<table>
<thead>
<tr>
<th>Title</th>
<th>Solving Sparse Semidefinite Programs by Matrix Completion (Part II) (Mathematical Science of Optimization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Fukuda, Mituhiro; Nakata, Kazuhide; Fujisawa, Katsuki; Kojima, Masakazu; Murota, Kazuo</td>
</tr>
<tr>
<td>Citation</td>
<td>数理解析研究所講究録 (2000), 1174: 130-137</td>
</tr>
<tr>
<td>Issue Date</td>
<td>2000-10</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/2433/64466">http://hdl.handle.net/2433/64466</a></td>
</tr>
<tr>
<td>Type</td>
<td>Departmental Bulletin Paper</td>
</tr>
<tr>
<td>Textversion</td>
<td>publisher</td>
</tr>
</tbody>
</table>
Solving Sparse Semidefinite Programs by Matrix Completion (part II)

東京大学 工学系研究科 中田 和秀 (Kazuhide Nakata)
京都大学 工学研究科 藤沢 克樹 (Katsuki Fujisawa)
東京工業大学 情報理工学研究科 福田 光浩 (Mituhiro Fukuda)
東京工業大学 情報理工学研究科 小島 政和 (Masakazu Kojima)
京都大学 数理解析研究所 室田 一雄 (Kazuo Murota)

This article succeeds the previous article (Solving Sparse Semidefinite Programs by Matrix Completion (part I). In this article, we will propose a primal-dual interior-point method based on the sparse factorization formula which arises from positive definite matrix completion [3]. We call this method the completion method. Next we will present some numerical results which compare with the original method, the conversion method and the completion method.

1 Completion Method

One disadvantage of the conversion method in the previous article is an increase in the number of equality constraints. In this section, we propose a primal-dual interior-point method based on positive semidefinite matrix completion which we can apply directly to the original SDP without adding any equality constraints.

1.1 Search Direction

Various search directions [1, 4, 5, 6, 7, 8, 9, 10] have been proposed so far for primal-dual interior-point methods. Among others, we restrict ourselves to the HRVW/KSH/M search direction [4, 6, 7].

Let $(\overline{X}, \overline{Y}, \overline{z})$ be a point obtained at the $k$th iteration of a primal-dual interior-point method using the HRVW/KSH/M search direction ($k \geq 1$) or given initially ($k = 0$). We assume that $\overline{X} \in S^{n}_{++}(F, ?)$ and $\overline{Y} \in S^{n}_{++}(E, 0)$.

In order to compute the HRVW/KSH/M search direction, we use the whole matrix values for both $\overline{X} \in S^{n}_{++}(F, ?)$ and $\overline{Y} \in S^{n}_{++}(E, 0)$, so that we need to make a positive definite matrix completion of $\overline{X} \in S^{n}_{++}(F, ?)$. Let $\hat{X} \in S^{n}_{++}$ be the maximum-determinant positive definite matrix completion of $\overline{X} \in S^{n}_{++}(F, ?)$. Then we compute the HRVW/KSH/M search direction $(dX, dY, dz)$ by solving the system of linear equations

$$
\begin{align*}
A_{p} \cdot dX &= g_{p} \quad (p = 1, 2, \ldots, m), \quad dX \in S^{n} \\
\sum_{p=1}^{m} A_{p}dX_{p} + dY &= H, \quad dY \in S^{n}(E, 0), \quad dz \in R^{m}, \\
\overline{dX} \overline{Y} + \hat{X} dY &= K, \quad dX = (\overline{dX} + \overline{dX}^{T})/2,
\end{align*}
$$

(1)
where \( g_p = b_p - A_p \cdot \hat{X} \in \mathbb{R} \) (\( p = 1, 2, \ldots, m \)) (the primal residual), \( H = A_0 - \sum_{p=1}^{m} A_p \tilde{x}_p - \hat{Y} \in S^n(E, 0) \) (the dual residual), \( K = \mu I - \hat{X}\hat{Y} \) (an \( n \times n \) constant matrix), and \( \tilde{dX} \) denotes an \( n \times n \) auxiliary matrix variable. The search direction parameter \( \mu \) is usually chosen to be \( \beta \hat{X} \cdot \hat{Y}/n \) for some \( \beta \in [0, 1] \). We can reduce the system of linear equations (1) to

\[
\begin{align*}
Bdz &= s, \quad dY = H - \sum_{p=1}^{m} A_p dz_p, \\
\tilde{dX} &= (K - \hat{X}dY)^{-1}, \quad dX = (\tilde{dX} + \tilde{dX}^T)/2,
\end{align*}
\]

(2)

where

\[
B_{pq} = \text{Trace } A_p \hat{X} A_q \hat{Y}^{-1} \quad (p = 1, 2, \ldots, m, \quad q = 1, 2, \ldots, m),
\]

\[
s_p = g_p - \text{Trace } A_p (K - \hat{X}H)\hat{Y}^{-1} \quad (p = 1, 2, \ldots, m).
\]

Note that \( B \) is a positive definite symmetric matrix.

As we have seen in the previous article, The maximum-determinant positive definite matrix completion \( \hat{X} \) of \( \hat{X} \in S^n_++(F, ?) \) can be expressed in terms of the sparse factorization formula

\[
\hat{X} = W^{-T}W^{-1},
\]

(3)

where \( W \) is a lower triangular matrix which enjoys the same sparsity structure as \( \hat{X} \). Also \( \hat{Y} \in S^n_++(E, 0) \) is factorized as \( \hat{Y} = NN^T \) without any fill-in's except for entries in \( F \setminus E \), where \( N \) is a lower triangular matrix. We can effectively utilize these factorizations of \( \hat{X} \) and \( \hat{Y} \) for the computation of the search direction \( (dX, dY, dz) \). In particular, the coefficients \( B_{pq} \) (\( p = 1, 2, \ldots, m, \quad q = 1, 2, \ldots, m \)) in the system (2) of linear equations is computed by

\[
B_{ij} = \sum_{k=1}^{n} (W^{-T}W^{-1}e_k)^T A_j(N^{-T}N^{-1}[A_i]_{sk}) \quad (i, j = 1, 2, \ldots, m),
\]

\[
s_i = b_i + \sum_{k=1}^{n} (W^{-T}W^{-1}e_k)^T H(N^{-T}N^{-1}[A_i]_{sk}) - \mu e_k^T N^{-T}N^{-1}[A_i]_{sk} \quad (i = 1, 2, \ldots, m)
\]

Here \([A_i]_{sk}\) denote the \( k \)th column of the matrix \( A_i \). Since we just store the sparse and lower triangular matrix \( W \) and \( N \) instead of \( \hat{X} \) and \( \hat{Y} \), we can not use the current computational formulas. Instead, we compute \( B, \tilde{dX} \) and \( s \) based on matrix-vector multiplications. For instance, to determine the the product \( h = W^{-1}v \), we solve the linear system \( Wh = v \), where \( W \) is lower triangular and sparse.

### 1.2 Step Size

As we will see below in the computation of the step length and the next iterate, we need the partial symmetric matrix with entries \([dX]_{ij}\) specified in \( F \), but not the whole search direction matrix \( dX \in S^n \) in the primal space (hence the partial symmetric matrix with entries \([d\tilde{X}]_{ij}\) specified in \( F \) but not the whole matrix \( d\tilde{X} \)). Hence it is possible to carry out all the matrix computation above using only partial matrices with entries specified in \( F \). Therefore we can expect to save both CPU time and memory in our computation of the search direction. To clarify the distinction between the whole primal search direction matrix \( dX \in S^n \) and the corresponding partial symmetric matrix
with entries specified in $F$ in the discussions below, we use the notation $d\hat{X}$ for the former whole matrix in $S^n$ and $d\overline{X}$ for the latter partial symmetric matrix in $S^n(F,:)$. Now, supposing that we have computed the HRVW/KSH/M search direction $(d\hat{X}, d\overline{Y}, dz) \in S^n \times S^n(E,0) \times \mathbb{R}^m$, and we describe how to compute a step length $\alpha > 0$ and the next iterate $(X', Y', z') \in S^n \times S^n(E,0) \times \mathbb{R}^m$. Usually we compute the maximum $\bar{\alpha}$ of $\alpha$’s satisfying

$$\hat{X} + \alpha d\hat{X} \in S_+^n \quad \text{and} \quad \hat{Y} + \alpha d\hat{Y} \in S_+^n,$$

and let $(X', Y', z') = (\hat{X}, \hat{Y}, \hat{z}) + \gamma \bar{\alpha}(d\hat{X}, d\overline{Y}, dz)$ for some $\gamma \in (0,1)$. Then $X' \in S_+^n$ and $Y' \in S_+^n(E,0)$. The computation of $\bar{\alpha}$ is necessary to know how long we can take the step length along the search direction $(d\hat{X}, d\overline{Y}, dz)$. The computation of $\bar{\alpha}$ is usually carried out by calculating the minimum eigenvalues of the matrices

$$\hat{M}^{-1}d\hat{X}\hat{M}^{-T} \quad \text{and} \quad N^{-1}d\overline{Y}N^{-T},$$

where $\hat{X} = \hat{M}\hat{M}^T$ and $\hat{Y} = NN^T$ denote the factorizations of $\hat{X}$ and $\hat{Y}$, respectively.

Instead of (4), we propose to employ

$$X_{C_rC_r} + \alpha dX_{C_rC_r} \in S_{++}^{C_r} \quad \text{and} \quad Y + \alpha dY \in S_+^n(E,0).$$

(5)

Recall that $\{C_r \subseteq V : r = 1, 2, \ldots, \ell\}$ denotes the family of maximal cliques of $G(V, F^n)$ and $\ell \leq n$. Let $\bar{\alpha}$ be the maximum of $\alpha$’s satisfying (5), and let

$$(X', Y', z') = (\overline{X}, \overline{Y}, \overline{z}) + \gamma \alpha(d\overline{X}, d\overline{Y}, dz) \in S^n(F,:) \times S_{++}^n(E,0) \times \mathbb{R}^m$$

for some $\gamma \in (0,1)$. $X' \in S^n(F,:)$ has a positive definite matrix completion, so that the point $(X', Y', z') \in S_+^n(F,:) \times S_+^n(E,0) \times \mathbb{R}^m$ can be the next iterate. In this case, the computation of $\bar{\alpha}$ is reduced to the computation of the minimum eigenvalues of the matrices

$$\hat{M}_{r}^{-1}d\hat{X}_{C_rC_r}\hat{M}_{r}^{-T} \quad \text{and} \quad N^{-1}d\overline{Y}N^{-T},$$

where $\hat{X}_{C_rC_r} = \hat{M}_{r}\hat{M}_{r}^T$ denotes a factorization of $\hat{X}_{C_rC_r}$ ($r = 1, 2, \ldots, \ell$). Thus the computation of the minimum eigenvalue of $\hat{M}_{r}^{-1}X_{C_rC_r}\hat{M}_{r}^{-T}$ has been replaced by the computation of the minimum eigenvalues of $\ell$ smaller submatrices $\hat{M}_{r}^{-1}d\hat{X}_{C_rC_r}\hat{M}_{r}^{-T} \quad \text{for} \quad r = 1, 2, \ldots, \ell$. On the other hand, when we compute the minimum eigenvalue of $N^{-1}d\overline{Y}N^{-T}$, we can compute it easily by Lanczos methods, because $N$ and $d\overline{Y}$ are sparse and we can compute the multiplication between $N^{-1}d\overline{Y}N^{-T}$ and vector $\nu$ using technic we mentioned.

### 2 Numerical Experiments

All numerical experiments in this section were executed on DEC Alpha Station (CPU Alpha 21164-600MHz with 1024MB). For the original method, we used the SDPA 5.0[2]. For the conversion method, we first converted the SDPs into block diagonal SDPs and solved them by the SDPA[2]. For the completion method, we used a new software called SDPA-C which we incorporated the sparse factorization formula.
2.1 Norm Minimization Problems

Let $F_i \in \mathbb{R}^{q \times r}$ ($0 \leq i \leq t$). The norm minimization problem is defined as:

$$
\text{minimize } \left\| F_0 + \sum_{i=1}^{t} F_i z_i \right\| \\
\text{subject to } z_i \in \mathbb{R} \ (1 \leq i \leq t).
$$

We can reduce this problem to an SDP:

$$
\begin{align*}
\text{maximize} & \quad -z_{t+1} \\
\text{subject to} & \quad \sum_{i=1}^{t} \left( \begin{array}{cc} O & F_i^T \\ F_i & O \end{array} \right) z_i + \left( \begin{array}{cc} I & O \\ O & I \end{array} \right) z_{t+1} + \left( \begin{array}{cc} O & F_0^T \\ F_0 & O \end{array} \right) \in S_+^{q+r}.
\end{align*}
$$

Table 1 shows the performance comparisons among the original method (SDPA), the conversion method applied before solving with SDPA, and the completion method. As $q$ becomes larger, the aggregate sparsity pattern and extended sparsity pattern becomes denser.

<table>
<thead>
<tr>
<th>$q$</th>
<th>$n$</th>
<th>$m$</th>
<th>$CPU$ (s)</th>
<th>$memory$ (MB)</th>
<th>$CPU$ (s)</th>
<th>$memory$ (MB)</th>
<th>$CPU$ (s)</th>
<th>$memory$ (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000</td>
<td>10</td>
<td>3492.0</td>
<td>321</td>
<td>4.7</td>
<td>11.3</td>
<td>100.0</td>
<td>5.18</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>10</td>
<td>4263.0</td>
<td>321</td>
<td>9.7</td>
<td>16.0</td>
<td>157.7</td>
<td>5.98</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
<td>10</td>
<td>5387.0</td>
<td>321</td>
<td>28.4</td>
<td>26.9</td>
<td>378.5</td>
<td>10.2</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>10</td>
<td>6941.4</td>
<td>321</td>
<td>86.4</td>
<td>42.6</td>
<td>696.1</td>
<td>17.8</td>
</tr>
<tr>
<td>20</td>
<td>1000</td>
<td>10</td>
<td>7233.3</td>
<td>321</td>
<td>326.6</td>
<td>84.1</td>
<td>2484.1</td>
<td>44.9</td>
</tr>
<tr>
<td>50</td>
<td>1000</td>
<td>10</td>
<td>7844.1</td>
<td>321</td>
<td>2195.3</td>
<td>192.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the case of norm minimization problems, the conversion method is better than the other methods.

2.2 Quadratic Programs with Box Constraints

Let $Q \in S^n$ and $q \in \mathbb{R}^n$. The quadratic program with box constraints is defined as:

$$
\begin{align*}
\text{minimize} & \quad \frac{1}{2} x^T Q x + q^T x \\
\text{subject to} & \quad -1 \leq x_i \leq 1 \ (i = 1, 2, \ldots, n).
\end{align*}
$$
We have the following semidefinite programming relaxation of the above problem

\[
\text{minimize} \quad \frac{1}{2} \begin{pmatrix} 0 & q^T & 0^T \\ q & Q & O \\ 0 & O & O \end{pmatrix} \cdot X \\
\text{subject to} \quad \begin{pmatrix} 1 & 0^T & 0^T \\ 0 & O & O \\ 0 & O & O \end{pmatrix} \cdot X = 1, \\
\begin{pmatrix} 0 & E_{ii} & O \\ 0 & O & E_{ii} \end{pmatrix} \cdot X = 1 \quad (i = 1, 2, \ldots, n), \quad X \in S_{+}^{2n+1}.
\]

Here \( E_{ii} \) denotes the \( n \times n \) symmetric matrix with \((i, i)\)th element 1 and all others 0.

Table 2 shows the performance comparisons of these methods applied to this particular class of problems. \( \alpha \) denotes the sparsity of the objective function matrix. The entry “1001,1000” in the \( n \) column means that the primal matrix variable \( X \) has block matrices of sizes 1001×1001 and 1000×1000.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( n )</th>
<th>( m )</th>
<th>original</th>
<th>conversion</th>
<th>completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (s)</td>
<td>memory (MB)</td>
<td>CPU (s)</td>
<td>memory (MB)</td>
<td>CPU (s)</td>
<td>memory (MB)</td>
</tr>
<tr>
<td>0.0020</td>
<td>1001,1000</td>
<td>1001</td>
<td>3782.2</td>
<td>316</td>
<td>1153.8</td>
</tr>
<tr>
<td>0.0025</td>
<td>1001,1000</td>
<td>1001</td>
<td>3771.0</td>
<td>316</td>
<td>1681.6</td>
</tr>
<tr>
<td>0.0030</td>
<td>1001,1000</td>
<td>1001</td>
<td>3648.4</td>
<td>316</td>
<td>2179.4</td>
</tr>
<tr>
<td>0.0035</td>
<td>1001,1000</td>
<td>1001</td>
<td>3634.6</td>
<td>316</td>
<td>2561.7</td>
</tr>
<tr>
<td>0.0040</td>
<td>1001,1000</td>
<td>1001</td>
<td>3629.3</td>
<td>316</td>
<td>3014.8</td>
</tr>
</tbody>
</table>

In the case of SDP relaxations of quadratic programs with box constraints, the completion method is better than the other methods.

2.3 Max-cut Problems over Lattice Graphs

Let \( G = (V, E) \) be a lattice graph which is of size \( k1 \times k2 \) with a vertex set \( V = \{1, 2, \ldots, n\} \), and an edge set \( E = \{(i, j) : i, j \in V, i < j\} \). We assign a weight \( C_{ij} = C_{ji} \) to each edge \((i, j) \in E\). The maximum cut problem is to find a partition \((L, R)\) of \( V \) that maximizes the cut \( c(L, R) = \Sigma_{i \in L, j \in R} C_{ij} \). Introducing a variable vector \( u \in \mathbb{R}^n \), we can reformulate the problem as a nonconvex quadratic program:

\[
\text{maximize} \quad \frac{1}{2} \sum_{i < j} C_{ij}(1 - u_i u_j) \quad \text{subject to} \quad u_i^2 = 1 \quad (1 \leq i \leq n).
\]

Here each feasible solution \( u \in \mathbb{R}^n \) of this problem corresponds to a cut \((L, R)\) with \( L = \{i \in V : u_i = -1\} \) and \( R = \{i \in V : u_i = 1\} \). If we define \( C \) to be the \( n \times n \) symmetric matrix with elements \( C_{ji} = C_{ij} \) \((i, j) \in E\) and \( C_{ij} = 0 \) \(((i, j) \notin E)\), and the \( n \times n \) symmetric matrix \( A_0 \in \mathbb{S}^n \)
by \( A_0 = \text{diag}(Ce) - C \), where \( e \in \mathbb{R}^n \) denotes the vector of ones and \( \text{diag}(Ce) \) the diagonal matrix of the vector \( Ce \in \mathbb{R}^n \), we can obtain the following semidefinite programming relaxation of the maximum cut problem:

\[
\begin{align*}
\text{minimize} & \quad -A_0 \bullet X \\
\text{subject to} & \quad E_{ii} \bullet X = 1/4 (1 \leq i \leq n), \quad X \in S_+^n.
\end{align*}
\]

Table 3 compares the three algorithms for this problem. As \( k_1 \) becomes larger, the aggregate sparsity patterns remain sparse, but the extended sparsity pattern becomes denser.

<table>
<thead>
<tr>
<th>( k_1 \times k_2 )</th>
<th>( n )</th>
<th>( m )</th>
<th>CPU (s)</th>
<th>memory (MB)</th>
<th>CPU (s)</th>
<th>memory (MB)</th>
<th>CPU (s)</th>
<th>memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 \times 500</td>
<td>1000</td>
<td>1000</td>
<td>2984.2</td>
<td>315</td>
<td>133.0</td>
<td>53.0</td>
<td>183.7</td>
<td>15.5</td>
</tr>
<tr>
<td>4 \times 250</td>
<td>1000</td>
<td>1000</td>
<td>2971.5</td>
<td>315</td>
<td>168.5</td>
<td>73.3</td>
<td>213.7</td>
<td>16.9</td>
</tr>
<tr>
<td>5 \times 200</td>
<td>1000</td>
<td>1000</td>
<td>2815.1</td>
<td>315</td>
<td>184.7</td>
<td>83.6</td>
<td>233.1</td>
<td>17.8</td>
</tr>
<tr>
<td>8 \times 125</td>
<td>1000</td>
<td>1000</td>
<td>2972.6</td>
<td>315</td>
<td>221.5</td>
<td>85.7</td>
<td>256.6</td>
<td>18.8</td>
</tr>
<tr>
<td>10 \times 100</td>
<td>1000</td>
<td>1000</td>
<td>2835.9</td>
<td>315</td>
<td>263.2</td>
<td>96.8</td>
<td>320.3</td>
<td>19.3</td>
</tr>
<tr>
<td>20 \times 50</td>
<td>1000</td>
<td>1000</td>
<td>3125.2</td>
<td>315</td>
<td>730.5</td>
<td>118.0</td>
<td>347.8</td>
<td>21.9</td>
</tr>
<tr>
<td>25 \times 40</td>
<td>1000</td>
<td>1000</td>
<td>3118.8</td>
<td>315</td>
<td>804.9</td>
<td>180.0</td>
<td>337.7</td>
<td>21.8</td>
</tr>
</tbody>
</table>

For the semidefinite programming relaxation of maximum cut problems over lattice graphs, the conversion and the completion methods are better.

### 2.4 Semidefinite Programming Relaxation of Graph-partition Problem

Let \( G = (V, E) \) be a lattice graph which is of size \( k_1 \times k_2 \) with a vertex set \( V = \{1, 2, \ldots, n\} \), and an edge set \( E = \{(i,j) : i, j \in V, i < j\} \). We assign a weight \( C_{ij} = C_{ji} \) to each edge \((i,j) \in E\). We assume that \( n \) is an even number. The graph partition problem is to find a uniform partition \((L, R)\) of \( V \), i.e., a partition \((L, R)\) of \( V \) with the same cardinality \(|L| = |R| = n/2\), that minimizes the cut \( c(L, R) = \sum_{i \in L, j \in R} C_{ij} \). This problem is formulated as a nonconvex quadratic program:

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \sum_{i<j} C_{ij}(1 - u_i u_j) \\
\text{subject to} & \quad (\sum_{i=1}^{n} u_i)^2 = 0, \quad u_i^2 = 1 (1 \leq i \leq n).
\end{align*}
\]

As in the maximum cut problem, we can derive a semidefinite programming relaxation of the graph partition problem:

\[
\begin{align*}
\text{minimize} & \quad A_0 \bullet X \\
\text{subject to} & \quad E_{ii} \bullet X = 1/4 (1 \leq i \leq n), \quad E \bullet X = 0, \quad X \in S_+^n.
\end{align*}
\]

Here \( A_0 \) and \( E_{ii} \) \((1 \leq i \leq n)\) are the same matrices as in the previous section, and \( E \) denotes the \( n \times n \) matrix with all elements 1. Table 4 compares the three algorithms for this problem. As \( k_1 \)
Table 4: Numerical results on relaxation of the graph partition problems

<table>
<thead>
<tr>
<th>$k_1 \times k_2$</th>
<th>n</th>
<th>m</th>
<th>original times (s)</th>
<th>original memory (MB)</th>
<th>conversion times (s)</th>
<th>conversion memory (MB)</th>
<th>completion times (s)</th>
<th>completion memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x 500</td>
<td>1000</td>
<td>1001</td>
<td>4065.7</td>
<td>315</td>
<td>221.6</td>
<td>53.1</td>
<td>439.9</td>
<td>18.0</td>
</tr>
<tr>
<td>4 x 250</td>
<td>1000</td>
<td>1001</td>
<td>4073.1</td>
<td>315</td>
<td>212.6</td>
<td>74.8</td>
<td>563.0</td>
<td>22.8</td>
</tr>
<tr>
<td>5 x 200</td>
<td>1000</td>
<td>1001</td>
<td>4065.3</td>
<td>315</td>
<td>511.7</td>
<td>125</td>
<td>667.7</td>
<td>26.8</td>
</tr>
<tr>
<td>8 x 125</td>
<td>1000</td>
<td>1001</td>
<td>2843.1</td>
<td>315</td>
<td>1341.2</td>
<td>206</td>
<td>806.8</td>
<td>25.4</td>
</tr>
<tr>
<td>10 x 100</td>
<td>1000</td>
<td>1001</td>
<td>4065.7</td>
<td>315</td>
<td>740.3</td>
<td>168</td>
<td>892.7</td>
<td>27.2</td>
</tr>
<tr>
<td>20 x 50</td>
<td>1000</td>
<td>1001</td>
<td>4064.5</td>
<td>315</td>
<td>2481.5</td>
<td>315</td>
<td>1323.0</td>
<td>31.6</td>
</tr>
<tr>
<td>25 x 40</td>
<td>1000</td>
<td>1001</td>
<td>3912.3</td>
<td>315</td>
<td>3918.0</td>
<td>315</td>
<td>1333.2</td>
<td>34.1</td>
</tr>
</tbody>
</table>

becomes larger, the aggregate sparsity patterns remain sparse, but the extended sparsity pattern becomes denser.

For the semidefinite programming relaxation of graph partition problems over lattice graphs, the conversion and the completion methods are better.

3 Concluding Remarks

In this article, we explained the mechanisms of the completion method. Next we presented some numerical results which compared the original method, the conversion method and the completion method. As a result, we confirmed the effectiveness of the conversion method and the completion method.

References


