論文 / 著書情報
Article / Book Information

Title	An efficient second-order cone programming approach for optimal selection in tree breeding
Authors	Makoto Yamashita, Tim J. Mullin, Sena Safarina
Citation	Optimization Letters, Volume 12, 7, pp. 1683–1697
Pub. date	2018, 1

Optim Lett (2018) 12:1683–1697
https://doi.org/10.1007/s11590-018-1229-y

ORIGINAL PAPER

An efficient second-order cone programming approach for optimal selection in tree breeding

Makoto Yamashita[1] · **Tim J. Mullin**[2,3] · **Sena Safarina**[1]

Received: 15 June 2015 / Accepted: 9 January 2018 / Published online: 22 January 2018

Abstract It is common in forest tree breeding that selection of populations must consider conservation of genetic diversity, while at the same time attempting to maximize response to selection. To optimize selection in these situations, the constraint on genetic diversity can be mathematically described with the numerator relationship matrix as a quadratic constraint. Pong-Wong and Woolliams formulated the optimal selection problem using semidefinite programming (SDP). Their SDP approach gave an accurate optimal value, but required rather long computation time. In this paper, we propose an second-order cone programming (SOCP) approach to reduce the heavy computation cost. First, we demonstrate that a simple SOCP formulation achieves the same numerical solution as the SDP approach. A simple SOCP formulation is, however, not much more efficient compared to the SDP approach, so we focused on the sparsity structure of the numerator relationship matrix, and we develop a more efficient SOCP formulation using Henderson's algorithm. Numerical results show that the proposed formulation, which we call a *compact SOCP*, greatly reduced computation time. In a case study, an optimal selection problem that demanded 39,200 s under the SDP approach was solved in less than 2 s by the compact SOCP formulation. The proposed approach is now available as a part of the software package OPSEL.

Keywords Semidefinite programming · Second-order cone programming · Tree breeding · Optimal selection · Group coancestry · Relatedness · Genetic gain

✉ Makoto Yamashita
Makoto.Yamashita@is.titech.ac.jp

[1] Department of Mathematical and Computing Science, Tokyo Institute of Technology, 2-12-1-W8-29, Ookayama, Meguro-ku, Tokyo 152-8552, Japan

[2] The Swedish Forestry Research Institute (Skogforsk), Box 3, 918 21 Sävar, Sweden

[3] The Swedish Forestry Research Institute (Skogforsk), 224 rue du Grand Royal Est, Shefford, QC J2M 1R5, Canada

1 Introduction

Breeders of forest trees must often consider conservation of genetic diversity, while at the same time maximizing response to selection. The selection of new breeding parents must make rapid progress in terms of genetic gain, while maintaining genetic diversity for future genetic improvement. Buyers of planting stock produced by selected parental genotypes managed in *seed orchards* want maximum performance, while satisfying a restriction, sometimes legislated, on the diversity deployed to the forest.

The use of mathematical optimization approaches has been applied increasingly by breeders to the optimum selection problem [1,15,22,25]. Meuwissen [15] expressed this selection of *optimal contributions* as an optimization problem of the form:

$$\begin{array}{ll} \max & : \boldsymbol{g}^T \boldsymbol{x} \\ \text{subject to} & : \boldsymbol{e}^T \boldsymbol{x} = 1 \\ & \quad \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \\ & \quad \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \leq 2\theta. \end{array} \tag{1}$$

Through this paper, we use m to denote the number of the candidate members. We use the vector $\boldsymbol{e} \in \mathbb{R}^m$ to denote a vector of ones, and the superscript T to denote the transpose of a vector or a matrix. The variable vector $\boldsymbol{x} \in \mathbb{R}^m$ corresponds to the contributions (or the proportions) of the candidates, hence the first constraint $\boldsymbol{e}^T \boldsymbol{x} = 1$ requires that the total contribution of candidate members be unity. In the second constraint, $\boldsymbol{l} \in \mathbb{R}^m$ and $\boldsymbol{u} \in \mathbb{R}^m$ are element-wise lower and upper bounds.

The estimated breeding values (EBVs) [13] are often used as the coefficient vector $\boldsymbol{g} \in \mathbb{R}^m$ in the objective function, since the maximization of the weighted EBVs is expected to produce the best genetic performance. From the viewpoint of numerical optimization, we can assume that $\boldsymbol{g}$ is given.

The most interesting property in (1) is the last constraint $\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \leq 2\theta$. This constraint was originally introduced to consider relatedness or *coancestry* among candidates due to common ancestors in their pedigree. These effects accumulate over cycles of breeding as the pedigree becomes more complex. Cockerham [4] extended the definition of coancestry coefficients to describe *group coancestry*. The group coancestry of the contributions $\boldsymbol{x}$ is calculated with the formula $\frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{2}$, where $\boldsymbol{A} \in \mathbb{R}^{m \times m}$ is the numerator relationship matrix of Wright [29] (we will review a formula for the numerator relationship matrix in Sect. 2). The constraint $\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \leq 2\theta$ is therefore employed to keep group coancestry $\frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{2}$ under an appropriate level $\theta \in \mathbb{R}$.

To solve the optimal selection problem (1) efficiently, Meuwissen [15] developed an iterative method based on Lagrangian multipliers and his method has been widely used in breeding, for example, [7,10,28]. It is a characteristic of this method that some variables x_i may be fixed to their lower or upper bounds (l_i or u_i) forcibly, and it was further demonstrated by [22] that the output solution of Meuwissen's method is not always truly optimal.

Pong-Wong and Woolliams [22] utilized semidefinite programming (SDP) to derive a formula equivalent to (1). An SDP is a convex optimization problem that maximizes a linear objective function over the constraints described as linear matrix inequalities. Many studies in the 1990s extended interior-point methods from linear programming problems to SDP problems (e.g., [8,11]). Based on primal-dual interior-point methods, several software solver packages have been developed, such as SDPA [30], SDPARA [31], SDPA-C [32], SDPT3 [27], and SeDuMi [26]. Utilizing the SDP formulation, Pong-Wong and Woolliams [22] obtained the exact optimal value of (1).

An important obstacle encountered in the SDP approach is the heavy computation cost. The number of candidate members discussed in [22] was thus limited, $m \leq 10$. Ahlinder et al. reported in [1] that the SDP approach required 5 h of computation time for a problem of the size $m = 12{,}000$. This computation time is rather long for operational application and may require significant truncation of the candidate list prior to solving the SDP.

The objective of this paper is to propose an efficient approach based on second-order cone programming (SOCP). SOCP is a convex optimization that maximizes a linear objective function over linear constraints and second-order cone constraints. Lobo et al. [12] discussed a wide range of SOCP applications, for example, filter design and truss design, and Sasakawa and Tsuchiya [24] applied SOCP to magnetic shield design. Alizadeh and Goldfarb [2] surveyed theoretical and algorithmic aspects of SOCP. The software packages SDPT3 [27] and SeDuMi [26] can solve not only SDP but also SOCP using primal-dual interior-point methods. In addition, ECOS [5] was implemented recently to solve SOCP problems.

We first show that an SOCP formulation can attain the exact optimal solution of (1), in the same way as SDP. From the fact that SOCP is a special case of SDP, we might expect that a simple SOCP formulation would be sufficient to reduce the computation time, but preliminary numerical tests revealed that the simple SOCP formulation was no better than the SDP approach.

We thus focus on exploiting the sparsity embedded in the numerator relationship matrix $\boldsymbol{A}$ and establish a *sparse SOCP* formulation. Furthermore, we integrate Henderson's algorithm [9] to resolve a bottleneck in the sparse formulation and propose a *compact SOCP* formulation. This compact formulation allows us to remove dense matrices from the computation steps of the SOCP solution.

Numerical tests with data from a sample of Scots pine (*Pinus sylvestris* L.) selection candidates showed that the compact SOCP formulation greatly reduced computation time. For the case of $m = 10{,}100$, we attained a computation speed improvement of 20,000-times compared to the SDP approach. In addition, the compact SOCP formulation is more efficient from the viewpoint of memory utilization.

The remainder of this paper is organized as follows: Sect. 2 describes the SDP approach of Pong-Wong and Woolliams and discusses a simple SOCP formulation; in Sect. 3, we propose SOCP formulations and derive an efficient method; Sect. 4 presents numerical results to verify the reduction in computation time for problems of various sizes; and finally, Sect. 5 gives conclusions and discusses future directions.

2 SDP formulation and simple SOCP formulation

Since a principal characteristic of the optimal selection problem (1) is determined by the numerator relationship matrix $\boldsymbol{A}$, we first review an algorithm to evaluate $\boldsymbol{A}$. We then describe the SDP formulation of Pong-Wong and Woolliams [22], and compare the performance of the SDP formulation and a simple SOCP formulation.

To evaluate $\boldsymbol{A}$, we separate the set of pedigree candidate members $\mathcal{P} := \{1, 2, \ldots, m\}$ into the three disjoint groups:

$$\mathcal{P} = \mathcal{P}_0 \cup \mathcal{P}_1 \cup \mathcal{P}_2,$$

where

$$\begin{cases} \mathcal{P}_0 = \{i \in \mathcal{P} : \text{both parents } p(i) \text{ and } q(i) \text{ are unknown}\} \\ \mathcal{P}_1 = \{i \in \mathcal{P} : \text{one parent } p(i) \text{ is known and the other parent } q(i) \text{ is unknown}\} \\ \mathcal{P}_2 = \{i \in \mathcal{P} : \text{both parents } p(i) \text{ and } q(i) \text{ are known}\}. \end{cases}$$

Figure 1 gives an example of a pedigree having $m = 9$ members and illustrates its genealogical chart. In this example, $\mathcal{P}_0 = \{1, 2\}, \mathcal{P}_1 = \{5\}, \mathcal{P}_2 = \{3, 4, 6, 7, 8, 9\}$, and the parents of the 8th member are $p(8) = 7$ and $q(8) = 6$. If parent $p(i)$ or $q(i)$ is unknown, we assign $p(i) = 0$ or $q(i) = 0$, respectively, and we can assume $i > p(i) \geq q(i)$ for all $i \in \mathcal{P}$ without loss of generality.

The numerator relationship matrix $\boldsymbol{A}$ was defined by Wright [29], and its simplified formula was presented in [9]. The formula in [9] gives the elements $A_{11}, \ldots, A_{nn}$ in a recursive style as follows:

$$\begin{cases} A_{ij} = A_{ji} = \frac{A_{j,p(i)} + A_{j,q(i)}}{2} & \text{for} \quad i = 1, \ldots, m, \ j = 1, \ldots, i-1 \\ A_{ii} = 1 + \frac{A_{p(i),q(i)}}{2} & \text{for} \quad i = 1, \ldots, m, \end{cases}$$

where $A_{pq} = 0$ if $p = 0$ or $q = 0$. When we apply this recursion to the example of Fig. 1, we obtain the corresponding matrix $\boldsymbol{A}$ as follows:

pedigree id $[i]$	parents $[p(i)$ and $q(i)]$
1	unknown and unknown
2	unknown and unknown
3	1 and 2
4	1 and 2
5	2 and unknown
6	3 and 4
7	1 and 5
8	6 and 7
9	5 and 7

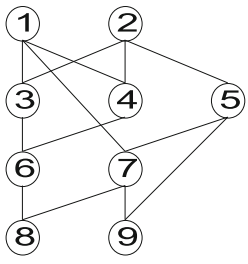

Fig. 1 An example of a coded pedigree and its diagram

$$\boldsymbol{A} = \frac{1}{32}\begin{pmatrix} 32 & 0 & 16 & 16 & 0 & 16 & 16 & 16 & 8 \\ 0 & 32 & 16 & 16 & 16 & 16 & 8 & 12 & 12 \\ 16 & 16 & 32 & 16 & 8 & 24 & 12 & 18 & 10 \\ 16 & 16 & 16 & 32 & 8 & 24 & 12 & 18 & 10 \\ 0 & 16 & 8 & 8 & 32 & 8 & 16 & 12 & 24 \\ 16 & 16 & 24 & 24 & 8 & 40 & 12 & 26 & 10 \\ 16 & 8 & 12 & 12 & 16 & 12 & 32 & 22 & 24 \\ 16 & 12 & 18 & 18 & 12 & 26 & 22 & 38 & 17 \\ 8 & 12 & 10 & 10 & 24 & 10 & 24 & 17 & 48 \end{pmatrix}. \tag{2}$$

Pong-Wong and Woolliams [22] devised an SDP formulation to solve the problem (1). They observed that the matrix $\boldsymbol{A}$ is always positive definite, and they used the Schur complement to replace the group-coancestry constraint $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \leq 2\theta$ with an equivalent condition $\begin{pmatrix} 2\theta & \boldsymbol{x}^T \\ \boldsymbol{x} & \boldsymbol{A}^{-1} \end{pmatrix} \in \mathbb{S}_+^{1+m}$, where $\mathbb{S}_+^{1+m}$ is the space of positive semidefinite matrices of dimension $1+m$. Consequently, they solved the following SDP problem that is equivalent to the optimal selection problem (1):

$$\begin{aligned} \text{max} \quad & : \boldsymbol{g}^T\boldsymbol{x} \\ \text{subject to} \; & : \boldsymbol{e}^T\boldsymbol{x} = 1 \\ & \quad \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \\ & \quad \begin{pmatrix} 2\theta & \boldsymbol{x}^T \\ \boldsymbol{x} & \boldsymbol{A}^{-1} \end{pmatrix} \in \mathbb{S}_+^{1+m}. \end{aligned} \tag{3}$$

Detailed conversions of this problem into a standard SDP format that can be passed to SDP solvers are described in [1,22]. Version 1 of the software package OPSEL [18] automatically performs the conversion and passes the problem to SDPA [30] for solving.

Table 1 displays the computed optimal values and the computation times for Meuwissen's implementation (GENCONT) [16] and the SDP formulation (3). We executed numerical tests using Matlab R2015a on a Windows 8.1 PC with Xeon CPU E3-1231 (3.40 GHz, 4 cores) and 8 GB of RAM. We used Windows, since GENCONT can run on only Windows. To solve the SDP (3), we employed SDPA [30] with four-core parallel processing.

Table 1 Optimal values and computation times for GENCONT and SDP formulations (time in seconds)

m (size of pedigree)	2045	10,100
GENCONT		
Optimal value	438.56	OOM*
Time	67.43	
SDP formulation (3)		
Optimal value	439.12	47.76
Time	70.21	39,200.78

**OOM* out of memory

We observe from Table 1 that the SDP formulation (3) attained a better optimal value than GENCONT. Actually, as shown in [22], the optimal value obtained from (3) was the exact optimal value, while the Lagrangian multiplier method implemented in GENCONT cannot guarantee optimality. Furthermore, for the large problem ($m = 10,100$), GENCONT gave up on the computation, while the SDP formulation again obtained the exact value. On the other hand, the disadvantage of the SDP formulation is its computation time. Even using parallel processing implemented in SDPA, the SDP formulation was slower than GENCONT when $m = 2045$. Furthermore, the computation time for the large problem exceeded 10 h.

To reduce the heavy computation time of the SDP formulation, we express the group-coancestry constraint $\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} \leq 2\theta$ as a second-order cone (SOC) constraint. The SOC constraint for $\boldsymbol{v} \in \mathbb{R}^n$ and $v_0 \in \mathbb{R}$ is the inequality constraint $||\boldsymbol{v}|| \leq v_0$, where $||\boldsymbol{v}|| := \sqrt{\boldsymbol{v}^T \boldsymbol{v}} = \sqrt{\sum_{k=1}^{n} v_k^2}$ is the Euclidean norm of $\boldsymbol{v}$. Since the matrix $\boldsymbol{A}$ is positive definite, the Cholesky factorization gives an upper triangular matrix $\boldsymbol{U}$ such that $\boldsymbol{A} = \boldsymbol{U}^T \boldsymbol{U}$. Due to the relation $\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} = \boldsymbol{x}^T \boldsymbol{U}^T \boldsymbol{U}\boldsymbol{x} = ||\boldsymbol{U}\boldsymbol{x}||^2$, the group-coancenstry constraint $\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} \leq 2\theta$ can be substituted by an SOC constraint $||\boldsymbol{U}\boldsymbol{x}|| \leq \sqrt{2\theta}$. Consequently, we can convert (1) into an SOCP problem, which will be referred to as a *simple SOCP formulation* or more concisely *simple SOCP*:

$$\begin{array}{ll} \max & : \boldsymbol{g}^T \boldsymbol{x} \\ \text{subject to} & : \boldsymbol{e}^T \boldsymbol{x} = 1 \\ & \quad \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \\ & \quad ||\boldsymbol{U}\boldsymbol{x}|| \leq \sqrt{2\theta}. \end{array} \tag{4}$$

From the equivalence, it is clear that this simple SOCP formulation also gives the exact optimal value of the optimal-selection problem (1).

It is well known that SOC constraints are a special case of positive semidefinite constraints. We thus expected that the simple SOCP (4) could be solved more quickly than SDP (3). Table 2 reports the results of a preliminary numerical experiment for the SDP and simple SOCP formulations. We used ECOS [5] as the SOCP solver for solving (4). For the small problem ($m = 2045$), simple SOCP successfully reduced the computation time from 70.21 to 0.28 s, a speed improvement of 250-times. For the large problem ($m = 10,100$), however, the speed improvement was only 7-times. When we consider the computational complexity, simple SOCP would be even slower than SDP for still larger problems. Despite that SOCP is a special case of SDP, this result suggests that a simple SOCP is not so promising.

Table 2 Computation time on SDP and simple SOCP formulations (time in seconds)

m (size of pedigree)	2045	10,100
SDP (3)	70.21	39,200.78
Simple SOCP (4)	0.28	5604.25

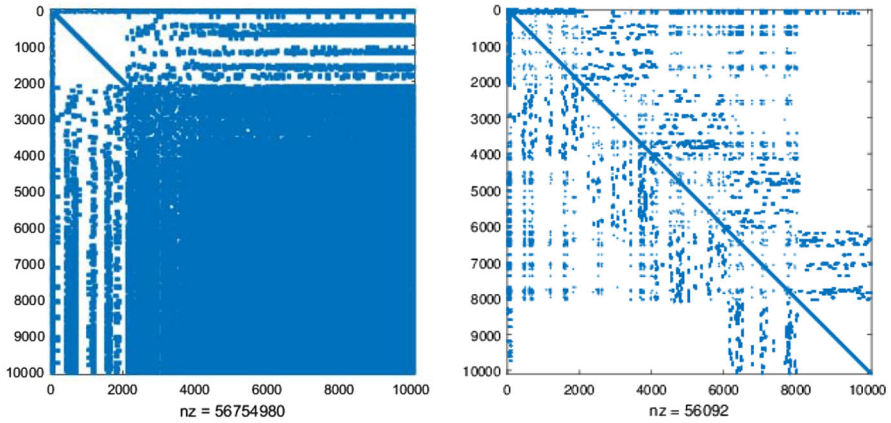

Fig. 2 The positions of non-zero elements of $\boldsymbol{A}$ (left) and $\boldsymbol{A}^{-1}$ (right) for the problem of size $m = 10,100$

3 Efficient formulation based on second-order cone programming

To obtain a more efficient formulation based on SOCP, we investigate properties hidden in the simple SOCP (4). In particular, noting that SDP (3) relies on only $\boldsymbol{A}^{-1}$, we focus on the sparsity of $\boldsymbol{A}$ and its inverse $\boldsymbol{A}^{-1}$. Taking the inverse of $\boldsymbol{A}$ in (2) reveals that $\boldsymbol{A}^{-1}$ contains many more zero-elements than $\boldsymbol{A}$;

$$\boldsymbol{A}^{-1} = \frac{1}{42}\begin{pmatrix} 105 & 42 & -42 & -42 & 21 & 0 & -42 & 0 & 0 \\ 42 & 98 & -42 & -42 & -28 & 0 & 0 & 0 & 0 \\ -42 & -42 & 105 & 21 & 0 & -42 & 0 & 0 & 0 \\ -42 & -42 & 21 & 105 & 0 & -42 & 0 & 0 & 0 \\ 21 & -28 & 0 & 0 & 98 & 0 & -21 & 0 & -42 \\ 0 & 0 & -42 & -42 & 0 & 108 & 24 & -48 & 0 \\ -42 & 0 & 0 & 0 & -21 & 24 & 129 & -48 & -42 \\ 0 & 0 & 0 & 0 & 0 & -48 & -48 & 96 & 0 \\ 0 & 0 & 0 & 0 & -42 & 0 & -42 & 0 & 84 \end{pmatrix}.$$

Figure 2 illustrates the positions of the non-zero elements of $\boldsymbol{A}$ and $\boldsymbol{A}^{-1}$ for the problem of size $m = 10,100$. The dimensions of $\boldsymbol{A}$ and $\boldsymbol{A}^{-1}$ correspond to size m.

Figure 2 again indicates that $\boldsymbol{A}^{-1}$ is much sparser than $\boldsymbol{A}$. The numbers of non-zero elements in $\boldsymbol{A}$ and $\boldsymbol{A}^{-1}$ are 56,754,980 and 56,092, respectively, hence their density against the fully-dense matrix (m^2 non-zero elements) are 55.6 and 0.0549%, respectively. When we apply the Cholesky factorization to $\boldsymbol{A}$, the upper triangular matrix $\boldsymbol{U}$ inherits the dense property. The number of non-zero elements in $\boldsymbol{U}$ is 23,171,296, and its density against the fully-dense upper triangular matrix is still 45.4%.

3.1 Sparse SOCP formulation

We can exploit the property that $\boldsymbol{A}^{-1}$ is remarkably sparse compared to $\boldsymbol{A}$ and $\boldsymbol{U}$. The first key step in our approach is to replace $\boldsymbol{x}$ with $\boldsymbol{A}^{-1}\boldsymbol{y}$, where $\boldsymbol{y}$ is a new variable

defined by $\boldsymbol{y} := \boldsymbol{A}\boldsymbol{x}$. We can transform $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \leq 2\theta$ into another quadratic constraint $\boldsymbol{y}^T\boldsymbol{A}^{-1}\boldsymbol{y} \leq 2\theta$. This new constraint $\boldsymbol{y}^T\boldsymbol{A}^{-1}\boldsymbol{y} \leq 2\theta$ is composed of the sparse matrix $\boldsymbol{A}^{-1}$ and does not involve the dense matrix $\boldsymbol{A}$. The second step is to utilize the Cholesky factor of $\boldsymbol{A}^{-1}$. It is given by $\boldsymbol{U}^{-T}$ (the transposed matrix of $\boldsymbol{U}^{-1}$) from the relation $\boldsymbol{A}^{-1} = (\boldsymbol{U}^{-T})^T\boldsymbol{U}^{-T}$, so $\boldsymbol{y}^T\boldsymbol{A}^{-1}\boldsymbol{y} \leq 2\theta$ is now transformed into an SOC constraint $||\boldsymbol{U}^{-T}\boldsymbol{y}|| \leq \sqrt{2\theta}$. Using these two steps, we derive another SOCP problem which will be referred to as a *sparse SOCP formulation* or more concisely *sparse SOCP*:

$$\begin{array}{ll} \max & : (\boldsymbol{A}^{-1}\boldsymbol{g})^T\boldsymbol{y} \\ \text{subject to} & : (\boldsymbol{A}^{-1}\boldsymbol{e})^T\boldsymbol{y} = 1 \\ & \quad \boldsymbol{l} \leq \boldsymbol{A}^{-1}\boldsymbol{y} \leq \boldsymbol{u} \\ & \quad ||\boldsymbol{U}^{-T}\boldsymbol{y}|| \leq \sqrt{2\theta}. \end{array} \tag{5}$$

We should emphasize that when $\boldsymbol{y}^*$ is obtained as the optimal solution of (5), the optimal solution $\boldsymbol{x}^*$ of the original problem (1) can be computed through the relation $\boldsymbol{x}^* = \boldsymbol{A}^{-1}\boldsymbol{y}^*$ without depending on the dense matrix $\boldsymbol{A}$.

In Table 3, we compare the performance of SDP (3), simple SOCP (4), sparse SOCP (5), and compact SOCP (6) formulations. (We will discuss the compact SOCP formulation later in Sect. 3.2). The first column 'nnz' is the total number of non-zero elements that appear in a standard format that are acceptable by SDP/SOCP solvers (Technically speaking, we utilized the SeDuMi format [26] to count the non-zero elements). The second column is the computation time to convert a pedigree like Fig. 1 and EBVs (estimated breeding values) into the SDP or SOCP formulations, and the third column is the computation time of the solvers. We applied SDPA with four-core parallel processing to the SDP formulation and ECOS to the SOCP formulations. The fourth (last) column is the total computation time.

Table 3 indicates that for the large problem ($m = 10,100$), the computation time of sparse SOCP was much shorter than that of simple SOCP. The sparse SOCP seems to have more complex structure than the simple SOCP, as it contained $\boldsymbol{A}^{-1}$ three times in the input data. Nevertheless, the total number of non-zero elements was reduced from

Table 3 Performance comparison of the SDP and SOCP formulations (time in seconds)

Formulation	nnz	Time (conversion)	Time (solver)	Time (total)
m (size of pedigree) = 2045				
SDP (3)	24,300	0.52	69.55	70.21
Simple SOCP (4)	18,201	0.10	0.04	0.28
Sparse SOCP (5)	30,348	0.37	0.05	0.55
Compact SOCP (6)	30,249	0.01	0.05	0.19
m (size of pedigree) = 10,100				
SDP (3)	121,703	26.97	39,173.03	39,200.78
Simple SOCP (4)	23,231,801	15.95	5587.53	5604.25
Sparse SOCP (5)	159,570	24.14	0.68	25.60
Compact SOCP (6)	153,001	0.37	0.62	1.76

23,231,801 in the simple SOCP to 159,570 in the sparse SOCP, due to the change from $\boldsymbol{U}$ (which contains 23,171,296 non-zero elements) to $\boldsymbol{U}^{-T}$ (36,607 non-zero elements). This led to a reduction in computation time by the SOCP solver.

3.2 Compact SOCP formulation

To reduce the total time further, we consider the computation time required to prepare the SOCP formulations. In the case of $m = 10,100$ in Table 3, the conversion comprised 94% of the total time. In particular, the construction of dense matrix $\boldsymbol{A}$ and its inversion are the principal bottlenecks. We evaluate $\boldsymbol{A}$ to formulate the sparse SOCP, but $\boldsymbol{A}$ will be no longer required when we solve the resultant SOCP problem with interior-point methods.

It is thus highly desirable to generate directly the sparse matrix $\boldsymbol{A}^{-1}$ without the dense matrix $\boldsymbol{A}$, and this leads us to a compact algorithm proposed by Henderson [9] to construct $\boldsymbol{A}^{-1}$. In Henderson's original algorithm, the vector of inbreeding coefficients defined by $\boldsymbol{h} := diag(\boldsymbol{A}) - \boldsymbol{e}$ appears in the computation process. Here, $diag(\boldsymbol{A})$ is a vector composed of the diagonal elements of $\boldsymbol{A}$, so Henderson's algorithm still depends on $\boldsymbol{A}$. Quaas [23] devised an efficient short-cut to compute $\boldsymbol{h}$ without constructing the matrix $\boldsymbol{A}$ itself, and Masuda et al. [14] utilized this method to implement their YAMS package. The compact algorithm of [9] with the enhancement [23] is shown in Table 4. In the algorithm, the indicator function $\delta(p)$ stands for $\delta(0) = 1$ and $\delta(p) = 0$ for $p \neq 0$, and A^{-1}_{ij} denotes the (i, j)th element of $\boldsymbol{A}^{-1}$.

By using the values b_i computed in Table 4, the mathematical formula for $\boldsymbol{A}^{-1}$ can be given as follows:

$$\begin{aligned}\boldsymbol{A}^{-1} = &\sum_{i\in\mathcal{P}_0} b_i \boldsymbol{e}_i \boldsymbol{e}_i^T + \sum_{i\in\mathcal{P}_1} b_i \left(\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{p(i)}\right)\left(\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{p(i)}\right)^T \\ &+ \sum_{i\in\mathcal{P}_2} b_i \left(\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{p(i)} - \frac{1}{2}\boldsymbol{e}_{q(i)}\right)\left(\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{p(i)} - \frac{1}{2}\boldsymbol{e}_{q(i)}\right)^T,\end{aligned}$$

where $\boldsymbol{e}_i$ is a vector whose components are all zero except the ith component being one.

It is known that $b_i > 0$ for any $i = 1, 2, \ldots, m$ from properties of the inbreeding coefficients, therefore, the group-coancestry constraint can be transformed into another SOC constraint:

$$\begin{aligned}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \leq 2\theta &\Leftrightarrow \boldsymbol{y}^T\boldsymbol{A}^{-1}\boldsymbol{y} \leq 2\theta \\ &\Leftrightarrow \sum_{i\in\mathcal{P}_0} b_i y_i^2 + \sum_{i\in\mathcal{P}_1} b_i \left(y_i - \frac{1}{2}y_{p(i)}\right)^2 \\ &\qquad + \sum_{i\in\mathcal{P}_2} b_i \left(y_i - \frac{1}{2}y_{p(i)} - \frac{1}{2}y_{q(i)}\right)^2 \leq 2\theta \\ &\Leftrightarrow ||\boldsymbol{B}\boldsymbol{y}|| \leq \sqrt{2\theta},\end{aligned}$$

Table 4 A compact algorithm to obtain the inverse of the numerator relationship matrix A^{-1} based on [9]

$A^{-1} \leftarrow O$.
for $i = 1, 2, \ldots, m$
  $b_i \leftarrow \frac{4}{(1+\delta(p(i)))(1-h_{p(i)})+(1+\delta(q(i)))(1-h_{q(i)})}$
  if $i \in \mathcal{P}_0$ **then**
    add b_i to A^{-1}_{ii}
  elseif $i \in \mathcal{P}_1$ **then**
    add b_i to A^{-1}_{ii}
    add $-\frac{b_i}{2}$ to $A^{-1}_{i,p(i)}$, and $A^{-1}_{p(i),i}$
    add $\frac{b_i}{4}$ to $A^{-1}_{p(i),p(i)}$
  elseif $i \in \mathcal{P}_2$ **then**
    add b_i to A^{-1}_{ii}
    add $-\frac{b_i}{2}$ to $A^{-1}_{i,p(i)}$, $A^{-1}_{p(i),i}$, $A^{-1}_{i,q(i)}$, and $A^{-1}_{q(i),i}$
    add $\frac{b_i}{4}$ to $A^{-1}_{p(i),p(i)}$, $A^{-1}_{p(i),q(i)}$, $A^{-1}_{q(i),p(i)}$, and $A^{-1}_{q(i),q(i)}$
  endif
endfor

where $\boldsymbol{B}$ is the matrix whose ith row vector is determined by

$$B_{i*} = \begin{cases} \sqrt{b_i}\boldsymbol{e}_i^T & \text{for} \quad i \in \mathcal{P}_0 \\ \sqrt{b_i}\left(\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{p(i)}\right)^T & \text{for} \quad i \in \mathcal{P}_1 \\ \sqrt{b_i}\left(\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{p(i)} - \frac{1}{2}\boldsymbol{e}_{q(i)}\right)^T & \text{for} \quad i \in \mathcal{P}_2. \end{cases}$$

Using the new SOC constraint $||\boldsymbol{B}\boldsymbol{y}|| \leq \sqrt{2\theta}$, we develop another SOCP which will be referred to as a *compact SOCP formulation* or concisely *compact SOCP*:

$$\begin{array}{ll} \max & : (\boldsymbol{A}^{-1}\boldsymbol{g})^T\boldsymbol{y} \\ \text{subject to} & : (\boldsymbol{A}^{-1}\boldsymbol{e})^T\boldsymbol{y} = 1 \\ & \quad \boldsymbol{l} \leq \boldsymbol{A}^{-1}\boldsymbol{y} \leq \boldsymbol{u} \\ & \quad ||\boldsymbol{B}\boldsymbol{y}|| \leq \sqrt{2\theta}. \end{array} \tag{6}$$

The combination of the compact algorithm in Table 4 and the SOC constraint $||\boldsymbol{B}\boldsymbol{y}|| \leq \sqrt{2\theta}$ gives us a formulation that does not depend on any dense matrices. Indeed, the number of non-zero elements of the matrix $\boldsymbol{B}$ is just 30,100 for the problem of size $m = 10{,}100$, much smaller than that of $\boldsymbol{U}$ (23,171,296) for the simple SOCP and comparable to that of $\boldsymbol{U}^{-T}$ (36,607) for the sparse SOCP formulation.

From Table 3, we observe that the solver time for the compact SOCP was slightly less than for the sparse SOCP. Further reduction in computation time was achieved

during the formulation of the SOCP from the pedigree. In the case $m = 10,100$, the conversion was reduced from 24.14 s to 0.37 s.

4 Numerical tests

We conducted numerical evaluations of the SDP and SOCP formulations on several datasets. The datasets are problems of sizes 2045, 5050, 15,100, 15,222, 50,100, 100,100 and 300,100. The datasets with sizes 2045 and 15,222 were derived from actual data for Scots pine and loblolly pine ((*Pinus taeda* L.), respectively (available at the Dryad Digital Repository http://dx.doi.org/10.5061/dryad.9pn5m). The other data were produced by simulation of five cycles of breeding in a closed population using the approach of [20,21].

The computation environment for the small problems (i.e., $m \leq 10,100$) was Matlab R2015a on a Windows 8.1 PC with Xeon E3-1231 (3.40 GHz) and 8 GB RAM. For the larger problems ($m \geq 15,100$), we used Matlab R2014b on a Debian Linux server with Opteron 4386 (3.10 GHz) and 128 GB RAM, since 8 GB was insufficient for the SDP formulation. We utilized SDPA [30] with four-core parallel processing as the SDP solver and ECOS [5] as the SOCP solver. For the compact SOCP (6), we also conducted the numerical experiment on CVX [6] using SDPT3 [27] and MOSEK [17] as the SOCP solvers. The use of CVX makes it much easier to write SOCP problems in the Matlab environment, and SDPT3 is the default SOCP/SDP solver in CVX while MOSEK is one of state-of-the-art commercial solvers for SDP/SOCP.

Although we mainly used Matlab for the comparison, we also implemented compact SOCP with C++. We can directly know the positions of non-zero elements that appear in the matrix $\boldsymbol{B}$ from the pedigree list, therefore, a specified data structure can accelerate the computation time to build $\boldsymbol{B}$. If we implement simple or sparse SOCP with C++, we would need to embed certain numerical routines for the Cholesky factorization to compute $\boldsymbol{U}$ or $\boldsymbol{U}^{-T}$. In addition, the sparse Cholesky factorization requires a preprocessing by, for example, AMD [3] to derive better performance. One of the advantages in compact SOCP is that we can manage the computation without numerical routines for the Cholesky factorization for obtaining $\boldsymbol{B}$. Our C++ implementation for compact SOCP internally calls ECOS as the SOCP solver.

Table 5 shows the numerical results for the SDP and SOCP formulations. We can see that compact SOCP (6) solved the problems much faster than other formulations. In particular, for the problem $m = 15,222$, the compact SOCP with C++ required only 2.21 s, compared with 22,566 s for the SDP formulation, an improvement in speed by a factor of 10 210.

Another significant advantage of compact SOCP is memory consumption. The SDP formulation consumed 31 GB RAM to solve the problem $m = 15,222$, and it was unable to handle $m \geq 50,100$, despite having 128 GB available. A rough estimate suggests that the memory required for the largest problem $m = 300,100$ would be 12,000 GB. Simple SOCP also suffered from a large memory requirement to store the dense matrix $\boldsymbol{U}$. The sparse SOCP did not require the dense matrix $\boldsymbol{A}$ in the resultant SOCP problem, but it still utilized $\boldsymbol{A}$ for computing $\boldsymbol{A}^{-1}$, and thus required a long conversion time in the same way as the SDP formulation. In contrast, the compact

Table 5 Numerical results on SDP and SOCP formulations (time in seconds)

Formulation	nnz*	Time (conversion)	Time (solver)	Time (total)
m (size of pedigree) = 2045				
SDP (3)	24,300	0.52	69.55	70.21
Simple SOCP (4)	18,201	0.10	0.04	0.28
Sparse SOCP (5)	30,348	0.37	0.05	0.55
Compact SOCP (6)	30,249	0.01	0.05	0.20
Compact SOCP with SDPT3	30,249	0.03	3.23	3.37
Compact SOCP with MOSEK	30,249	0.02	0.73	0.76
Compact SOCP with C++	30,249	0.01	0.06	0.09
m (size of pedigree) = 5050				
SDP (3)	60,853	4.63	887.32	892.30
Simple SOCP (4)	6,812,127	1.60	696.10	698.15
Sparse SOCP (5)	78,405	3.67	0.19	4.21
Compact SOCP (6)	76,533	0.04	0.19	0.58
Compact SOCP with SDPT3	76,533	0.01	6.61	6.89
Compact SOCP with MOSEK	76,533	0.03	0.77	0.81
Compact SOCP with C++	76,533	0.01	0.21	0.28
m (size of pedigree) = 15,100				
SDP (3)	181,703	157.41	21,836.63	21,994.87
Simple SOCP (4)	54,063,065	26.17	38,733.01	38,760.00
Sparse SOCP (5)	234,760	145.53	2.13	148.49
Compact SOCP (6)	227,989	0.04	2.06	2.92
Compact SOCP with SDPT3	227,989	0.04	28.08	28.95
Compact SOCP with MOSEK	227,989	0.04	2.32	3.27
Compact SOCP with C++	227,989	0.02	1.95	1.99
m (size of pedigree) = 15,222				
SDP (3)	181,947	161.99	22,403.30	22,566.11
Simple SOCP (4)	7,889,551	17.96	618.18	636.95
Sparse SOCP (5)	227,758	150.07	2.13	153.01
Compact SOCP (6)	227,203	0.04	2.20	3.05
Compact SOCP with SDPT3	227,203	0.04	177.97	178.84
Compact SOCP with MOSEK	227,203	0.04	1.98	2.88
Compact SOCP with C++	227,203	0.02	2.16	2.21
m (size of pedigree) = 50,100				
SDP (3)				OOM**
Simple SOCP (4)				OOM
Sparse SOCP (5)	759,294	4989.55	7.58	4999.90
Compact SOCP (6)	753,023	0.15	7.71	10.63
Compact SOCP with SDPT3	753,023	0.14	273.69	276.66
Compact SOCP with MOSEK	753,023	0.12	16.15	19.14
Compact SOCP with C++	753,023	0.08	7.48	7.69

Table 5 continued

Formulation	nnz*	Time (conversion)	Time (solver)	Time (total)
m (size of pedigrees) = 100,100				
SDP (3)				OOM
Simple SOCP (4)				OOM
Sparse SOCP (5)				> 24 h***
Compact SOCP (6)	1,502,983	0.35	18.44	24.76
Compact SOCP with SDPT3	1,502,983	0.25	902.60	908.63
Compact SOCP with MOSEK	1,502,983	0.24	11.06	17.23
Compact SOCP with C++	1,502,983	0.16	17.57	17.92
m (size of pedigrees) = 300,100				
SDP (3)				OOM
Simple SOCP (4)				OOM
Sparse SOCP (5)				OOM
Compact SOCP (6)	4,503,065	1.10	82.41	99.65
Compact SOCP with SDPT3	4,503,065	0.84	10,023.77	10,040.02
Compact SOCP with MOSEK	4,503,065	0.78	37.14	56.15
Compact SOCP with C++	4,503,065	0.51	78.54	79.62

*nnz = the total number of non-zero elements that appear in input data
**_OOM_ out of memory
***> 24 h = the computation failed to complete within 24 h

SOCP required less than 766 MB RAM to solve even the huge problem $m = 300,100$, due mainly to the compact SOCP not having to process any dense matrices.

From the numerical results, we also observe that the C++ implementation of compact SOCP is faster than the implementation in Matlab. The discrepancy between Matlab (99.65 s) and C++ (79.62 s) in the largest problem was due to a specified data structure written in C++. In particular, the structure was useful when we arranged the pedigree information before building the SOCP problems.

As for the performance of CVX using SDPT3 and MOSEK, the latter was much faster, thus we focus on the comparison between CVX using MOSEK and the C++ implementation. Up to the relatively large problems ($m \leq 100,100$), the total time for CVX using MOSEK was not so different from that of the C++ implementation. For the largest problem ($m = 300,100$), CVX using MOSEK was faster than the C++ implementation, mainly because the commercial solver MOSEK [17] was more efficient than ECOS [5] for large problems in the compact SOCP. ECOS is, however, a free software package and the problem size $m = 300,100$ is huge even for operational use in breeding. Thus, the C++ implementation using ECOS is certainly attractive from a practical standpoint.

5 Conclusions and future directions

We examined the SOCP formulations for the optimal selection problem arising from tree breeding. We employed the transformation $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{y}$ based on the sparsity

of $\boldsymbol{A}^{-1}$ and the efficient method for building $\boldsymbol{A}^{-1}$ by Henderson's algorithm with the Quaas enhancement. As a result, the compact SOCP formulation removed the requirement for processing of large dense matrices from the computations. The numerical results demonstrated that the compact SOCP formulation obtained the optimal solution significantly faster than the SDP formulation available previously to breeders.

While the SOCP formulations proposed in this paper may look simple to researchers in the field of mathematical optimization, the reduction in computation time to solve the optimal selection problem will help improve operational application in tree breeding. We expect that this paper will be one of bridges to introduce efficient approaches cultivated in mathematical optimization to breeding. The proposed approach is now available as a part of Version 2 of the software package OPSEL [19], available free at http://www.skogforsk.se/opsel.

In this paper, we discussed optimizing *unequal* contributions of parental genotypes deployed in mating designs and seed orchards. In some situations it might be preferable to optimize selection of populations for *equal* contributions deployment. This would require a method to solve a mixed-integer SOCP problem, i.e., an SOCP problem in which some variables are constrained to be integers. The structure of the SOCP formulation developed in this paper will be a basis for an efficient method now under development to consider the mixed-integer SOCP problem for equal-deployment problems.

Acknowledgements The authors are grateful to Dr. Yutaka Masuda of Obihiro Univeristy of Agriculture and Veterinary Medicine for providing access to the source code of the YAMS package. Our work was partially supported by funding from JSPS KAKENHI (Grant-in-Aid for Scientific Research (C), 15K00032), the European Union Horizon 2020 Research and Innovation Program under Grant Agreement No. 676876 (Project GenTree), and Föreningen Skogsträdsförädling (The Tree Breeding Association, Sweden).

The authors also would like to thank the reviewers for valuable suggestions that have improved the quality of this paper.

References

1. Ahlinder, J., Mullin, T.J., Yamashita, M.: Using semidefinite programming to optimize unequal deployment of genotypes to a clonal seed orchard. Tree Genet. Genomes **10**(1), 27–34 (2014)
2. Alizadeh, F., Goldfarb, D.: Second-order cone programming. Math. Program. B **95**(1), 3–51 (2003)
3. Amestoy, P.R., Davis, T.A., Duff, I.S.: Algorithm 837: AMD, an approximate minimum degree ordering algorithm. ACM Trans. Math. Softw. **30**(3), 381–388 (2004)
4. Cockerham, C.C.: Group inbreeding and coancestry. Genetics **56**(1), 89–104 (1967)
5. Domahidi, A., Chu, E., Boyd, S.: ECOS: an SOCP solver for embedded systems. In: Proceedings of European control conference, pp. 3071–3076 (2013)
6. Grant, M., Boyd, S., Ye, Y.: CVX: Matlab software for disciplined convex programming (2009). http://cvxr.com/cvx/cvx_usrguide.pdf.
7. Grundy, B., Villanueva, B., Wooliams, J.A.: Dynamic selection procedures for constrained inbreeding and their consequences for pedigree development. Genet. Res. **72**(2), 159–168 (1998)
8. Helmberg, C., Rendl, F., Vanderbei, R.J., Wolkowicz, H.: An interior-point method for semidefinite programming. SIAM J. Optim. **6**(2), 342–361 (1996)

9. Henderson, C.R.: A simple method for computing the inverse of a numerator relationship matrix used in prediction of breeding values. Biometrics **32**(1), 69–83 (1976)
10. Hinrichs, D., Meuwissen, T.H.E.: Analyzing the effect of different approaches of penalized relationship in multistage selection schemes. J. Anim. Sci. **89**(11), 3426–3432 (2011)
11. Kojima, M., Shindoh, S., Hara, S.: Interior-point methods for the monotone semidefinite linear complementarity problems in symmetric matrices. SIAM J. Optim. **7**, 86–125 (1997)
12. Lobo, M.S., Vandenberghe, L., Boyd, S., Lebret, H.: Applications of second-order cone programming. Linear Algebr. Appl. **284**(1), 193–228 (1998)
13. Lynch, M., Walsh, B.: Genetics and Analysis of Quantitative Traits, vol. 1. Sinauer, Sunderland (1998)
14. Masuda, Y., Baba, T., Suzuki, M.: Application of supernodal sparse factorization and inversion to the estimation of (co) variance components by residual maximum likelihood. J. Anim. Breed. Genet. **131**(3), 227–236 (2014)
15. Meuwissen, T.H.E.: Maximizing the response of selection with a predefined rate of inbreeding. J. Anim. Sci. **75**, 934–940 (1997)
16. Meuwissen, T.H.E.: GENCONT: an operational tool for controlling inbreeding in selection and conservation schemes. In: Proceeding of 7th world congress on genetics applied to livestock production (2002)
17. Mosek, A.P.S: The MOSEK optimization software. Online at http://www.mosek.com (2010)
18. Mullin, T.J.: OPSEL 1.0: a computer program for optimal selection in forest tree breeding by mathematical programming. Technical Report Nr. 841-2014, Arbetsrapport från Skogforsk (2014)
19. Mullin, T.J.: OPSEL 2.0: a computer program for optimal selection in tree breeding. Technical Report Nr. 954-2017, Arbetsrapport från Skogforsk (2017)
20. Mullin, T.J., Hallander, J., Rosvall, O., Andersson, B.: Using simulation to optimise tree breeding programmes in Europe: an introduction to POPSIM. Technical Report Nr. 711-2010, Arbetsrapport från Skogforsk (2010)
21. Mullin, T.J., Park, Y.S.: Stochastic simulation of population management strategies for tree breeding: a new decision-support tool for personal computers. Silvae Genet. **44**(2), 132–140 (1995)
22. Pong-Wong, R., Woolliams, J.A.: Optimisation of contribution of candidate parents to maximise genetic gain and restricting inbreeding using semidefinite programming. Genet. Sel. Evol. **39**, 3–25 (2007)
23. Quaas, R.L.: Computing the diagonal elements and inverse of a large numerator relationship matrix. Biometrics **32**, 949–953 (1976)
24. Sasakawa, T., Tsuchiya, T.: Optimal magnetic shield design with second-order cone programming. SIAM J. Sci. Comput. **24**(6), 1930–1950 (2003)
25. Schierenbeck, S., Pimentel, E., Tietze, M., Körte, J., Reents, R., Reinhardt, F., Simianer, H., König, S.: Controlling inbreeding and maximizing genetic gain using semi-definite programming with pedigree-based and genomic relationships. J. Dairy Sci. **94**(12), 6143–6152 (2011)
26. Sturm, J.F.: Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. Optim. Methods Softw. **11, 12**(1–4), 625–653 (1999)
27. Toh, K.C., Todd, M.J., Tütüncü, R.H.: SDPT3–a MATLAB software package for semidefinite programming, version 1.3. Optim. Methods Softw. **11, 12**(1–4), 545–581 (1999)
28. Woolliams, J.: Genetic contributions and inbreeding. In: Oldenbroek, K. (ed.) Utilisation and Conservation of Farm Animal Genetic Resources, pp. 147–165. Wageningen Academic Publishers, Wageningen (2007)
29. Wright, S.: Coeffcients of inbreeding and relationship. Am. Nat. **56**, 330–338 (1922)
30. Yamashita, M., Fujisawa, K., Fukuda, M., Kobayashi, K., Nakata, K., Nakata, M.: Latest developments in the SDPA family for solving large-scale SDPs. In: Anjos, M.F., Lasserre, J.B. (eds.) Handbook on Semidefinite, Cone and Polynomial Optimization: Theory, Algorithms, Software and Applications, Chapter 24, pp. 687–714. Springer, New York (2012)
31. Yamashita, M., Fujisawa, K., Fukuda, M., Nakata, K., Nakata, M.: Algorithm 925: parallel solver for semidefinite programming problem having sparse schur complement matrix. ACM Trans. Math. Softw. **39**(1), 6 (2012)
32. Yamashita, M., Nakata, K.: Fast implementation for semidefinite programs with positive matrix completion. Optim. Methods Softw. **30**(5), 1030–1049 (2015)