius ŽILINSKAS

Kopozityvumas svarbus kombinatoriniame ir kvadratinime programavime, nes kombinatorinio ir kvadratinio programavimo uždavinius galima tiksliai suformuluoti tiesinės funkcijos optimizavimui kopozityvių matricų kūgje. Kopozityviojo programavimo uždaviniai gali būti sprendžiami tikrinant matricų, sugeneruotų su skirtingomis kintamojo reikšmėmis, kopozityvumą, o sprendinys yra ekstremali reikšmė, kuria matrica yra kopozityvė. Tačiau toks sprendimo būdas turi keletą trūkumų. Trūkumams išvengti, šiame straipsnyje yra pasiūlytas simpleksų skaidymo algoritmas kopozityviam programavimui. Algoritmas tiriamas eksperimentiškai sprendžiant aibe testinių uždavinių.