INFORMATICA, 2011, Vol. 22, No. 4, 601–614

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 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 ε-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 $x^T Ax \geqslant 0$ for all $x \in \mathbb{R}^n_+$, where $\mathbb{R}^n_+ := \{x \in \mathbb{R}^n : x_i \geqslant 0 \text{ for all } i\}$ denotes the non-negative orthant.

We consider copositive programming problem formulated as

$$\max\{y\colon Q - yD \in \mathcal{C}\}, \tag{1}$$

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

$$\min\langle Q, X\rangle \quad \text{s.t. } \langle D, X\rangle = b, \qquad X = xx^T,\ x \geqslant 0.$$

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

$$\max\{y\colon Q - yJ \in \mathcal{C}\}, \tag{2}$$

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)

$$\omega(G) = \min\{t\colon tQ - J \in \mathcal{C}\}, \tag{3}$$

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)

$$A \text{ is copositive} \quad \Leftrightarrow \quad x^T Ax \geqslant 0 \quad \text{for all } x \in \mathbb{R}^n_+ \text{ with } \|x\|_1 = 1.$$

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 $x^T Ax \geqslant 0$ 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

$$v_i^T A v_j \geqslant 0 \quad \text{for all } i, j = 1, \ldots, n. \tag{4}$$

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 $\mathcal{P} = \{\Delta^S\}$. Check if $v_i^T A v_j \geqslant 0$ for all vertices v_i, v_j of all simplices in the partition $\mathcal{P}$. If yes, stop: the matrix is copositive. If no, select a simplex from $\Delta \in \mathcal{P}$ 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 $v^T Av < 0$ for one vertex v of one of the simplices which means that A is not copositive.

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

For numerical reasons the notion of ε-copositivity was introduced by Bundfuss and Dür (2008): a matrix is called A *ε-copositive*, if $x^T Ax \geqslant -\varepsilon$ 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 (y or 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

$$v_i^T(Q - yD)v_j = v_i^T Q v_j - y v_i^T D v_j \geqslant 0,$$

and

$$y \leqslant \frac{v_i^T Q v_j}{v_i^T D v_j}.$$

Therefore, the matrix A is not copositive, if

$$y > \frac{v^T Q v}{v^T D v}$$

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

$$y - \varepsilon \leqslant \frac{v_i^T Q v_j}{v_i^T D v_j}$$

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

$$y \leqslant v_i^T Q v_j,$$

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

$$y > v^T Q v$$

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

$$y - \varepsilon \leqslant v_i^T Q v_j$$

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

$$v_i^T(tQ - J)v_j = t v_i^T Q v_j - v_i^T J v_j = t v_i^T Q v_j - 1 \geqslant 0,$$

and

$$t \geqslant \frac{1}{v_i^T Q v_j}.$$

$Q = J - A_G$ and v do not have negative entries. Therefore, the matrix A is not copositive, if

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

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

$$t + \varepsilon \geqslant \frac{1}{v_i^T Q v_j}$$

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, $\arg\max_v 1/(v^T Q v)$ 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. $y = \min(v^T Q v)/(v^T D v)$ (alternatively $y = \min v^T Q v$ or $t = \max 1/(v^T Q v)$), where v is a vertex of a simplex in the partition.
3. Check, if $y - \varepsilon \leqslant (v_i^T Q v_j)/(v_i^T D v_j)$ (alternatively $y - \varepsilon \leqslant v_i^T Q v_j$ or $t + \varepsilon \geqslant 1/(v_i^T Q v_j)$) 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 $y = \min v^T Q v$ and $t = 1/\min(v^T Q v)$ 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 $y = \min(v^T Q v)/(v^T D v)$ 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 predefined. Simplices for which $y - \varepsilon \leqslant (v_i^T Q v_j)/(v_i^T D v_j)$ (alternatively $y - \varepsilon \leqslant v_i^T Q v_j$

Algorithm 1: $\max\{y\colon Q - yD \in \mathcal{C}\}$, $\max\{y\colon Q - yJ \in \mathcal{C}\}$, or $\min\{t\colon tQ - J \in \mathcal{C}\}$

Input: n; $Q \in \mathcal{S}$; $D \in \mathcal{S}$; $\varepsilon > 0$

$V \leftarrow (e_1, \ldots, e_n)$; $QS \leftarrow Q$; $DS \leftarrow D$; $l \leftarrow 0$; $y \leftarrow \infty$ (or $t \leftarrow 0$)
while $l \geqslant 0$ **do**
  for $i \in \{1, \ldots, n\}$ **do**
    if $y > QS_{ii}/DS_{ii}$ (alt. $y > QS_{ii}$ or $t < 1/QS_{ii}$) **then**
      $y \leftarrow QS_{ii}/DS_{ii}$ (alt. $y \leftarrow QS_{ii}$ or $t \leftarrow 1/QS_{ii}$)
      $s \leftarrow V_{\{1,\ldots,n\}i}$ (or $\{j \in \{1, \ldots, n\}\colon V_{ji} > 0\}$)
    end if
  end for
  if $\forall i, j\colon y - \varepsilon \leqslant QS_{ij}/DS_{ij}$ (alt. $y - \varepsilon \leqslant QS_{ij}$ or $t + \varepsilon \geqslant 1/QS_{ij}$) **then**
    $l \leftarrow l - 1$
    while $l \geqslant 0$ and $i_l^* = -1$ **do**
      $V_{\{1,\ldots,n\}j_l^*} \leftarrow v_l^*$; $QS_{\{1,\ldots,n\}j_l^*} \leftarrow q_l^*$; $QS_{j_l^*\{1,\ldots,n\}} \leftarrow q_l^*$
      $DS_{\{1,\ldots,n\}j_l^*} \leftarrow d_l^*$; $DS_{j_l^*\{1,\ldots,n\}} \leftarrow d_l^*$; $l \leftarrow l - 1$
    end while
    if $l \geqslant 0$ **then**
      $V_{\{1,\ldots,n\}i_l^*} \leftrightarrow V_{\{1,\ldots,n\}j_l^*}$; $V_{\{1,\ldots,n\}i_l^*} \leftrightarrow v_l^*$
      $QS_{\{1,\ldots,n\}i_l^*} \leftarrow q_l^*$; $QS_{i_l^*\{1,\ldots,n\}} \leftarrow q_l^*$ $q_l^* \leftarrow QS_{\{1,\ldots,n\}j_l^*}$
      $QS_{\{1,\ldots,n\}j_l^*} \leftarrow V^T Q V_{\{1,\ldots,n\}j_l^*}$; $QS_{j_l^*\{1,\ldots,n\}} \leftarrow V^T Q V_{\{1,\ldots,n\}j_l^*}$
      $DS_{\{1,\ldots,n\}i_l^*} \leftarrow d_l^*$; $DS_{i_l^*\{1,\ldots,n\}} \leftarrow d_l^*$; $d_l^* \leftarrow DS_{\{1,\ldots,n\}j_l^*}$
      $DS_{\{1,\ldots,n\}j_l^*} \leftarrow V^T D V_{\{1,\ldots,n\}j_l^*}$; $DS_{j_l^*\{1,\ldots,n\}} \leftarrow V^T D V_{\{1,\ldots,n\}j_l^*}$
      $i_l^* \leftarrow -1$; $l \leftarrow l + 1$
    end if
  else
    choose $i, j \in \{1, \ldots, n\}$ so that $\min QS_{ij}/DS_{ij}$ ($\min QS_{ij}$)
    $\alpha \leftarrow QS_{ii} - yDS_{ii}$ (alt. $QS_{ii} - y$ or $tQS_{ii} - 1$);
    $\beta \leftarrow QS_{jj} - yDS_{jj}$ (alt. $QS_{jj} - y$ or $tQS_{jj} - 1$);
    $\gamma \leftarrow QS_{ij} - yDS_{ij}$ (alt. $QS_{ij} - y$ or $tQS_{ij} - 1$)
    if $\frac{\gamma}{\gamma-\alpha} \leqslant \frac{\beta}{\beta-\gamma}$ **then**
      $\lambda \leftarrow \max\left\{\frac{\gamma}{\gamma-\alpha}, \min\left\{\frac{\beta-\gamma}{\alpha-2\gamma+\beta}, \frac{\beta}{\beta-\gamma}\right\}\right\}$
    else
      $\lambda \leftarrow \frac{\beta-\gamma}{\alpha-2\gamma+\beta}$
    end if
    $\sigma \leftarrow \lambda V_{\{1,\ldots,n\}i} + (1-\lambda) V_{\{1,\ldots,n\}j}$; $i_l^* \leftarrow i$; $j_l^* \leftarrow j$
    $v_l^* \leftarrow V_{\{1,\ldots,n\}i}$; $V_{\{1,\ldots,n\}i} \leftarrow \sigma$
    $q_l^* \leftarrow QS_{\{1,\ldots,n\}i}$; $QS_{\{1,\ldots,n\}i} \leftarrow V^T Q\sigma$; $QS_{i\{1,\ldots,n\}} \leftarrow V^T Q\sigma$
    $d_l^* \leftarrow DS_{\{1,\ldots,n\}i}$; $DS_{\{1,\ldots,n\}i} \leftarrow V^T D\sigma$; $DS_{i\{1,\ldots,n\}} \leftarrow V^T D\sigma$; $l \leftarrow l + 1$
  end if
end while

Output: y (or t), s

or $t + \varepsilon \geqslant 1/(v_i^T Q v_j)$) 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 $v_i^T Q v_j$ (and $v_i^T D v_j$ 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

$$\max \left\{ y\colon \begin{pmatrix} -3 & 0.5 \\ 0.5 & -4 \end{pmatrix} - y \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \in \mathcal{C} \right\}.$$

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):

$$\max f(x) = \frac{3x_1^2 - x_1 x_2 + 4x_2^2}{x_1^2 + x_2^2} \quad \text{s.t. } x_1 + x_2 = 1,\ x_1 \geqslant 0,\ x_2 \geqslant 0,$$

formulated as a minimization problem

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

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

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

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

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

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

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

Table 1

Experimental results on example problems

	$\max\{y\colon Q - yD \in \mathcal{C}\}, \varepsilon = 10^{-6}$				$\max\{y\colon Q - yJ \in \mathcal{C}\}, \varepsilon = 10^{-6}$			
Q	y	n_{simp}	$\max l$	Time (s)	y	n_{simp}	$\max l$	Time (s)
Q_1	0.5	19	4	0.0	0.5	19	4	0.0
Q_2	0.3333	71679	22	0.53	0.3333	71679	22	0.35
Q_3	−16.3333	23	6	0.0	−16.3333	23	6	0.0
Q_4	0.4839	89	13	0.0	0.4839	89	13	0.0

	$\min\{t\colon tQ - J \in \mathcal{C}\}, \varepsilon = 1$			
Q	t	n_{simp}	$\max l$	Time (s)
Q_1	2	19	4	0.0
Q_2	3	31301	21	0.18

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 $\varepsilon = 1$. 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.

Table 2
Experimental results on generated maximum clique problems

Graph	Nodes	Edges	Clique number	t	n_{simp}	$\max l$	Time (s)
Brock14	14	51	5	5	1873355	33	15.73
Brock16	16	59	5	5	3929357	36	43.71
Brock18	18	78	5	5	100442543	50	1487.45
Brock20	20	95	5	5	936595215	52	18154.00
Jagota14	14	31	6	6	94425	24	0.71
Jagota16	16	57	8	8	14659409	43	157.13
Jagota18	18	84	10	10	2366989393	67	34478.00
Morgen14	14	50	5	5	2565871	37	20.97
Morgen16	16	59	5	5	1963895	36	21.39
Morgen18	18	60	5	5	10032327	38	141.71
Morgen20	20	67	5	5	298940103	43	5925.00
Morgen22	22	68	5	5	91131959	43	2109.27
Morgen24	24	69	5	5	80165637	45	2280.00
Sanchis14	14	50	5	5	1184933	31	10.09
Sanchis16	16	50	5	5	560033	28	6.05
Sanchis18	18	50	5	5	2578523	28	35.80
Sanchis20	20	50	5	5	17294981	33	315.58
Sanchis22	22	50	5	5	40407507	34	905.79
Sanchis24	24	50	5	5	172426589	38	4885.00
c-fat14-1	14	52	6	6	29692167	39	250.27
c-fat16-1	16	69	7	7	2071185519	52	23524.00
c-fat18-1	18	72	6	6	7515802065	53	109344.00
Hamming4-4	16	8	2	2	511	8	0.00
Johnson6-2-4	15	45	3	3	88483	21	0.82
Johnson6-4-4	15	45	3	3	90013	21	0.85
Johnson7-2-4	21	105	3	3	123008083	33	2829.00
Keller2	16	40	2	2	10329	15	0.10

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

Table 3
Results for the DIMACS benchmark problems. The algorithm was run for the maximum allowed time

Graph	Nodes	Edges	Clique number	t	n_{simp}	$\max l$	Allowed time (s)
Brock200_1	200	14834	21	16	708261	1989	10^4
Brock200_2	200	9876	12	10	710547	980	10^4
Brock200_3	200	12048	15	11	699520	1330	10^4
Brock200_4	200	13089	17	14	708598	1504	10^4
Brock400_1	400	59723	27	20	18115	4984	10^4
Brock400_2	400	59786	29	20	17955	4946	10^4
Brock400_3	400	59681	31	20	18027	4920	10^4
Brock400_4	400	59765	33	20	18162	4978	10^4
Brock800_1	800	207505	23	17	19290	8376	10^5
Brock800_2	800	208166	24	17	19317	8482	10^5
Brock800_3	800	207333	25	17	19275	8385	10^5
Brock800_4	800	207643	26	16	19379	8389	10^5
Hamming6-2	64	1824	32	32	15805366	883	10^4
Hamming6-4	64	704	4	4	16612788	112	10^4
Hamming8-2	256	31616	128	128	128307	14441	10^4
Hamming8-4	256	20864	16	16	128632	1628	10^4
Hamming10-2	1024	518656	512	512	3899	3899	10^5
Hamming10-4	1024	434176	40	33	3857	3857	10^5
Johnson8-2-4	28	210	4	4	230691654	56	10^4
Johnson8-4-4	70	1855	14	14	14850322	405	10^4
Johnson16-2-4	120	5460	8	8	3202090	488	10^4
Johnson32-2-4	496	107880	16	16	7818	4308	10^4
Keller4	171	9435	11	8	696414	794	10^4
Keller5	776	225990	27	20	19377	8608	10^5
Keller6	3361	4619898	$\geqslant 59$	37	1493	1493	10^6
MANN_a9	45	918	16	16	57018692	320	10^4
MANN_a27	378	70551	126	121	242995	23055	10^5
MANN_a45	1035	533115	345	336	74763	74763	10^6
MANN_a81	3321	5506380	$\geqslant 1100$	302	1550	1550	10^6

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, $\max l$ states the maximal level of simplices, allowed time (in seconds) is indicated in the last column.

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 thouse 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

Andersson, L.E., Chang, G., Elfving, T. (1995). Criteria for copositive matrices using simplices and barycentric coordinates. *Linear Algebra and its Applications*, 220(15), 9–30.

Bomze, I.M. (2009). Copositive optimization. In: Floudas, C.A., Pardalos, P.M. (Eds.), *Encyclopedia of Optimization*, 2nd edn. Springer, New York, pp. 561–564.

Bomze, I.M., De Klerk, E. (2002). Solving standard quadratic optimization problems via linear, semidefinite and copositive programming. *Journal of Global Optimization*, 24(2), 163–185.

Bomze, I.M., Dür, M., de Klerk, E., Roos, C., Quist, A.J., Terlaky, T. (2000). On copositive programming and standard quadratic optimization problems. *Journal of Global Optimization*, 18(4), 301–320.

Brockington, M., Culberson, J.C. (1996). Camouflaging independent sets in quasi-random graphs. In: Johnson, D.S., Trick, M.A. (Eds.), *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 26. AMS, Providence, pp. 75–88.

Bundfuss, S., Dür, M. (2008). Algorithmic copositivity detection by simplicial partition. *Linear Algebra and its Applications*, 428(7), 1511–1523.

Bundfuss, S., Dür, M. (2009). An adaptive linear approximation algorithm for copositive programs. *SIAM Journal on Optimization*, 20(1), 30–53.

Burer, S. (2009). On the copositive representation of binary and continuous nonconvex quadratic programs. *Mathematical Programming*, 120(2), 479–495.

de Klerk, E., Pasechnik, D.V. (2002). Approximation of the stability number of a graph via copositive programming. *SIAM Journal on Optimization*, 12(4), 875–892.

Dukanovic, I., Rendl, F. (2010). Copositive programming motivated bounds on the stability and the chromatic numbers. *Mathematical Programming*, 121(2), 249–268.

Dür, M. (2010). Copositive programming – a survey. In: Diehl, M., Glineur, F., Jarlebring, E., Michiels, W. (Eds.), *Recent Advances in Optimization and its Applications in Engineering*. Springer, Berlin, pp. 3–20.

Hasselberg, J., Pardalos, P.M., Vairaktarakis, G. (1993). Test case generators and computational results for the maximum clique problem. *Journal of Global Optimization*, 3(4), 463–482.

Jagota, A. (1992). *Approximating max-clique with a Hopfield network*. Tech. rep. 92-33, University at Buffalo, Computer Science, available from author only.

Johnson, D.S., Trick, M.A. (Eds.) (1996). *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 26, AMS, Providence.

Kaplan, W. (2000). A test for copositive matrices. *Linear Algebra and its Applications*, 313(1–3), 203–206.

Kaplan, W. (2001). A copositivity probe. *Linear Algebra and its Applications*, 337(1–3), 237–251.

Lagarias, J.C., Shor, P.W. (1992). Keller's cube-tiling conjecture is false in high dimensions. *Bulletin of the Americal Mathematical Society*, 27(2), 279–283.

Murty, K.G., Kabadi, S.N. (1987). Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39(2), 117–129.

Phillips, A.T. (2009). Quadratic fractional programming: Dinkelbach method. In: Floudas, C.A., Pardalos, P.M. (Eds.), *Encyclopedia of Optimization*, 2nd edn. Springer, New York, pp. 3149–3153.

Povh, J., Rendl, F. (2007). A copositive programming approach to graph partitioning. *SIAM Journal on Optimization*, 18(1), 223–241.

Povh, J., Rendl, F. (2009). Copositive and semidefinite relaxations of the quadratic assignment problem. *Discrete Optimization*, 6(3), 231–241.

Sanchis, L.A., Jagota, A. (1996). Some experimental and theoretical results on test case generators for the maximum clique problem. *INFORMS Journal on Computing*, 8(2), 87–102.

Väliaho, H. (1986). Criteria for copositive matrices. *Linear Algebra and its Applications*, 81, 19–34.

Yamamoto, R., Konno, H. (2007). An efficient algorithm for solving convex-convex quadratic fractional programs. *Journal of Optimization Theory and Applications*, 133(2), 241–255.

Yang, S., Li, X. (2009). Algorithms for determining the copositivity of a given symmetric matrix. *Linear Algebra and its Applications*, 430(2–3), 609–618.

Žilinskas, A., Žilinskas, J. (2009). Branch and bound algorithm for multidimensional scaling with city-block metric. *Journal of Global Optimization*, 43(2–3), 357–372.

Žilinskas, J. (2008). Branch and bound with simplicial partitions for global optimization. *Mathematical Modelling and Analysis*, 13(1), 145–159.

Žilinskas, J., Dür, M. (2011). Depth-first simplicial partition for copositivity detection, with an application to MaxClique. *Optimization Methods and Software*, 26(3), 499–510.

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 simpleksų skaidymu

Julius ŽILINSKAS

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