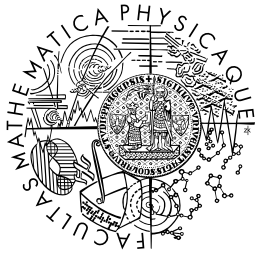

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Martin Nemjo

Testing copositivity by an interval branch-and-bound approach

Department of Applied Mathematics

Supervisor of the bachelor thesis: prof. Mgr. Milan Hladík, Ph.D

Study programme: Computer Science

Prague 2024

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to wholeheartedly thank prof. Mgr. Milan Hladík, Ph.D for his extremely valuable time and insights into this project. He dedicated a lot of time, was very patient, and very quick to respond to any problem, which made this work possible. I would also like to thank my girlfriend, my family and my friends, for supporting me during this thesis.

Title: Testing copositivity by an interval branch-and-bound approach

Author: Martin Nemjo

Department: Department of Applied Mathematics

Supervisor: prof. Mgr. Milan Hladík, Ph.D, Department of Applied Mathematics

Abstract: In this thesis, we analyze the copositivity problem, determining whether a given matrix is copositive. Despite its NP-hard nature, a variety of testing approaches exists. We explore the mathematical properties and necessary and sufficient conditions for copositivity. Our work includes implementing a branch-and-bound algorithm in MATLAB with the INTLAB library. The algorithm uses three different bounding techniques based on interval analysis, spectral properties, and quadratic programming. We briefly address challenges encountered, such as random matrix generation, and propose effective strategies to mitigate them. Additionally, we create a modular and extensible testing environment for possible future extensions. Finally, we compare all considered methods and the practical performance of our method to a simplex-based branch-and-bound algorithm. In testing, we show that our method outperforms the simplex-based one on copositive matrices.

Keywords: copositivity, copositive matrix, quadratic programming, branch-and-bound, interval analysis, copositive optimization

Název práce: Testování kopositivity pomocí intervalové branch-and-bound metody

Autor: Martin Nemjo

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: prof. Mgr. Milan Hladík, Ph.D, Katedra aplikované matematiky

Abstrakt: V tejto práci analyzujeme problém kopozitivity, teda určenie, či je daná matica kopozitívna. Napriek tomu, že ide o NP-ťažký problém, existuje množstvo testovacích metód. Skúmame matematické vlastnosti, nevyhnutné a postačujúce podmienky pre kopozitivitu. Naša práca zahŕňa implementáciu branch-and-bound algoritmu v MATLABe s balíčkom INTLAB. Tento algoritmus používa tri rozličné metódy na odhadovanie, založené na intervalovej analýze, spektrálnych vlastnostiach a kvadratickom programovaní. Stručne spomenieme problémy, s ktorými sme sa stretli, ako je napríklad generovanie náhodných matíc na testovanie, a navrhneme efektívne spôsoby na ich mitigáciu. Navyše sme vytvorili modulárne a ľahko rozšíriteľné testovacie prostedie pre možné budúce rozšírenia práce. Nakoniec porovnávame všetky spomenuté metódy a kvalitu našej metódy voči branch-and-bound algoritmu založenom na delení simplexov v praxi. V testoch demonštrujeme, že naša metóda prekonáva túto simplexovú metódu na kopozitívnych maticiach.

Klíčová slova: kopozitivita, kopozitívna matica, kvadratické programovanie, branch-and-bound, intervalová analýza, kopozitívna optimalizácia

Contents

Introduction

The concept of copositivity is a relatively novel term coined less than 100 years ago. Since then, it has given rise to a whole new area of research, which gave us a new perspective on many other NP-complete problems and their reformulations. It helped to prove the NP-completeness of new, tighter questions by reducing them to the copositive problem. It also opened the door for a completely new area of optimization, copositive optimization, which generalizes semi-definite optimization with many fascinating geometrical properties.

Besides mathematical interest, copositivity has been widely used in economics, mainly financial optimization and applications in supply chains. Because of the problem's close connection to graph theory, specifically the maximum clique problem, it has applications in signal processing, image reconstruction, or machine learning.

The goal of this thesis

This thesis aims to implement and test a new approach for testing copositivity, using spectral properties and interval arithmetic, to create a branch-and-bound method. Ideally, the method should be comparable to some of the best approaches currently used (or proposed) in the scientific field, mainly a simplex branch-and-bound method as described later in the paper. We also aim to test different approaches and focus on problems related to copositivity, such as generating copositive matrices or providing a testing environment for future approaches and their testing.

For the implementation, we use MATLAB, with INTLAB arithmetic for numeric comparisons.

Structure of the work

In the beginning, we define basic notions and definitions, then we move on to showing some approaches from the scientific community to the problem, where we pay most attention to the simplex branch-and-bound method, as this is the method we wanted to compare our method to. Then, we describe the mathematical theorems and matrix properties that we use in our method. Next, we discuss the implementation, technical challenges, and how we dealt with them. Finally, we present statistical data from real-run experiments, which give us insights into the performance of methods.

1 Basic definitions

1.1 Notation

We denote matrices as capital letters (A, B, C, etc), and vectors with small letters (x, y, z, etc). We denote a real-valued matrix with m rows and n columns as $A \in \mathbb{R}^{m \times n}$. In the context of this work, when referring to a matrix, we mean symmetric real-valued square matrix (as this is most relevant for the copositive problem, the symmetric part is justified later in lemma 1.2.8). Real vectors of size n are denoted as $x \in \mathbb{R}^n$. The entry of the matrix in the i-th row and j-th column is denoted as $A_{i,j}$, and the k-th entry of a vector x is denoted as x_k. The transposition of matrix A is denoted as A^T.

1.2 Definitions

Definition 1.2.1 (Vector $e(i)$)**.** *Given $n \in \mathbb{N}$ and $i \in \{1, 2, \ldots, n\}$. We define $e(i)$ as vector from $\{0, 1\}^n$ which satisfies:*

- $j \neq i \implies e(i)_j = 0$
- $j = i \implies e(i)_j = 1$

(that is a vector whose all entries are 0, except for the i-th entry, which is a 1).

Definition 1.2.2 (Vector-real number inequality)**.** *Given $y \in \mathbb{R}, x \in \mathbb{R}^n$. We say that $x \geq y$ if each entry of x is greater than or equal to y.*

Definition 1.2.3 (Vector-vector inequality)**.** *Given $x, y \in \mathbb{R}^n$. We say that $x \geq y$ if*

$$(\forall i \in \{1, 2, \ldots, n\})(x_i \geq y_i)$$

Definition 1.2.4 (Matrix-matrix inequality)**.** *Given $A, B \in \mathbb{R}^{m \times n}$. We say that $A \geq B$ if*

$$(\forall i \in \{1, 2, \ldots, m\})(\forall j \in \{1, 2, \ldots, n\})(A_{i,j} \geq B_{i,j})$$

In other words, $A \geq B$ if, for all corresponding entries, the value in the matrix A is greater than or equal to the one in the matrix B.

Definition 1.2.5 (Copositive matrix)**.** *A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is called copositive if for all $x \in \mathbb{R}^n$ with $x \geq 0$ we have $x^T Ax \geq 0$. Otherwise, it is called non-copositive.*

Definition 1.2.6 (Strictly copositive matrix)**.** *A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is called copositive if for all $x \in \mathbb{R}^n$ with $x \geq 0, x \neq 0$ we have $x^T Ax > 0$.*

Definition 1.2.7 (ϵ-copositive matrix)**.** *A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is called ϵ-copositive if for all $x \in \mathbb{R}^n$ with $x \geq 0$ and $\max_i x_i \leq 1$ we have $x^T Ax \geq -\epsilon$.*

Remark. Note that without loss of generality, we can only consider symmetric matrices because of the following lemma:

Lemma 1.2.8 (Symmetric lemma)**.** *For any matrix $A \in \mathbb{R}^{n\times n}$ and any vector $x \in \mathbb{R}^n$ we have the following equality:*

$$x^T Ax = \frac{1}{2}x^T(A + A^T)x$$

Proof. We have $x^T Ax = (x^T Ax)^T = x^T A^T x$, where the first equality holds because the matrix is of size 1×1, and the second is just the transposition rule for matrix multiplication. Altogether we obtain $2x^T Ax = x^T(A + A^T)x$, which completes the proof. □

Definition 1.2.9 (Real interval)**.** *Given $a, b \in \mathbb{R}$, where $a \leq b$. A real interval $[a, b]$ is the set of numbers $[a, b] = \{y \in \mathbb{R} \mid a \leq y \leq b\}$. We call a the infimum (denoted* inf*) and b the supremum (denoted* sup*) of the interval $[a, b]$.*

Definition 1.2.10 (Interval matrix)**.** *Given matrices $A, B \in \mathbb{R}^{m\times n}, A \leq B$. An interval matrix $\mathbf{A} = [A, B]$ is the set of matrices $\mathbf{A} = \{X \in \mathbb{R}^{m\times n} \mid A \leq X \leq B\}$.*

Definition 1.2.11 (Interval vector)**.** *Given $x, y \in \mathbb{R}^n, x \leq y$. An interval vector $\mathbf{x} = [x, y]$ is the set of vectors $\mathbf{x} = \{z \in \mathbb{R}^n \mid x \leq z \leq y\}$.*

Definition 1.2.12 (Real interval operations)**.** *Given $a, b, c, d \in \mathbb{R}, a \leq b, c \leq d$. We define the following operations for intervals:*

1. $[a, b] + [c, d] = [a + c, b + d]$
2. $[a, b] - [c, d] = [a - c, b - d]$
3. $[a, b] \times [c, d] = [m, M]$*, where m is the minimum and M is the maximum of the set $\{ac, ad, bc, bd\}$.*
4. $[a, b] / [c, d] = [a, b] \times \frac{1}{[c,d]}$*, where*
   - *(a)* $c = 0 \implies \frac{1}{[c,d]} = \frac{1}{[0,d]} = \left[\frac{1}{d}, \infty\right]$
   - *(b)* $d = 0 \implies \frac{1}{[c,d]} = \frac{1}{[c,0]} = \left(-\infty, \frac{1}{c}\right]$
   - *(c)* $0 \in (c, d) \implies \frac{1}{[c,d]} = \left(-\infty, \frac{1}{c}\right] \cup \left[\frac{1}{d}, \infty\right)$
   - *(d) otherwise -* $\frac{1}{[c,d]} = \left[\frac{1}{d}, \frac{1}{c}\right]$

Definition 1.2.13 (Standard simplex)**.** *Given $n \in \mathbb{N}$. Then the set $S = \{x \in \mathbb{R}^n \mid ||x||_1 = 1\}$ is called the standard simplex.*

Definition 1.2.14 (Simplicial partition)**.** *Given $n \in \mathbb{N}$, simplex S in $\mathbb{R}^n$. Then the simplicial partition of the simplex S is a set of simplices $S_1, S_2, \ldots, S_m$ satisfying both*

1. $S = \cup_{i=1}^m S_i$
2. $(\forall i, j \in \{1, 2, \ldots, m\}, i \neq j)(int(S_i) \cap int(S_j) = \emptyset)$

2 Problem analysis

2.1 Problem statement

The first formulation of the problem is often attributed to Motzkin all the way in the year 1952 [1], and the term copositive programming was first coined in the year 2000 [2]. In terms of optimization, we can surely call this a novel problem. Let us give a precise formulation:

Given a symmetric $A \in \mathbb{R}^{n \times n}$, is the matrix copositive?

This has been shown to be a co-NP-complete problem [3], and many other problems have since been reformulated (or reduced) to this problem, helping us to classify them as NP-complete. Most notable NP-complete problems related to copositivity are quadratic optimization [2], list coloring problem [3] or, arguably the most famous with copositivity, the maximum clique problem [4].

2.2 Conditions for copositivity

2.2.1 Necessary conditions

Diagonal property

Claim 2.2.1. *Given a copositive matrix $A \in \mathbb{R}^{n \times n}$. Then each diagonal element of A, that is $A_{i,i}$ for $i \in \{1, 2, \ldots, n\}$ is non-negative.*

Proof. Suppose for contradiction that a copositive $A \in \mathbb{R}^{n \times n}$ with a negative element on its diagonal exists; let us denote it as $A_{i,i}$. Then for the vector $x = e(i)$ we have $e(i)^T A e(i) = A_{i,i} < 0$, which contradicts the copositivity of A. □

This may seem relatively trivial, but this gives us a fascinating insight into how many (symmetric) matrices of order n we can expect to be copositive, with entries being chosen uniformly random in the range [-1, 1]. Given that each such matrix has n diagonal entries, and each of those is negative with probability $\frac{1}{2}$, we have that the probability of the matrix being copositive is at most $\frac{1}{2^n}$.

Spectral property

Claim 2.2.2. *Given a copositive matrix $A \in \mathbb{R}^{n \times n}$. Then, at least one of the eigenvalues of A has to be non-negative.*

Proof. Suppose for contradiction that there exists a copositive $A \in \mathbb{R}^{n \times n}$ with all negative eigenvalues. Then, the matrix $-A$ has all its eigenvalues positive, and thus, it is positive-definite. Positive-definity implies copositivity, so we have that $-A$ is also copositive. However, then for all $x \in \mathbb{R}^n, x \geq 0$ we have both:

$$x^T A x \geq 0$$

$$x^T (-A) x = -x^T A x \geq 0$$

This necessarily implies that $A = 0$. However, this contradicts the fact that all eigenvalues of A are negative (as this matrix has all eigenvalues equal to 0), and thus, such a matrix cannot exist, and the proof is complete. □

While, again, this observation with spectral properties may seem obvious, it shows that copositivity is fundamentally connected to the spectral properties of the matrix. We explored other spectral properties as well and discussed them more in section 3.4. There are far more sophisticated conditions, both necessary and sufficient, that other researchers have explored [5][6][7].

2.2.2 Sufficient conditions

Positive-definity

Claim 2.2.3. *Given a (symmetric) positive-definite matrix $A \in \mathbb{R}^{n\times n}$. Then, the matrix A is also copositive.*

Proof. This comes directly from the definition - if $x^T Ax \geq 0$ holds for all $x \in \mathbb{R}^n$, then it also holds for all $y \in \mathbb{R}^n, y \geq 0$, as it is a subset of $\mathbb{R}^n$. □

This observation shows us that while testing copositive matrices is generally a co-NP-complete problem, some copositive matrices can be tested really quickly. Not only that, but this also gives us a certificate of copositivity that is only polynomially long. Similar approaches for copositive certificates have been tried and implemented [8].

Entry-wise properties

Claim 2.2.4. *Given a (symmetric) matrix $A \in \mathbb{R}^{n\times n}$. If A is entry-wise non-negative, that is*

$$(\forall i \in \{1, 2, \ldots, n\})(\forall j \in \{1, 2, \ldots, n\})(A_{i,j} \geq 0)$$

then A is copositive.

Proof. This is relatively straightforward: we just rewrite the expression $x^T Ax$ using the definition of matrix multiplication, and we obtain a sum where all of the elements are non-negative; thus, the sum will be non-negative. □

Other entry-wise conditions have been studied extensively, and we can say for $n = 2, 3, 4$ we have full necessary and sufficient conditions for copositivity [9]:

Claim 2.2.5 (Necessary and sufficient conditions for $n = 2$)**.** *Given a (symmetric) matrix $A \in \mathbb{R}^{2\times 2}$. Then A is copositive if and only if all the following hold:*

- $a_{1,1} \geq 0$
- $a_{2,2} \geq 0$
- $a_{1,2} + \sqrt{a_{1,1}a_{2,2}} \geq 0$

Claim 2.2.6 (Necessary and sufficient conditions for $n = 3$)**.** *Given a (symmetric) matrix $A \in \mathbb{R}^{3\times 3}$. Then A is copositive if and only if all the following hold:*

- $a_{1,1} \geq 0$
- $a_{2,2} \geq 0$
- $a_{3,3} \geq 0$
- $a_{1,2} + \sqrt{a_{1,1}a_{2,2}} \geq 0$
- $a_{1,3} + \sqrt{a_{1,1}a_{3,3}} \geq 0$
- $a_{2,3} + \sqrt{a_{2,2}a_{3,3}} \geq 0$
- $$\sqrt{a_{1,1}a_{2,2}a_{3,3}} + a_{1,2}\sqrt{a_{3,3}} + a_{1,3}\sqrt{a_{2,2}} + a_{2,3}\sqrt{a_{1,1}} + \sqrt{2\left(a_{1,2} + \sqrt{a_{1,1}a_{2,2}}\right)\left(a_{1,3} + \sqrt{a_{1,1}a_{3,3}}\right)\left(a_{2,3} + \sqrt{a_{2,2}a_{3,3}}\right)} \geq 0$$

As we can see, the number of conditions tends to increase fast, so we choose to omit the conditions for a matrix of size $n = 4$ and encourage the reader to go to the source as cited above for the claim, as well as proofs (or citations therein, which contain them).

These entry-wise properties are especially significant for matrices of size $n \leq 4$, where each copositive matrix can be written as the sum of a semi-positive-definite matrix and an entry-wise non-negative matrix [10].

Other approaches

There are numerous other necessary and sufficient conditions for this problem that offer unique ways of looking at the copositivity problem [11][6][12]. Some directly give rise to algorithms, which we briefly mention in section 2.4. We do not give them that much focus because we want to focus on the practical and testing approach.

2.3 Practical applications

Despite being a relatively new field of study, both copositive matrices and copositive optimization found their way into many areas of research. This section is not a full list of practical applications; rather, it is just an illustration and motivation for this topic.

Copositive programming shows up, for example, in stochastic appointment scheduling problem [13], which has practical applications in medicine, network flow optimization, or general delivery systems.

As previously mentioned, copositivity is tightly related to many combinatorial problems, such as clique number of a graph [10], stability number of a graph, [14] chromatic number of a graph [15], fractional chromatic number of a graph [16] or 3-partitioning of vertices [17]. One not necessarily graph-related application

where copositivity is used is the quadratic assignment problem [18].

Copositivity is further used in quadratic programming, for example, in binary quadratic problems [19], or fractional quadratic problems [20].

2.4 Approaches of other researchers

2.4.1 Simplex branch-and-bound method

We will discuss this approach in greater detail, as this is the approach we compare our methods to and compare it the most in this work.

The simplex branch-and-bound method was first mentioned in [4] and was later cited many times, with suggestions for improvements and other comments, for example, see [21] [22] [23].

The paper first proves that without loss of generality, when testing copositivity, it is sufficient to focus just on $x \in \mathbb{R}^n, x \geq 0$ with $||x||_1 = 1$, thus points in the standard simplex. Furthermore, to check for the inequality on the whole simplex, it is sufficient to consider only points $v_1, v_2, \ldots, v_m$, where the simplex is the convex cover of these vertices. From these, we obtain a sufficient condition for copositivity:

Claim 2.4.1. *Given $n \in \mathbb{N}, S$ the standard simplex on $\mathbb{R}^n$, symmetric $A \in \mathbb{R}^{n \times n}$ and a simplicial partition of the standard simplex $S = \{S_1, S_2, \ldots, S_m\}$. Let $v_1^k, v_2^k, \ldots, v_n^k$ be the vertices of the simplex S_k. Then if*

$$(\forall k \in \{1, 2, \ldots, m\})\,(\forall i, j \in \{1, 2, \ldots, n\})\left(\left(v_i^k\right)^T A v_j^k \geq 0\right)$$

Then, the matrix A is copositive.

This gives a direct algorithm to check for copositivity:

Algorithm 1 Simplex branch-and-bound method

$Q \leftarrow$ empty queue
add the standard simplex S on $\mathbb{R}^n$ into Q
while Q is not empty **do**
    remove the first element in Q and assign the value to s
    choose $v_1, v_2, \ldots, v_n$, such that $\text{conv}(v_1, v_2, \ldots, v_n) = s$
    for $i = 1, 2, \ldots, n$ **do**
        if $v_i^T A v_i < 0$ **then** return "A is not copositive" and v_i as certificate
        end if
    end for
    $m \leftarrow \min_{i,j \in \{1,2,\ldots,n\}} \{v_i^T A v_j\}$
    if $m < 0$ **then**
        partition s into two smaller simplicies s_1, s_2
        add s_1 and s_2 into Q
    else ▷ the matrix is copositive on the simplex s, we can discard it
    end if
end while
return "A is copositive" ▷ we discarded all of the intervals, and we only discard an interval when the matrix is copositive, so it is copositive on all of them

This algorithm leaves space for the partitioning of the simplex into two smaller ones. In the original paper [4], there is a suggestion for the splitting. This is the way we implemented splitting in our work as well. There have been other approaches, such as the one in the paper [21], which generally performs better. We will focus on the approach from the original paper. Here is the splitting procedure for the simplex $s = \text{conv}(v_1, v_2, \ldots, v_n)$:

Algorithm 2 Simplicial partition

$(i, j) = \text{argmin}_{(i,j) \in \{1,2,\ldots,n\}^2} v_i^T A v_j$
$\alpha \leftarrow v_i^T A v_i$
$\beta \leftarrow v_j^T A v_j$
$\gamma \leftarrow v_i^T A v_j$
if $\frac{\gamma}{\gamma - \alpha} > \frac{\beta}{\beta - \gamma}$ **then**
    return "A is not copositive"
end if
$\lambda \leftarrow \max\{\frac{\gamma}{\gamma - \alpha}, \min\{\frac{\beta - \gamma}{\alpha - 2\gamma + \beta}, \frac{\beta}{\beta - \gamma}\}\}$
$\sigma \leftarrow \lambda v_i + (1 - \lambda) v_j$
$s_1 = \text{conv}(v_1, v_2, \ldots, v_{i-1}, \sigma, v_{i+1}, v_{i+2}, \ldots, v_n)$
$s_1 = \text{conv}(v_1, v_2, \ldots, v_{j-1}, \sigma, v_{j+1}, v_{j+2}, \ldots, v_n)$
return (s_1, s_2)

In the paper, it was proved that this algorithm terminates for both strictly copositive and non-copositive matrices. The problem with termination arises with copositive matrices that are not strictly copositive. As this condition was sufficient but not necessary, for some matrices, the algorithm may keep splitting the simplices indefinitely, failing to terminate. Because of this, the notion of ϵ-copositivity for some small $\epsilon > 0$ has to be used to guarantee termination.

2.4.2 Linear complementarity approach

In the paper [24], a new approach is introduced that is not based on simplex partitioning. We first reformulate the copositivity problem as a quadratic programming problem:

$$\min f(x) = \frac{1}{2}x^T Ax$$
$$e^T x = 1$$
$$x \geq 0$$

where e is the vector, whose all coordinates are ones. Matrix A is copositive if and only if the optimum $\bar{x}$ found by the algorithm satisfies $f(\bar{x}) \geq 0$.
We can use KKT-conditions for optimality, to find $\lambda \in \mathbb{R}, w \in \mathbb{R}^n$, such that

$$Ax = \lambda e + w$$
$$x \geq 0, w \geq 0$$
$$x^T w = 0, e^T x = 1$$

Notice that if any triple (x, λ, w) satisfies these, then we have

$$f(x) = \frac{1}{2}x^T Ax = \frac{\lambda}{2}$$

We can use this to construct a new program with linear complementarity constraints (let us call it PLCP):

$$\min \frac{\lambda}{2}$$
$$w = Ax - \lambda e$$
$$x \geq 0, w \geq 0$$
$$x^T w = 0, e^T x = 1$$
$$\lambda \in \mathbb{R}$$

For which the paper proves the following claim:

Claim 2.4.2. *Given a symmetric matrix $A \in \mathbb{R}^n$. Then A is copositive if and only if the PLCP program has an optimal solution $(\bar{x}, \bar{\lambda}, \bar{w})$ with $\bar{\lambda} \geq 0$.*

Which shows the ties to the copositivity testing problem. The paper later goes into more conditions and applies Lemke's method and mixed integer linear programming to formulate an algorithm for testing, which was tested on matrices up to order 496×496. We encourage the reader to see the paper for more details and numerical results, alongside comparisons with the simplex branch-and-bound method.

Finally, let us mention that this is not the only paper mentioning the idea of using mixed integer linear programming and copositivity; for example, also see [25].

2.4.3 Other branch-and-bound methods

Because of the recursive structure of copositive matrices [26], there exist many branch-and-bound approaches, and this is by no means a full list but rather an illustration of other methods not explored in our testing.

Many branch-and-bound approaches come from the following observation:

Claim 2.4.3. *Given $n \in \mathbb{N}$, $A \in \mathbb{R}^{n\times n}$ copositive. Then, all its principal submatrices of order $n-1$ are copositive.*

Proof. By contradiction assume the existence of a copositive matrix $A \in \mathbb{R}^{n\times n}$ with a principal submatrix $B \in \mathbb{R}^{(n-1)\times(n-1)}$, that is non-copositive, thus there exists $y \in \mathbb{R}^{n-1}$, such that $y^T By < 0$. Denote i as the index of the row (and column) missing from A in B. If then we define $x \in \mathbb{R}^n$ as

- $x_j = 0$ for $j = i$
- $x_j = y_j$ for $j < i$
- $x_j = y_{j-1}$ for $j > i$

In other words, y with 0 inserted into i-th position. This gives us $x^T Ax = y^T By < 0$, thus contradicting the copositivity of A. □

This gives us a necessary condition for copositivity. From this, multiple approaches exist to adapt this idea into a sufficient condition, thus giving us various branch-and-bound algorithms [27]. Let us mention a few:

Gaddum's copositivity test

Claim 2.4.4. *[28] Given $n \in \mathbb{N}, A \in \mathbb{R}^{n\times n}$ with all its principal submatrices of order $n-1$ copositive. Then the following are equivalent:*

- *A is copositive*
- *the value of the matrix game induced by A is non-negative*

This gives us another interesting look at applications of matrices and their ties with Nash equilibrium and matrix games.

Cottle–Habetler–Lemke copositivity test

Claim 2.4.5. *[29][30] Given $n \in \mathbb{N}, A \in \mathbb{R}^{n\times n}$ with all its principal submatrices of order $n-1$ copositive. Then the following are equivalent:*

- *A is copositive*
- *$det(A) \geq 0$ or $adj(A)$ has a negative element*

Where adj(A) is the transpose of the matrix cofactors, as usual.

Claim 2.4.6. *[31] Given $n \in \mathbb{N}, A \in \mathbb{R}^{n\times n}$ with all its principal submatrices of order $n-1$ copositive. Then the following are equivalent:*

- *A is not copositive*
- *A^{-1} exists and is entry-wise non-copositive*

Finally, let us mention one with a recursive structure that does not rely on principal submatrices.

Claim 2.4.7. *[26] Given $n \in \mathbb{N}$, symmetric $A \in \mathbb{R}^{n \times n}$. Let us denote a, b, C as:*

$$A = \begin{bmatrix} a & b^T \\ b & C \end{bmatrix}$$

Then A is copositive if and only if all the following conditions hold

1. *$a \geq 0$*
2. *C is copositive*
3. *$y^T(aC - bb^T)y \geq 0$ for $y \in \mathbb{R}^{n-1}, b^T y \leq 0$*

The most problematic part of this algorithm would be the last condition - in branching, this would get increasingly complicated, taking exponentially long to verify.

3 Implementation approaches

3.1 Introduction to the branch-and-bound method

The branch-and-bound method is a method of solving optimization problems that enumerates the state space and discards states that cannot contain the optimal solution. The state space starts at a single node and then branches into further sub-problems, where different estimates (bounds) can be made to simplify the problem. This algorithm is better than an exhaustive search, as the bounds can help navigate it to the optimal solution quicker.

Because we bound the problem on smaller and smaller sub-problems, it is often used alongside interval methods [32] [33]. Interval methods help reliably estimate the upper and lower bounds of a particular function, which helps the search from the branch-and-bound method.

The branch-and-bound method has been used for many other NP-hard optimization problems [34], such as the travelling salesman problem [35] [36], where the term is believed to be coined.

3.2 Methodological Overlaps

In this section, we discuss the approaches we use to solve the copositivity problem. We provide three distinct methods of obtaining a lower bound for a given interval vector. While they are very different in nature, they have many overlaps.

3.2.1 Generating initial queue

In this method, we are testing whether a given matrix $A \in \mathbb{R}^{n\times n}$ is copositive on a given vector interval $\mathbf{x} \in \mathbb{R}^n$. We can model the coposivity problem as a whole with intervals $[0, \infty)$, so this setting is very general. First, we will show that we need not consider large values and can only consider values in the interval $[0, 1]$:

Claim 3.2.1. *Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$. If the matrix A is not copositive, then there exists $x \in \mathbb{R}^n$ with entries in the interval $[0, 1]$ with $x^T Ax < 0$.*

Proof. From the definition of copositivity, if the matrix A is not copositive, then there must exist a vector $y \in \mathbb{R}^n, y \geq 0$ with $y^T Ay < 0$. Note that $y \neq 0$, because otherwise the inequality would not hold, and thus we have $M = \max_i y_i > 0$, and we can define $x = \frac{1}{M}y$, which satisfies the inequality $x^T Ax < 0$, and the values of x are all in the interval $[0, 1]$ as claimed. □

This means that we can only look for x with values in the interval $[0, 1]$. We proved even more - if the matrix A is not copositive, then there must exist x with values in the interval $[0, 1]$ and at least one entry whose value is equal to 1.

In general, we will be working with a queue of intervals on which we want to check the copositivity of the matrix. The initial queue will take advantage of the fact from the paragraph above and thus will consist of n interval vectors, with $n-1$ entries being the interval $[0,1]$ and the remaining entry being $[1,1]$ (degenerate) interval. Here is a pseudocode:

Algorithm 3 Generate initial queue

```
initialize an empty queue Q
for i = 1, 2, ..., n do
    create a vector x = [0,1]^n
    x[i] ← [1,1]
    add x to Q
end for
return Q
```

3.2.2 Common subproblems

Because all of our approaches are based on the branch-and-bound approach, we have different options in both branching and bounding, respectively, which we describe in detail below.

Finding non-copositivity candidate

When we are checking if the matrix is copositive on a given interval vector, we would like to generate (or somehow obtain) a possible candidate for non-copositivity. Finding the optimal candidate, which minimizes the value of $y^T Ay$, is equivalent to solving the copositive problem, which is NP-hard. We have tried multiple approaches, comparing and discussing them further in section 5.6.

Splitting intervals

If we cannot say whether or not a matrix is copositive on a given interval, we need to somehow split the current interval vector into several smaller ones based on some criterion. Again, we tried multiple approaches, including look-ahead methods or the simplest "split the largest interval" method. We discuss them in more depth in section 5.5.

3.3 Interval method

This approach relies on interval arithmetic, and in each step, for a given interval vector, calculates the lower and upper bound for the node as the inf and sup of the interval obtained by matrix multiplication.

Algorithm 4 Interval-based branch-and-bound copositivity testing

$Q \leftarrow$ generate initial queue
while Q is non empty **do**
    remove the first element in Q and assign the value to $\mathbf{x}$
    pick a vector y in the interval vector $\mathbf{x}$
    if $y^T Ay < 0$ **then**
        return "A is not copositive" and y as certificate
    else
        calculate the interval $\mathbf{z} = \mathbf{x}^T A\mathbf{x}$
        if $\inf(\mathbf{z}) < 0$ **then**
            split the interval $\mathbf{x}$ into smaller ones, add them to Q
        else
            ▷ the matrix A is copositive on this interval, we can discard it
        end if
    end if
end while
return "A is copositive" ▷ we discarded all of the intervals, and we only discard an interval when the matrix is copositive, so it is copositive on all of them

3.4 Spectral method

The spectral method is based on the spectral theorem, a basic result from linear algebra:

Theorem 3.4.1 (Spectral theorem)**.** *Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$. Then there exists an orthogonal matrix $P \in \mathbb{R}^{n\times n}$ and a diagonal matrix $\Lambda \in \mathbb{R}^{n\times n}$, such that*

$$A = P^T \Lambda P$$

Then we can modify the expression from the definition of copositivity:

$$x^T Ax = x^T P^T \Lambda P x = (Px)^T \Lambda P x$$

If we denote $y = Px$, then we obtain a term $y^T \Lambda y$ that is similar to the one in the definition of copositivity but with a diagonal matrix instead. Because of this, we can rewrite this using the definition of matrix multiplication:

$$y^T \Lambda y = \sum_{i=1}^{n} \lambda_i y_i^2$$

Thus, we only need to calculate the eigenvalues of the matrix A and the matrix P once, and then in each iteration for the given interval matrix $\mathbf{x}$ calculate the value $P\mathbf{x}$ and then use the summation expression from above to get the lower bound. Here is the pseudocode:

Algorithm 5 Spectral-based branch-and-bound copositivity testing

calculate $\lambda_1, \lambda_2, \ldots, \lambda_n$ eigenvalues of A and the respective eigenvectors, and save them in a matrix P
$Q \leftarrow$ generate initial queue
while Q is non empty **do**
    remove the first element in Q and assign the value to $\mathbf{x}$
    pick a vector y in the interval vector $\mathbf{x}$
    if $y^T A y < 0$ **then**
        return "A is not copositive" and y as certificate
    else
        calculate the interval vector $\mathbf{z} = P\mathbf{x}$
        calculate the interval $w = \sum_{i=1}^{n} \lambda_i \mathbf{z}_i^2$
        if $\inf(w) < 0$ **then**
            split the interval $\mathbf{x}$ into smaller ones, add them to Q
        else
            ▷ the matrix A is copositive on this interval, we can discard it
        end if
    end if
end while
return "A is copositive" ▷ we discarded all of the intervals, and we only discard an interval when the matrix is copositive, so it is copositive on all of them

3.5 Quadratic program with linear conditions

The idea of this approach is to rewrite the original problem and approximate the variables with new ones, creating a quadratic optimization problem with linear constraints. This is also known as McCormick envelopes [37].

We start by rewriting

$$x^T A x = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{i,j} x_i x_j = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2A_{i,j} x_i x_j + \sum_{i=1}^{n} A_{i,j} x_i^2$$

And now we create new variables $z_{i,j}$ for $i = 1, 2, \ldots, n-1$ and $j = i+1, \ldots, n$. These variables are used in order to approximate the value of $x_i x_j$.

On a current interval vector $\mathbf{x}$, and for a vector $x \in \mathbf{x}$ we have the following trivial observations:

1. $x_i \geq \inf(\mathbf{x}_i)$
2. $x_i \leq \sup(\mathbf{x}_i)$

From these observations, we can write the inequalities

1. $(x_i - \inf(\mathbf{x}_i))(x_j - \inf(\mathbf{x}_j)) \geq 0$
2. $(x_i - \inf(\mathbf{x}_i))(\sup(\mathbf{x}_j) - x_j) \geq 0$

3. $(\sup(\mathbf{x}_i) - x_i)(x_j - \inf(\mathbf{x}_j)) \geq 0$
4. $(\sup(\mathbf{x}_i) - x_i)(\sup(\mathbf{x}_j) - x_j) \geq 0$

Each of these inequalities gives us an inequality for the value of $x_i x_j$, in which we replace them with the variables $z_{i,j}$. Notice the conditions are linear in all x_i (as both $\inf(\mathbf{x}_i), \sup(\mathbf{x}_i)$ are just constants), so we obtain the following quadratic program:

$$\min \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2A_{i,j} z_{i,j} + \sum_{i=1}^{n} A_{i,i} x_i^2$$

variables

$$z_{i,j} \;\; i \in \{1, 2, \ldots, n-1\}, j \in \{i+1, \ldots, n\}$$
$$x_i \;\; i \in \{1, 2, \ldots, n\}$$

conditions

$$\sup(\mathbf{x_i}) \geq x_i \geq \inf(\mathbf{x_i}) \;\; i \in \{1, 2, \ldots, n\}$$

$$z_{i,j} - x_i \inf(\mathbf{x_j}) - \inf(\mathbf{x_i}) x_j + \inf(\mathbf{x_i}) \inf(\mathbf{x_j}) \geq 0$$

(inequality 1)

$$\sup(\mathbf{x_j}) x_i - z_{i,j} - \sup(\mathbf{x_i}) \sup(\mathbf{x_j}) + \inf(\mathbf{x_i}) x_j \geq 0$$

(inequality 2)

$$\sup(\mathbf{x_i}) x_j - \sup(\mathbf{x_i}) \inf(\mathbf{x_j}) - x_i x_j + \inf(\mathbf{x_j}) x_i \geq 0$$

(inequality 3)

$$\sup(\mathbf{x_i}) \sup(\mathbf{x_j}) - \sup(\mathbf{x_i}) x_j - \sup(\mathbf{x_j}) x_i + x_i x_j \geq 0$$

(inequality 4)

This returns both a lower bound on the value of $x^T Ax$, as well as the value of x, where the minimum is obtained for this auxiliary program. We can use this as a possible non-copositivity candidate, which is useful in practice. Here is a pseudocode of this approach:

Algorithm 6 Quadratic program branch-and-bound copositivity testing

$Q \leftarrow$ generate initial queue
while Q is non empty **do**
    remove the first element in Q and assign the value to $\mathbf{x}$
    pick a vector y in the interval vector $\mathbf{x}$
    if $y^T Ay < 0$ **then**
        return "A is not copositive" and y as certificate
    else
        calculate the lower bound b of the quadratic program, alongside with the variable z, for which this minimum is achieved
        if $z^T Az < 0$ **then**
            return "A is not copositive" and z as certificate ▷ this is optional; we do not have to use z for testing
        end if
        if $b < 0$ **then**
            split the interval $\mathbf{x}$ into smaller ones, add them to Q
        else
            ▷ the matrix A is copositive on this interval, we can discard it
        end if
    end if
end while
return "A is copositive" ▷ we discarded all of the intervals, and we only discard an interval when the matrix is copositive, so it is copositive on all of them

4 Implementation details

4.1 General notes

We wrote the methods and testing functions using MATLAB [38], with the INTLAB package [39] and optimization toolbox [40]. All our work is available in the GitLab repository [41].

This thesis does not contain explicit user documentation, as its goal was not to provide a user-friendly interface for testing but to implement and test a new approach. The program variables and functions have to be manually changed inside the code, which we did not consider user-friendly. Thus, we did not consider adding it meaningful. Regarding programmer documentation, the code is documented inside the repository itself in the form of extensive `README.md` files. Of course, this documentation is also complemented by comments in the code. This section provides a high-level overview of the approach.

4.2 Technical limitations

As the copositivity problem is NP-hard, we cannot guarantee termination in a reasonable amount of time for an arbitrary matrix. Because of this, we implemented a time restriction on the execution time inside the testing functions themselves, slowing down the execution. However, this is far more practical than waiting for the code to finish on every matrix when testing large amounts of matrices.

Another technical limitation is the termination itself - even for smaller matrices, if the matrix is not strictly copositive, some variations of the branch-and-bound method (such as the simplex method [4]) may not terminate. Because of this, we have to result in a slightly weaker definition of copositivity, namely ϵ-copositivity for some chosen value of ϵ. This will significantly change the result of the program, given a large value of ϵ (the functions will terminate with the result of ϵ-copositive more often, which is undesirable).

4.3 Method comparison functions

This section refers to the functions, which are programmed to compare various methods against each other in terms of speed and/or number of steps. The code is fairly simple to understand and well documented in the source code itself, so we will only focus on the most interesting parts to discuss in greater detail.

4.3.1 Generating matrices for testing

Generating random symmetric matrices would seem like a logical choice; without loss of generality, we can re-scale the matrix so that its entries are in the interval $[-1, 1]$. This, however, would result in almost all generated matrices being non-copositive (the bigger the matrix, the worse this gets). This would then favor

methods that are better at detecting non-copositive matrices and skew the results toward them. One could argue that if that is truly the nature of random symmetric matrices, then it is, in fact, the best way. We did not find this interesting enough and instead chose to modify the matrices to achieve a ratio of non-copositive to copositive matrices close to 50/50. To do this, we came up with two different approaches.

Adding values to the diagonal entries

The idea of this method is to choose some value $\lambda(n)$, which is dependent on the size of tested matrix n, and to add $\lambda(n)$ to all diagonal entries of the matrix. The idea of this approach is simple - adding values to the diagonal entries moves the matrix closer to being positive definite, which is a sufficient condition for copositivity. We experimentally found the best value of $\lambda(n)$ for chosen values of n and used that to generate matrices this way. The optimal value depends on the ratio; we went for 50/50.

Adding values to all the entries

The idea of this method is similar to the one in the last subsection, but instead, we add the value $\lambda(n)$ to all the values in the matrix. The idea is that if all the matrix entries are non-negative, then the matrix is clearly copositive. Again, we experimentally found the values and used the same ratio as before.

Of course, generating random symmetric matrices (and then modifying them in one of the mentioned ways) is not the only way to test for, and we account for that. Because of that, all the tests call some function that generates the matrix for testing instead. This function is always called when we need a matrix to test on, so for larger quantities, this function needs to be relatively fast. Otherwise, the already NP-difficult testing may take a long time. As the code is relatively customizable and modular, it is straightforward to change the function inside the code to accommodate for specific matrix generation to reflect better the matrices on which these methods may be used.

4.3.2 Files and structure

These tests are single function files with no parameters. There are settings to modify and change the behavior of these tests that are included (with comments on what they exactly do) at the beginning of each function. All of these files are in the `testing` directory, and we will now go through them and briefly explain what they do.

1. `compact_testing` - a simple function that does testing of set methods on many matrices, then displays the results in a compact format.

2. `compact_specific_testing` - a more general modification of the previous function. This also supports generating random matrices from a custom function or just generation of matrices until a condition is met - for a full explanation, see the details in the code.

3. `endless_comparison_better` - an endless (meaning that by default it runs forever, a "while true" loop situation) function for finding examples of matrices, where a set method is better than the other. It generates a random matrix, tests set methods on that matrix, and if the result is significantly better (what this means precisely can be modified in the code; basically, we do not want the difference to be because of CPU load fluctuations), displays it.

4. `endless_comparison_worse` - almost the same function as the one above, occasionally useful as its standalone function.

5. `lambda_value_finder` - this function is related to section 4.3.1: we are trying to find the $\lambda(n)$ values, as described in the linked subsection. The ratio, matrix sizes, and the number of examples to test are all settings that can be easily modified at the beginning of the function.

6. `test_results` - this function is used for storing matrices, where we want to display exceptionally better or worse running time, or the number of steps, between two functions. It contains many examples and can easily be re-run to confirm the significance of behavior differences of tested methods.

4.4 Copositivity testing functions

4.4.1 General overview

In section 3, we discussed the math behind numerous methods for testing copositivity using branch-and-bound. We implemented the interval method, spectral method, and the quadratic programming methods as the basis. From these, we also implemented all possible combinations of these methods. On a given interval vector $\mathbf{x}$, we can compute various values to get multiple lower bounds and choose the best one (at the cost of doing more calculations in all the steps). We have three different methods; for each one, we can choose to include that method or not, and in addition, in the quadratic program method, we can use the given non-copositivity candidate or not, giving us a total of 26 different methods (the order in which we compute them is important as well, cause it can discard unnecessary calculations).

The simplest methods are directly inside our repository's `tests` directory. Inside this directory, there are multiple subdirectories:

1. `combined` - contains the implementations of all combinations of different methods

2. `lookahead_tests` - contains two subdirectories, `spectral` and `interval`, and includes the implementation of the spectral and interval methods, respectively, where when branching, we do not split the longest interval in the current interval vector, but instead split the interval in the index, in which it gives us the best lower/upper bound for the next iteration. We discuss this in more detail in section 5.5.

4.4.2 Finding a non-copositivity candidate

These functions are implemented in the `utilities` directory. They are called with two arguments - an interval vector and a matrix, on which to find the candidate. The function always returns a single candidate. The currently used function can be changed easily inside the code, making it fairly modular. With a slight addition to the code, one can easily implement a function that returns a list of candidates instead. We implemented a total of three different functions:

1. `get_random_non_copositivity_candidate` - picks a random point from the interval vector

2. `get_edge_non_copositivity_candidate` - the idea of this method is to find x, where the value of x^TAx is minimal on the given interval. We take the derivative $\frac{d}{dx}x^TAx = 2Ax$ for x in the middle of the current interval vector. Then we move in the direction of $-2Ax$ from x until we reach an edge, that is inf or sup in one of the coordinates in the current interval vector.

3. `get_vertex_non_copositivity_candidate` - similar to the one above this, but instead, we take note of the sign of $-2Ax$ in each coordinate, and we move from x in each coordinate until we either reach inf or sup of the current interval vector (based of the sign).

Note that in section 5.6, we also use finding the middle of the current interval matrix as a method for testing. This function is uninteresting, as it does not take into consideration the matrix at all, and thus, we did not seem to find it interesting to include as its own. It does guarantee the termination of the algorithm for non-copositive matrices but is slower than other methods in practice. A similar function `get_middle_point` is used often, but that does not take two parameters as described above and thus does not fit the description in this category.

As mentioned before, adding a custom non-copositivity finding function is relatively easy - any function matching the input and return arguments with a custom heuristic can be easily added instead of these tests.

4.4.3 Functions arguments and usage

All of the methods have the same interface to interact with them - each method (or combination of methods) is a function in a separate file, with the same arguments as input into the function:

1. `matrix` - matrix for testing copositivity

2. `epsilon` - this is the parameter for returning the result of `epsilon-copositive`, as described more in section 4.2

3. `output` - this is a boolean parameter, whether or not to print the "certificate" of non-copositivity, that is, a particular value of $x \geq 0$, where $x^TAx < 0$

4. `maximum_execution_time` - maximum execution time of the function in seconds (is not precise, the function may run more than this amount, but not by much), the necessity of this is explained more in section 4.2

The function is called with these parameters and returns a tuple, where the first value is a string that is either `copositive`, `epsilon-copositive`, `non-copositive` or `not-finished`, which indicates the result of the operation.

The second value is a non-negative integer that indicates the number of steps the calculation took to get to the result. This can mean various things, depending on the method used, but most commonly refers to the number of interval vectors $\mathbf{x}$ extracted from the queue during the calculation that is processed in any way. It is useful for finding faster examples and helping illustrate which classes of matrices a particular method struggles with, so we decided to leave it like this.

This implementation is easily extensible, and any further method, whether or not using branch-and-bound, can be implemented like this and will work with our current structure. The only requirement is that these function arguments be precisely in this order and that the tuple (time, number of steps) be returned as described.

4.4.4 Naming convention

The naming of these functions is pretty straightforward: the basic methods always end with `_test`, and the simplest methods implemented begin with `interval`, `spectral` and `linear` (for the quadratic test with linear conditions, spectral test, interval test respectively). If the quadratic program method and its non-copositivity candidate are used, then the name `linear` is extended with `_example`. If multiple methods are used, they extend the name, starting with `_` and then the method name.

For example, for the test that first uses the interval method, and if that fails, goes to the quadratic test with the non-copositivity candidate, the name will be `interval_linear_example_test`. As another example, the method, which uses spectral, then interval, and lastly quadratic programming test without checking for the non-copositivity candidate from the program, will be named `spectral_interval_linear_test`.

5 Testing and results

5.1 Environment

We run the tests on the ASUS TUF Gaming notebook with the following specifications:

GPU

Intel(R) UHD Graphics + NVIDIA GeForce GTX 1660 Ti

CPU

Inter(R) Core(TM) i7-10870H CPU @ 2.20GHz

OS

Arch Linux

Memory

16GB RAM

5.2 Common parameter values

In all of our tests, we use the same value for these variables (their exact names may slightly differ, but they refer to the same idea):

- `epsilon` = 10^{-10} - refers for approximation to ϵ-copositivity as described in section 4.2
- `maximum_execution_time` = 60 - limits the maximum execution time of each function, in seconds; for more information, again refer to section 4.2
- `number_of_matrices_to_test` = 1000

To make sure the ratio of copositive matrices is close to $\frac{1}{2}$, we use the diagonal adjustment method, as mentioned in section 4.3.1. Also, unless it is clear from context otherwise, each time a new matrix is generated for testing, its size is a random number from 5 to 9 (including both endpoints).

5.3 Optimal lambda values

This refers to finding the optimal value of $\lambda(n)$ as described in section 4.3.1. The experiment was ran using the `lambda_value_finder.m` function with the following settings:

- `lambda_min_value` = 0
- `lambda_step_value` = 0.01
- `desired_ratio` = 0.5

5.3.1 Diagonal method

For the diagonal method, the parameter `lambda_max_value` was set to 2.

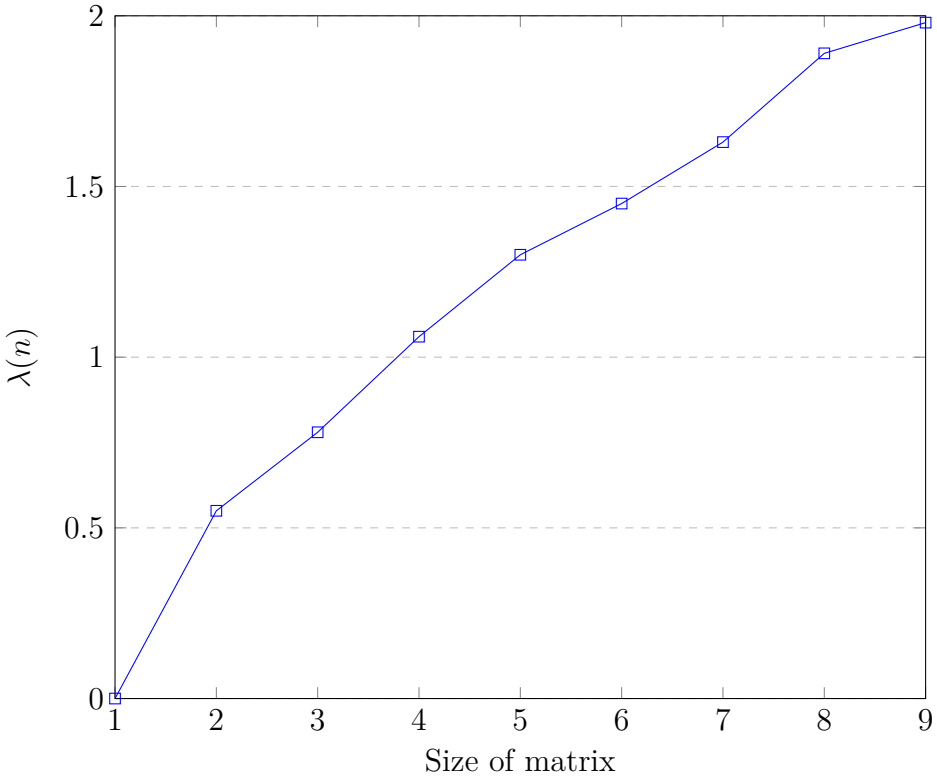

Figure 5.1 Optimal value of $\lambda(n)$ found by the experiment for the diagonal method

n	$\lambda(n)$
1	0
2	0.55
3	0.78
4	1.06
5	1.30
6	1.45
7	1.63
8	1.89
9	1.98

Table 5.1 Optimal value of $\lambda(n)$ found by the experiment for the diagonal method

5.3.2 All entries method

For the all entries method, the parameter `lambda_max_value` was set to 1.

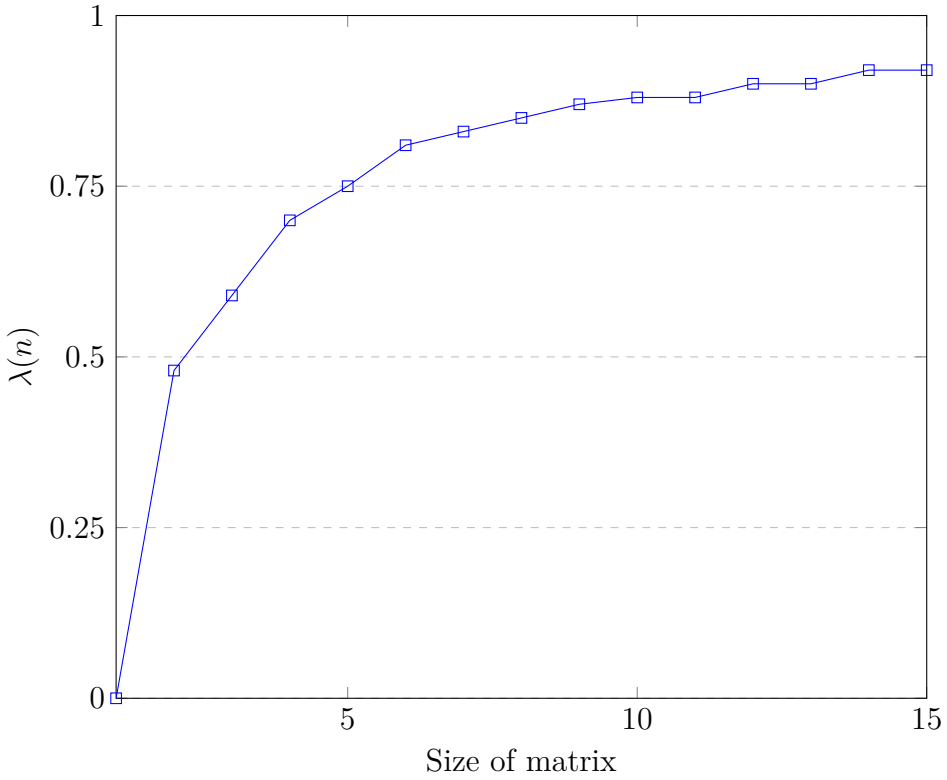

Figure 5.2 Optimal value of $\lambda(n)$ found by the experiment for the all entries method

n	$\lambda(n)$
1	0
2	0.48
3	0.59
4	0.70
5	0.75
6	0.81
7	0.83
8	0.85
9	0.87
10	0.88
11	0.88
12	0.90
13	0.90
14	0.92
15	0.92

Table 5.2 Optimal value of $\lambda(n)$ found by the experiment for the all entries method

We can see that for larger matrices, we need almost all entries to be positive for a matrix to be copositive, which tells us how much more likely the random matrix is to be non-copositive (of course, adding one would make the matrix always copositive).

5.4 Combined methods

Another idea was to combine the methods - meaning when checking each interval vector, we try using multiple methods to get multiple estimates, as some methods may behave better on certain types of matrices and certain types of interval vectors.

In the following figures, we had to create shortcuts because of the long names of the methods used, and we will explain them briefly now. We denote spectral as `S`, interval as `I`, and linear as `L`. When using linear with an example, we denote as `Lx`. The order of the letters denotes the order of methods in which they are used. For example, `SILx` denotes that we first check spectral, then interval, then linear with an example for a non-copositive candidate. As another example, `LS` denotes linear (without the example) and then spectral.

Because these tests contain so many methods, the testing is very time-consuming, and the parameter `number_of_matrices_to_test` was set to 250.

Copositive matrices

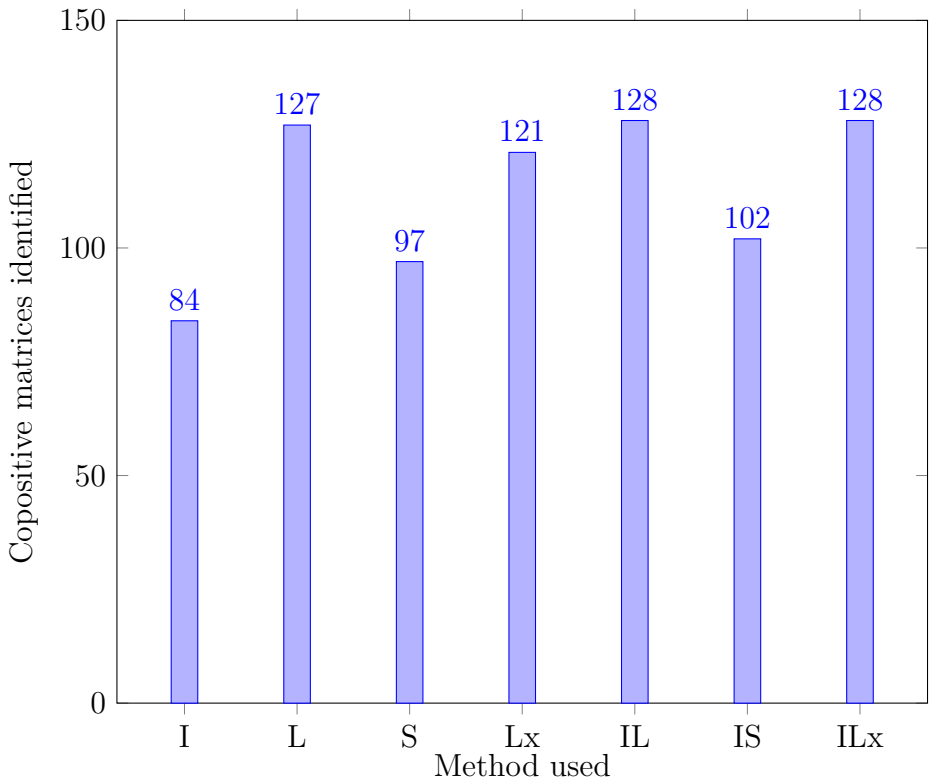

Figure 5.3 Number of copositive matrices identified in combined method testing by each method (1)

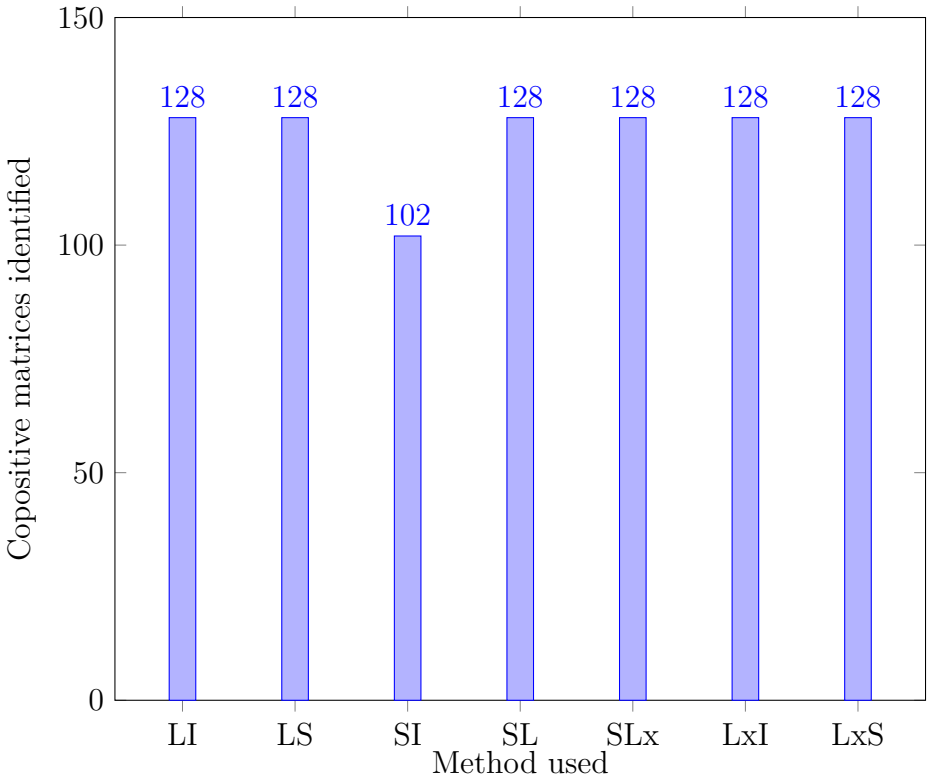

Figure 5.4 Number of copositive matrices identified in combined method testing by each method (2)

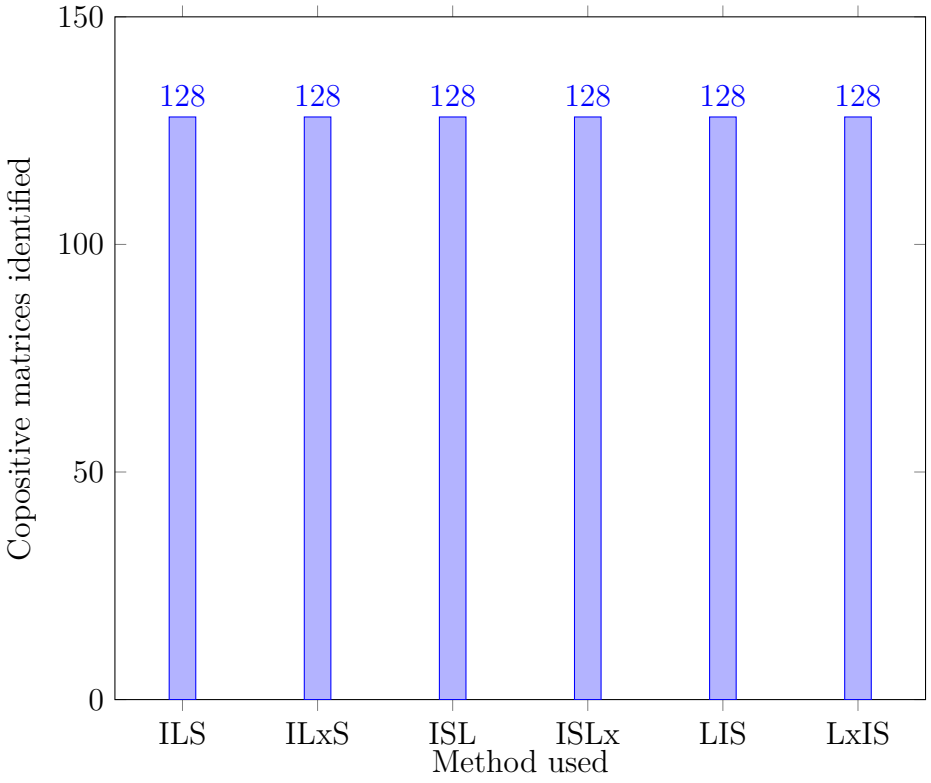

Figure 5.5 Number of copositive matrices identified in combined method testing by each method (3)

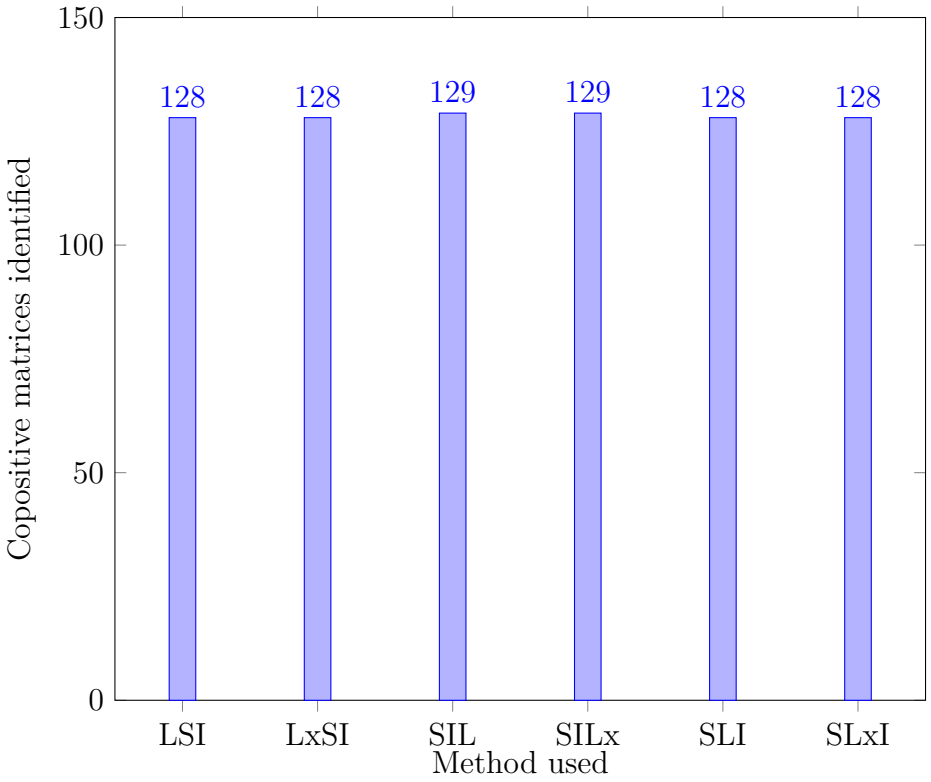

Figure 5.6 Number of copositive matrices identified in combined method testing by each method (4)

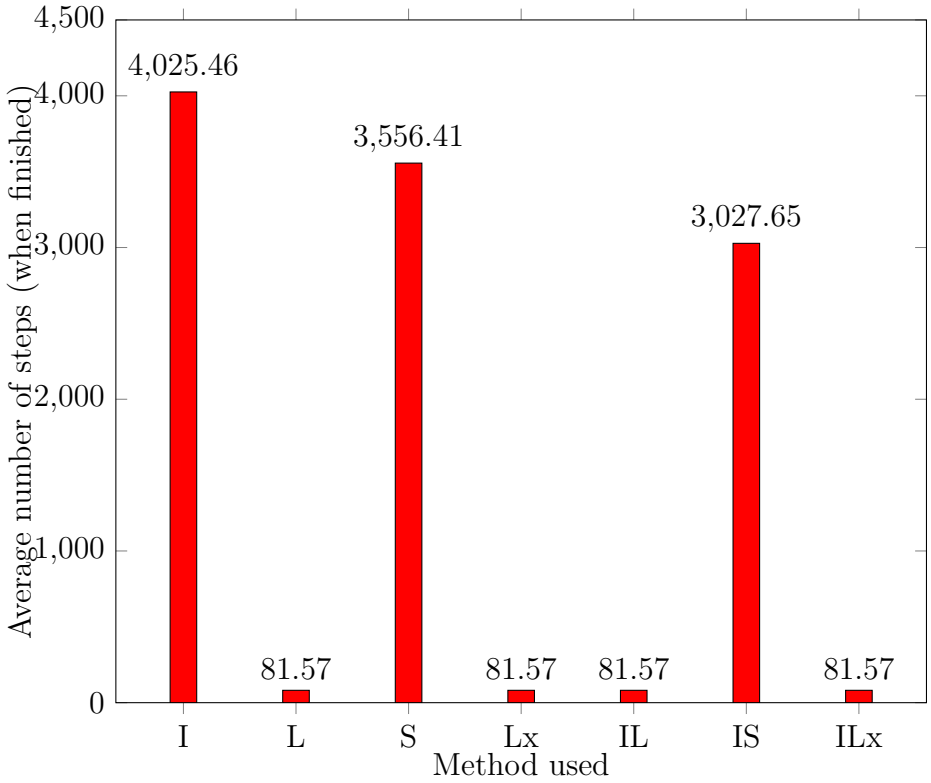

Figure 5.7 Average number of steps (when finished) for copositive matrices identified in combined method testing by each method (1)

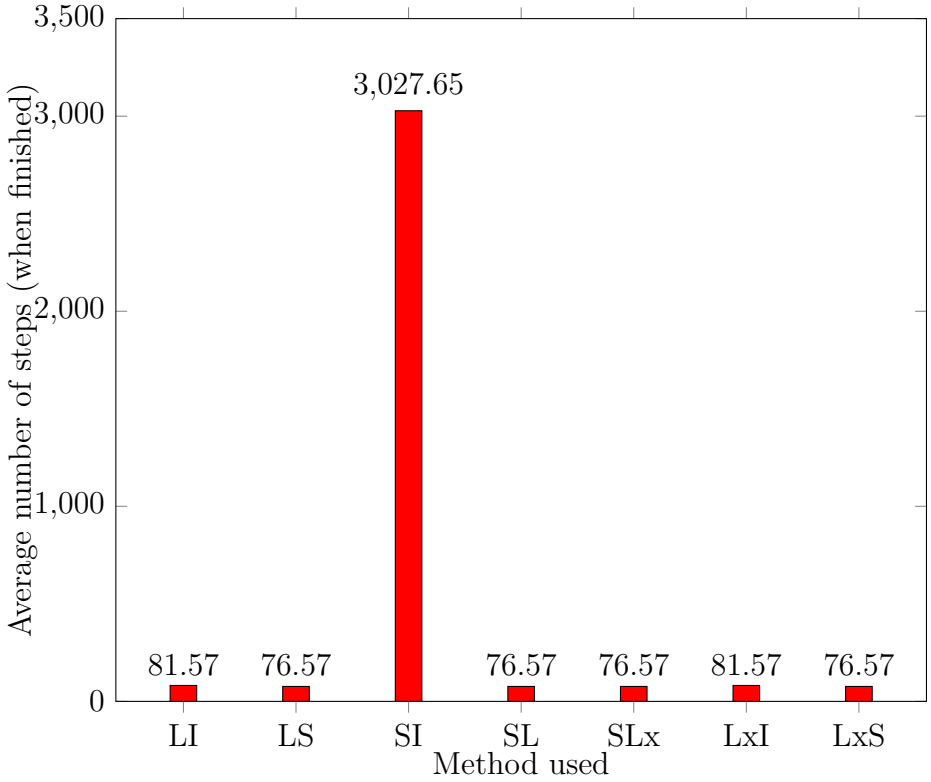

Figure 5.8 Average number of steps (when finished) for copositive matrices identified in combined method testing by each method (2)

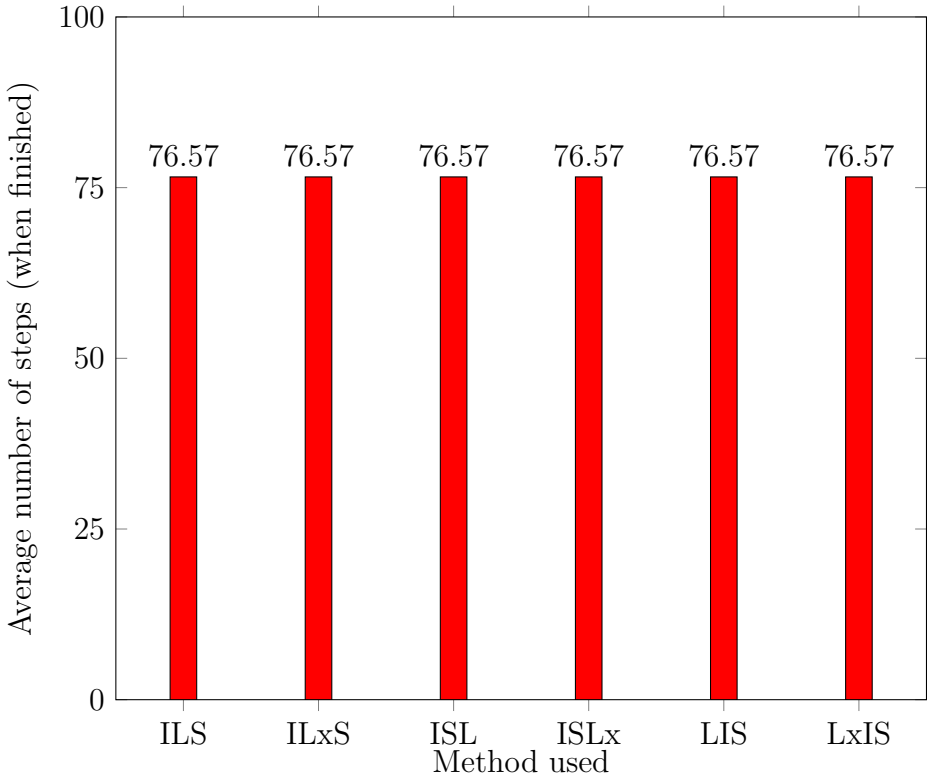

Figure 5.9 Average number of steps (when finished) for copositive matrices identified in combined method testing by each method (3)

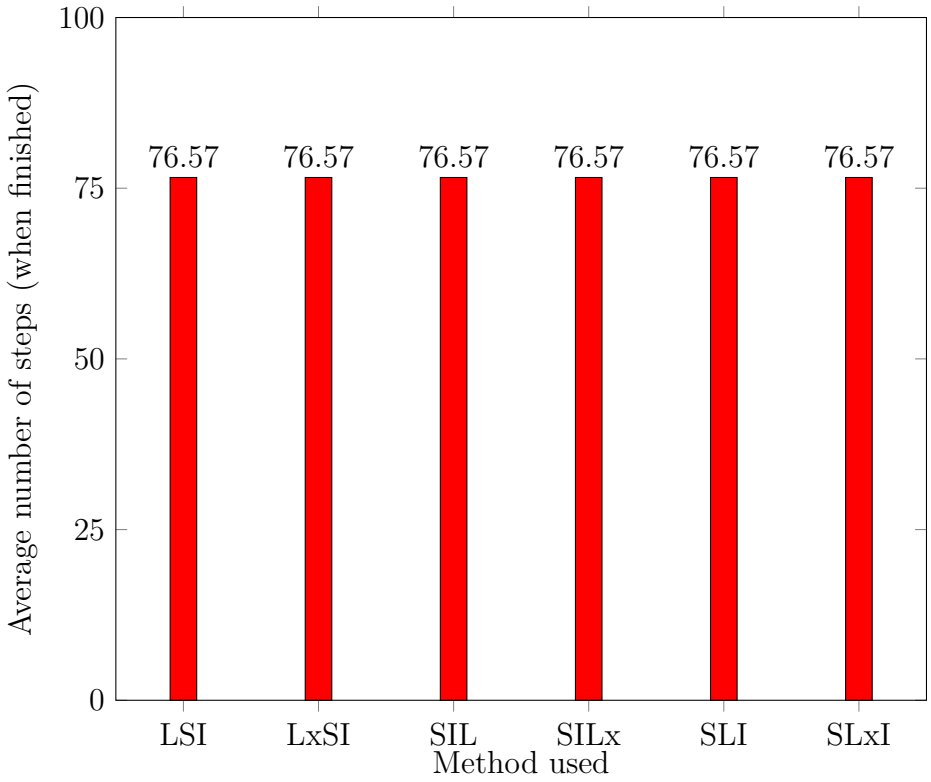

Figure 5.10 Average number of steps (when finished) for copositive matrices identified in combined method testing by each method (4)

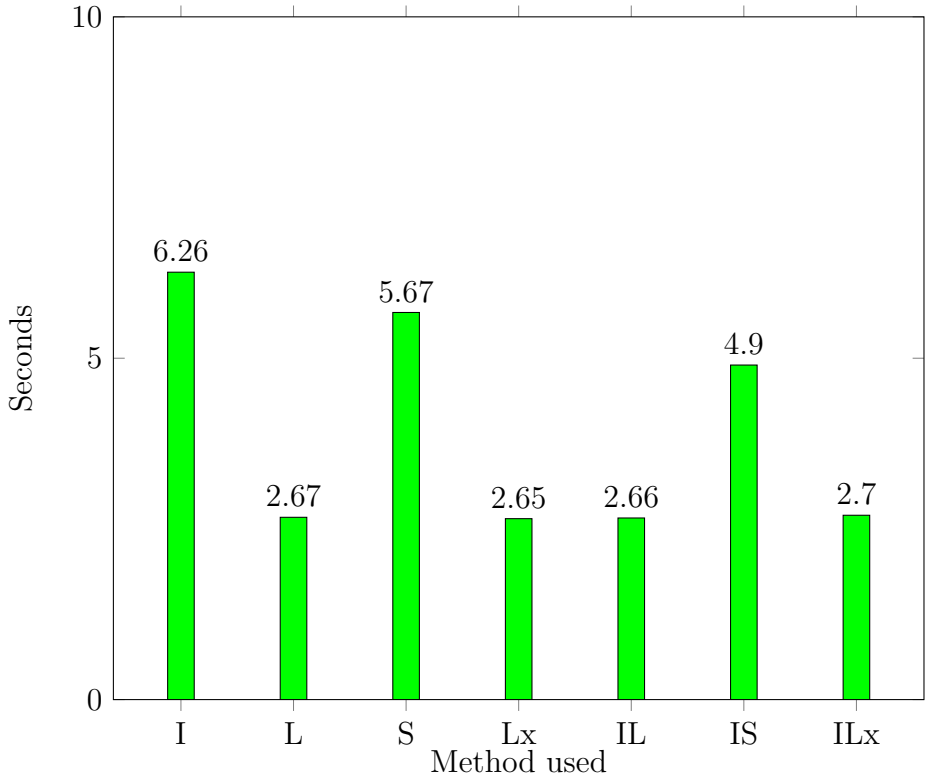

Figure 5.11 Average time took (when finished) for copositive matrices identified in combined method testing by each method (1)

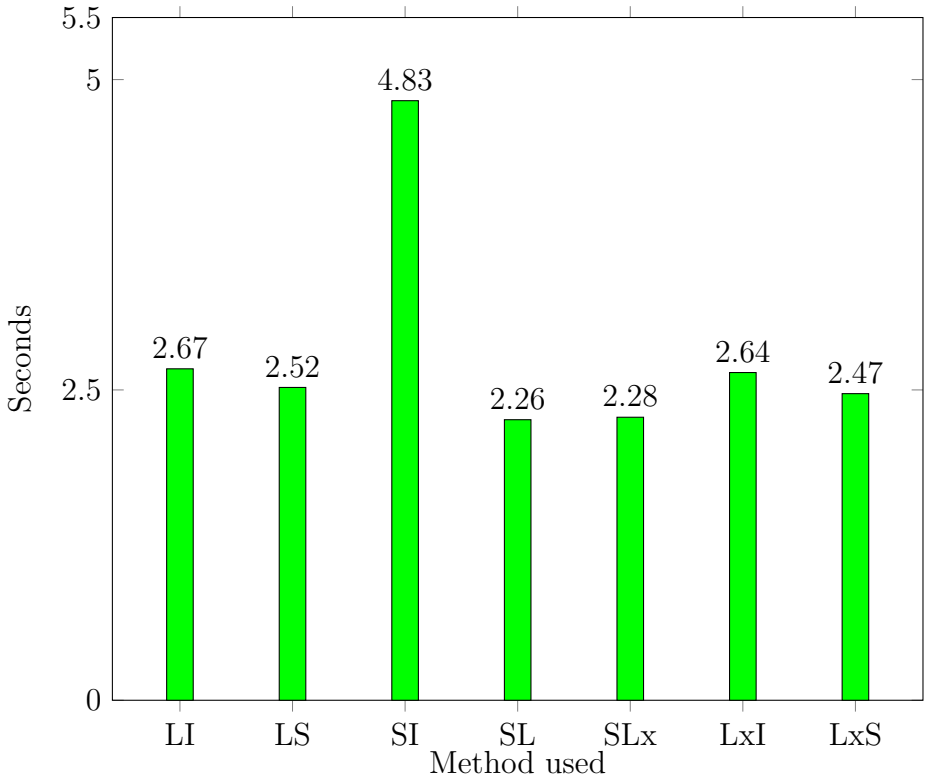

Figure 5.12 Average time took (when finished) for copositive matrices identified in combined method testing by each method (2)

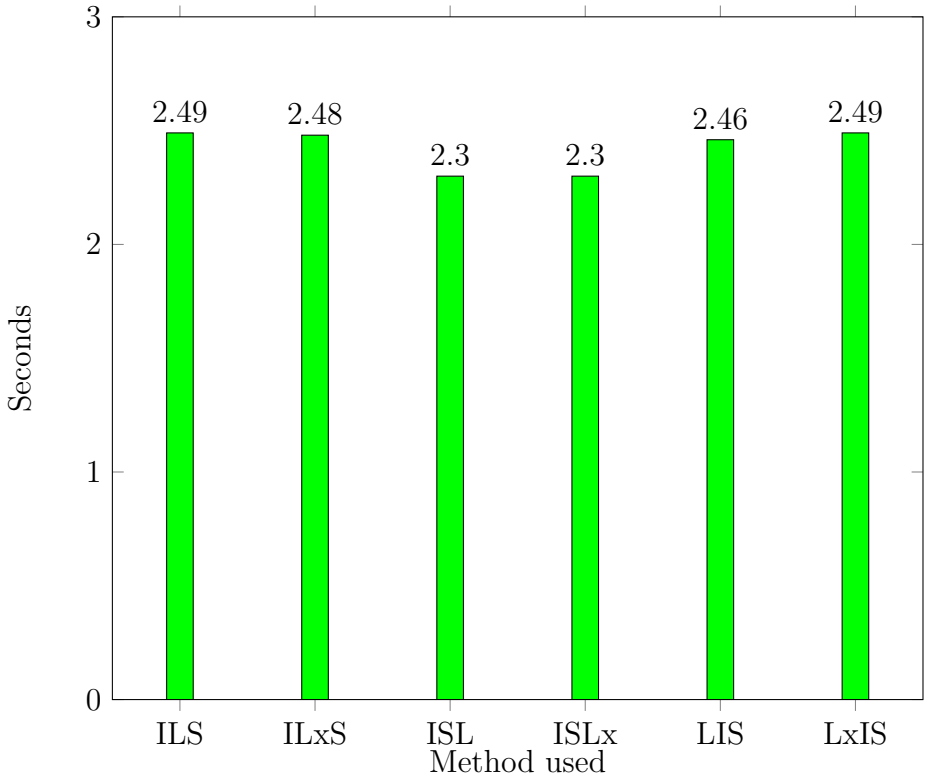

Figure 5.13 Average time took (when finished) for copositive matrices identified in combined method testing by each method (3)

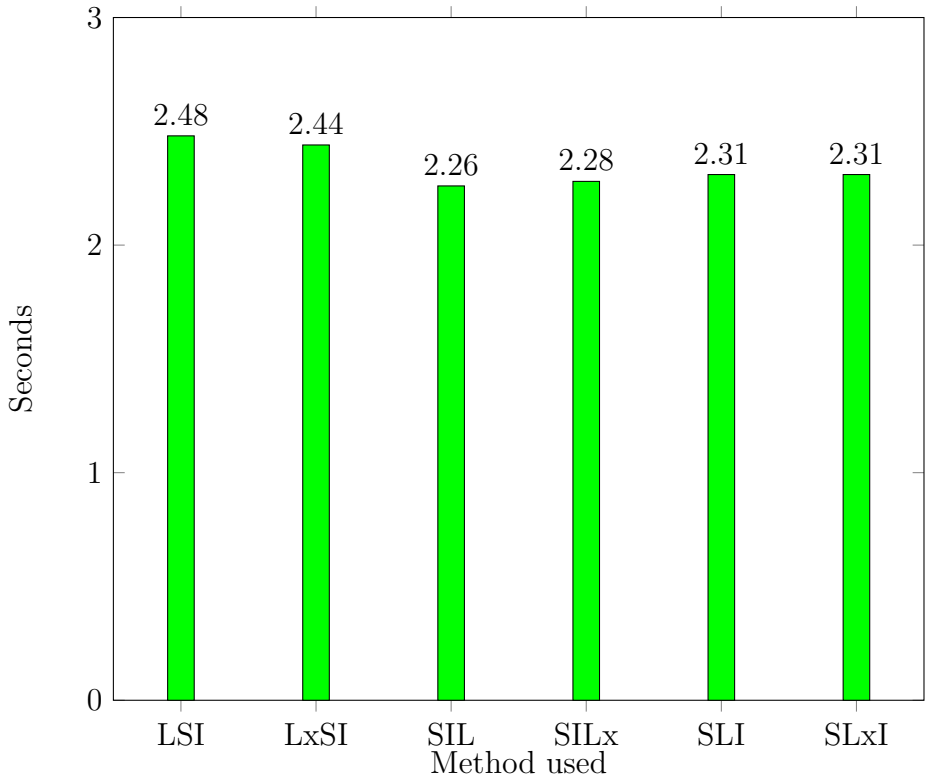

Figure 5.14 Average time took (when finished) for copositive matrices identified in combined method testing by each method (4)

Non-copositive matrices

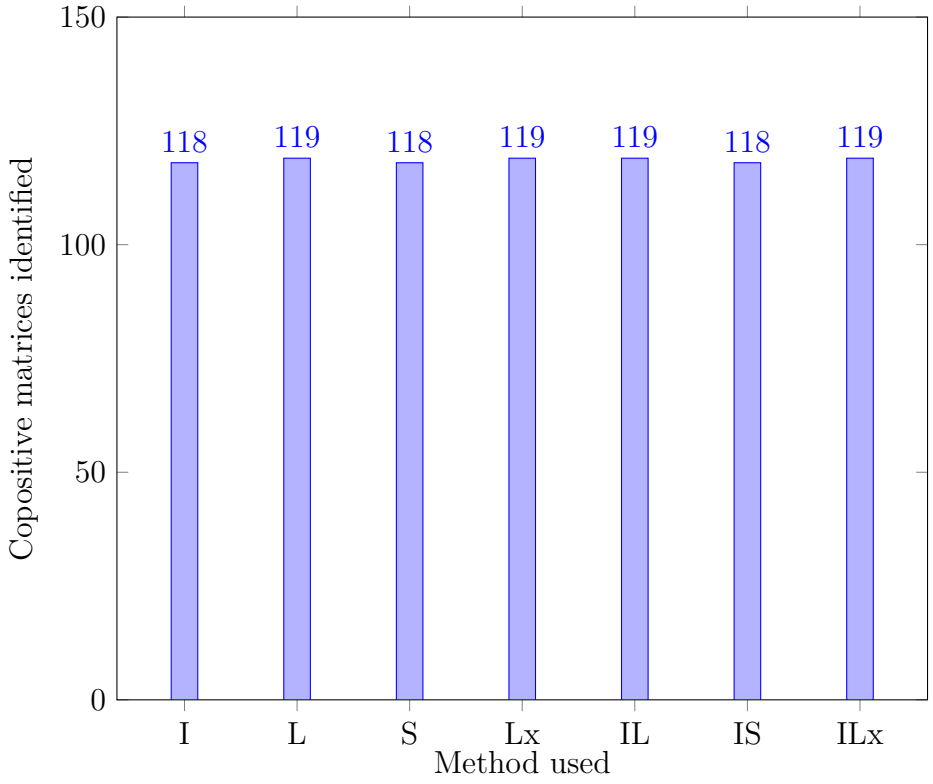

Figure 5.15 Number of non-copositive matrices identified in combined method testing by each method (1)

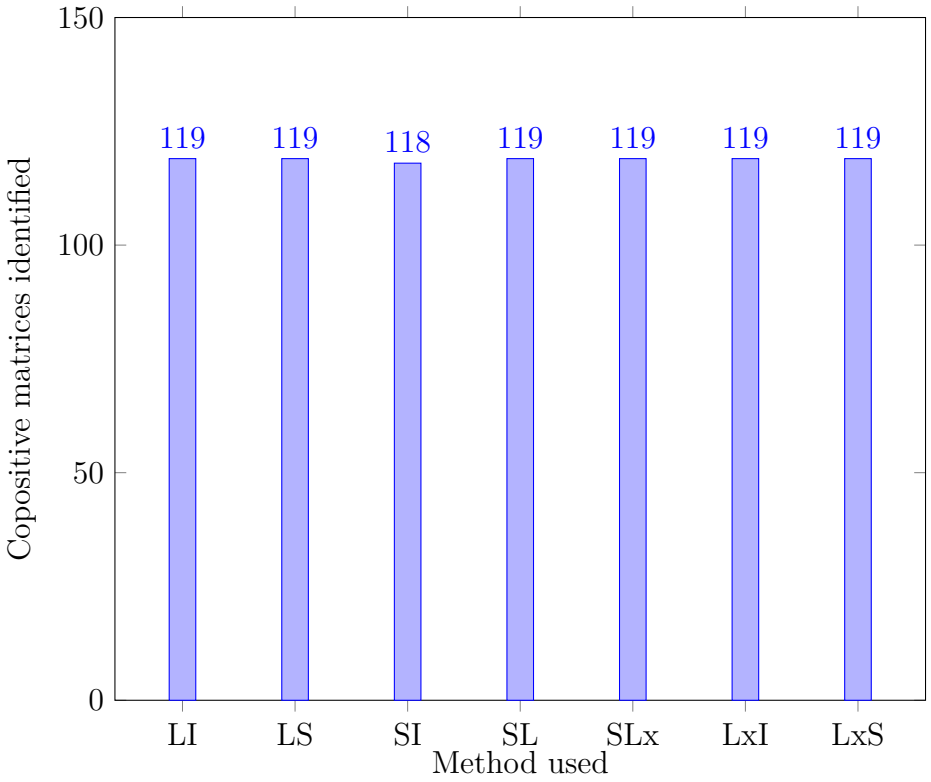

Figure 5.16 Number of non-copositive matrices identified in combined method testing by each method (2)

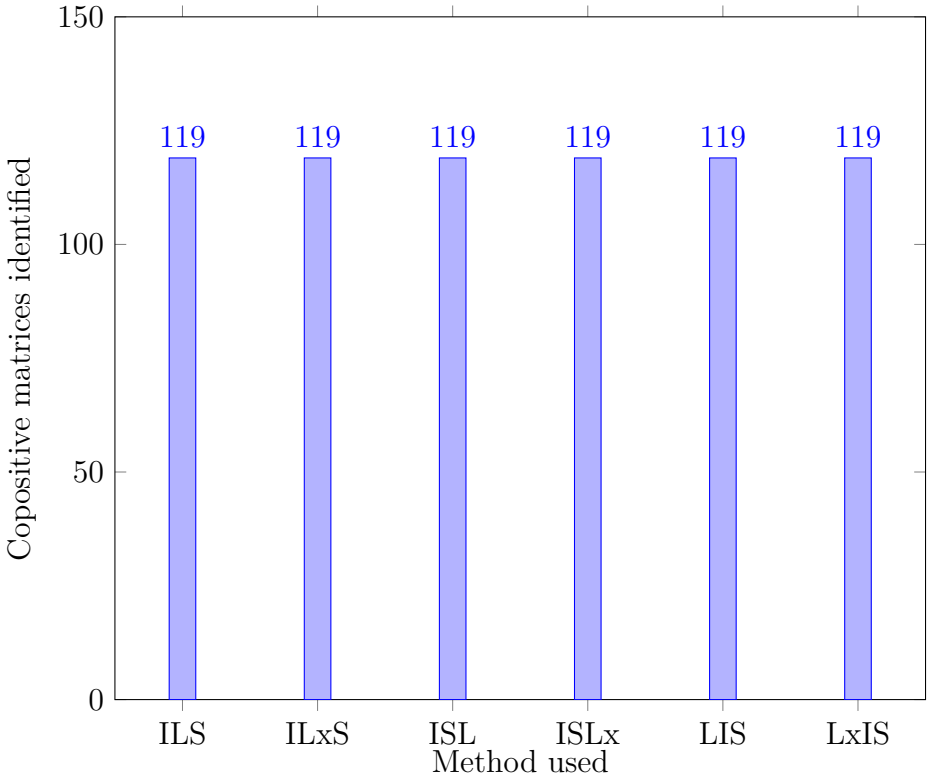

Figure 5.17 Number of non-copositive matrices identified in combined method testing by each method (3)

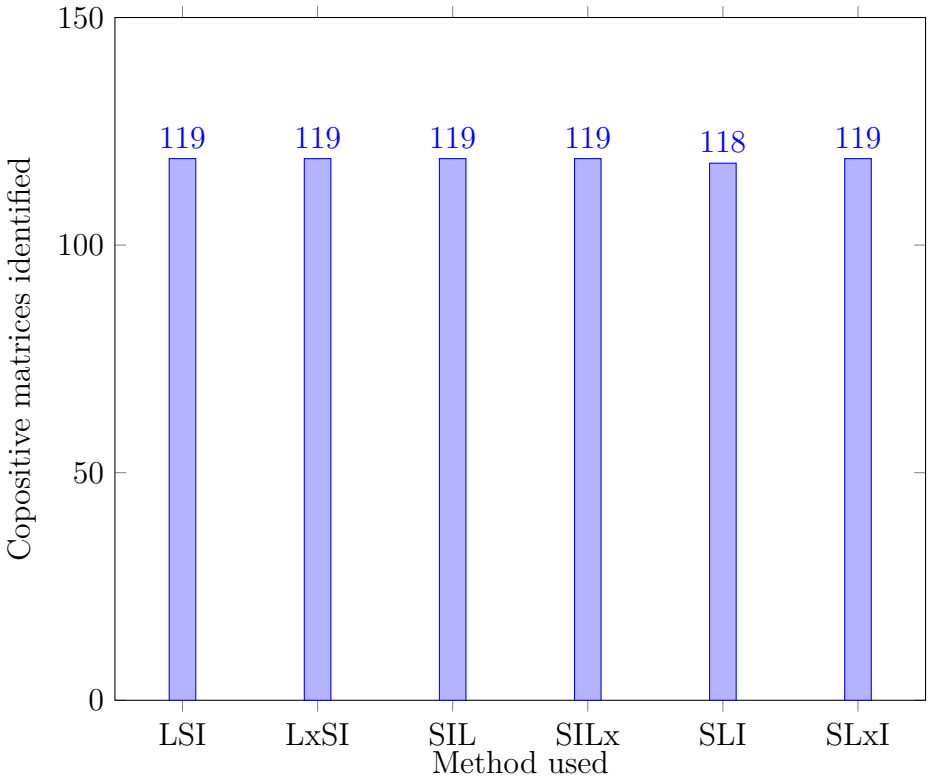

Figure 5.18 Number of non-copositive matrices identified in combined method testing by each method (4)

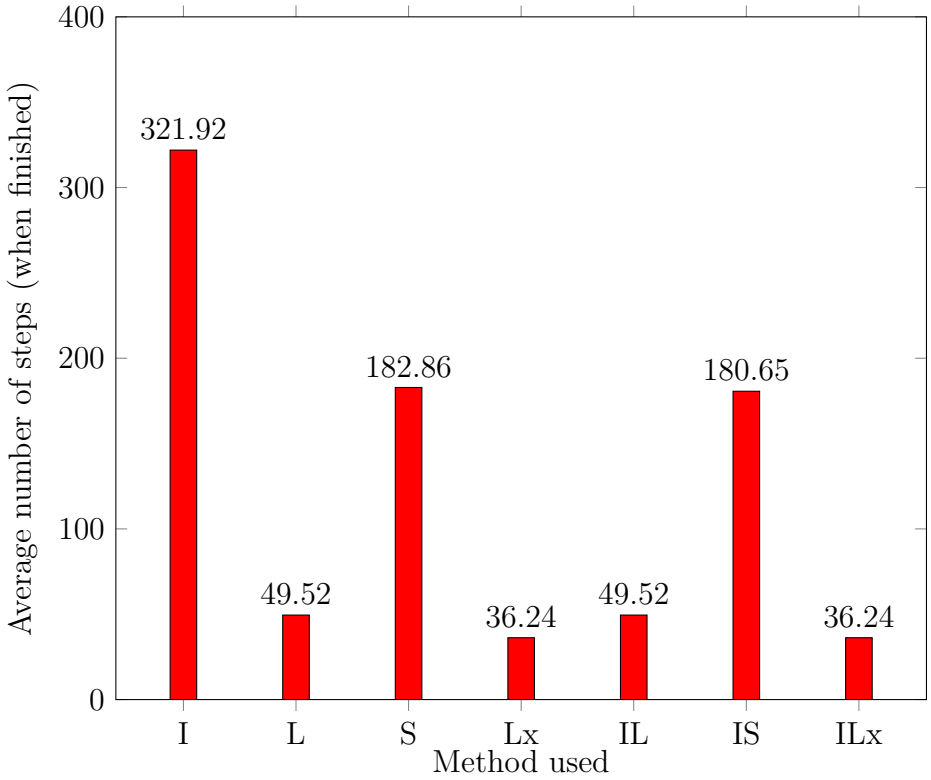

Figure 5.19 Average number of steps (when finished) for non-copositive matrices identified in combined method testing by each method (1)

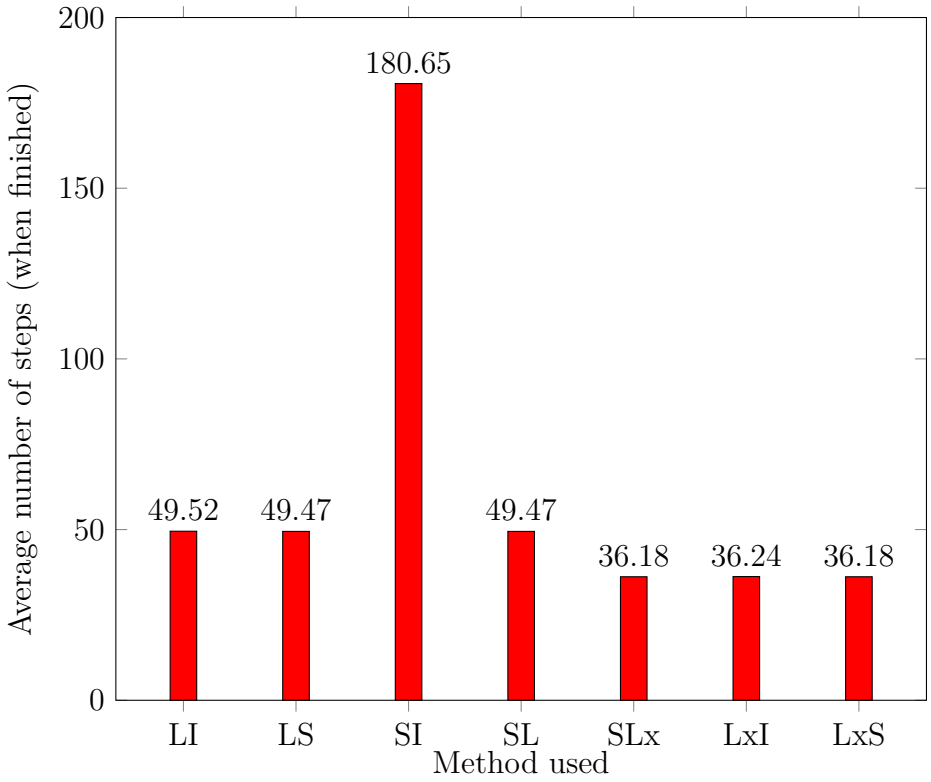

Figure 5.20 Average number of steps (when finished) for non-copositive matrices identified in combined method testing by each method (2)

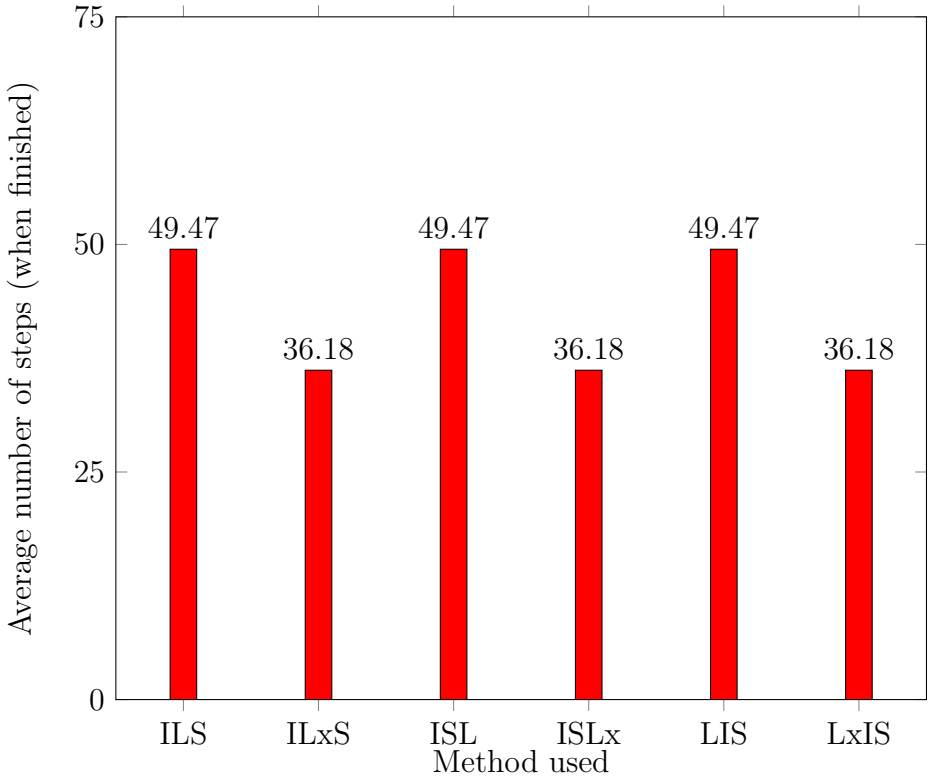

Figure 5.21 Average number of steps (when finished) for non-copositive matrices identified in combined method testing by each method (3)

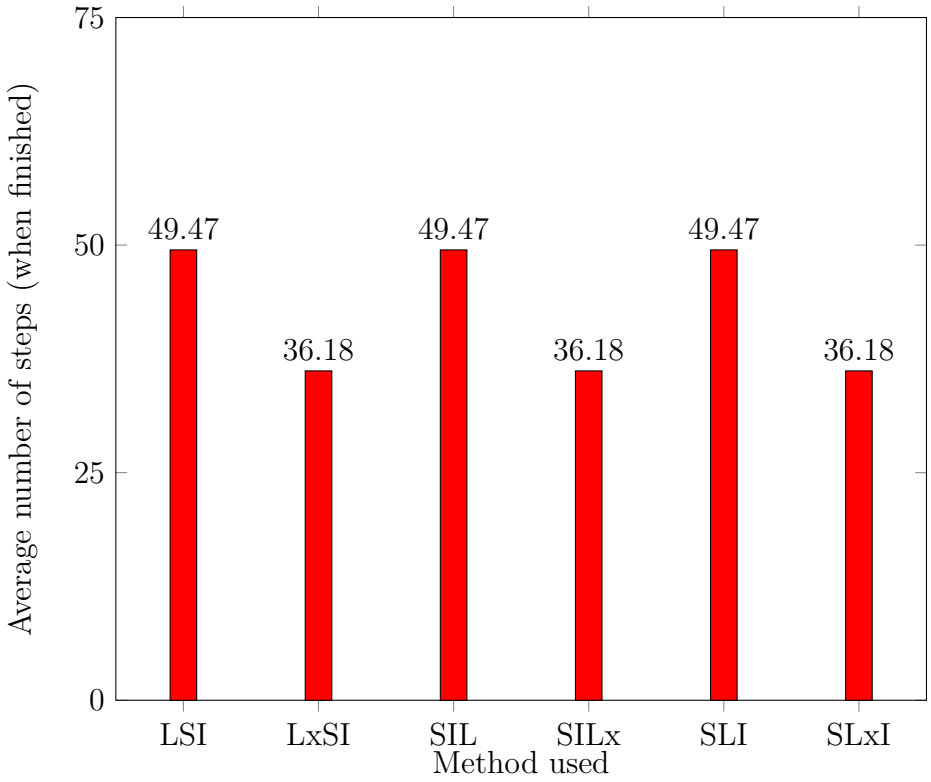

Figure 5.22 Average number of steps (when finished) for non-copositive matrices identified in combined method testing by each method (4)

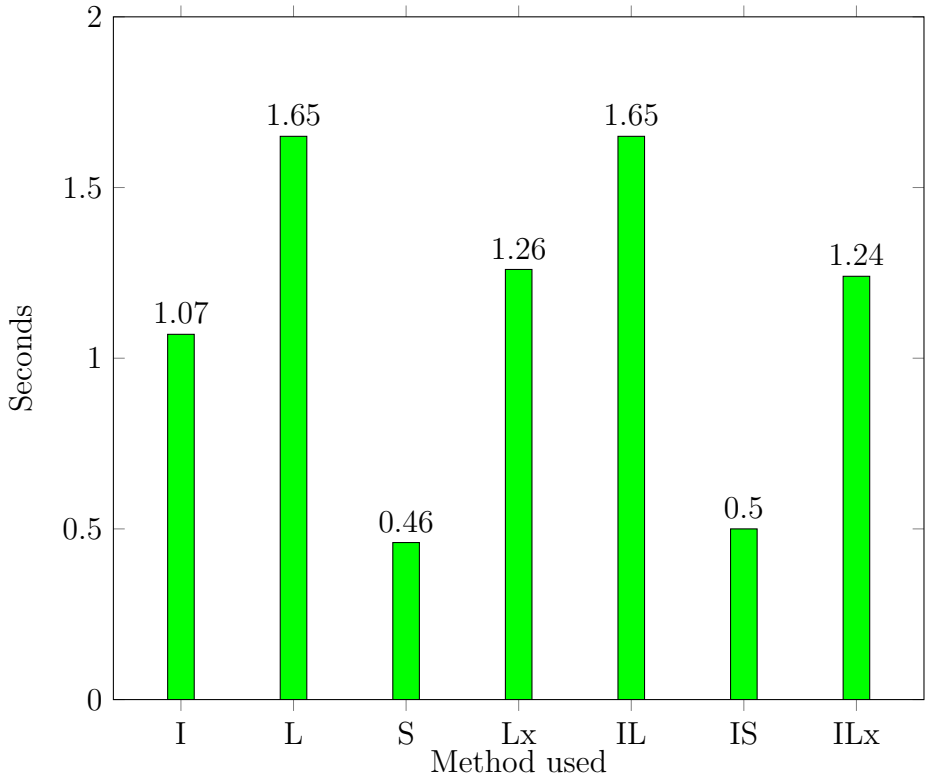

Figure 5.23 Average time took (when finished) for non-copositive matrices identified in combined method testing by each method (1)

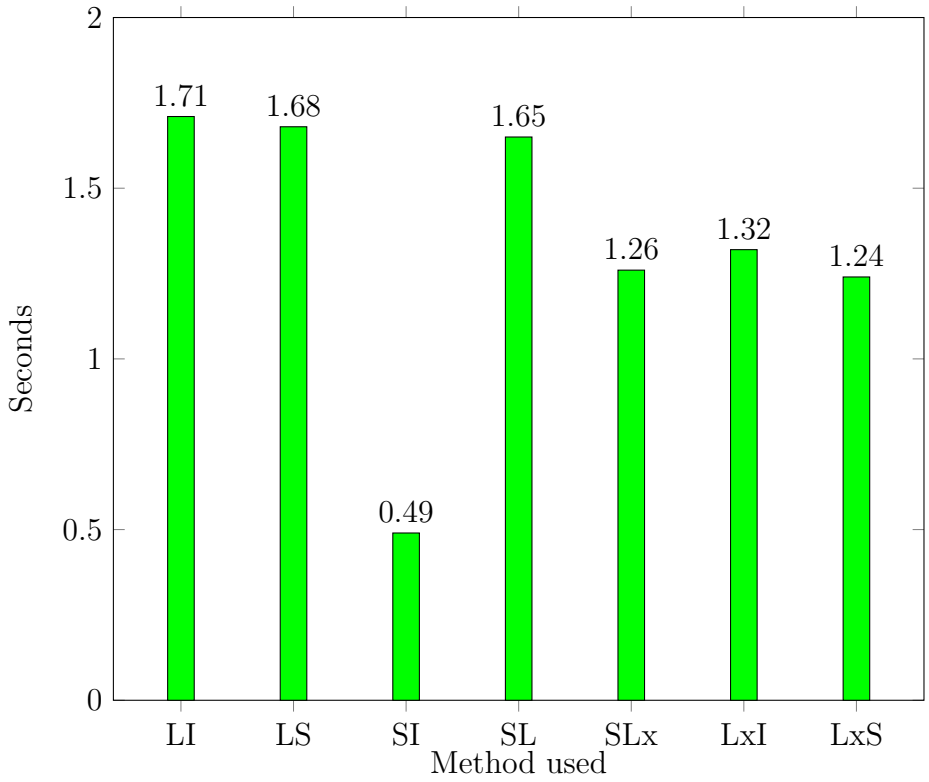

Figure 5.24 Average time took (when finished) for non-copositive matrices identified in combined method testing by each method (2)

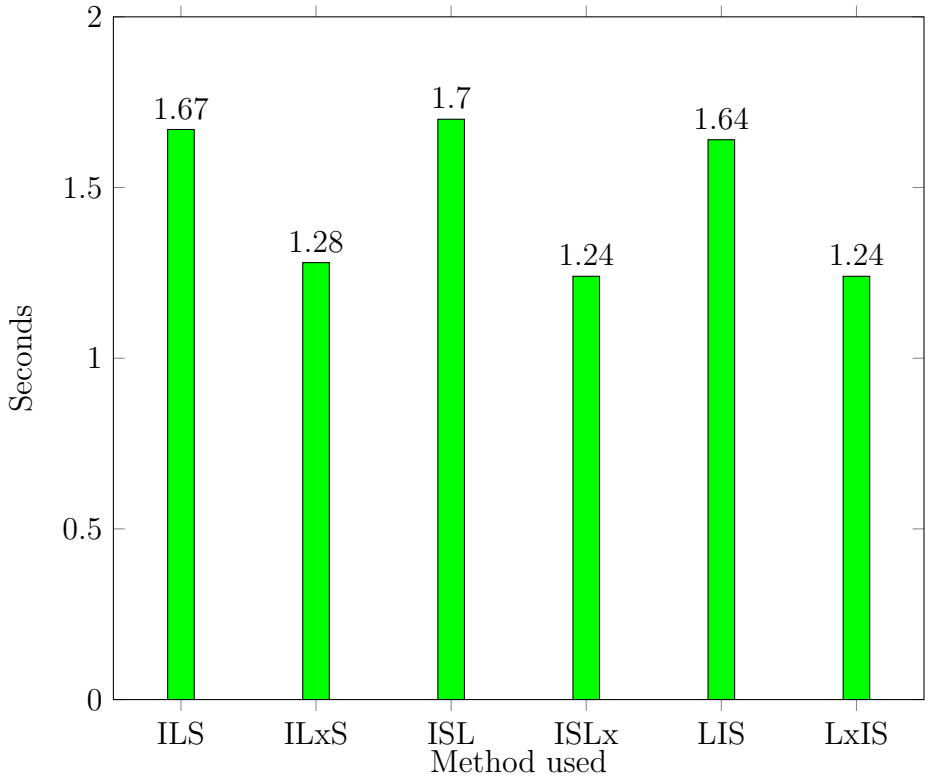

Figure 5.25 Average time took (when finished) for non-copositive matrices identified in combined method testing by each method (3)

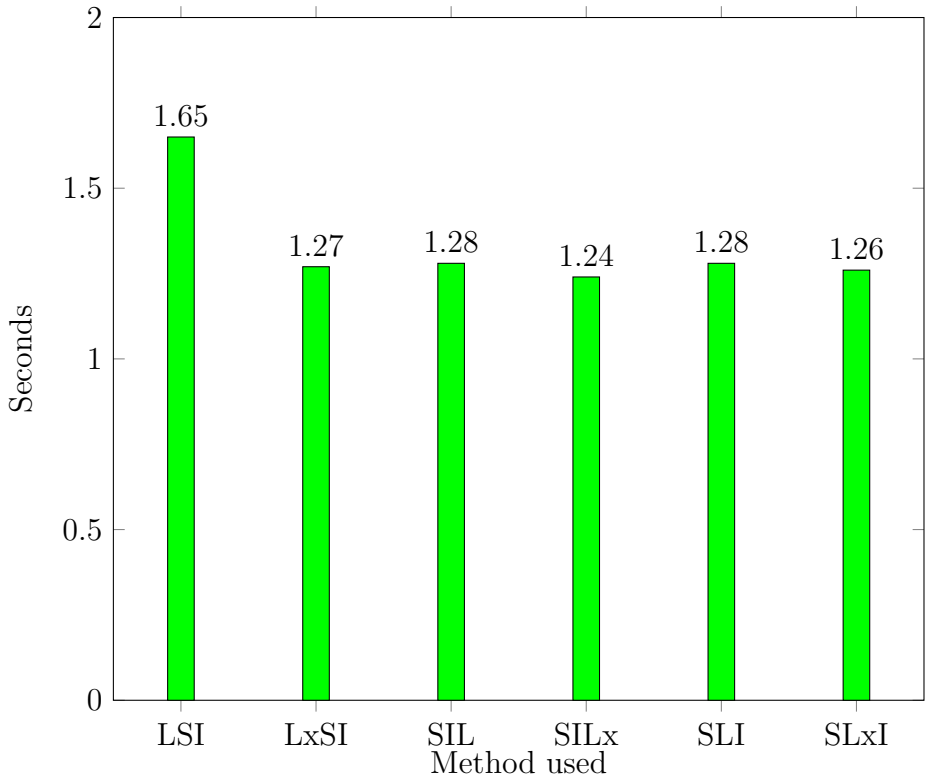

Figure 5.26 Average time took (when finished) for non-copositive matrices identified in combined method testing by each method (4)

Discussion

We can observe that a combination of all three approaches performs very slightly better in general. The combination itself does not seems to matter that much and all methods in general perform really similarly. This is mainly because in almost all of the steps, we have to calculate all three bounds. The `SILx` approach seems to have a very slight edge, and we will continue with it for the rest of the testing as our best method.

Notably, from the singular methods, the quadratic program seems to perform the best, and in our testing, it does exceptionally well with matrices with relatively large diagonal values.

5.5 Look-ahead splitting

In this experiment, we decided to look at options related to splitting intervals, as mentioned in section 3.2.2. In the basic method, which guarantees termination for non-copositive matrices, we split the largest of all intervals from the current interval matrix into two new ones and add them to the queue. However, in theory, if we suspect the matrix is not copositive, it may be beneficial to prioritize some interval splits over others, as they may find the solution quicker. On the other hand, this will provide no boost in performance for copositive matrices, as to mark a matrix copositive, we have to discard all of the intervals in the queue (for more details, refer to section 3.3 or section 3.4), so the additional calculations will

only slow this down.

We decided to try a new approach instead - try to calculate both the lower bound and the upper bound from the interval (or spectral) method and try to choose the one with the lowest/highest lower bound or the one with the lowest-/highest higher bound. As we did not think that it was clear which method would give the best result, we decided to test all of them. This gives us a total of 8 possible approaches, the details of which can be viewed in the source code itself. For brevity, they are named just as digits from 1 to 8 in the figures below. We compared these to the standard approach of splitting the largest interval and, of course, as always, the simplex method.

When we split the interval into two, we obtain two new intervals; let us call them `first` and `second`. We calculate the bound on these new interval vectors and call them `first_value` and `second_value`. Now, we choose the best split based on the following criteria (the numbering is consistent with the method notation in figures):

1. the largest $\min(\inf(\texttt{first\_value}), \inf(\texttt{second\_value}))$
2. the largest $\max(\inf(\texttt{first\_value}), \inf(\texttt{second\_value}))$
3. the smallest $\max(\inf(\texttt{first\_value}), \inf(\texttt{second\_value}))$
4. the smallest $\min(\inf(\texttt{first\_value}), \inf(\texttt{second\_value}))$
5. the largest $\min(\sup(\texttt{first\_value}), \sup(\texttt{second\_value}))$
6. the largest $\max(\sup(\texttt{first\_value}), \sup(\texttt{second\_value}))$
7. the smallest $\max(\sup(\texttt{first\_value}), \sup(\texttt{second\_value}))$
8. the smallest $\min(\sup(\texttt{first\_value}), \sup(\texttt{second\_value}))$

5.5.1 Interval method

Copositive matrices

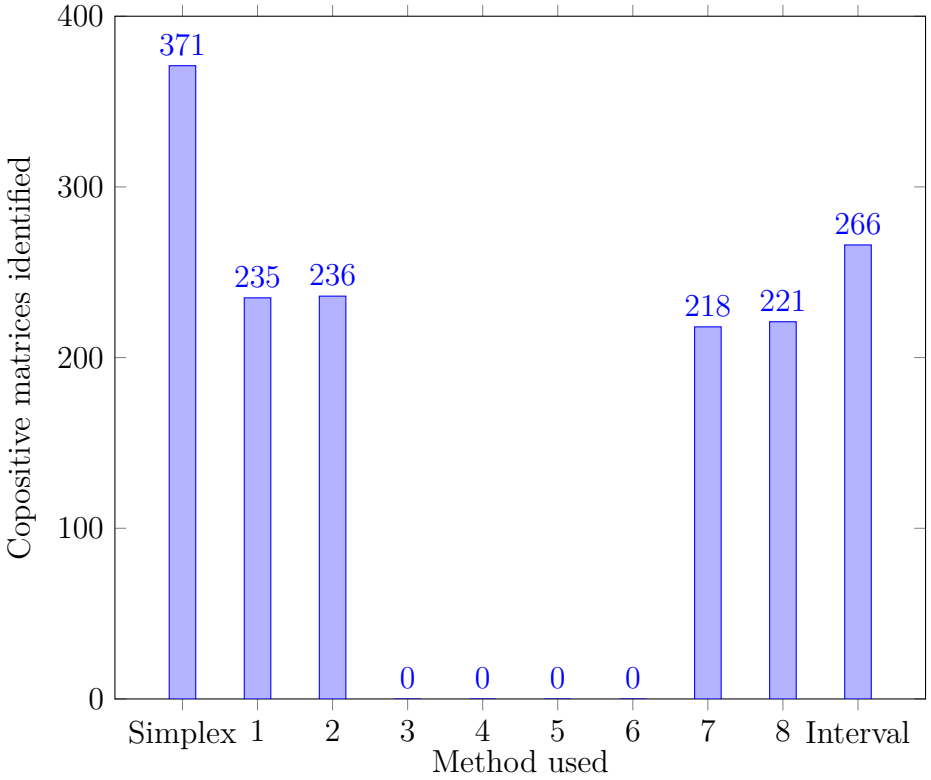

Figure 5.27 Number of copositive matrices identified in look-ahead splitting by each method (interval)

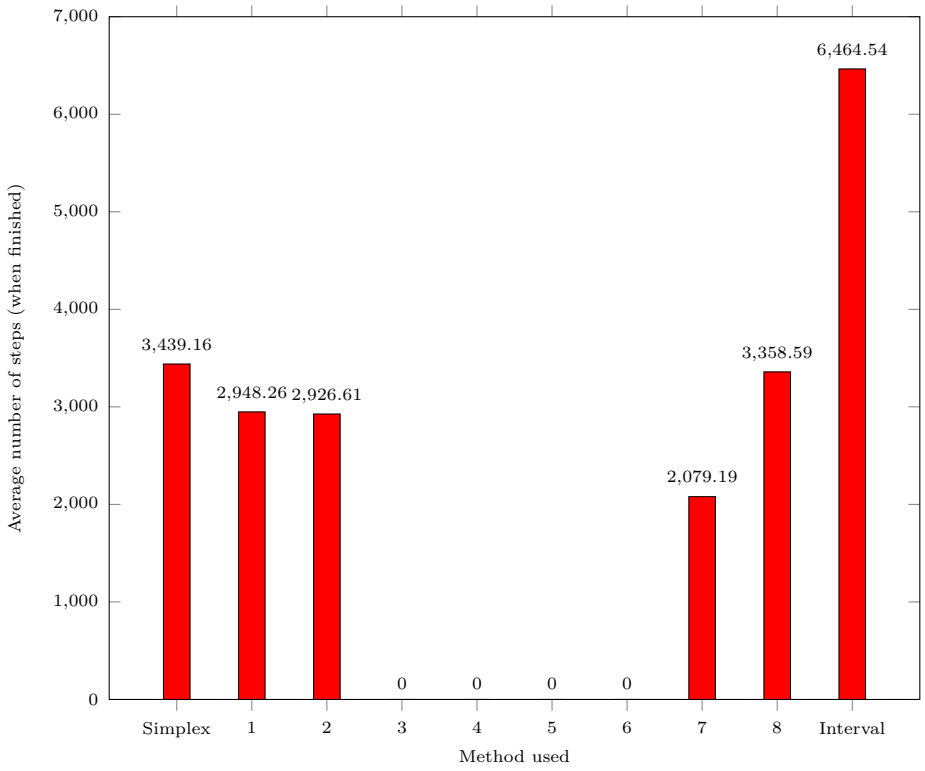

Figure 5.28 Average number of steps (when finished) for copositive matrices identified in look-ahead splitting by each method (interval)

Note that the Average number of steps (when finished) is described as 0 because the average is not well defined (they did not identify a single copositive matrix).

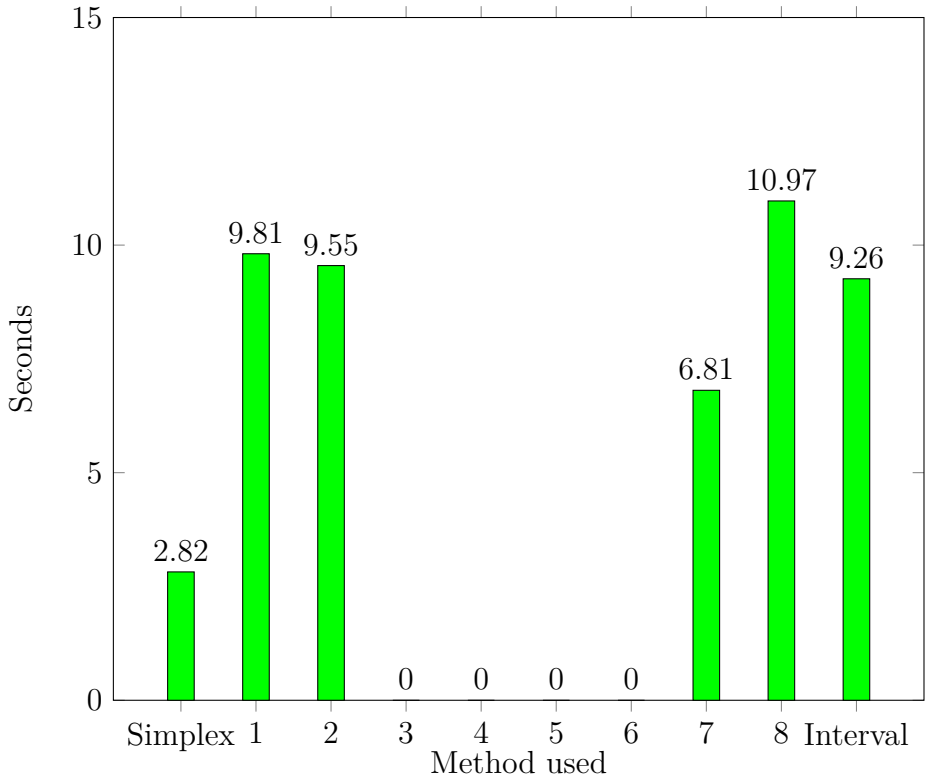

Figure 5.29 Average time took (when finished) for copositive matrices identified in the look-ahead splitting by each method (interval)

Again, the 0 represents that the average could not be calculated.

Non-copositive matrices

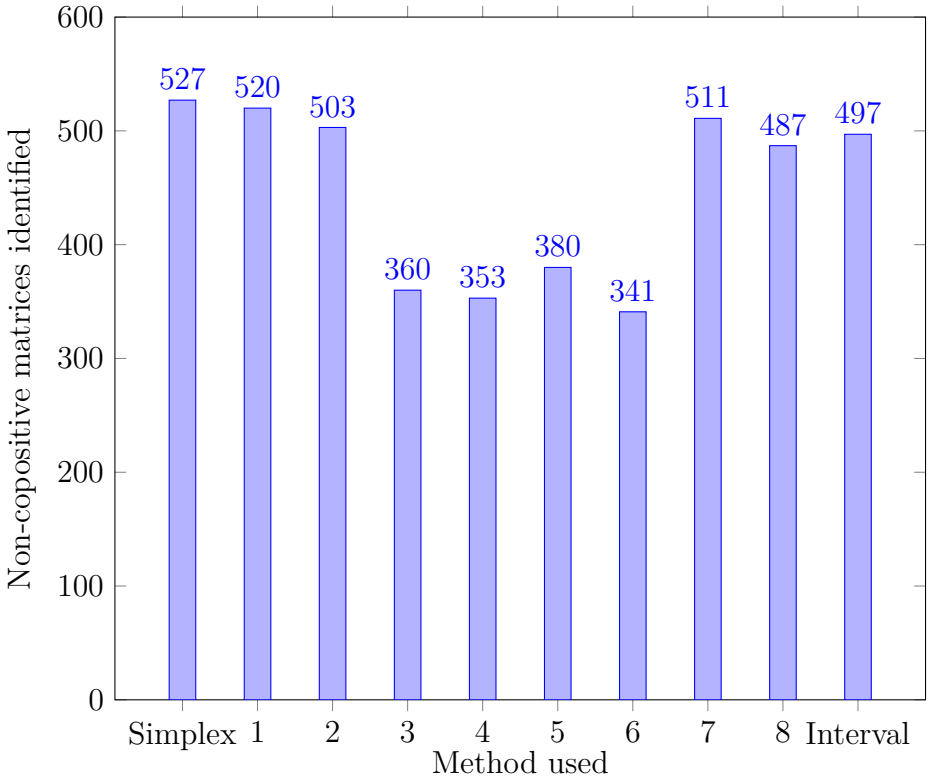

Figure 5.30 Number of non-copositive matrices identified in look-ahead splitting by each method (interval)

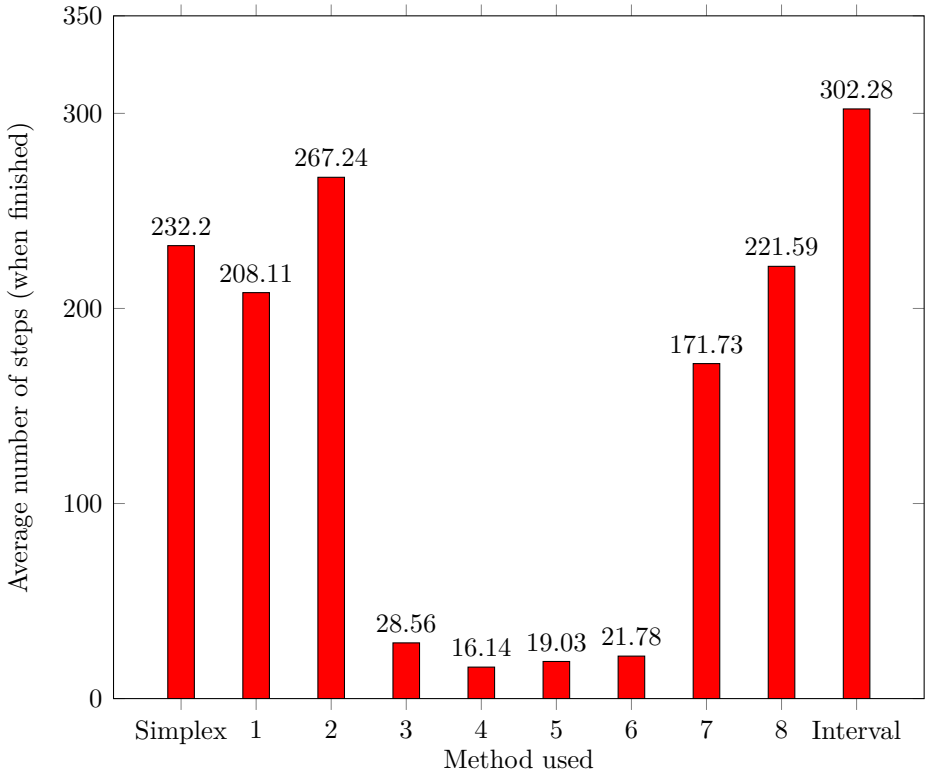

Figure 5.31 Average number of steps (when finished) for non-copositive matrices identified in look-ahead splitting by each method (interval)

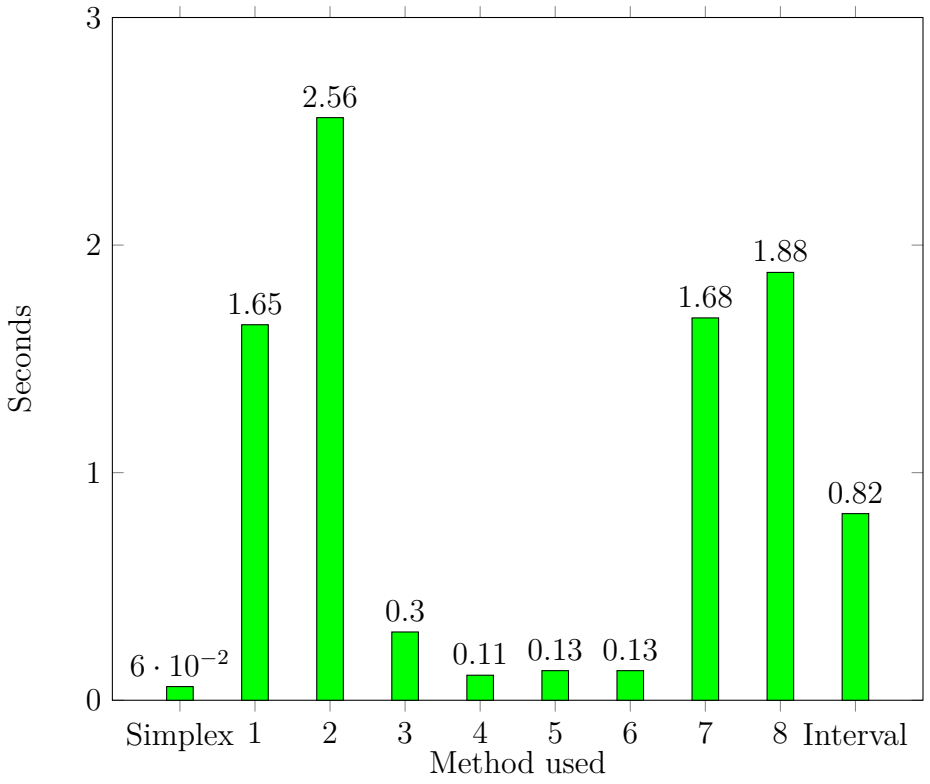

Figure 5.32 Average time took (when finished) for non-copositive matrices identified in the look-ahead splitting by each method (interval)

Discussion

Here, we can see that some of the approaches are extremely bad on either copositive or non-copositive matrices, namely methods 3, 4, 5, and 6 could not identify a single copositive matrix in a given time restriction. Some of these look-ahead approaches actually outperformed our non-look-ahead method in, for example, non-copositive matrices in "Average number of steps (when finished)", but ultimately they did not seem good enough to replace the standard biggest interval splitting, so we used that for further tests.

5.5.2 Spectral method

Copositive matrices

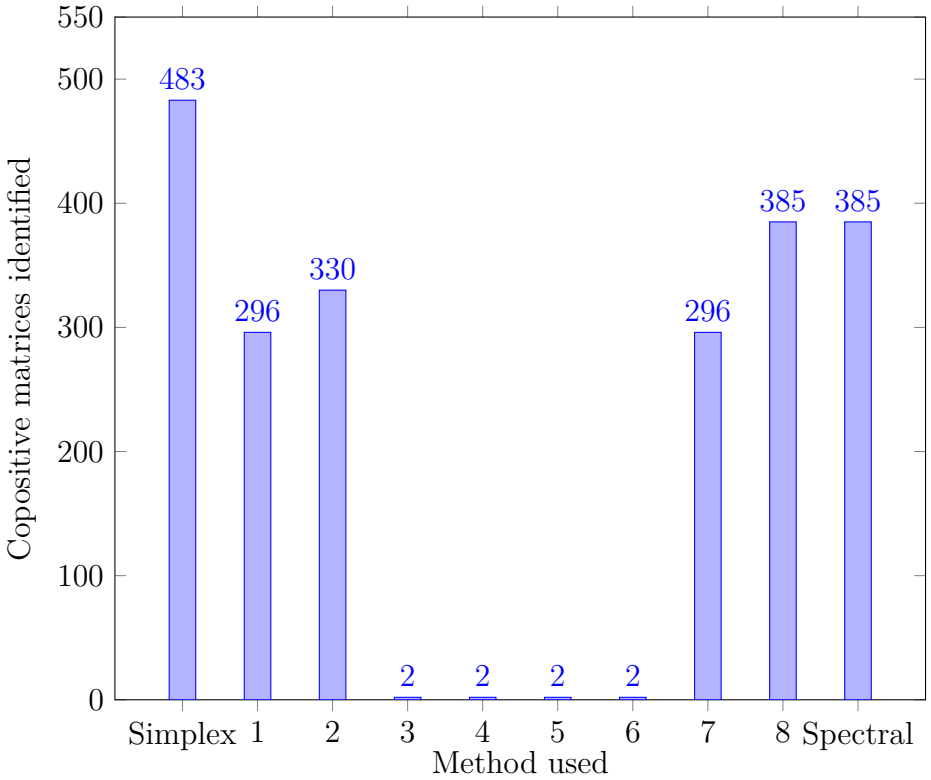

Figure 5.33 Number of copositive matrices identified in look-ahead splitting by each method (spectral)

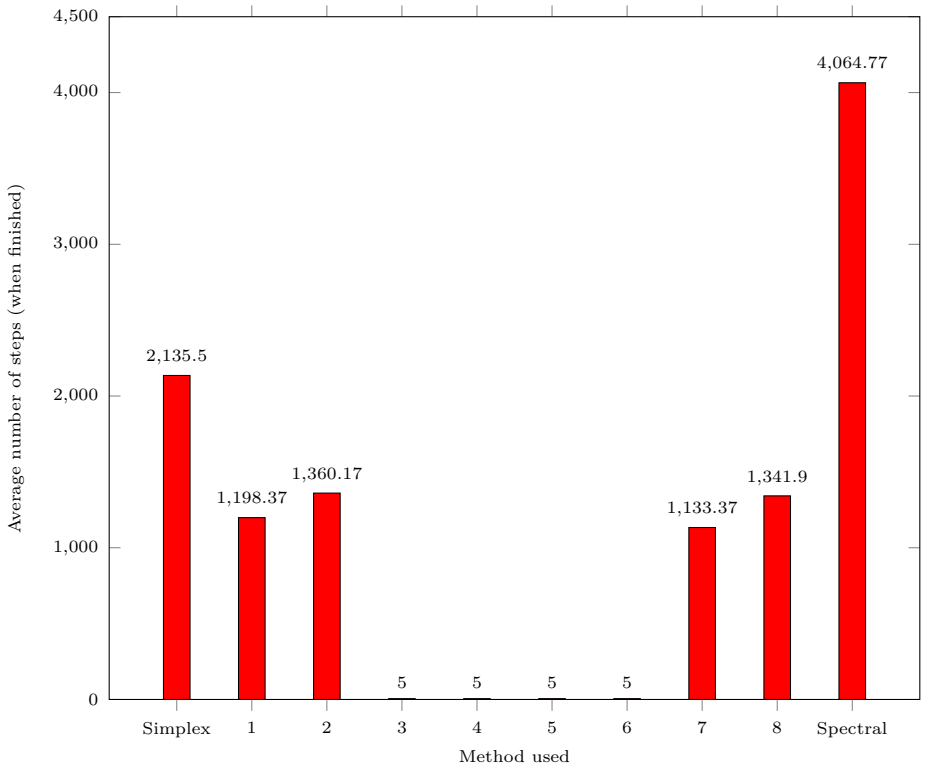

Figure 5.34 Average number of steps (when finished) for copositive matrices identified in look-ahead splitting by each method (spectral)

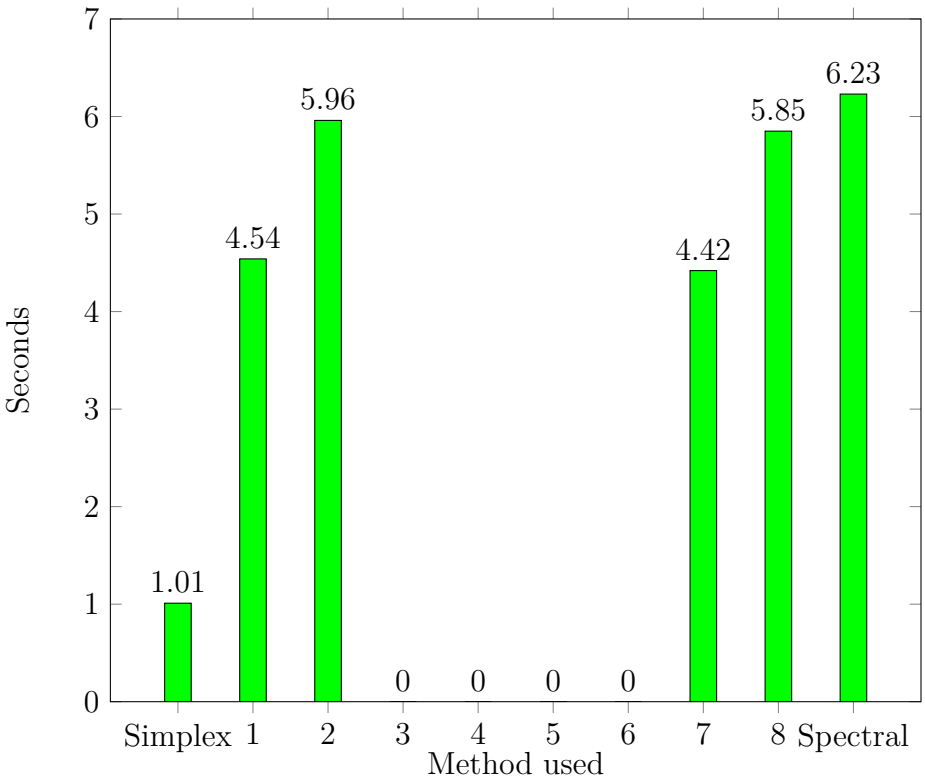

Figure 5.35 Average time took (when finished) for copositive matrices identified in the look-ahead splitting by each method (spectral)

The exact value for the average in methods 3, 4, 5 and 6 was 0.00307. This is insignificant, as they only identified 2 matrices, so it seems it was guessed as copositive without splitting at all.

Non-copositive matrices

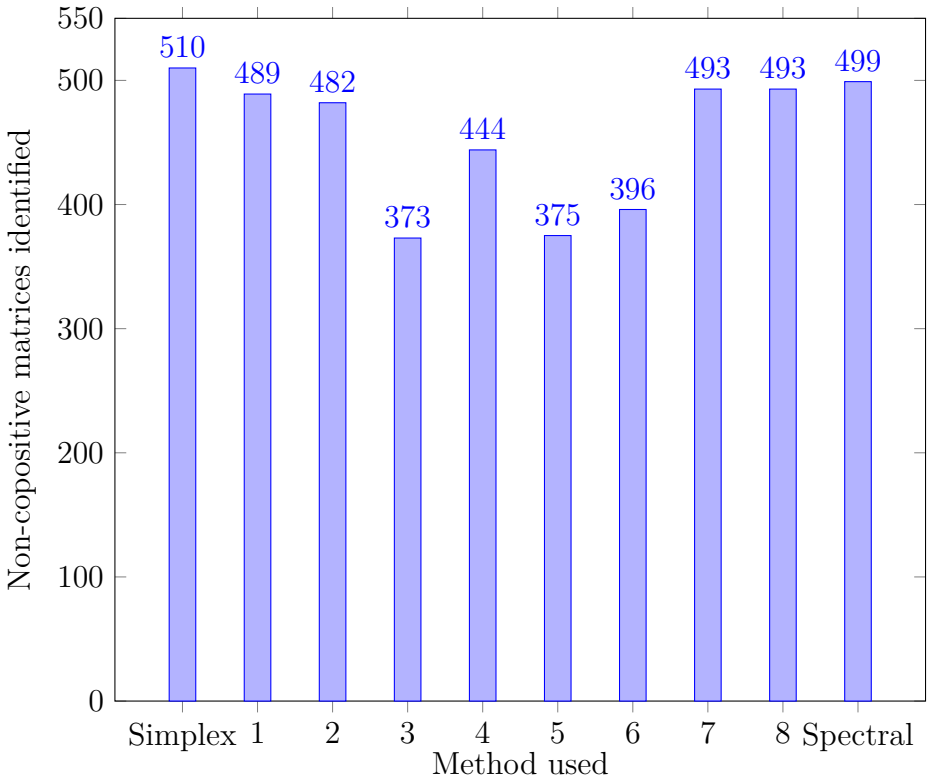

Figure 5.36 Number of non-copositive matrices identified in look-ahead splitting by each method (spectral)

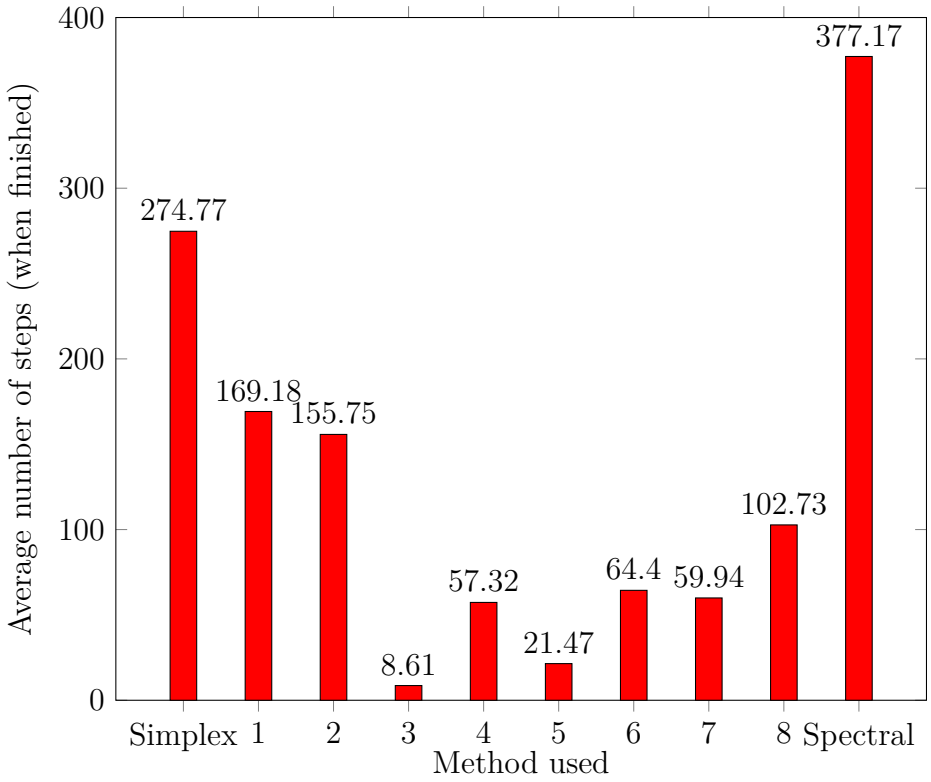

Figure 5.37 Average number of steps (when finished) for non-copositive matrices identified in look-ahead splitting by each method (spectral)

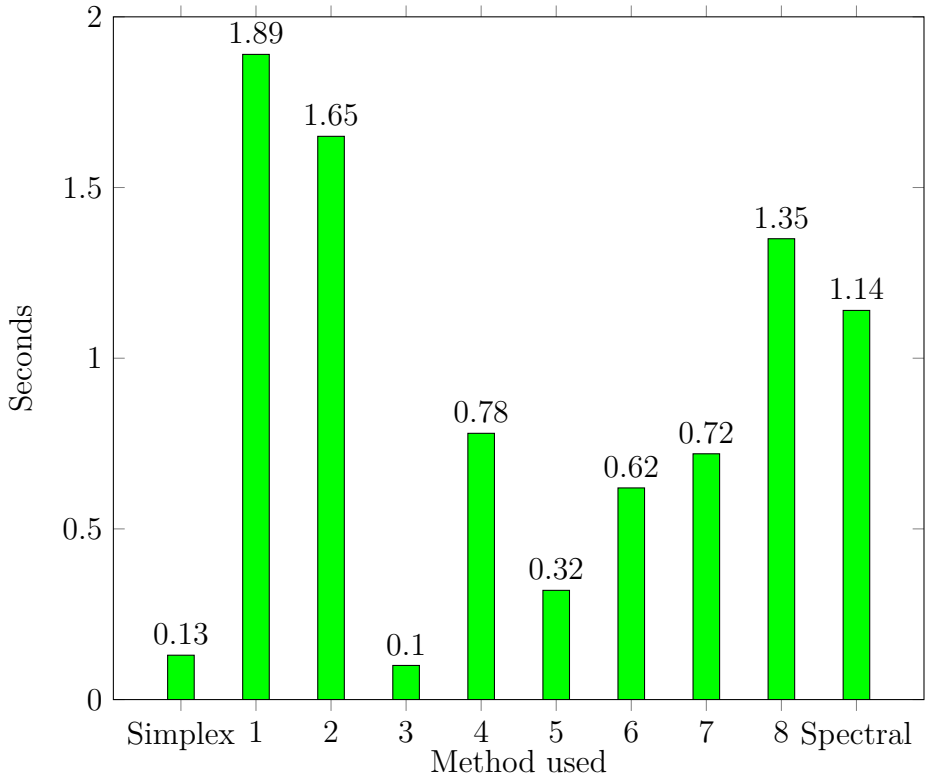

Figure 5.38 Average time took (when finished) for non-copositive matrices identified in the look-ahead splitting by each method (spectral)

Discussion

We see very similar results to the results in the previous subsection, where while some of the look-ahead methods seem to do better on copositive (or non-copositive) matrices, for example, taking many few steps, they fail to do so reliably, and for both types of matrices. Because of this, again, we choose to discard these approaches and continue with the simplest, largest interval splitting.

5.6 Non-copositive candidate

As described in section 4.4.2, we have many approaches to choose a possible non-copositive candidate when bounding. As always, we compare them with the simplex method. The `R` stands for random in the tests and the number after it denotes how many random candidates we tried on a given interval vertex.

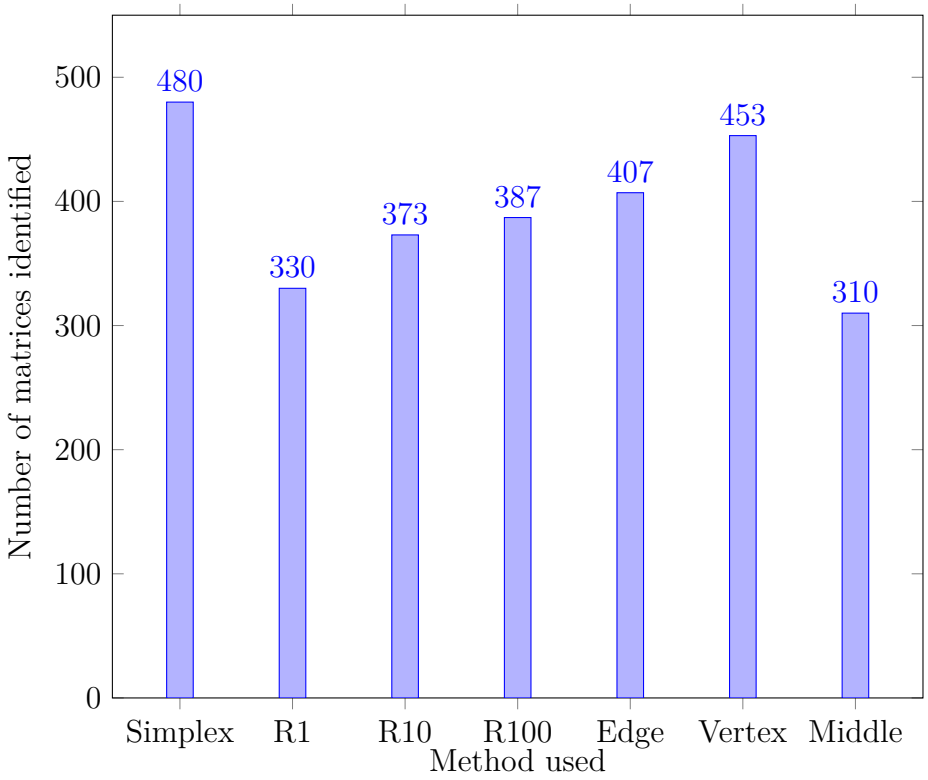

Figure 5.39 Number of non-copositive matrices identified by different candidate methods in testing

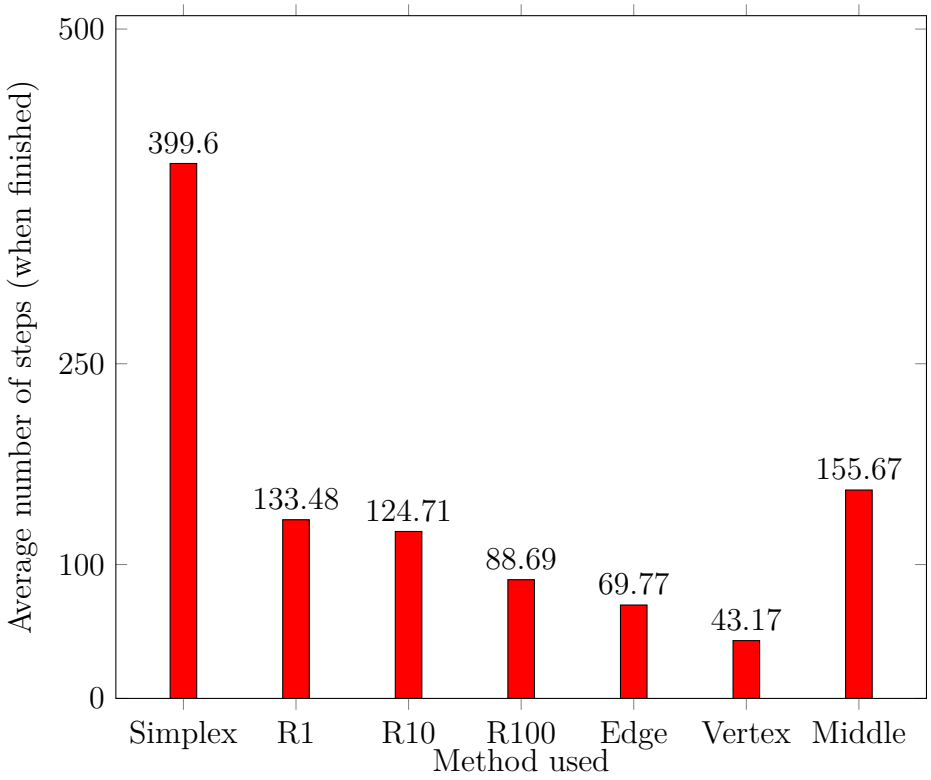

Figure 5.40 Average number of steps (when finished) for non-copositive matrices identified by different candidate methods in testing

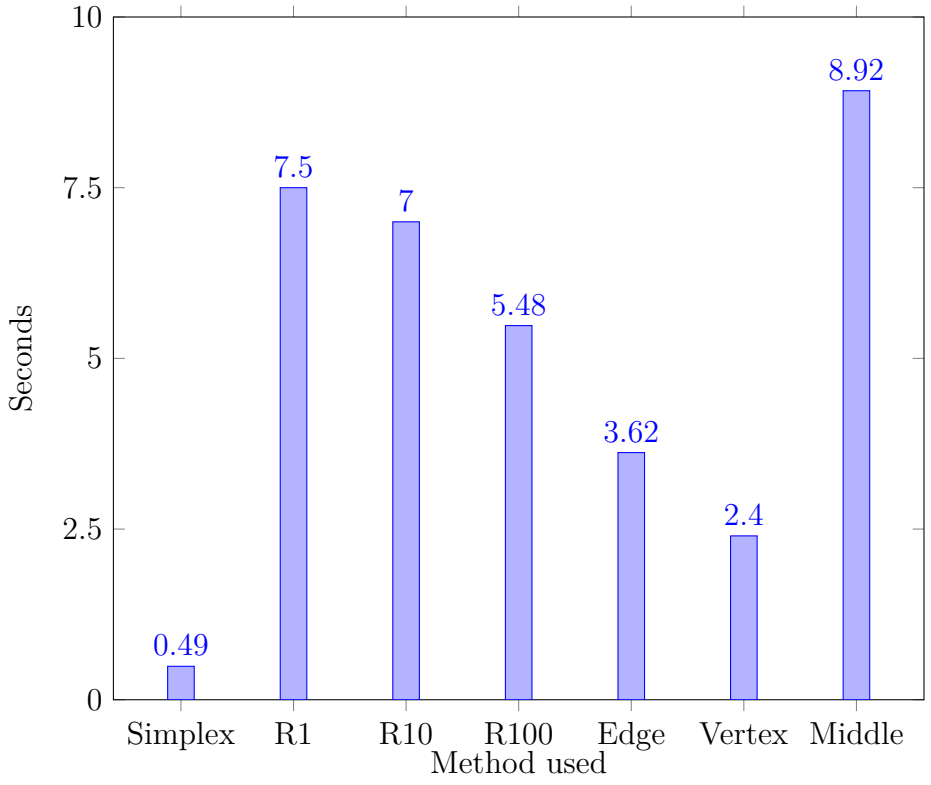

Figure 5.41 Average time took (when finished) for non-copositive matrices identified by different candidate methods in testing

Discussion

These figures show that the simplex method is really good compared with our ideas in identifying non-copositive matrices. It gives by far the best time, and even tho it takes many more steps than our methods, its steps are so much faster that it makes up for it. From our methods, Middle came as the worst, followed by random testing. Random testing, however, seems to get better as we test for more candidates. An interesting idea would be to look into the optimal value (if such a thing even exists) of random candidates for which to test. Our best methods were Edge and Vertex, with Vertex slightly outperforming the Edge approach. Based on these results, we decided to go with the Vertex approach as our best method, although it is straightforward to switch to Edge in the source code directly, just by changing the function called.

5.7 Large general testing

In this testing set, we decided to test a more significant number of matrices on just two methods - the simplex method and the `spectral_interval_linear_example_test`, as we thought this method showed the most promise at that time. In these tests, we set the number of matrices at $n = 10000$.

Copositive matrices

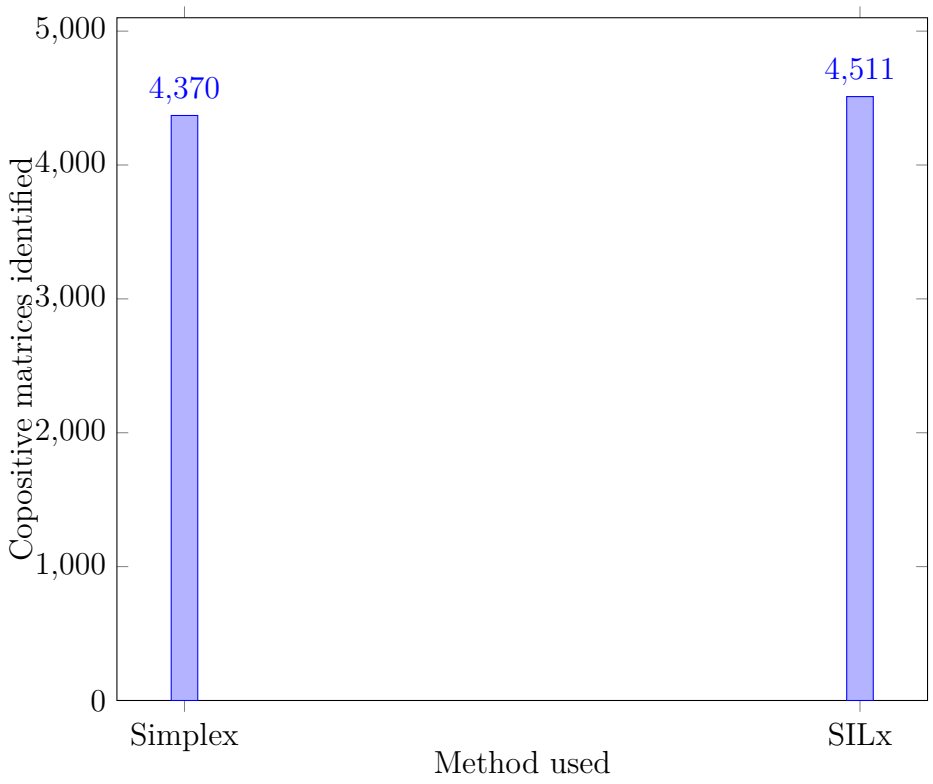

Figure 5.42 Number of copositive matrices identified in simplex/SILx method comparison testing

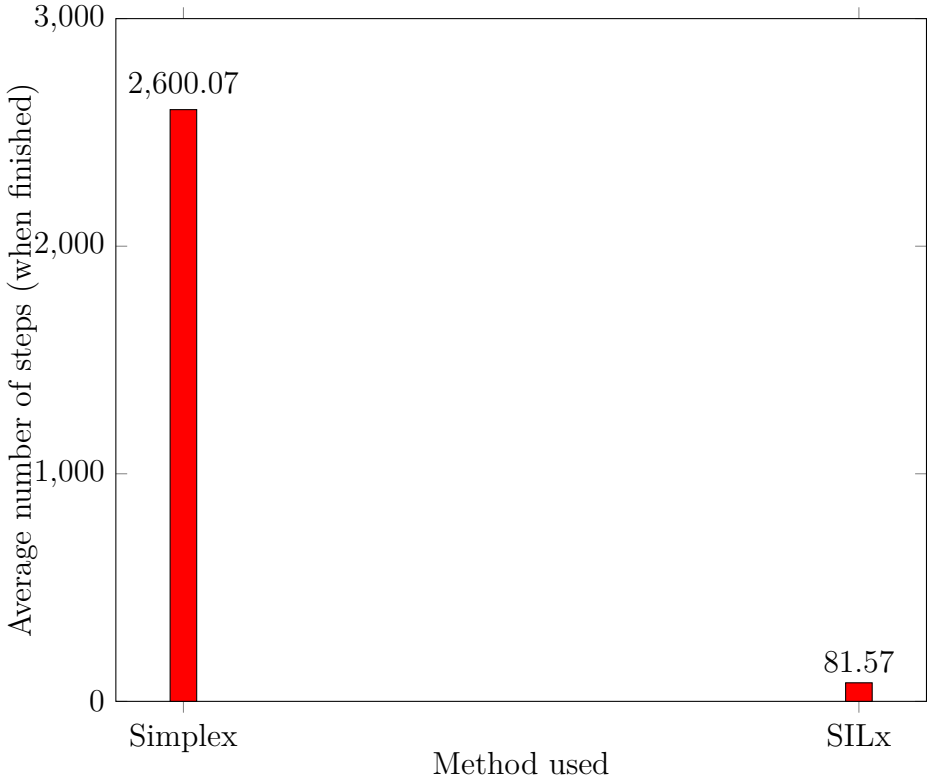

Figure 5.43 Average number of steps (when finished) for copositive matrices identified in simplex/SILx method comparison testing

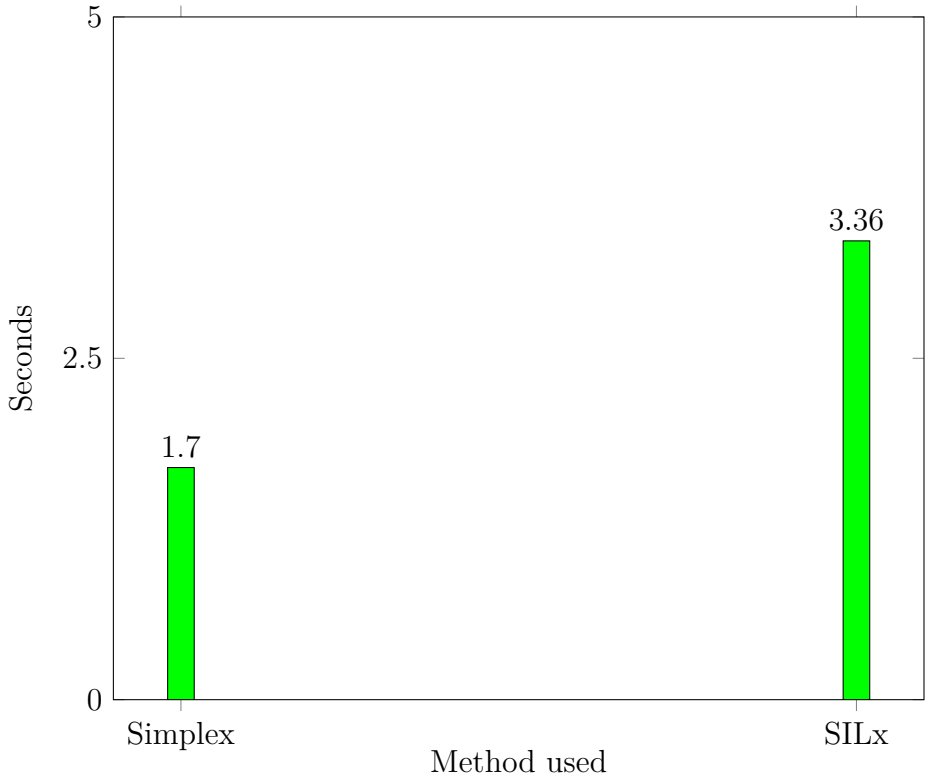

Figure 5.44 Average time took (when finished) for copositive matrices identified in simplex/SILx method comparison testing

Non-copositive matrices

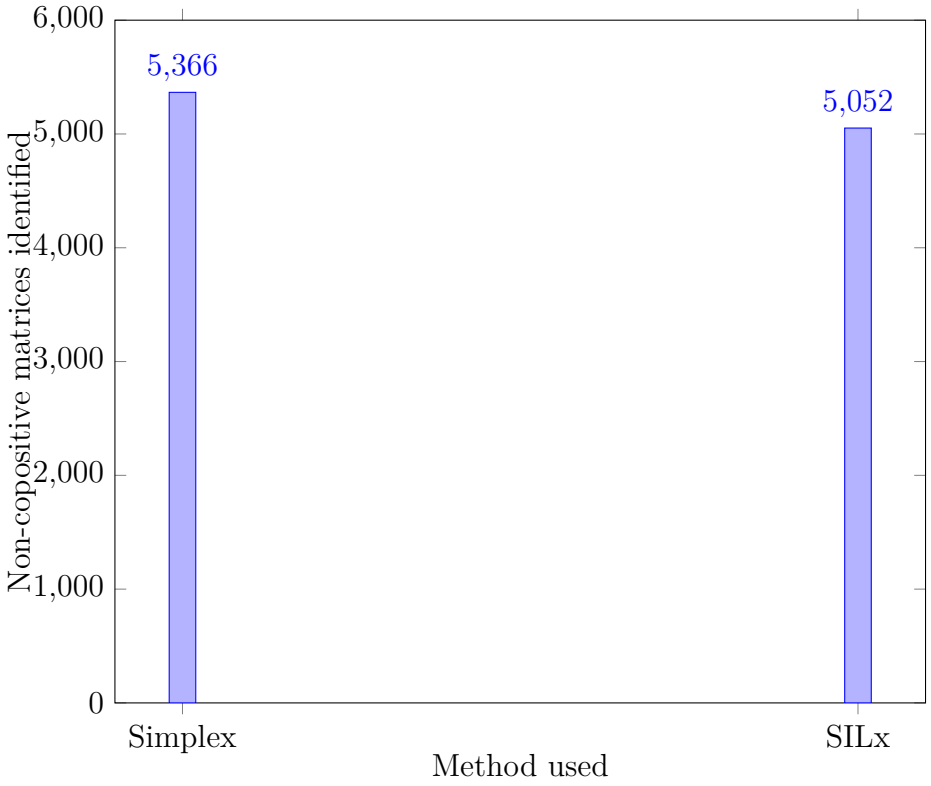

Figure 5.45 Number of non-copositive matrices identified in simplex/SILx method comparison testing

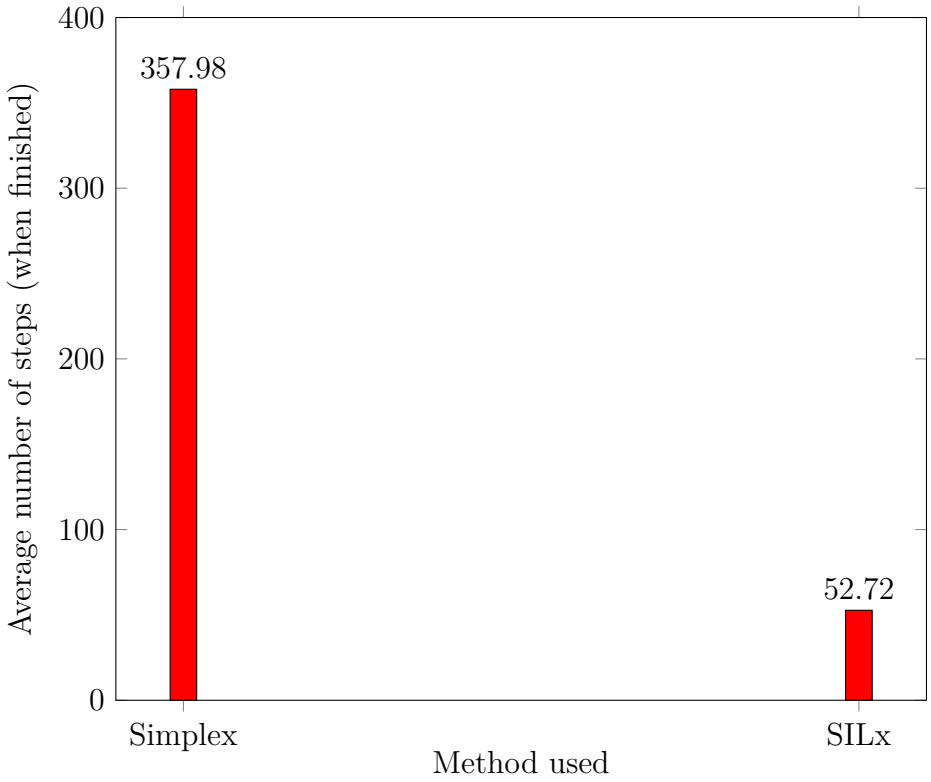

Figure 5.46 Average number of steps (when finished) for non-copositive matrices identified in simplex/SILx method comparison testing

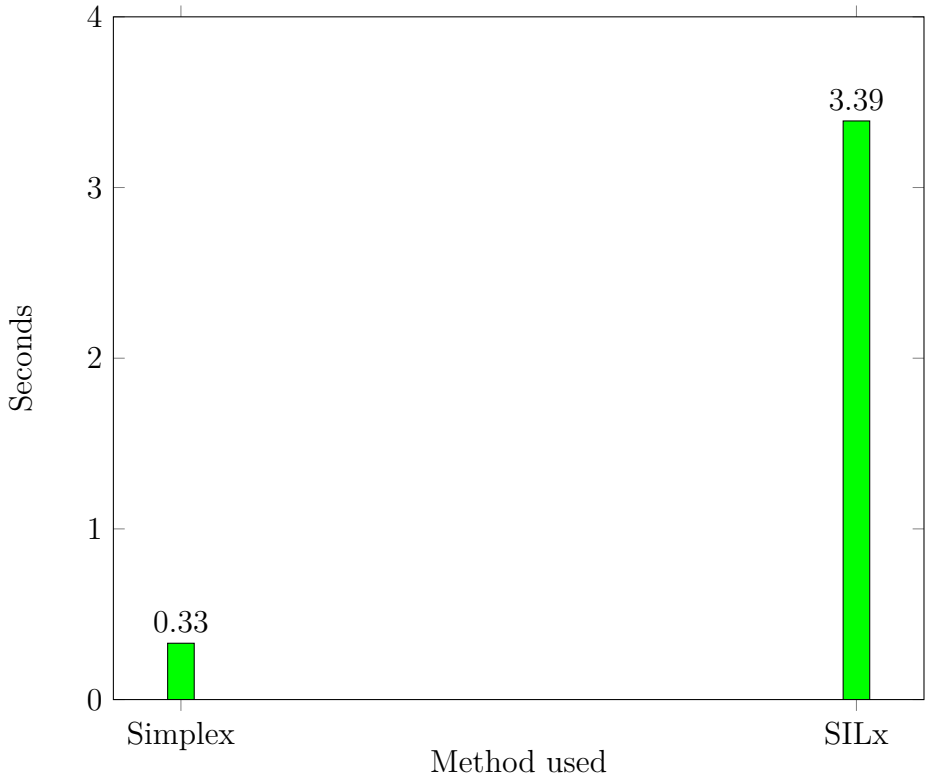

Figure 5.47 Average time took (when finished) for non-copositive matrices identified in simplex/SILx method comparison testing

Discussion

We can notice that the simplex branch-and-bound method did not finish on all copositive matrices - meaning that there were copositive matrices that our method identified, while the simplex did not (took too much time). This shows promise, and we took notice of such matrices to try and isolate what type of matrices these are. The simplex method seems to outperform our method on non-copositive matrices by far, but on copositive matrices, our method does seem to have a very slight edge.

5.8 Conclusion

In the end, by using vertex-based-non-copositivity candidate generation with a combined approach of spectral, interval, and quadratic test methods, we were able to construct a test that performs better on general copositive matrices than the simplex branch-and-bound method, which we consider a good result.
We noticed that matrices, where our method does **much** better than the simplex method, often have rather large values on the diagonal; thus, we tried to add even more to the diagonal (even beyond compensating for non-copositivity, as explained in section 4.3.1). This means the number of copositive matrices will be larger than the number of non-copositive matrices, which, as explained, is the nature of random matrices, so this may seem somewhat artificial. Another problem with this approach is that the matrices would often just become positive semi-definite, which is not an interesting case for the algorithm, as these matrices are always copositive.

These matrices have to be generated by a separate function that does the compensation. Of course, the value added to the diagonal depends on the size of the matrix, and this thing is challenging to figure out on its own - the larger the value, the better our methods seem compared to the simplex branch-and-bound approach, but the slower the random matrix generation seems to be.

We mention this for two reasons:

1. A suggestion for the simplex branch-and-bound method, showing its weakness on a particular type of matrices

2. Suggestions for researchers interested in further improving methods from these papers

Conclusion

In this thesis we looked at the problem of testing copositivty of a given matrix. We analyzed the problem mathematically, looked at problems with testing, including it being NP-complete problem. We further examined different approaches other researchers took, before diving into mathematical properties, including necessary and sufficient conditions for copositivty. Then we came up with three methods for testing ourselves - based on interval analysis, spectral properties of copositive matrices, and quadratic programming. We tried implementing these approaches in MATLAB, described our difficulties (such as generating random matrices for testing), and then did statistical analysis on these methods.

Results of this work

As a practical result of this work, we created a simple, yet modular and highly customizable means for further testing and comparing. Our methods provide a really easy to use interface, with easily changeable settings for other researchers after us.
We tried many different approaches for finding non-copositivity candidate, as well as interval splitting, for both bounding and branching respectively, which can be used in any branch-and-bound algorithm for copositivity testing.
We provide statistical results for future generating of matrices, which we tested on large sample of random matrices, and can prove very useful for others. This method of generating, alongside the option for custom generation, means that basically anyone interested in copositivity testing in MATLAB can make use of our work.
We implemented our own methods for testing, compared them with the simplex method, and provide a list of results, where our method is better, than the simplex-based one.

Future work

This work left some details unanswered, which can provide a new view for some of the parts of testing. We briefly analyzed random non-copositivity candidate in bounding, and it showed promise, perhaps testing with larger number of non-copositivity candidates, or even better, finding the optimal number of candidates to check for, to detect non-copositivity.
In addition we failed to clearly identify the matrices, on which our method behaves much better than the simplex method. This may be interesting for our method especially, as a means to further improve it, or as a possible suggestion for the simplex method, to show its weaknesses, and where it can improve.

Bibliography

1. Standards, National Bureau of. Report 1818. Quarterly Report, April through June 1952. 1952.
2. Bomze, Immanuel M.; Dür, Mirjam; Klerk, Etienne de; Roos, Kees; Quist, Arie J.; Terlaky, Tamás. On Copositive Programming and Standard Quadratic Optimization Problems. *Journal of Global Optimization*. 2000, vol. 18, pp. 301–320. Available also from: https://api.semanticscholar.org/CorpusID:14720527.
3. Murty, Katta G.; Kabadi, Santosh N. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*. 1987, vol. 39, pp. 117–129. Available also from: https://api.semanticscholar.org/CorpusID:30500771.
4. Bundfuss, Stefan; Dür, Mirjam. Dür M.: Algorithmic copositivity detection by simplicial partition. Linear Algebra Appl. 428, 1511-1523. *Linear Algebra and its Applications*. 2008, vol. 428, pp. 1511–1523. Available from doi: 10.1016/j.laa.2007.09.035.
5. Kaplan, Wilfred. A test for copositive matrices. *Linear Algebra and its Applications*. 2000, vol. 313, no. 1, pp. 203–206. issn 0024-3795. Available from doi: https://doi.org/10.1016/S0024-3795(00)00138-5.
6. Qi, Liqun. Symmetric nonnegative tensors and copositive tensors. *Linear Algebra and its Applications*. 2013, vol. 439, no. 1, pp. 228–238. issn 0024-3795. Available from doi: https://doi.org/10.1016/j.laa.2013.03.015.
7. Johnson, Charles R.; Reams, Robert. Spectral theory of copositive matrices. *Linear Algebra and its Applications*. 2005, vol. 395, pp. 275–281. issn 0024-3795. Available from doi: https://doi.org/10.1016/j.laa.2004.08.008.
8. Dickinson, Peter J.C. A new certificate for copositivity. *Linear Algebra and its Applications*. 2019, vol. 569, pp. 15–37. issn 0024-3795. Available from doi: https://doi.org/10.1016/j.laa.2018.12.025.
9. Ping, Li; Yu, Feng Yu. Criteria for copositive matrices of order four. *Linear Algebra and its Applications*. 1993, vol. 194, pp. 109–124. issn 0024-3795. Available from doi: https://doi.org/10.1016/0024-3795(93)90116-6.
10. Dür, Mirjam. Copositive Programming – a Survey. In: 2010, pp. 3–20. isbn 978-3-642-12597-3. Available from doi: 10.1007/978-3-642-12598-0_1.
11. Väliaho, Hannu. Criteria for copositive matrices. *Linear Algebra and its Applications*. 1986, vol. 81, pp. 19–34. issn 0024-3795. Available from doi: https://doi.org/10.1016/0024-3795(86)90246-6.
12. Song, Yisheng; Qi, Liqun. Necessary and sufficient conditions for copositive tensors. *Linear and Multilinear Algebra*. 2013, vol. 63, no. 1, pp. 120–131. issn 1563-5139. Available from doi: 10.1080/03081087.2013.851198.
13. Kong, Qingxia; Lee, Chung-Yee; Teo, Chung; Zheng, Zhichao. Scheduling Arrivals to a Stochastic Service Delivery System Using Copositive Cones. *Operations Research*. 2013, vol. 61, pp. 711–726. Available from doi: 10.2307/23474013.

14. Klerk, Etienne; Pasechnik, Dmitrii. Approximation of the Stability Number of a Graph via Copositive Programming. *SIAM Journal on Optimization*. 2002, vol. 12, pp. 875–892. Available from doi: 10.1137/S1052623401383248.

15. Gvozdenović, Nebojša; Laurent, Monique. The Operator Ψ $for the Chromatic Numbe *SIAM Journal on Optimization*. 2008, vol. 19, no. 2, pp. 572–591. Available from doi: 10.1137/050648237.

16. Dukanovic, Igor; Rendl, Franz. Copositive programming motivated bounds on the stability and the chromatic numbers. *Math. Program.* 2010, vol. 121, pp. 249–268. Available from doi: 10.1007/s10107-008-0233-x.

17. Povh, Janez; Rendl, Franz. A Copositive Programming Approach to Graph Partitioning. *SIAM Journal on Optimization*. 2007, vol. 18, pp. 223–241. Available from doi: 10.1137/050637467.

18. Povh, Janez; Rendl, Franz. Copositive and semidefinite relaxations of the quadratic assignment problem. *Discrete Optimization*. 2009, vol. 6, pp. 231–241. Available from doi: 10.1016/j.disopt.2009.01.002.

19. Burer, Samuel. On the copositive representation of binary and continuous nonconvex quadratic programs. *Math. Program.* 2009, vol. 120, pp. 479–495. Available from doi: 10.1007/s10107-008-0223-z.

20. Preisig, J. C. Copositivity and the Minimization of Quadratic Functions with Nonnegativity and Quadratic Equality Constraints. *SIAM Journal on Control and Optimization*. 1996, vol. 34, no. 4, pp. 1135–1150. Available from doi: 10.1137/S0363012993251894.

21. Tanaka, Akihiro; Yoshise, Akiko. AN LP-BASED ALGORITHM TO TEST COPOSITIVITY. *Pacific Journal of Optimization*. 2015, vol. 11, pp. 101–120.

22. Žilinskas, Julius; Dür, Mirjam. Depth-first simplicial partition for copositivity detection, with an application to MaxClique. *Optimization Methods and Software*. 2011, vol. 26, no. 3, pp. 499–510. Available from doi: 10.1080/10556788.2010.544310.

23. Sponsel, Julia; Bundfuss, Stefan; Dür, Mirjam. An improved algorithm to test copositivity. *Journal of Global Optimization*. 2012, vol. 52, pp. 537–551. Available from doi: 10.1007/s10898-011-9766-2.

24. Brás, Carmo; Eichfelder, Gabriele; Judice, Joaquim. Copositivity tests based on the linear complementarity problem. *Computational Optimization and Applications*. 2014, vol. 63. Available from doi: 10.1007/s10589-015-9772-2.

25. Anstreicher, Kurt M. Testing copositivity via mixed–integer linear programming. *Linear Algebra and its Applications*. 2021, vol. 609, pp. 218–230. issn 0024-3795. Available from doi: https://doi.org/10.1016/j.laa.2020.09.002.

26. Bomze, Immanuel M. Remarks on the recursive structure of copositivity. *Journal of Information and Optimization Sciences*. 1987, vol. 8, no. 3, pp. 243–260. Available from doi: 10.1080/02522667.1987.10698891.

27. Hiriart-Urruty, Jean-Baptiste; Seeger, Alberto. A Variational Approach to Copositive Matrices. *SIAM Review*. 2010, vol. 52, pp. 593–629. Available from doi: 10.1137/090750391.

28. Gaddum, Jerry William. Linear inequalities and quadratic forms. *Pacific Journal of Mathematics*. 1958, vol. 8, pp. 411–414. Available from doi: 10.2140/pjm.1958.8.411.

29. Cottle, R.W.; Habetler, G.J.; Lemke, C.E. QUADRATIC FORMS SEMI-DEFINITE OVER CONVEX CONES. In: *Proceedings of the Princeton Symposium on Mathematical Programming*. Princeton: Princeton University Press, 1971, pp. 551–566. isbn 9781400869930. Available from doi: doi:10.1515/9781400869930-033.

30. Cottle, R.W.; Habetler, G.J.; Lemke, C.E. On classes of copositive matrices. *Linear Algebra and its Applications*. 1970, vol. 3, no. 3, pp. 295–310. issn 0024-3795. Available from doi: https://doi.org/10.1016/0024-3795(70)90002-9.

31. Hadeler, K.P. On copositive matrices. *Linear Algebra and its Applications*. 1983, vol. 49, pp. 79–89. issn 0024-3795. Available from doi: https://doi.org/10.1016/0024-3795(83)90095-2.

32. Jaulin, Luc; Kieffer, Michel; Didrit, Olivier; Walter, Eric. *Applied Interval Analysis with Examples in Parameter and State Estimation, Robust Control and Robotics*. 2001. isbn 1852332190.

33. Hansen, Eldon R.; Walster, G. William. *Global Optimization Using Interval Analysis*. 2nd. New York: Marcel Dekker, 2004.

34. Clausen, Jens. Branch and Bound Algorithms -. 2003.

35. Little, John D. C.; Murty, Katta G.; Sweeney, Dura W.; Karel, Caroline. An Algorithm for the Traveling Salesman Problem. *Operations Research* [online]. 1963, vol. 11, no. 6, pp. 972–989 [visited on 2024-07-14]. issn 0030364X, issn 15265463. Available from: http://www.jstor.org/stable/167836.

36. Balas, Egon; Toth, Paolo. Branch and Bound Methods for the Traveling Salesman Problem. In: 1983. Available also from: https://api.semanticscholar.org/CorpusID:119356236.

37. McCormick, Garth P. Computability of global solutions to factorable nonconvex programs: Part I – Convex underestimating problems. *Math. Program.* 1976, vol. 10, no. 1, pp. 147–175.

38. Inc., The MathWorks. *MATLAB version: 24.1 (R2024a)*. Natick, Massachusetts, United States: The MathWorks Inc., 2024. Available also from: https://www.mathworks.com.

39. Rump, Siegfried M. INTLAB – INTerval LABoratory. In: Csendes, Tibor (ed.). *Developments in Reliable Computing*. Dordrecht: Kluwer Academic Publishers, 1999, pp. 77–104.

40. Inc., The MathWorks. *Optimization Toolbox version: 24.1 (R2024a)*. Natick, Massachusetts, United States: The MathWorks Inc., 2024. Available also from: https://www.mathworks.com.

41. Nemjo, Martin. *Testing copositivity by branch-and-bound* [https://gitlab.mff.cuni.cz/teaching/nprg045/hladik/testing-copositivity-by-branch-and-bound]. Gitlab, 2024.

List of Figures

List of Algorithms

List of Tables

A Attachments

A.1 First Attachment