Mathematical Programming 79 (1997) 235–253

Exploiting sparsity in primal–dual interior-point methods for semidefinite programming

Katsuki Fujisawa*, Masakazu Kojima, Kazuhide Nakata

Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152, Japan

Received 27 February 1997; accepted 10 April 1997

Abstract

The Helmberg–Rendl–Vanderbei–Wolkowicz/Kojima–Shindoh–Hara/Monteiro and Nesterov–Todd search directions have been used in many primal–dual interior-point methods for semidefinite programs. This paper proposes an efficient method for computing the two directions when the semidefinite program to be solved is large scale and sparse. © 1997 The Mathematical Programming Society, Inc. Published by Elsevier Science B.V.

Keywords: Interior-point methods; Semidefinite programming; Sparsity

1. Introduction

This paper deals with two types of search directions which have been used in many primal–dual interior-point methods (see, e.g., [1,2,6–9,11–14,16,17,19]) for semidefinite programming. The first is the HRVW/KSH/M direction which was independently proposed by two groups of researchers, Helmberg, Rendl, Vanderbei and Wolkowicz [6] and Kojima, Shindoh and Hara [8], and later rediscovered by Monteiro [12] in a new formulation. The second is the NT direction by Nesterov and Todd [13,14]. The AHO direction given by Alizadeh, Haeberly and Overton [1] is another important direction in primal–dual interior-point methods for semidefinite programming. Global and local convergence of primal–dual interior-point methods using the HRVW/KSH/M, the NT and the AHO directions have been studied extensively in many papers (see, e.g., [1,2,6–9,11–14,16,19]), and some computational results on the three directions have been reported in the papers [1,2,6,17]. Some computer programs [3,5,18] of

* Corresponding author.

0025-5610/97/$17.00
Published by Elsevier Science B.V.
PII S0025-5610(97)00045-2

primal–dual interior-point methods using the three directions are available through the Internet. The purpose of this paper is to present an efficient method for computing the HRVW/KSH/M and the NT directions when the semidefinite program to be solved is large scale and sparse.

Let $R^{n\times n}$ denote the set of all $n \times n$ real matrices. We regard $R^{n\times n}$ as n^2-dimensional Euclidean space. Let $\mathcal{S}^n$ denote the set of all $n \times n$ symmetric real matrices; $\mathcal{S}^n$ forms an $n(n+1)/2$-dimensional linear subspace of $R^{n\times n}$. For each pair of $\boldsymbol{X}$ and $\boldsymbol{Z}$ in $R^{n\times n}$, $\boldsymbol{X} \bullet \boldsymbol{Z}$ stands for the inner product of $\boldsymbol{X}$ and $\boldsymbol{Z}$, i.e., $\mathrm{Tr}\, \boldsymbol{X}^{\mathrm{T}}\boldsymbol{Z}$, the trace of $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Z}$. We write $\boldsymbol{X} \succ \boldsymbol{O}$ if $\boldsymbol{X} \in \mathcal{S}^n$ is positive definite, i.e., $\boldsymbol{u}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{u} > 0$ for every nonzero $\boldsymbol{u} \in R^n$, and $\boldsymbol{X} \succeq \boldsymbol{O}$ if $\boldsymbol{X} \in \mathcal{S}^n$ is positive semidefinite, i.e., $\boldsymbol{u}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{u} \geqslant 0$ for every $\boldsymbol{u} \in R^n$. Here $\boldsymbol{O}$ denotes the $n \times n$ zero matrix.

Let $\boldsymbol{C} \in \mathcal{S}^n$, $\boldsymbol{A}_i \in \mathcal{S}^n$ $(1 \leqslant i \leqslant m)$ and $b_i \in R$ $(1 \leqslant i \leqslant m)$. Consider the semidefinite program and its dual:

$$\left.\begin{array}{lll} \mathcal{P}: & \text{minimize} & \boldsymbol{C} \bullet \boldsymbol{X} \\ & \text{subject to} & \boldsymbol{A}_i \bullet \boldsymbol{X} = b_i \ (1 \leqslant i \leqslant m), \ \boldsymbol{X} \succeq \boldsymbol{O}. \\ \mathcal{D}: & \text{maximize} & \displaystyle\sum_{i=1}^{m} b_i y_i \\ & \text{subject to} & \displaystyle\sum_{i=1}^{m} \boldsymbol{A}_i y_i + \boldsymbol{Z} = \boldsymbol{C}, \ \boldsymbol{Z} \succeq \boldsymbol{O}. \end{array}\right\} \tag{1}$$

Throughout the paper we assume that the set of $n\times n$ symmetric matrices $\boldsymbol{A}_i$ $(1 \leqslant i \leqslant m)$ is linearly independent. This implies that $m \leqslant n(n+1)/2$. We say that $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ is a feasible solution (an interior-feasible solution, or an optimal solution, respectively) of the SDP (1) if $\boldsymbol{X}$ is a feasible solution (an interior-feasible solution, i.e., a feasible solution satisfying $\boldsymbol{X} \succ \boldsymbol{O}$, or a minimizing solution, respectively) of $\mathcal{P}$ and $(\boldsymbol{y}, \boldsymbol{Z})$ is a feasible solution (an interior-feasible solution, i.e., a feasible solution satisfying $\boldsymbol{Z} \succ \boldsymbol{O}$, or a maximizing solution, respectively) of $\mathcal{D}$.

Generic primal–dual interior-point method.

Step 0. Set up a stopping criterion, and choose an initial point $(\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0)$ such that $\boldsymbol{X}^0 \succ \boldsymbol{O}$ and $\boldsymbol{Z}^0 \succ \boldsymbol{O}$. Let $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z}) = (\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0)$.

Step 1. If the current iterate $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ satisfies the stopping criterion, stop the iteration.

Step 2. Choose a search direction $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$.

Step 3. Choose a primal step length α_{p} and a dual step length α_{d} such that

$$\boldsymbol{X} + \alpha_{\mathrm{p}} d\boldsymbol{X} \succ \boldsymbol{O} \quad \text{and} \quad \boldsymbol{Z} + \alpha_{\mathrm{d}} d\boldsymbol{Z} \succ \boldsymbol{O}.$$

Let

$$\boldsymbol{X} = \boldsymbol{X} + \alpha_{\mathrm{p}} d\boldsymbol{X} \quad \text{and} \quad (\boldsymbol{y}, \boldsymbol{Z}) = (\boldsymbol{y}, \boldsymbol{Z}) + \alpha_{\mathrm{d}}(d\boldsymbol{y}, d\boldsymbol{Z}).$$

Step 4. Go to Step 1.

Most of the primal-dual interior-point methods developed so far may be obtained if we specify

- a stopping criterion at Step 0,
- an initial point (X^0, y^0, Z^0) at Step 0,
- a search direction (dX, dy, dZ) at Step 2,
- a primal step length α_p and a dual step length α_d at Step 3.

in the generic primal-dual interior-point method described above.

Section 2 describes the HRVW/KSH/M and the NT directions.

Section 3 studies efficient computation of the HRVW/KSH/M and the NT directions. We first reduce their computation to a system of linear equations $\boldsymbol{B}dy = \boldsymbol{r}$ in $dy \in R^m$ with an $m \times m$ symmetric matrix $\boldsymbol{B}$ and $\boldsymbol{r} \in R^m$ (see (6) and (7)). In the linear programming case, $\boldsymbol{B}$ corresponds to $\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^{\mathrm{T}}$, where $\boldsymbol{A}$ denotes an $m \times n$ matrix associated with the equality constraint $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ of the primal problem and $\boldsymbol{D}$ a primal-dual diagonal scaling matrix, and $\boldsymbol{B}$ is usually sparse when $\boldsymbol{A}$ is sparse. But, in the SDP case,

- $\boldsymbol{B}$ is generally full dense even when the data matrices $\boldsymbol{A}_i$ $(1 \leqslant i \leqslant m)$ are sparse,
- the computation of $\boldsymbol{B}$ is much more expensive than in the linear programming case, and it is the most costly part of an iteration if dense computations are used.

We concentrate our effort on exploiting sparsity in the computation of $\boldsymbol{B}$. We present three formulae for $\boldsymbol{B}$, and their efficient combination which achieves great savings when some $\boldsymbol{A}_i$s are dense and some others are sparse.

Section 4 is devoted to computational results on the three formulae for $\boldsymbol{B}$ and their efficient combination given in Section 3. We will observe there that the combined formula works effectively, and it leads to other parts of the computation in an iteration being more important than the computation of $\boldsymbol{B}$ in some cases.

2. Search directions

The HRVW/KSH/M direction is described as a solution (dX, dy, dZ) of the three types of equations

$$\boldsymbol{A}_i \bullet d\boldsymbol{X} = p_i \ (1 \leqslant i \leqslant m), \quad d\boldsymbol{X} \in \mathcal{S}^n, \tag{2}$$

$$\sum_{i=1}^{m} \boldsymbol{A}_i dy_i + d\boldsymbol{Z} = \boldsymbol{D}, \quad d\boldsymbol{Z} \in \mathcal{S}^n, \tag{3}$$

$$\widehat{d\boldsymbol{X}}\boldsymbol{Z} + \boldsymbol{X}d\boldsymbol{Z} = \boldsymbol{K}', \quad \widehat{d\boldsymbol{X}} \in R^{n\times n}, \ d\boldsymbol{X} = (\widehat{d\boldsymbol{X}} + \widehat{d\boldsymbol{X}}^{\mathrm{T}})/2. \tag{4}$$

Here we regard $\widehat{d\boldsymbol{X}} \in R^{n\times n}$ as an auxiliary variable matrix, and $p_i \in R$, $\boldsymbol{D} \in \mathcal{S}^n$ and $\boldsymbol{K}' \in R^{n\times n}$ denote a scalar constant, an $n \times n$ constant symmetric matrix, and an $n \times n$ constant matrix, respectively, which are determined by the current point $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ and some other factors. They differ from one method to another. For example, we take

$$p_i = 0 \ (1 \leqslant i \leqslant m), \quad \boldsymbol{D} = \boldsymbol{O} \quad \text{and}$$

$$\boldsymbol{K}' = \beta \frac{\boldsymbol{X} \bullet \boldsymbol{Z}}{n} \boldsymbol{I} - \boldsymbol{X}\boldsymbol{Z} \quad \text{for some } \beta \in (0, 1)$$

in the feasible-path-following method using the HRVW/KSH/M direction. But their actual values are not relevant in the succeeding discussions. See, for example, [1,2,6–9,11,12,16,17,19] for various primal–dual interior-point methods using the HRVW/KSH/M direction. Under the linear independence assumption on the set $\{\boldsymbol{A}_i \mid 1 \leqslant i \leqslant m\}$ of constraint matrices, it is known [8] that for any $\boldsymbol{X} \succ \boldsymbol{O}$, $\boldsymbol{Z} \succ \boldsymbol{O}$, $p_i \in R$ $(1 \leqslant i \leqslant m)$, $\boldsymbol{D} \in \mathcal{S}^n$, and $\boldsymbol{K}' \in R^{n \times n}$, the system of equations (2), (3) and (4) has a unique solution $(d\boldsymbol{X}, dy, d\boldsymbol{Z})$.

The NT direction shares equations (2) and (3) with the HRVW/KSH/M direction, but we replace (4) by

$$d\boldsymbol{X}\boldsymbol{W}^{-1} + \boldsymbol{W}d\boldsymbol{Z} = \boldsymbol{K}'', \tag{5}$$

where $\boldsymbol{W} = \boldsymbol{X}^{1/2}(\boldsymbol{X}^{1/2}\boldsymbol{Z}\boldsymbol{X}^{1/2})^{-1/2}\boldsymbol{X}^{1/2} \in \mathcal{S}^n$, and $\boldsymbol{K}'' \in R^{n \times n}$ are determined by the current point $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ and some other factors. The NT direction is given as a solution $(d\boldsymbol{X}, dy, d\boldsymbol{Z})$ of the system of equations (2), (3) and (5). The same comments on the values of $p_i \in R$, $\boldsymbol{D} \in \mathcal{S}^n$ and $\boldsymbol{K}'' \in R^{n \times n}$ and on the existence and uniqueness of the direction apply to the NT direction as in the case of the HRVW/KSH/M direction. See [17] for more details.

3. Computing directions

Todd, Toh and Tütüncü [17] presented some numerically stable methods for computing the AHO, the NT and the HRVW/KSH/M directions, and reported numerical results showing that numerical stability depends on how we compute the directions. Alizadeh, Haeberly and Overton [2] also proposed some numerically stable methods for the three directions. We will investigate below how to compute the HRVW/KSH/M and the NT directions. However, the main issue here is not the numerical stability but computational efficiency when some or all of the data matrices $\boldsymbol{A}_i$ $(1 \leqslant i \leqslant m)$ of the SDP are sparse.

In the HRVW/KSH/M direction case, we reduce the system of equations (2), (3) and (4) to

$$\left.\begin{aligned} &\boldsymbol{B}'dy = \boldsymbol{r}', \\ &d\boldsymbol{Z} = \boldsymbol{D} - \sum_{j=1}^{m} \boldsymbol{A}_j dy_j, \\ &\widehat{d\boldsymbol{X}} = (\boldsymbol{K}' - \boldsymbol{X}d\boldsymbol{Z})\boldsymbol{Z}^{-1}, \ d\boldsymbol{X} = (\widehat{d\boldsymbol{X}} + \widehat{d\boldsymbol{X}}^{\mathrm{T}})/2, \end{aligned}\right\} \tag{6}$$

where

$$B'_{ij} = \boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_j \quad (1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant m), \tag{8'}$$
$$r'_i = p_i - (\boldsymbol{K}' - \boldsymbol{X}\boldsymbol{D})\boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_i \quad (1 \leqslant i \leqslant m).$$

The $n \times n$ matrix $\boldsymbol{X}$, the $n \times n$ matrix $\boldsymbol{Z}^{-1}$ and the $m \times m$ matrix $\boldsymbol{B}'$ are symmetric, and dense in general even when all $\boldsymbol{A}_i$ $(1 \leqslant i \leqslant m)$ are sparse. Hence solving the system of equations (6) in $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ by using a direct method such as the Cholesky factorization requires $O(m^3) + O(n^3)$ arithmetic operations. On the other hand, if we use the above formulae for the coefficient matrix $\boldsymbol{B}'$ and the right hand side vector $\boldsymbol{r}'$ in a straightforward way, the computation of $\boldsymbol{B}'$ requires $O(mn^3 + m^2n^2)$ arithmetic operations and the computation of $\boldsymbol{r}'$ $O(n^3 + mn^2)$ arithmetic operations, respectively. Here the operation counts are for the case that the matrices $\boldsymbol{A}_i$ $(1 \leqslant i \leqslant m)$ are dense. Therefore computing the coefficient matrix $\boldsymbol{B}'$ is more crucial than computing $\boldsymbol{r}'$ and solving $\boldsymbol{B}'d\boldsymbol{y} = \boldsymbol{r}'$ in the entire computation of the HRVW/KSH/M direction. We will investigate how efficiently we can compute the matrix $\boldsymbol{B}'$ when some or all of the matrices $\boldsymbol{A}_i$ $(1 \leqslant i \leqslant m)$ are sparse.

We can reduce the system of equations (2), (3) and (5) describing the NT direction to a similar system of equations as (6):

$$\left.\begin{aligned} &\boldsymbol{B}''d\boldsymbol{y} = \boldsymbol{r}'', \\ &d\boldsymbol{Z} = \boldsymbol{D} - \sum_{j=1}^{m} \boldsymbol{A}_j dy_j, \\ &d\boldsymbol{X} = (\boldsymbol{K}'' - \boldsymbol{W}d\boldsymbol{Z})\boldsymbol{W}, \end{aligned}\right\} \tag{7}$$

where

$$B''_{ij} = \boldsymbol{W}\boldsymbol{A}_i\boldsymbol{W} \bullet \boldsymbol{A}_j \quad (1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant m), \tag{8''}$$
$$r''_i = p_i - (\boldsymbol{K}'' - \boldsymbol{W}\boldsymbol{D})\boldsymbol{W} \bullet \boldsymbol{A}_i \quad (1 \leqslant i \leqslant m).$$

Similar comments apply to the computation of the $m \times m$ symmetric matrix $\boldsymbol{B}''$ in (8''), the computation of the right-hand side vector $\boldsymbol{r}''$, and solving the system (7) of equations in the NT direction $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ as in the case of the HRVW/KSH/M direction above.

The remainder of this section is devoted to an efficient computation of the $m \times m$ symmetric matrices $\boldsymbol{B}'$ and $\boldsymbol{B}''$. To deal with both directions simultaneously, we consider

$$B_{ij} = \boldsymbol{T}\boldsymbol{A}_i\boldsymbol{U} \bullet \boldsymbol{A}_j \tag{8}$$

$(1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant m)$, where $\boldsymbol{T} \in \mathcal{S}^n$ and $\boldsymbol{U} \in \mathcal{S}^n$, If we take $\boldsymbol{T} = \boldsymbol{X}$ and $\boldsymbol{U} = \boldsymbol{Z}^{-1}$, then $\boldsymbol{B} = \boldsymbol{B}'$. If $\boldsymbol{T} = \boldsymbol{U} = \boldsymbol{W}$ then $\boldsymbol{B} = \boldsymbol{B}''$. Note that $\boldsymbol{B}$ is symmetric; hence we only need to compute the upper triangular part of $\boldsymbol{B}$, i.e., B_{ij} $(1 \leqslant i \leqslant j \leqslant m)$.

There are many factors which take part in the CPU time for computing the matrix $\boldsymbol{B}$. First we focus our attention on the number of multiplications, and later we incorporate some other factors.

Let f_i denote the number of nonzero elements in $\boldsymbol{A}_i$, and $\boldsymbol{\Sigma}$ the set of permutations of the indices $1, 2, \ldots, m$ (i.e., one-to-one mappings from $\{1, 2, \ldots, m\}$ onto itself). Each $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ determines an order in computation of the elements B_{ij} $(1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant m)$ such that

$$\left.\begin{array}{l} \rightarrow \\ B_{\sigma(1)\sigma(1)},\ B_{\sigma(1)\sigma(2)},\ \ldots\ B_{\sigma(1)\sigma(m)}, \\ \qquad \rightarrow \\ \qquad B_{\sigma(2)\sigma(2)},\ \ldots\ B_{\sigma(2)\sigma(m)}, \\ \qquad\quad \rightarrow \\ \qquad\qquad \ldots \\ \qquad\qquad B_{\sigma(m)\sigma(m)}. \end{array}\right\} \tag{9}$$

Recall that $\boldsymbol{B}$ is symmetric; hence $B_{\sigma(j)\sigma(i)} = B_{\sigma(i)\sigma(j)}$ $(1 \leqslant i < j \leqslant m)$.

Let $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ and $i \in \{1, 2, \ldots, m\}$ be fixed. We introduce three kinds of formulae for computing $B_{\sigma(i)\sigma(j)}$ $(i \leqslant j \leqslant m)$ below.

- $\mathcal{F}_1$: Compute $\boldsymbol{F} = \boldsymbol{A}_{\sigma(i)}\boldsymbol{U}$, which requires $nf_{\sigma(i)}$ multiplications, and $\boldsymbol{G} = \boldsymbol{TF}$, which requires n^3 multiplications. For each $j = i, i+1, \ldots, m$, compute $B_{\sigma(i)\sigma(j)} = \boldsymbol{G} \bullet \boldsymbol{A}_{\sigma(j)}$, which requires $f_{\sigma(j)}$ multiplications. Then the total number of multiplications to compute all $B_{\sigma(i)\sigma(j)}$ $(i \leqslant j \leqslant m)$ turns out to be

$$nf_{\sigma(i)} + n^3 + \sum_{i \leqslant j \leqslant m} f_{\sigma(j)}. \tag{10}$$

- $\mathcal{F}_2$: Compute $\boldsymbol{F} = \boldsymbol{A}_{\sigma(i)}\boldsymbol{U}$, which requires $nf_{\sigma(i)}$ multiplications. For each $j = i, i+1, \ldots, m$, compute

$$B_{\sigma(i)\sigma(j)} = \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{A}_{\sigma(j)}]_{\alpha\beta}\left(\sum_{\gamma=1}^{n} T_{\alpha\gamma}[\boldsymbol{F}]_{\gamma\beta}\right),$$

which requires $(n+1)f_{\sigma(j)}$ multiplications. Then the total number of multiplications to compute all $B_{\sigma(i)\sigma(j)}$ $(i \leqslant j \leqslant m)$ is given by

$$nf_{\sigma(i)} + (n+1)\sum_{i \leqslant j \leqslant m} f_{\sigma(j)}. \tag{11}$$

- $\mathcal{F}_3$: For each $j = i, i+1, \ldots, m$, compute

$$\sum_{\gamma=1}^{n}\sum_{\varepsilon=1}^{n}\left(\sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{A}_{\sigma(i)}]_{\alpha\beta}T_{\alpha\gamma}U_{\beta\varepsilon}\right)[\boldsymbol{A}_{\sigma(j)}]_{\gamma\varepsilon},$$

which requires $(2f_{\sigma(i)} + 1)f_{\sigma(j)}$ multiplications. Hence the total number of multiplications to compute all $B_{\sigma(i)\sigma(j)}$ $(i \leqslant j \leqslant m)$ is given by

$$(2f_{\sigma(i)} + 1)\sum_{i \leqslant j \leqslant m} f_{\sigma(j)} \tag{12}$$

We now consider some other factors. The number of additions is a similar order as the number of multiplications (see Remark B of Section 5 for more details). But the overhead in dealing with the sparse data matrices A_i $(1 \leqslant i \leqslant m)$ depends deeply on their data structure and also on the memory management of the operating system of the computer which we use. We may assume that

(i) all A_i $(1 \leqslant i \leqslant m)$ are stored in a common sparse data matrix structure,

(ii) the matrices T and U are stored in the standard matrix array since they are generally dense.

Usually we store each nonzero element of A_i together with its row and column indices in computer memory. (In our numerical experiments, we use the sparse data structure as shown in Fig. 1.) Suppose that we multiply a nonzero element $[A_i]_{pq}$ of A_i by an element H_{rs} of a dense matrix H, where H can be T, U, F or G in formulae $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$, and the row index r and/or the column index s of H_{rs} depend on the indices p and/or q of $[A_i]_{pq}$. What takes extra time are reading the indices p and q and accessing to the element H_{rs} of the dense matrix H. Also we need to consider the overhead in using (initializing, writing and reading) the dense matrices F and G in formula $\mathcal{F}_1$, and the overhead in using (initializing, writing and reading) the dense matrix F in formula $\mathcal{F}_2$.

Incorporating all factors mentioned above into our consideration would be too complicated. Instead, we simply modify formulae (10), (11) and (12) to define "the weighted number $d_{ki}(\sigma)$ of multiplications" to compute all $B_{\sigma(i)\sigma(j)}$ $(i \leqslant j \leqslant m)$ by formula $\mathcal{F}_k$ $(k = 1, 2, 3)$:

$$d_{1i}(\sigma) = \kappa n f_{\sigma(i)} + n^3 + \kappa \sum_{i \leqslant j \leqslant m} f_{\sigma(j)} \tag{10'}$$

$$d_{2i}(\sigma) = \kappa n f_{\sigma(i)} + \kappa(n+1) \sum_{i \leqslant j \leqslant m} f_{\sigma(j)}, \tag{11'}$$

$$d_{3i}(\sigma) = \kappa(2\kappa f_{\sigma(i)} + 1) \sum_{i \leqslant j \leqslant m} f_{\sigma(j)}, \tag{12'}$$

where $\kappa \geqslant 1$ is a constant. We may regard κ as an overhead in dealing with the sparse data matrices A_i $(1 \leqslant i \leqslant m)$. If replace $\kappa f_{\sigma(i)}$ by $f_{\sigma(i)}$ then (10)', (11)' and (12)' coincide with their original definitions (10), (11) and (12).

Although the weighted number of multiplications only partially explains the CPU time for computing B by formulae $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$, it enjoys some nice theoretical properties as we see in Theorem 1 and it is useful enough in practice as we see in Section 4 where we take $\kappa = 1.5$, and report numerical results on formulae $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$ and their efficient combination (the combined formula $\mathcal{F}_*(\kappa)$ below). Because of these reasons, we employ the weighted number of multiplications for measuring the efficiency of the formulae $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$.

Let $\sigma \in \Sigma$. Define

$$d_{*i}(\sigma) = \min\{d_{1i}(\sigma), d_{2i}(\sigma), d_{3i}(\sigma)\} \quad (1 \leqslant i \leqslant m), \tag{13}$$

$$d_*(\sigma) = \sum_{1 \leqslant i \leqslant m} d_{*i}(\sigma). \tag{14}$$

Then, for each $i = 1, 2, \ldots, m$, $d_{*i}(\sigma)$ denotes the minimum weighted number of multiplications to compute $B_{\sigma(i)\sigma(j)}$ $(i \leqslant j \leqslant m)$ by using any of formulae $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$, and their sum $d_*(\sigma)$ over all $i = 1, 2, \ldots, m$ denotes the minimum weighted number of multiplications that are required to compute all the elements of the matrix $\boldsymbol{B}$ in the order (9). $d_*(\sigma)$ depends on $\sigma \in \Sigma$, i.e., how we permute the indices $1, 2, \ldots, m$ before applying formulae $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$ to computation of the elements of the matrix $\boldsymbol{B}$. We want to choose a permutation that minimizes $d_*(\sigma)$ over all $\sigma \in \Sigma$. The theorem below states that the minimum of $d_*(\sigma)$ over all $\sigma \in \Sigma$ is attained if we sort the indices according to $f_1, f_2, \ldots, f_m$ in a descending order.

Theorem 1. (i) *σ^* is a minimizer of $d_*(\sigma)$ over all $\sigma \in \Sigma$ if and only if it satisfies*

$$f_{\sigma^*(1)} \geqslant f_{\sigma^*(2)} \geqslant \cdots \geqslant f_{\sigma^*(m)}. \tag{15}$$

(ii) *Suppose that $\sigma^* \in \Sigma$ satisfies (15). Then there exist $q_1 \in \{0, 1, 2, \ldots, m\}$ and $q_2 \in \{q_1, q_1 + 1, \ldots, m\}$ such that*

$$\left.\begin{array}{lll} d_{1i}(\sigma^*) \leqslant d_{2i}(\sigma^*), & d_{1i}(\sigma^*) \leqslant d_{3i}(\sigma^*) & \text{if } 0 < i \leqslant q_1, \\ d_{2i}(\sigma^*) < d_{1i}(\sigma^*), & d_{2i}(\sigma^*) \leqslant d_{3i}(\sigma^*) & \text{if } q_1 < i \leqslant q_2, \\ d_{3i}(\sigma^*) < d_{1i}(\sigma^*), & d_{3i}(\sigma^*) < d_{2i}(\sigma^*) & \text{if } q_2 < i \leqslant m. \end{array}\right\} \tag{16}$$

Proof. See the Appendix.

Based on the theorem above, we now propose:

Combined formula $\mathcal{F}_*(\kappa)$.

Step A. Count the number f_i of nonzero elements in $\boldsymbol{A}_i$ $(1 \leqslant i \leqslant m)$.

Step B. Sort the set of indices $1, 2, \ldots, m$ according to $f_1, f_2, \ldots, f_m$ in a descending order as given in (15), where σ^* is a permutation of the index set $\{1, 2, \ldots, m\}$. For each $i = 1, 2, \ldots, m$, compute $d_{1i}(\sigma^*)$ by (10′), $d_{2i}(\sigma^*)$ by (11′) and $d_{3i}(\sigma^*)$ by (12′). Find $q_1 \in \{0, 1, 2, \ldots, m\}$ and $q_2 \in \{q_1, q_1 + 1, \ldots, m\}$ satisfying (16).

Step C. For every $i \in \{1, 2, \ldots, m\}$,
- use formula $\mathcal{F}_1$ if $0 < i \leqslant q_1$, or
- use formula $\mathcal{F}_2$ if $q_1 < i \leqslant q_2$, or
- use formula $\mathcal{F}_3$ otherwise (i.e., if $q_2 < i \leqslant m$),

to compute $B_{\sigma^*(i)\sigma^*(j)}$ $(i \leqslant j \leqslant m)$.

We can easily incorporate these three steps into a generic primal-dual interior-point method using the HRVW/KSH/M direction or the NT direction. We place Step A where we read data matrices $\boldsymbol{A}_i$ $(1 \leqslant i \leqslant m)$, and Step B right after Step A. Hence we

$$A_i = \begin{pmatrix} 11 & 0 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 25 \\ 13 & 0 & 33 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 25 & 0 & 0 & 0 \end{pmatrix} \quad \Rightarrow$$

row →	1	1	2	3
column →	1	3	5	3
value →	11	13	25	33

Fig. 1. Data structure for sparse data matrices.

execute both steps once in the method. On the other hand, we place Step C where we compute a direction. Hence we need to execute Step C repeatedly at each iteration of the primal–dual interior-point method.

4. Numerical results

In this section, we present some numerical results on formulae $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$ and $\mathcal{F}_*(\kappa)$ for four kinds of SDPs. One is a randomly generated SDP, and the others are SDP relaxations of the quadratic assignment problem, the graph partitioning problem and the maximum clique problem. See, for example, [6,15,20]. We applied a modified version of the SDPA [5] using the HRVW/KSH/M search direction to all SDPs. The numerical results were computed on DEC Alpha (CPU 21164-300MHz with 256MB memory). In general, the overhead parameter κ in dealing with the sparse data matrices $\boldsymbol{A}_i$ $(1 \leqslant i \leqslant m)$ depends on the data structure for $\boldsymbol{A}_i$ $(1 \leqslant i \leqslant m)$. We employed the sparse data structure illustrated in Fig. 1. We made preliminary numerical experiments to estimate $\kappa = 1.5$.

4.1. Randomly generated SDPs

Let p be a positive integer. Define the function $f(\cdot; m, n, p) : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, n^2\}$ by

$$f(i; m, n, p) = \left[\frac{(n^2 - 1)(m - i)^p}{(m - 1)^p} + 1\right],$$

where $[a]$ denotes the largest integer not less than a. To construct an SDP, we randomly generated an $n \times n$ data matrix $\boldsymbol{A}_i \in \mathcal{S}^n$ with each nonzero element uniformly distributed in the interval $[-10.0, 10.0]$ and an expected number of nonzero elements equal to $f(i; m, n, p)$ $(1 \leqslant i \leqslant m)$ (If $\boldsymbol{A}_i = \boldsymbol{O}$ then we modified it so that it had at least one nonzero element.) The parameter p controls how fast the expected number of nonzero elements in the $n \times n$ symmetric matrix $\boldsymbol{A}_i$ decreases as i increases from 1 to m. For any choice of p, $\boldsymbol{A}_1$ has n^2 nonzero elements and $\boldsymbol{A}_m$ at least one nonzero element. We took $p = \log_2 n$, so that $\boldsymbol{A}_{\lfloor m/2 \rfloor}$ expectedly had about n nonzeros. Table 1 shows the numerical results. The columns $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$ and $\mathcal{F}_*(1.5)$ denote the average CPU time in seconds to compute the $m \times m$ matrix $\boldsymbol{B}$ per iteration when we used formulae $\mathcal{F}_1$, $\mathcal{F}_2$,

Table 1
Randomly generated SDPs

p	n	m	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_*(1.5)$	q_1	q_2	one iteration $-\mathcal{F}$-$*(1.5)$
5	32	32	0.07	0.08	1.65	0.04	11	20	0.08
5	32	64	0.14	0.30	5.87	0.09	26	40	0.10
5	32	128	0.30	1.02	23.19	0.22	62	80	0.14
6	64	64	1.10	1.79	71.56	0.59	22	39	0.71
6	64	128	2.24	7.02	279.45	1.33	52	76	0.77
6	64	256	4.75	23.81	–	3.14	118	152	1.09
7	128	128	19.89	–	–	11.98	49	79	6.43
7	128	256	42.82	–	–	24.07	99	147	7.06

$\mathcal{F}_3$ and $\mathcal{F}_*(1.5)$ (the combined formula $\mathcal{F}_*(\kappa)$ with $\kappa = 1.5$), respectively. The last column denotes "the average CPU time of one iteration in seconds − the average CPU time to compute $\boldsymbol{B}$ in seconds" when we used formula $\mathcal{F}_*(1.5)$. One iteration consists of computation of the matrix $\boldsymbol{B}$, a search direction, a step length, a new iterate, etc. Each average CPU time in Table 1 was taken over the first 10 iterations. Recall also that q_1 and q_2 denote the positive numbers determined by (16), i.e., in the combined formula $\mathcal{F}_*(1.5)$, to compute $B_{\sigma^*(i)\sigma^*(j)}$ $(i \leqslant j \leqslant m)$, we used $\mathcal{F}_1$ if $0 < i \leqslant q_1$, $\mathcal{F}_2$ if $q_1 < i \leqslant q_2$ and $\mathcal{F}_3$ otherwise (i.e., if $q_2 < i \leqslant m$). We see from Table 1 that

- formula $\mathcal{F}_*(1.5)$ works efficiently, and
- as m increases, the CPU time to compute $\boldsymbol{B}$ becomes more crucial in the CPU time of one iteration.

Figs. 2 and 3 compare the weighted number

$$d_{*i}(\boldsymbol{\sigma}^*) = \min\{d_{1i}(\boldsymbol{\sigma}^*), d_{2i}(\boldsymbol{\sigma}^*), d_{3i}(\boldsymbol{\sigma}^*)\}$$

of multiplications with the average CPU time to compute $B_{\sigma^*(i)\sigma^*(j)}$ $(i \leqslant j \leqslant m)$ per iteration for the case of $p = 7$, $n = 128$ and $m = 256$. We observe that our theoretical efficiency measure $d_{ki}(\boldsymbol{\sigma}^*)$ for formula $\mathcal{F}_k$ works quite nicely $(k = 1, 2, 3)$.

4.2. *Quadratic assignment problems*

As SDP relaxations of quadratic assignment problems, we generated SDPs of the form (1) with $n = s^2 + 1$, $m = 2s + s^3 + 1$, $2s$ $\boldsymbol{A}_i$'s having s^4 nonzero elements, s^2 $\boldsymbol{A}_i$'s having 3 nonzero elements, $(s^3 - s^2)$ $\boldsymbol{A}_i$'s having 2 nonzero elements and one $\boldsymbol{A}_i$ having 1 nonzero element. Here s denotes the size of a quadratic assignment problem. (Our formulation is based on the paper [15], which is different from the projected SDP formulation given in the paper [20]). In this case, we have that

$$f_{\sigma^*(i)} = s^4 \quad (1 \leqslant i \leqslant 2s),$$

$$f_{\sigma^*(i)} \in \{1, 2, 3\} \quad (2s + 1 \leqslant i \leqslant 2s + s^3 + 1),$$

$$d_{1i}(\boldsymbol{\sigma}^*) < d_{2i}(\boldsymbol{\sigma}^*) \text{ and } d_{1i}(\boldsymbol{\sigma}^*) < d_{3i}(\boldsymbol{\sigma}^*) \quad (1 \leqslant i \leqslant 2s),$$

$$d_{3i}(\boldsymbol{\sigma}^*) < d_{1i}(\boldsymbol{\sigma}^*) \text{ and } d_{3i}(\boldsymbol{\sigma}^*) < d_{2i}(\boldsymbol{\sigma}^*) \quad (2s + 1 \leqslant i \leqslant 2s + s^3 + 1).$$

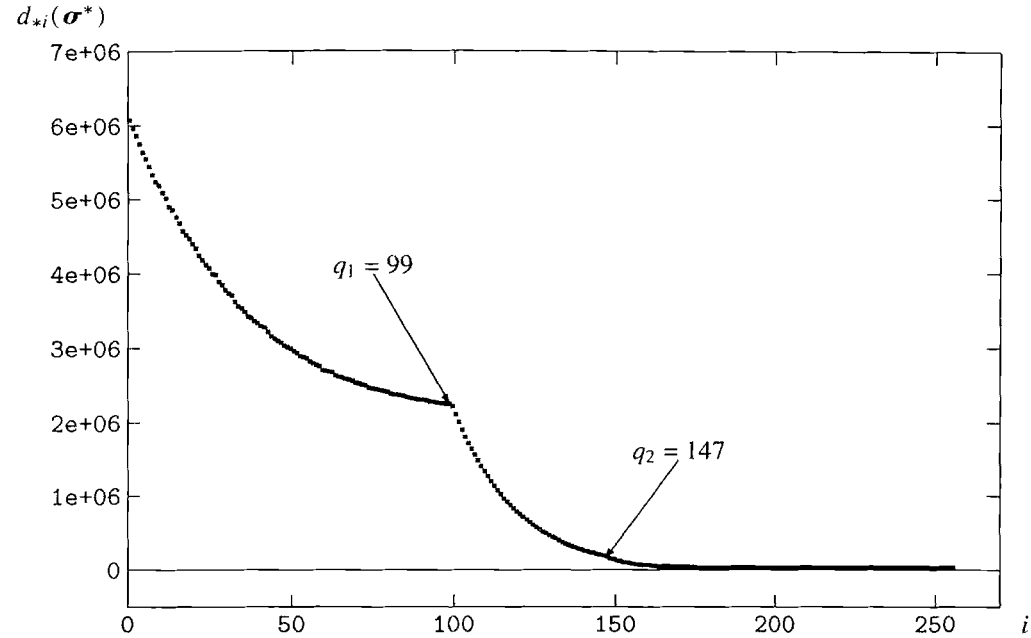

Fig. 2. Weighted number $d_{*i}(\sigma^*)$ of multiplications for the case: $p = 7$, $n = 128$ and $m = 256$.

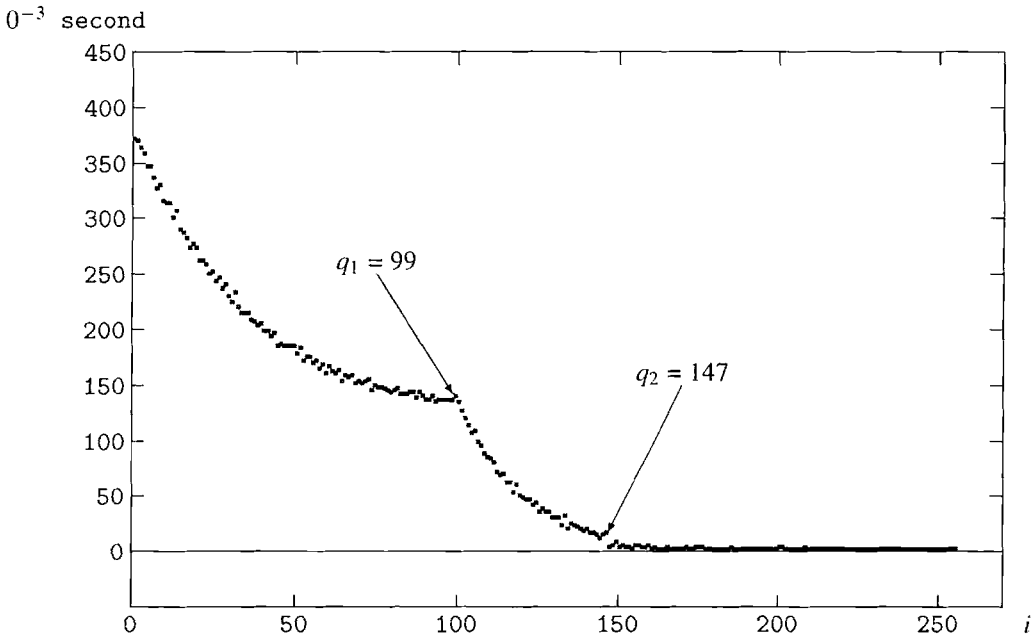

Fig. 3. Average CPU time to compute $B_{\sigma^*(i)\sigma^*(j)}$ $(i \leqslant j \leqslant m)$ for the case: $p = 7$, $n = 128$ and $m = 256$.

Table 2
Quadratic assignment problems

s	n	m	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_*(1.5)$	$q_1 = q_2$	one iteration $-\mathcal{F}$-$*(1.5)$
5	26	136	0.14	0.12	2.15	0.04	10	0.10
6	37	229	0.67	0.48	11.73	0.13	12	0.34
7	50	358	3.33	1.91	56.52	0.42	14	1.32
8	65	529	8.29	4.65	–	0.90	16	3.16
9	82	748	22.89	11.79	–	1.99	18	11.08
10	101	1021	61.28	29.47	–	4.50	20	36.75

Table 3
Graph partitioning problems

s	t	n	m	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_*(1.5)$	one iteration $-\mathcal{F}$-$*(1.5)$
124	1271	124	125	21.15	0.56	22.13	0.36	6.36
250	2421	250	251	360.09	5.07	356.96	2.99	59.50
500	5120	500	501	7247.16	52.04	6341.55	29.29	607.23

Therefore $q_1 = q_2 = 2s$. Table 2 shows the numerical results. The combined formula $\mathcal{F}_*(1.5)$ works very efficiently.

4.3. *Graph partitioning problems*

As SDP relaxations of graph partitioning problems, we generated SDPs of the form (1) with $n = s$, $m = s + 1$, one $\boldsymbol{A}_i$ having s^2 nonzero elements and s $\boldsymbol{A}_i$'s having 1 nonzero element. Here s denotes the number of nodes. (Our formulation is based on the paper [6, Section 3.2].) In this case, we have that

$$f_{\sigma^*(1)} = s^2,$$
$$f_{\sigma^*(i)} = 1 \quad (2 \leqslant i \leqslant s+1),$$
$$d_{11}(\boldsymbol{\sigma}^*) < d_{21}(\boldsymbol{\sigma}^*) \text{ and } d_{11}(\boldsymbol{\sigma}^*) < d_{31}(\boldsymbol{\sigma}^*),$$
$$d_{3i}(\boldsymbol{\sigma}^*) < d_{1i}(\boldsymbol{\sigma}^*) \text{ and } d_{3i}(\boldsymbol{\sigma}^*) < d_{2i}(\boldsymbol{\sigma}^*) \quad (2 \leqslant i \leqslant s+1).$$

Therefore $q_1 = 1$ and $q_2 = 1$. Table 3 shows the numerical result, where t denotes the number of edges. The combined formula $\mathcal{F}_*(1.5)$ works efficiently.

4.4. *Maximum clique problems*

As SDP relaxations of maximum clique problems, we generated SDPs of the form (1) with $n = s$, $m = u + 1$, one $\boldsymbol{A}_i$ having s nonzero elements and u $\boldsymbol{A}_i$'s having 2 nonzero elements. Here s denotes the number of nodes and u denotes the number of pairs of nodes having no edge between them (or $n(n-1)/2$ – "the number of edges"). (Our formulation is based on the paper [6, Section 3.1].) In this case, we have that

Table 4
Maximum clique problems

s	u	n	m	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_*(1.5)$	one iteration $-\mathcal{F}$-$*(1.5)$
150	561	150	562	123.15	5.35	0.36	0.36	13.20
150	1101	150	1102	238.90	19.70	1.46	1.46	44.75
200	621	200	622	336.58	9.30	0.51	0.48	28.82
200	992	200	993	540.16	28.73	1.27	1.32	50.28
250	651	250	652	980.63	19.01	0.59	0.61	63.43
250	972	250	973	1396.54	40.86	1.54	1.33	80.04
300	943	300	944	2472.22	43.01	1.40	1.29	120.21

$$
\begin{aligned}
&f_{\sigma^*(1)} = s,\\
&f_{\sigma^*(i)} = 2 \quad (2 \leqslant i \leqslant u+1),\\
&d_{31}(\boldsymbol{\sigma}^*) < d_{11}(\boldsymbol{\sigma}^*) \text{ and } d_{31}(\boldsymbol{\sigma}^*) < d_{21}(\boldsymbol{\sigma}^*) \quad \text{if } u < s/(4\kappa-2),\\
&d_{21}(\boldsymbol{\sigma}^*) < d_{11}(\boldsymbol{\sigma}^*) \text{ and } d_{21}(\boldsymbol{\sigma}^*) \leqslant d_{31}(\boldsymbol{\sigma}^*) \quad \text{if } s/(4\kappa-2) \leqslant u < s^2/(2\kappa),\\
&d_{11}(\boldsymbol{\sigma}^*) \leqslant d_{21}(\boldsymbol{\sigma}^*) \text{ and } d_{11}(\boldsymbol{\sigma}^*) < d_{31}(\boldsymbol{\sigma}^*) \quad \text{if } s^2/(2\kappa) \leqslant u,\\
&d_{3i}(\boldsymbol{\sigma}^*) < d_{1i}(\boldsymbol{\sigma}^*) \text{ and } d_{3i}(\boldsymbol{\sigma}^*) < d_{2i}(\boldsymbol{\sigma}^*) \quad (2 \leqslant i \leqslant u+1).
\end{aligned}
$$

Therefore

$$
\begin{aligned}
&q_1 = 0 \text{ and } q_2 = 0 \quad \text{if } u < s/(4\kappa-2),\\
&q_1 = 0 \text{ and } q_2 = 1 \quad \text{if } s/(4\kappa-2) \leqslant u < s^2/(2\kappa),\\
&q_1 = 1 \text{ and } q_2 = 1 \quad \text{if } s^2/(2\kappa) \leqslant u.
\end{aligned}
$$

Table 4 shows the numerical result. Again the combined formula $\mathcal{F}_*(1.5)$ works efficiently.

5. Concluding remarks

(A) We have assumed the matrix $\boldsymbol{F} = \boldsymbol{A}_{\sigma(i)}\boldsymbol{U}$ to be stored as a dense matrix when we count the number n^3 of multiplications to compute $\boldsymbol{G} = \boldsymbol{T}\boldsymbol{F}$ in formula $\mathcal{F}_1$, and when we count the number $(n+1)f_{\sigma(j)}$ of multiplications to compute

$$
B_{\sigma(i)\sigma(j)} = \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{A}_{\sigma(j)}]_{\alpha\beta}\Big(\sum_{\gamma=1}^{n}T_{\alpha\gamma}[\boldsymbol{F}]_{\gamma\beta}\Big)
$$

in formula $\mathcal{F}_2$. It should be noted that the γth row of the matrix $\boldsymbol{F} = \boldsymbol{A}_{\sigma(i)}\boldsymbol{U}$ becomes the zero vector whenever the γth row of $\boldsymbol{A}_{\sigma(i)}$ is the zero vector $(1 \leqslant \gamma \leqslant n)$. Let $g_{\sigma(i)}$ denote the number of nonzero rows of the symmetric data matrix $\boldsymbol{A}_{\sigma(i)}$. Then $\sqrt{f_{\sigma(i)}} \leqslant g_{\sigma(i)} \leqslant \min\{n,\ f_{\sigma(i)}\}$ since $\boldsymbol{A}_{\sigma(i)}$ is symmetric. If we incorporate the sparsity of the matrix $\boldsymbol{F} = \boldsymbol{A}_{\sigma(i)}\boldsymbol{U}$ into our consideration, we replace (10) by

$$nf_{\sigma(i)} + g_{\sigma(i)}n^2 + \sum_{i \leqslant j \leqslant m} f_{\sigma(j)},$$

and (11) by

$$nf_{\sigma(i)} + (g_{\sigma(i)} + 1) \sum_{i \leqslant j \leqslant m} f_{\sigma(j)},$$

respectively. Such a modification is effective when $g_{\sigma(i)} < n$. But we also mention that if $g_{\sigma(i)}$ is sufficiently small (or more precisely, if $g_{\sigma(i)} = O(1)$), formula $\mathcal{F}_3$ looks more efficient than formulae $\mathcal{F}_1$ and $\mathcal{F}_2$ even with such a modification; hence formulae $\mathcal{F}_1$ and $\mathcal{F}_2$ are not likely to be used.

(B) We can evaluate the numbers of additions in formulae $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$ to compute $B_{\sigma(i)\sigma(j)}$ $(i \leqslant j \leqslant m)$. The numbers of additions in formulae $\mathcal{F}_1$ and $\mathcal{F}_2$ coincide with the numbers of multiplications in formulae $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively. See (10) and (11). But the number of additions in formulae $\mathcal{F}_3$ is smaller than the number of multiplication given in (12);

$$\text{"the number of multiplication"} = (2f_{\sigma(i)} + 1) \sum_{i \leqslant j \leqslant m} f_{\sigma(j)}$$

$$> (f_{\sigma(i)} + 1) \sum_{i \leqslant j \leqslant m} f_{\sigma(j)} = \text{"the number of additions"}.$$

We learned through our numerical experiments that formula $\mathcal{F}_3$ works more efficiently than what we expect from our theoretical analysis in Section 4. In fact, we observe a little jump of the curve at $q_2 = 147$ in Fig. 3, and we also observe in Table 4 that formula $\mathcal{F}_3$ works as efficiently as formula $\mathcal{F}_*(1.5)$. These facts may be partially explained by the number of additions which we have not taken account of.

(C) We can combine our method for computing the coefficient matrix $\boldsymbol{B} = \boldsymbol{B}'$ of the system of equations in (6) (or the coefficient matrix $\boldsymbol{B} = \boldsymbol{B}''$ of the system of equations in (7)) in Section 3 with indirect or iterative methods (such as the conjugate direction method) for solving the system of equations $\boldsymbol{Bdy} = \boldsymbol{r}$. In those methods we repeatedly compute $\boldsymbol{u} = \boldsymbol{Bv}$ for different $\boldsymbol{v}$'s in R^m. It should be noted that each element u_i is of the form $u_i = \sum_{1 \leqslant j \leqslant m} B_{ij}v_j$ $(1 \leqslant i \leqslant m)$. Therefore we can compute $\boldsymbol{u} = \boldsymbol{Bv}$ without storing the entire matrix $\boldsymbol{B}$. This advantage of iterative methods is crucial when the number m of equality constraints of the SDP (1) is too large to store the entire matrix $\boldsymbol{B}$ in computer memory. In fact, the conjugate gradient method was applied to large scale SDPs arising from relaxations of the quadratic assignment problem (see, e.g., [10,20]).

(D) Suppose that the data matrices $\boldsymbol{A}_i$ $(0 \leqslant i \leqslant m)$ of the SDP (1) share a common block diagonal matrix structure as follows:

$$\boldsymbol{A}_i = \begin{pmatrix} \boldsymbol{A}_i^0 & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{A}_i^1 & \dots & \boldsymbol{O} \\ \cdot & \cdot & \dots & \cdot \\ \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{A}_i^l \end{pmatrix}. \tag{17}$$

Here $\boldsymbol{A}_i^k$ denotes an $n_k \times n_k$ symmetric matrix $(0 \leqslant k \leqslant l)$. In this case we can represent the coefficient matrix $\boldsymbol{B} = \boldsymbol{B}'$ of the system of equations in (6) (or the coefficient matrix $\boldsymbol{B} = \boldsymbol{B}''$ of the system of equations in (7)) as

$$\boldsymbol{B} = \sum_{0 \leqslant k \leqslant l} \boldsymbol{B}^k \quad \text{and} \quad B_{ij}^k = \boldsymbol{T}^k \boldsymbol{A}_i^k \boldsymbol{U}^k \bullet \boldsymbol{A}_j^k \quad (0 \leqslant k \leqslant l).$$

Therefore we can apply the method given in Section 3 to computation of each $\boldsymbol{B}^k$ separately $(0 \leqslant k \leqslant l)$. Such a block diagonal matrix structure often appears in semidefinite programs arising from the system and control theory. See [4]. Also, when we are given inequality constraints

$$\boldsymbol{A}_i^0 \bullet \boldsymbol{X}^0 \leqslant b_i \ (1 \leqslant i \leqslant m) \quad \text{and} \quad \boldsymbol{X}^0 \succeq \boldsymbol{O},$$

we can convert them to equality constraints

$$\boldsymbol{A}_i \bullet \boldsymbol{X} = b_i \ (1 \leqslant i \leqslant m) \quad \text{and} \quad \boldsymbol{X} \succeq \boldsymbol{O}$$

by defining the matrices $\boldsymbol{A}_i$ $(1 \leqslant i \leqslant m)$ by (17) with

$$l = m, \ n_k = 1 \ (1 \leqslant k \leqslant m) \quad \text{and} \quad \boldsymbol{A}_i^k = \begin{cases} 1 & \text{if } k = i, \\ 0 & \text{otherwise,} \end{cases}$$

where $\boldsymbol{X}$ is a variable symmetric matrix having the same block diagonal structure as $\boldsymbol{A}_i$:

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{X}^0 & \boldsymbol{O} & \dots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{X}^1 & \dots & \boldsymbol{O} \\ \cdot & \cdot & \cdots & \cdot \\ \boldsymbol{O} & \boldsymbol{O} & \dots & \boldsymbol{X}^l \end{pmatrix}.$$

Appendix. Proof of Theorem 1

We first prove two lemmas.

Lemma A.1. *Let $\boldsymbol{\sigma}$ be a permutation of $\{1, 2, \dots, m\}$. Suppose that*

$$f_{\sigma(l)} < f_{\sigma(l+1)}$$

for some $l \in \{1, 2, \dots, m-1\}$. Define the permutation $\boldsymbol{\tau}$ by

$$\tau(i) = \begin{cases} \sigma(i) & \text{if } 1 \leqslant i < l \text{ or } l+1 < i \leqslant m, \\ \sigma(l+1) & \text{if } i = l, \\ \sigma(l) & \text{if } i = l+1. \end{cases}$$

Then $d_*(\boldsymbol{\sigma}) > d_*(\boldsymbol{\tau})$.

Proof. For simplicity of notation, we assume that $\sigma(i) = i$ $(i = 1, 2, \ldots, m)$. We see from the assumption that

$$\tau(i) = \begin{cases} i & \text{if } 1 \leqslant i < l \text{ or } l+1 < i \leqslant m, \\ l+1 & \text{if } i = l, \\ l & \text{if } i = l+1, \end{cases} \tag{A.1}$$

$$f_l < f_{l+1},$$

$$d_{*l}(\boldsymbol{\sigma}) = d_{*i}(\boldsymbol{\tau}) \quad \text{if } 1 \leqslant i < l \text{ or } l+1 < i \leqslant m.$$

Hence it suffices to show that the inequality

$$d_{*l}(\boldsymbol{\sigma}) + d_{*(l+1)}(\boldsymbol{\sigma}) > d_{*l}(\boldsymbol{\tau}) + d_{*(l+1)}(\boldsymbol{\tau}) \tag{A.2}$$

holds under the assumption (A.1). Furthermore we know by definition that

$$d_{ql}(\boldsymbol{\tau}) \geqslant d_{*l}(\boldsymbol{\tau}) \quad \text{for any } q \in \{1,2,3\},$$

$$d_{r(l+1)}(\boldsymbol{\tau}) \geqslant d_{*(l+1)}(\boldsymbol{\tau}) \quad \text{for any } r \in \{1,2,3\}.$$

Therefore inequality (A.2) follows if

$$d_{*l}(\boldsymbol{\sigma}) + d_{*(l+1)}(\boldsymbol{\sigma}) \geqslant d_{ql}(\boldsymbol{\tau}) + d_{r(l+1)}(\boldsymbol{\tau}) \tag{A.3}$$

holds for some $q, r \in \{1,2,3\}$. To derive (A.3) for some $q, r \in \{1,2,3\}$, we take q and r as follows:

(a) If $d_{*l}(\boldsymbol{\sigma}) = d_{1l}(\boldsymbol{\sigma})$ and $d_{*(l+1)}(\boldsymbol{\sigma}) = d_{1(l+1)}(\boldsymbol{\sigma})$ then let $q = 1$ and $r = 1$.

(b) If $d_{*l}(\boldsymbol{\sigma}) = d_{1l}(\boldsymbol{\sigma})$ and $d_{*(l+1)}(\boldsymbol{\sigma}) = d_{2(l+1)}(\boldsymbol{\sigma})$ then let $q = 1$ and $r = 2$.

(c-1) If $d_{*l}(\boldsymbol{\sigma}) = d_{1l}(\boldsymbol{\sigma})$, $d_{*(l+1)}(\boldsymbol{\sigma}) = d_{3(l+1)}(\boldsymbol{\sigma})$ and $2\kappa \sum_{l \leqslant j \leqslant m} f_j \geqslant n$ then let $q = 1$ and $r = 3$.

(c-2) If $d_{*l}(\boldsymbol{\sigma}) = d_{1l}(\boldsymbol{\sigma})$, $d_{*(l+1)}(\boldsymbol{\sigma}) = d_{3(l+1)}(\boldsymbol{\sigma})$ and $2\kappa \sum_{l \leqslant j \leqslant m} f_j < n$ then let $q = 3$ and $r = 3$.

(d) If $d_{*l}(\boldsymbol{\sigma}) = d_{2l}(\boldsymbol{\sigma})$ and $d_{*(l+1)}(\boldsymbol{\sigma}) = d_{1(l+1)}(\boldsymbol{\sigma})$ then let $q = 1$ and $r = 2$.

(e) If $d_{*l}(\boldsymbol{\sigma}) = d_{2l}(\boldsymbol{\sigma})$ and $d_{*(l+1)}(\boldsymbol{\sigma}) = d_{2(l+1)}(\boldsymbol{\sigma})$ then let $q = 2$ and $r = 2$.

(f-1) If $d_{*l}(\boldsymbol{\sigma}) = d_{2l}(\boldsymbol{\sigma})$, $d_{*(l+1)}(\boldsymbol{\sigma}) = d_{3(l+1)}(\boldsymbol{\sigma})$ and $2\kappa \sum_{l \leqslant j \leqslant m} f_j \geqslant n$ then let $q = 2$ and $r = 3$.

(f-2) If $d_{*l}(\boldsymbol{\sigma}) = d_{2l}(\boldsymbol{\sigma})$, $d_{*(l+1)}(\boldsymbol{\sigma}) = d_{3(l+1)}(\boldsymbol{\sigma})$ and $2\kappa \sum_{l \leqslant j \leqslant m} f_j < n$ then let $q = 3$ and $r = 3$.

(g) If $d_{*l}(\boldsymbol{\sigma}) = d_{3l}(\boldsymbol{\sigma})$ and $d_{*(l+1)}(\boldsymbol{\sigma}) = d_{1(l+1)}(\boldsymbol{\sigma})$ then let $q = 1$ and $r = 3$.

(h-1) If $d_{*l}(\boldsymbol{\sigma}) = d_{3l}(\boldsymbol{\sigma})$, $d_{*(l+1)}(\boldsymbol{\sigma}) = d_{2(l+1)}(\boldsymbol{\sigma})$ and $2\kappa f_{l+1} \geqslant n$ then let $q = 2$ and $r = 3$.

(h-2) If $d_{*l}(\boldsymbol{\sigma}) = d_{3l}(\boldsymbol{\sigma})$, $d_{*(l+1)}(\boldsymbol{\sigma}) = d_{2(l+1)}(\boldsymbol{\sigma})$ and $2\kappa f_{l+1} < n$ then let $q = 3$ and $r = 3$.

(i) If $d_{*l}(\boldsymbol{\sigma}) = d_{3l}(\boldsymbol{\sigma})$ and $d_{*(l+1)}(\boldsymbol{\sigma}) = d_{3(l+1)}(\boldsymbol{\sigma})$ then let $q = 3$ and $r = 3$.

In each of the cases, it is easy to derive inequality (A.3). Here we only deal with cases (c-1) and (c-2). Suppose that

$$d_{*l}(\boldsymbol{\sigma}) = d_{1l}(\boldsymbol{\sigma}), \quad d_{*(l+1)}(\boldsymbol{\sigma}) = d_{3(l+1)}(\boldsymbol{\sigma}) \quad \text{and} \quad 2\kappa \sum_{l \leqslant j \leqslant m} f_j \geqslant n.$$

Then

$$\begin{aligned}&(d_{1l}(\boldsymbol{\sigma}) + d_{3(l+1)}(\boldsymbol{\sigma})) - (d_{1l}(\boldsymbol{\tau}) + d_{3(l+1)}(\boldsymbol{\tau}))\\ &\quad = \kappa(f_{l+1} - f_l)\Big(2\kappa \sum_{l \leqslant j \leqslant m} f_j - n\Big) + \kappa(f_{l+1} - f_l) > 0.\end{aligned}$$

Thus we have shown inequality (A.3) in case (c-1). Now assume that

$$d_{*l}(\boldsymbol{\sigma}) = d_{1l}(\boldsymbol{\sigma}), \quad d_{*(l+1)}(\boldsymbol{\sigma}) = d_{3(l+1)}(\boldsymbol{\sigma}) \quad \text{and} \quad 2\kappa \sum_{l \leqslant j \leqslant m} f_j < n.$$

Then

$$\begin{aligned}&(d_{1l}(\boldsymbol{\sigma}) + d_{3(l+1)}(\boldsymbol{\sigma})) - (d_{3l}(\boldsymbol{\tau}) + d_{3(l+1)}(\boldsymbol{\tau}))\\ &\quad = \kappa f_l\Big(n - 2\kappa \sum_{l \leqslant j \leqslant m} f_j\Big) + n^3 + \kappa(f_{l+1} - f_l) > 0.\end{aligned}$$

Thus we have shown inequality (A.3) in case (c-2). We can derive the other cases similarly. The details are omitted. □

Lemma A.2. *Let* $\boldsymbol{\sigma}$ *be a permutation of* $\{1, 2, \ldots, m\}$ *and* $i \in \{1, 2, \ldots, m-1\}$. *Suppose that* $f_{\sigma(i)} \geqslant f_{\sigma(i+1)}$ *for some* $i \in \{1, 2, \ldots, m-1\}$.

(a) *If* $d_{2i}(\boldsymbol{\sigma}) \leqslant d_{1i}(\boldsymbol{\sigma})$ *then* $d_{2(i+1)}(\boldsymbol{\sigma}) < d_{1(i+1)}(\boldsymbol{\sigma})$.

(b) *If* $d_{3i}(\boldsymbol{\sigma}) \leqslant d_{1i}(\boldsymbol{\sigma})$ *then* $d_{3(i+1)}(\boldsymbol{\sigma}) < d_{1(i+1)}(\boldsymbol{\sigma})$.

(c) *If* $d_{3i}(\boldsymbol{\sigma}) \leqslant d_{2i}(\boldsymbol{\sigma})$ *then* $d_{3(i+1)}(\boldsymbol{\sigma}) < d_{2(i+1)}(\boldsymbol{\sigma})$.

Proof. For simplicity of notation, we assume that $\sigma(j) = j$ $(j = 1, 2, \ldots, m)$, and use the notation d_{1j}, d_{2j} and d_{3j} for $d_{1j}(\boldsymbol{\sigma})$, $d_{2j}(\boldsymbol{\sigma})$ and $d_{3j}(\boldsymbol{\sigma})$, respectively.

(a) By (10)′ and (11)′ we see that

$$(d_{1(i+1)} - d_{2(i+1)}) - (d_{1i} - d_{2i}) = n\kappa f_i > 0.$$

Hence $(d_{1i} - d_{2i}) \geqslant 0$ implies $(d_{1(i+1)} - d_{2(i+1)}) > 0$.

(b) By (10)′ and (12)′ we see that

$$(d_{1(i+1)} - d_{3(i+1)})) = n^3 + \kappa f_{i+1}\Big(n - 2\kappa \sum_{i+1 \leqslant j \leqslant m} f_j\Big).$$

Hence, if $(n - 2\kappa \sum_{i+1 \leqslant j \leqslant m} f_j) \geqslant 0$ then $(d_{1(i+1)} - d_{3(i+1)}) > 0$. Now assume that $(n - 2\kappa \sum_{i+1 \leqslant j \leqslant m} f_j) < 0$. Then

$$\begin{aligned}&(d_{1(i+1)} - d_{3(i+1)}) - (d_{1i} - d_{3i})\\ &\quad = 2(\kappa f_i)^2 + \kappa(f_i - f_{i+1})\Big(2\kappa \sum_{i+1 \leqslant j \leqslant m} f_j - n\Big) > 0.\end{aligned}$$

Hence $(d_{1i} - d_{3i}) \geqslant 0$ implies $(d_{1(i+1)} - d_{3(i+1)}) > 0$.

(c) By (11)′ and (12)′ we see that if $n \geqq 2\kappa f_{i+1}$ then

$$(d_{2(i+1)} - d_{3(i+1)}) = n\kappa f_{i+1} + (n - 2\kappa f_{i+1})\kappa \sum_{i+1 \leqq j \leqq m} f_j$$

$$= \Big(n - 2\kappa \sum_{i+1 \leqq j \leqq m} f_j\Big)\kappa f_{i+1} + n\kappa \sum_{i+1 \leqq j \leqq m} f_j > 0.$$

Now assume that $n < 2\kappa f_{i+1}$. Then

$$(d_{2(i+1)} - d_{3(i+1)}) - (d_{2i} - d_{3i})$$

$$= \kappa(f_i - f_{i+1})\Big(2\kappa \sum_{i+1 \leqq j \leqq m} f_j - n\Big) + \kappa f_i(2\kappa f_i - n) > 0.$$

Hence $(d_{2i} - d_{3i}) \geqq 0$ implies $(d_{2(i+1)} - d_{3(i+1)}) > 0$. □

Now we are ready to prove Theorem 1. Let $\boldsymbol{\sigma}^*$ be a minimizer of $d_*(\boldsymbol{\sigma})$ over $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$. Assume on the contrary that $\boldsymbol{\sigma}^*$ does not satisfy the relation (15). Then there is an index $l \in \{1, 2, \ldots, m-1\}$ such that $f_{\sigma^*(l)} < f_{\sigma^*(l+1)}$. Hence, applying Lemma A.1, we can find a $\boldsymbol{\tau} \in \boldsymbol{\Sigma}$ such that $d_*(\boldsymbol{\tau}) < d_*(\boldsymbol{\sigma}^*)$. This is a contradiction. Thus we have shown that $\boldsymbol{\sigma}^* \in \boldsymbol{\Sigma}$ satisfies (15).

Now suppose that $\boldsymbol{\sigma}^* \in \boldsymbol{\Sigma}$ satisfies (15). Since the number of elements in $\boldsymbol{\Sigma}$ is finite, $d_* : \boldsymbol{\Sigma} \to R$ attains its minimum at some $\boldsymbol{\tau} \in \boldsymbol{\Sigma}$, which must satisfy the relation

$$f_{\tau(1)} \geqq f_{\tau(2)} \geqq \cdots \geqq f_{\tau(m)}. \tag{15′}$$

Since the relations (15) and (15)′ imply that $f_{\sigma(i)} = f_{\tau(i)}$ for every $i \in \{1, 2, \ldots, m\}$, we obtain that $d_*(\boldsymbol{\sigma}) = d_*(\boldsymbol{\tau})$. Therefore $\boldsymbol{\sigma}^* \in \boldsymbol{\Sigma}$ is a minimizer of $d_*(\boldsymbol{\sigma})$ over all $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$.

Assertion (ii) of the theorem follows from Lemma A.2. □

References

[1] F. Alizadeh, J.-P.A. Haeberly and M.L. Overton, Primal-dual interior-point methods for semidefinite programming, Working Paper, 1994.

[2] F. Alizadeh, J.-P.A. Haeberly and M.L. Overton, Primal-dual interior-point methods for semidefinite programming: convergence rates, stability and numerical results, *SIAM Journal on Optimization*, to appear.

[3] N. Brixius, F.A. Potra and R. Sheng, Solving semidefinite programming in mathematica, Reports on Computational Mathematics, No. 97/1996, Dept. of Mathematics, University of Iowa, 1996; Available at http://www.cs.uiowa.edu/brixius/sdp.html.

[4] S. Boyd, L.E. Ghaoui, E. Feron and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory* (SIAM, Philadelphia, PA, 1994).

[5] K. Fujisawa, M. Kojima and K. Nakata, SDPA (Semidefinite Programming Algorithm) – User's Manual –, Tech. Rept. B-308, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152, Japan, 1995, revised 1996; Available via anonymous ftp at ftp.is.titech.ac.jp in pub/OpRes/software/SDPA.

[6] C. Helmberg, F. Rendl, R.J. Vanderbei and H. Wolkowicz, An interior-point method for semidefinite programming, *SIAM Journal on Optimization* 6 (1996) 342–361.

[7] M. Kojima, M. Shida and S. Shindoh, Local convergence of predictor-corrector infeasible-interior-point algorithms for SDPs and SDLCPs, *Mathematical Programming*, to appear.
[8] M. Kojima, S. Shindoh and S. Hara, Interior-point methods for the monotone semidefinite linear complementarity problems, *SIAM Journal on Optimization* 7 (1997) 86–125.
[9] C.-J. Lin and R. Saigal, A predictor-corrector method for semi-definite linear programming, Working paper, Dept. of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI, 1995.
[10] C.-J. Lin and R. Saigal, Predictor corrector methods for semidefinite programming, Informs Atlanta Fall Meeting, 1996.
[11] Z.-Q. Luo, J.F. Sturm and S. Zhang, Superlinear convergence of a symmetric primal–dual path following algorithm for semidefinite programming, *SIAM Journal on Optimization*, to appear.
[12] R.D.C. Monteiro, Primal—dual path following algorithms for semidefinite programming, *SIAM Journal on Optimization*, to appear.
[13] Yu.E. Nesterov and M.J. Todd, Self-scaled cones and interior-point methods in nonlinear programming, Working Paper, CORE, Catholic University of Louvain, Louvain-la-Neuve, Belgium, 1994; also in: *Mathematics of Operations Research*, to appear.
[14] Yu.E. Nesterov and M.J. Todd, Primal–dual interior-point methods for self-scaled cones, *SIAM Journal on Optimization*, to appear.
[15] S.Poljak, F.Rendl and H.Wolkowicz, A recipe for semidefinite relaxation for (0,1) quadratic programming, *Journal of Global Optimization* 7 (1995) 51–73.
[16] F.A. Potra and R. Sheng, Superlinear convergence of infeasible-interior-point algorithms for semidefinite programming, Dept. of Mathematics, University of Iowa, Iowa City, IA, 1996.
[17] M.J. Todd, K.C. Toh and R.H. Tütüncü, On the Nesterov–Todd direction in semidefinite programming, Tech. Rept., School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, 1996.
[18] K.C. Toh, M.J. Todd and R.H. Tütüncü, SDPT3 – A MATLAB software package for semidefinite programming, Dept. of Mathematics, National University of Singapore, Singapore, 1996; Available at http://www.math.nus.sg/mattohkc/index.html.
[19] Y. Zhang, On extending primal–dual interior-point algorithms from linear programming to semidefinite programming, *SIAM Journal on Optimization*, to appear.
[20] Q. Zhao, S.E. Karisch, F. Rendl and H. Wolkowicz, Semidefinite programming relaxations for the quadratic assignment problem, CORR Rept. 95-27, Dept. of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada, 1996.