情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-MPS-79 No.2
2010/7/12

変形均衡割当て問題 — 2次元ベクトルの場合 —

嘉村友作[†1]　中森眞理雄[†1]

辺の重みが m 次元ベクトルの場合の均衡割当て問題を考える．n 個のベクトルを元とする 2 つの集合を A,B とする．A の元と B の元とを対応づける 1 : 1 対応の中で，対応づけられるベクトルの成分の和の最大値と最小値の差が最小になる 1 : 1 対応 π を見つける問題を考える．このような問題は，半導体露光装置の最適なレンズ組合せ法を考えるときに生じる．2 つの近似アルゴリズムを提案し，提案するアルゴリズムによる数値実験の結果を示す.

Modified Balanced Assignment Problem — 2 Dimensional Case —

Yuusaku Kamura[†1] and Mario Nakamori[†1]

We extend the well known balanced assignment problem to the case where costs are expressed by m-dimensional vectors. If $m = 1$, the problem is the balanced assignment problem. The formal description of our problem is as follows: given two sets A and B of vectors, find the one to one correspondence π between A and B such that the difference of the maximum and the minimum components of the sum of vectors under π is the minimum. Such a type of problem often occurs in combining lenses of semiconductor exposure systems. In the present paper we propose two approximate algorithms for quasi optimal solution with results of numerical experiments.

Keywords: Assignment problem, Balanced assignment problem, Optimal combination

†1 東京農工大学
Tokyo A&T University

1. Introduction

Suppose we have n suppliers $u_1, u_2, \ldots, u_n$ (to be denoted by a set A) and n customers $v_1, v_2, \ldots, v_n$ (to be denoted by B). If supplier u_i and customer v_j are chosen, the cost is c_{ij}. We are going to determine a one to one correspondence $\pi : A \to B$ under an appropriate objective function. This is the well known assignment problem.

Since making a one to one correspondence $\pi : A \to B$ is equivalent to permutating the set $\{1, 2, \ldots, n\}$, we hereafter call one to one correspondence as permutation.

The assignment problem has various versions with respect to the objectives[1)]. If we are going to minimize the total sum of the cost $\sum_{i=1}^{n} c_{i\pi(i)}$, the problem is called the *linear sum assignment problem*, and there have been proposed many efficient algorithms[2)].

If we are going to minimize the maximum cost of the corresponded pair, the problem is called the *bottleneck assignment problem*[3)]. The objective is $\min_\pi \max_{1\le i\le n} c_{i\pi(i)}$. Also, polynomial time algorithms have been proposed for the bottleneck assignment problem.

If we are going to minimize the difference of the maximum cost and the minimum cost of the corresponded pair, the problem is called *balanced assignment problem*[4)]. The objective is $\min_\pi\{\max_{1\le i\le n} c_{i\pi(i)} - \min_{1\le i\le n} c_{i\pi(i)}\}$. Again, polynomial time algorithms have been proposed for the balanced assignment problem. In the following we abbreviate the balanced assignment problem as **BOP**. In the present paper we extend the balanced assignment problem to the case that costs are multidimensional, i.e., vectors. Also, we assume that cost vector $\boldsymbol{c}_{ij}$ is represented as a sum of the supplier vector $\boldsymbol{a}_i$ and the customer vector $\boldsymbol{b}_j$.

A formal description of our problem is as follows. Let A and B be sets of m dimensional vectors. We denote each element of A by $\boldsymbol{a}_i = (a_i^{(1)}, a_i^{(2)}, \ldots, a_i^{(m)})$ and each element of B by $\boldsymbol{b}_i = (b_i^{(1)}, b_i^{(2)}, \ldots, b_i^{(m)})$.

We assume that $a_i^{(1)}, a_i^{(2)}, \ldots, a_i^{(m)}$ and $b_i^{(1)}, b_i^{(2)}, \ldots, b_i^{(m)}$ are nonnegative.

We define sum of vectors $\boldsymbol{a}_i + \boldsymbol{b}_j$ as

$$\boldsymbol{a}_i + \boldsymbol{b}_j = (a_i^{(1)} + b_j^{(1)}, a_i^{(2)} + b_j^{(2)}, \ldots, a_i^{(m)} + b_j^{(m)}).$$

Let us consider the following problem: find a permutaion π of a set $\{1, 2, \ldots, n\}$ such

Vol.2010-MPS-79 No.2
2010/7/12

that

$$T_m(\pi) = \max_i \left\{ a_i^{(k)} + b_{\pi(i)}^{(k)} \right\} - \min_i \left\{ a_i^{(k)} + b_{\pi(i)}^{(k)} \right\}$$

is the minimum.

Such a problem appears when we combine lenses for a semiconductor manufacturing system. This system includes many lenses (about 30) for exposing the circuit pattern on a silicon wafer. Although, these lenses are manufactured very precisely, each individual has its own error expressed by a vector. This error vector is high dimensional (more than 300), and it is required to minimize the maximum component of the combined error vector. In the present paper we simplify the problem such that the system includes only 2 lenses. Thus, we have n individuals for lens position A and n individuals for lens position B. The above vectors $\boldsymbol{a}_i$ and $\boldsymbol{b}_j$ are the error vector of the individual i for position A and that of the individual j for position B. The sum vector $\boldsymbol{a}_i + \boldsymbol{b}_j$ is the combined error. In our former research[5),6),7)] we considered the problem of making n sets of lenses such that the worst combined error is the minimum. The above one to one correspondence π stands for the combination.

It is sometimes required to minimize the dispersion of the errors. Thus in the present paper we are going to extend the balanced assignment problem to multidimensional case.

2. Formulation as an integer programming problem

Let $\boldsymbol{c}_{ij} = \boldsymbol{a}_i + \boldsymbol{b}_j = (a_i^{(1)} + b_j^{(1)}, a_i^{(2)} + b_j^{(2)}, \ldots, a_i^{(m)} + b_j^{(m)})$. Our problem is formulated to the following 0-1 integer programming problem. Here M is a sufficient big number.

Problem 1

Minimize $f = u - l$

subject to

$$\begin{aligned}
& c_{ij}^{(k)} x_{ij} \le u && (i,j = 1,2,\ldots,n; \\
& l \le c_{ij}^{(k)} x_{ij} + M(1 - x_{ij}) && \quad k = 1,2,\ldots,m), \\
& \sum_{i=1}^{n} x_{ij} = 1 && (j = 1,2,\ldots,n), \\
& \sum_{j=1}^{n} x_{ij} = 1 && (i = 1,2,\ldots,n), \\
& l \ge 0, \quad u \ge 0, \\
& x_{ij} \in \{0,1\} && (i,j = 1,2,\ldots,n).
\end{aligned}$$

Unlike the classical assignment problem (scalar case), no polynomial time algorithm of obtaining the integer solution of this problem, and unfortunately, the relaxation method is not effective for this type of integer programming problems[8)]. So instead of trying to solve Problem 1 directly, we propose approximate algorithms give a quasi optimal solution.

3. Property of the problem

Before describing our algorithm for Problem 1, let us consider the property of our problem.

At first, let us consider the case that $m = 1$, i.e., a_i and b_j are scalars. It is trivial that the total sum of $c_{i\pi(i)}$ does not depend on π, i.e.,

$$\sum_{i=1}^{n} c_{i\pi(i)} = \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_{\pi(i)} = \sum_{i=1}^{n} a_i + \sum_{j=1}^{n} b_j = const.$$

Next, let us consider the case that $m = 2$. In this case again the total sum of $c_{i\pi(i)}^{(1)}$ and that of $c_{i\pi(i)}^{(2)}$ do not depend on π. We denote $\frac{1}{n}\sum_{i=1}^{n}(a_i^{(1)} + b_i^{(1)})$ by μ_1 and $\frac{1}{n}\sum_{i=1}^{n}(a_i^{(2)} + b_i^{(2)})$ by μ_2. For general case of m, the situation is unchanged.

Thus, we have the following Property 1.

Property 1 For each k, the total sum of $c_{ij}^{(k)}$ does not depend on the selection of permutation π and it is constant. □

Hereafter, we consider only the case that $m = 2$, and also assume that $\mu_1 = \mu_2$.

4. Algorithm for the scalar balanced assignment problem

Since our algorithm for the 2 dimensional balanced assignment problem is based on the scalar BOP, we restate the algorithm shown in[4)].

Algorithm_BOP

$C = (c_{ij})_{i,j=1,2,\ldots,n}$: the cost matrix for an $n \times n$ assignment problem.

(i,j) : the i,j element of C.

$v_1 < v_2 < \cdots < v_k$: the values appearing in C.

$C(i,j) := \{(i,j) : v_l \le c_{ij} \le v_u\}$.

q is the number that satisfies

$$v_q = \max\{\max_j(\min_i c_{ij}), \max_i(\min_j c_{ij})\}.$$

Step 0 (Initialization)

Let $l := 1, u := q, T := \infty$. T is the minimum difference of the maximum value and minimum one so far.

If the edges in $C(l,u)$ can make a complete matching, then go to Step 1 . Otherwise go to Step 2 .

Step 1 ($C(l,u)$ contains an assignment)

Remove all c_{ij} that equal to v_l from $C(l,u)$. After this operation $C(l,u)$ changes to $C(l+1,u)$.

Check whether a complete matching exists or not.

(i) If a complete matching exits.

$l+1 = u \ \Rightarrow \ T := 0, l^* := u^* := u$ and stop.

$l+1 \neq u \ \Rightarrow \ l := l+1$ and go to Step 2.

(ii) If a complete matching does not exist.

$v_u - v_l < T \ \Rightarrow \ T := v_u - v_l, l^* := l, u^* := u$. Then $l := l+1$ and go to Step 2.

Step 2 ($C(l,u)$ does not contain an assignment)

If $u = k$, stop.

$u \neq k \ \Rightarrow$ add all c_{ij} that's weight are v_{u+1} to $C(l,u)$. This is $C(l,u+1)$. Check $C(l,u+1)$ contains the complete matching. Let $u := u+1$.

(i) If a complete matching exits, go to Step 1.

(ii) If a complete matching does not exist, go to Step 2.

The complexity of checking a complete matching exists is $O(n^{2.5})$[9], and the above Step 1 is iterated n^2 times in the worst case, so it is easy to see that the total time complexity of Algorithm_BOP is $O(n^{4.5})$. Furthermore, there has been proposed an $O(n^4)$ algorithm[4].

5. Algorithm for 2 dimensional balanced assignment problem

In this section we propose two algorithms. Both use Algorithm_BOP's searching method. That is, narrow step by step the range of edges' weight that edges in it are allowed to use to construct a complete matching between A and B.

In Algorithm 1 we take bigger element for the representation in each 2-dimensional vector, and deal with the scalar case balanced assignment problem.

In Algorithm 2 we transform each element of the vector to normal form and calculate the decination from the mean value. Then we solve the problem to minimize the total decinations.

5.1 Algorithm 1

First, we extend Algorithm_BOP to 2-dimensional vector case simple way. For each vector $\boldsymbol{a}_i = (a_i^{(1)}, a_i^{(2)})$, we take $\max\{a_i^{(1)}, a_i^{(2)}\}$ to decide the representative value. We describe it $a_i^{\max}$. $b_j^{\max}$ is also defined in the same way. Namely

$$a_i^{\max} = \max\{a_i^{(1)}, a_i^{(2)}\}, \quad b_j^{\max} = \max\{b_j^{(1)}, b_j^{(2)}\}.$$

Then let $c_{ij} = a_i^{\max} + b_j^{\max}$ and we solve the scalar case's balanced optimization problem that edges' weight are given by c_{ij} .

Algorithm 1

For $a_i^{\max} = \max\{a_i^{(1)}, a_i^{(2)}\}$, $b_j^{\max} = \max\{b_j^{(1)}, b_j^{(2)}\}$; $(i,j = 1,2,\ldots,n)$, define the scalar c_{ij} as follows: $c_{ij} = a_i^{\max} + b_j^{\max}$.

$C = (c_{ij})_{i,j=1,2,\ldots,n}$: the cost matrix for an $n \times n$ assignment problem.

$v_1 < v_2 < \cdots < v_k$: the values appearing in C.

$C(l,u)$: the set of c_{ij} satisfy $v_l \le c_{ij} \le v_u$.

Step 0 (Initialization)

Let $l := 1, u := 1, T := \infty$.

Step 1 (Set edges' weight w_{ij})

$c_{ij} < v_l$ or $c_{ij} > v_u \ \Rightarrow \ w_{ij} := \infty$.

$v_l \le c_{ij} \le v_u \ \Rightarrow \ w_{ij} := c_{ij}$.

Solve the linear sum assignment problem for w_{ij}. We write the objective function g_1 and it is defined as follows :

$$g_1 = \sum_{i=1}^{n} w_{ij} x_{ij}.$$

If $g_1 \neq \infty$, then go to Step 2. Otherwise go to Step 3.

Step 2 ($C(l,u)$ contains a complete matching)

Define the permutation $\pi(i)$ that $\pi(i) := j$ if $x_{ij} = 1$.

(i) Case $l = u$.

Let $T := 0, l^* := u^* := u$ and stop.

(ii) Case $l \neq u$.

If $T > T_2(\pi)$ then set $T := T_2(\pi), \pi^* := \pi, l^* := l, u^* := u$. Set $l := l + 1$ and go to Step 1.

Step 3 ($C(l,u)$ does not contain a complete matching)

(i) If $u = k$ then stop.

(ii) If $u \neq k$, then set $u := u + 1$ and go to Step 1.

When Algorithm 1 terminate, π^* gives the solution of Problem 1.

From Property 1, the total value of n weights is constant. We can say that the optimal matching is to make each weight as near to the mean value as possible. So if the matching minimizes the maximum value, we expect that it also makes the minimum one maximum. The idea of Algorithm 1 depends on this expectation.

However there is an obvious defect in the matching constructed by Algorithm 1. When we take $a_i^{\max}$ from $(a_i^{(1)}, a_i^{(2)})$, only selected one has relation to make a maching. For example, let $\pi(i_0) = j_0$, the representations, $a_{i_0}^{\max} = a_{i_0}^{(1)}$ and $b_{j_0}^{\max} = b_{j_0}^{(1)}$. It happens the case that $|(a_{i_0}^{(2)} + b_{j_0}^{(2)}) - \mu| > |(a_{i_0}^{(1)} + b_{j_0}^{(1)}) - \mu|$. Especially it occurs Cor_a or/and Cor_b is negative cases. Here Cor_a is the correlation coefficient of $a_i^{(1)}$ and $a_i^{(2)}$, similary Cor_b is that of $b_j^{(1)}$ and $b_j^{(2)}$.

5.2 Algorithm 2

In Algorithm 1, we have taken only one component from $\boldsymbol{a}_i = (a_i^{(1)}, a_i^{(2)})$ and $\boldsymbol{b}_j = (b_j^{(1)}, b_j^{(2)})$ to make the representations $a_i^{\max}$ and $b_j^{\max}$ respectively. Algorithm 2 is an improved method on this weakness.

Define the mean value of $a_i^{(1)}, (i = 1, 2, \ldots, n)$ as μ_{a_1} and the variance of them $\sigma_{a_1}^2$. Now we normalize $a_i^{(1)}$ and denote by $\tilde{a}_i^{(1)}$:

$$\tilde{a}_i^{(1)} = \frac{a_i^{(1)} - \mu_{a_1}}{\sigma_{a_1}}.$$

By this normalization $a_i^{(1)}$ is converted to the probablistic variable that the mean and the variance is 0 and 1, respectively. Similary we define $\mu_{a_2}, \mu_{b_1}, \mu_{b_2}$; $\sigma_{a_2}, \sigma_{b_1}, \sigma_{b_2}$ and $\tilde{a}_i^{(2)}, \tilde{b}_j^{(1)}, \tilde{b}_j^{(2)}$. Then $a_i^{(1)} + b_j^{(1)} = \mu_{a_1} + \mu_{a_2}$ is equivalent to $\tilde{a}_i^{(1)} + \tilde{b}_j^{(1)} = 0$. It is also for $a_i^{(2)} + b_j^{(2)}$.

So now we can consider the mapping between normlized vectors $(\tilde{a}_i^{(1)}, \tilde{a}_i^{(2)})$ and $(\tilde{b}_j^{(1)}, \tilde{b}_j^{(2)})$, instead of between $(a_i^{(1)}, a_i^{(2)})$ and $(b_j^{(1)}, b_j^{(2)})$. Then our problem is changed to find the permutation $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ that the all $2n$ values $\tilde{a}_i^{(1)} + \tilde{b}_{\pi(i)}^{(1)}, \tilde{a}_i^{(2)} + \tilde{b}_{\pi(i)}^{(2)}$ are as near to origin as possible (Fig. 2).

For the problem between $(\tilde{a}_i^{(1)}, \tilde{a}_i^{(2)})$ and $(\tilde{b}_{\pi(i)}^{(1)}, \tilde{b}_{\pi(i)}^{(2)})$, we consider an assignment problem whose objective function is

$$g_2 = \sum_{i=1}^{n} \{(\tilde{a}_i^{(1)} + \tilde{b}_{\pi(i)}^{(1)})^2 + (\tilde{a}_i^{(2)} + \tilde{b}_{\pi(i)}^{(2)})^2\}.$$

The closer each $a_i^{(1)} + b_{\pi(i)}^{(1)}$ and $a_i^{(2)} + b_{\pi(i)}^{(2)}$ are $\mu_{a_1} + \mu_{b_1}$ and $\mu_{a_2} + \mu_{b_2}$ respectively, the nearer $\tilde{a}_i^{(1)} + \tilde{b}_{\pi(i)}^{(1)}$ and $\tilde{a}_i^{(2)} + \tilde{b}_{\pi(i)}^{(2)}$ are 0. So we can say that the permutation minimizes g_2 gives the approximate solution for our problem. This problem is the linear sum assignment problem.

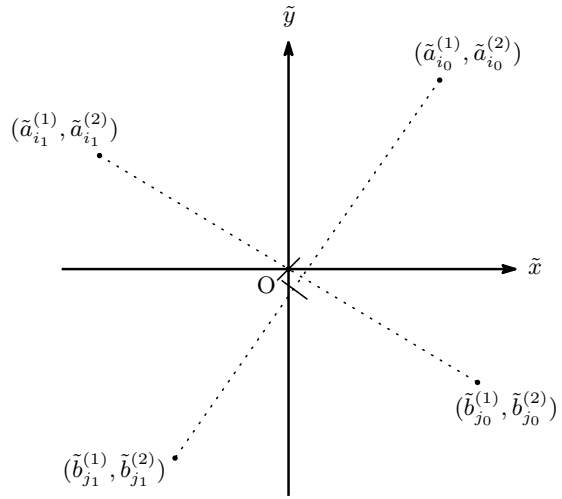

Fig 2: Find a permutation π that the center of $(\tilde{a}_i^{(1)}, \tilde{a}_i^{(2)})$ and $(\tilde{b}_{\pi(i)}^{(1)}, \tilde{b}_{\pi(i)}^{(2)})$ as near to origin as possible.

Vol.2010-MPS-79 No.2
2010/7/12

Problem 2

Minimize

$$g_2 = \sum_{i,j=1}^{n} \{(\tilde{a}_i^{(1)} + \tilde{b}_j^{(1)})^2 + (\tilde{a}_i^{(2)} + \tilde{b}_j^{(2)})^2\} x_{ij}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1, 2, \ldots, n),$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1, 2, \ldots, n),$$

$x_{ij} \in \{0, 1\}$.

It is obvious that g_2 gives the total sum of difference's squares from $\mathbf{0}$. Threfore for each permutation π introduced by the solution of Problem 2, we check $T_2(\pi)$ and adopt π^* minimizes $T_2(\pi)$ as the solution of Problem 1.

Algorithm 2

$\tilde{c}_{ij} := (\tilde{a}_i^{(1)} + \tilde{b}_j^{(1)})^2 + (\tilde{a}_i^{(2)} + \tilde{b}_j^{(2)})^2$.

$\tilde{C} = (\tilde{c}_{ij})_{i,j=1,2,\ldots,n}$: the cost matrix for an $n \times n$ assignment problem.

$v_1 < v_2 < \cdots < v_k$: the values appearing in $\tilde{C}$.

$\tilde{C}(l, u)$: the set of $\tilde{c}_{ij}$ satisfy $v_l \leq \tilde{c}_{ij} \leq v_u$.

Step 0 (Initialization)

Let $l := 1, u := 1, T := \infty$.

Step 1 (Solve the linear sum assignment problem)

$\tilde{c}_{ij} < v_l$ or $\tilde{c}_{ij} > v_u \ \Rightarrow \ w_{ij} := \infty$.

$v_l \leq \tilde{c}_{ij} \leq v_u \ \Rightarrow \ w_{ij} := \tilde{c}_{ij}$.

Solve the linear sum assignment problem for w_{ij}. Then if the objective function $g_2 \neq \infty$, go to Step 2. Otherwise go to Step 3.

Step 2 ($\tilde{C}(l, u)$ contains a complete matching)

Define the permutation $\pi(i)$ that $\pi(i) := j$ if $x_{ij} = 1$.

(i) Case $l = u$.

Let $T := 0, l^* := u^* := u$ and stop.

(ii) Case $l \neq u$.

If $T > T_2$ then set $T := T_2, \pi^* := \pi, l^* := l, u^* := u$. Set $l := l + 1$ and go to Step 1.

Step 3 ($\tilde{C}(l, u)$ does not contain a complete matching)

(i) If $u = k$ then stop.

(ii) If $u \neq k$, then set $u := u + 1$ and go to Step 1.

The linear sum assignment problem is solved $O(n^3)$ time complexity in the worst case[10]. In Algorithm 1 and Algorithm 2, it is solved at most n^2 times. Initialization and the checking process's time is constant. So we conclude the time complexity of Algorithm 1 and 2 is $O(n^5)$.

6. Numerical experiments

For vectors $\boldsymbol{a}_i = (a_i^{(1)}, a_i^{(2)})$ and $\boldsymbol{b}_j = (b_j^{(1)}, a_j^{(2)})$, $(i, j = 1, 2, \ldots, n)$, let $a_i^{(1)}, a_i^{(2)}, b_j^{(1)}, a_j^{(2)}$ follow the normal distribution that the mean is 10 and the variance 1, respectively.

We show the results of numerical experiments for $n = 100$. Each Cor_a and Cor_b, we make two different data sets. For the same data set, we solved by Algorithm 1 and Algorithm 2. Table 1 and 2 show the results for $n = 100$ solved by Algorithm 1 and Algorithm 2, respectively.

For all data sets, Algorithm 2 gives better results than that of Algorithm 1 on the whole. However when the sign of Cor_a and Cor_b is same, Algorithm 1 shows a good performance. So taking the overhead of creating objective function g_2 and the numerical error into consideration, we can say that the after checking the sign of Cor_a and Cor_b, then decide which algorithms use.

7. Conclusions

In this paper, we considered a permutation that minimizes the difference of the maximum value and the minimum one in $2n$ sums made by the permuation. We first formulated this problem as an integer programming problem. Then we proposed two $O(n^5)$ approximate algorithms for the problem based on Algorithm_BOP[4]. And we presented the results from computational experiments using our two algorithms.

Create the exact solution in some way and compare the quasi optimal solution by our proposed algorithms to it is left to further research.

Vol.2010-MPS-79 No.2
2010/7/12

参 考 文 献

1) D.W.Pentico, *Assignment problems: A golden anniversary survey*, European J. Oper. Res. 176, pp.774-793, 2007.
2) D.-Z.Du and P.M.Pardalos(Eds.), *Handbook of Combinatorial Optimization*, Kluwer, 1999.
3) H.N.Gabow and R.E.Tarjan, *Algorithms for Two Bottleneck Optimization Problems*, J. of Algorithms, Vol.9, pp.411-417, 1988.
4) S.Martello,W.R.Pulleyblank,P.Toth and D.de Werra, *Balanced Optimization Problems*, Operations Reserch letters Vol.3, No.5, pp.275-278, 1984.
5) Y.Kamura and M.Nakamori, *Combining Imperfect Components to Minimize the System's Error*, Proc. PDPTA'01, pp.1277-1283, 2001.
6) Y.Kamura, M.Nakamori and Y.Shinano, *Combining Imperfect Components (II) — The Case of Multidimensional Error*, Proc. PDPTA'02, pp.228-232, 2002.
7) Y.Kamura and M.Nakamori, *Combining Imperfect Components (III) — Minimax Optimization of Multidimensional Cost Error*, Proc. PDPTA'04, pp.311-316, 2004.
8) M.Mori and T.Matsui, *Operations Research*(in Japanese), Asakura publishing, 2004.
9) J.E.Hopcroft and R.M.Karp, *An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs*, SIAM J. Comput. Vol.2,No.4,pp.225-231, 1973.
10) R.E.Burkard, *Selected topics on assignment problems*, Discrete App. Math., 123, pp.257-302, 2002.

Table 1 : $n = 100$, Algorithm 1

		Data 1			Data 2		
Cor_a	Cor_b	Max	Min	diff	Max	Min	diff
-1.0	-1.0	20.6277	19.3723	1.2554	20.4162	19.5838	0.8324
-1.0	-0.5	21.1062	18.5528	2.5534	21.1474	18.4862	2.6612
-1.0	0.0	21.6211	17.4032	4.2179	21.4141	17.5529	3.8612
-1.0	0.5	21.5460	16.6397	4.9063	21.9223	17.3766	4.5457
-1.0	1.0	22.5844	15.7712	6.8132	21.9210	14.3053	7.6157
-0.5	-0.5	20.8955	17.6405	3.2550	20.9763	18.6864	2.2899
-0.5	0.0	21.5077	17.3642	4.1435	20.8298	18.2061	2.6237
-0.5	0.5	21.7811	16.6792	5.1019	21.0596	17.2530	3.8066
-0.5	1.0	22.0696	16.8683	5.2013	21.9174	17.0144	4.9030
0.0	0.0	21.0863	18.6364	2.4499	20.9735	17.6820	3.2915
0.0	0.5	21.1613	18.3384	2.8229	21.5932	18.8291	2.7641
0.0	1.0	20.9331	17.1424	3.7907	22.5109	17.1620	5.3489
0.5	0.5	20.9191	19.0730	1.8461	21.2065	18.4932	2.7133
0.5	1.0	20.8593	17.6928	3.1665	21.2709	17.3914	3.8795
1.0	1.0	20.3449	18.9761	1.3688	20.3802	19.6933	0.6869

Table 2 : $n = 100$, Algorithm 2

		Data 1			Data 2		
Cor_a	Cor_b	Max	Min	diff	Max	Min	diff
-1.0	-1.0	20.6277	19.3723	1.2554	20.4162	19.5838	0.8324
-1.0	-0.5	21.1062	18.6009	2.5053	21.3284	18.8522	2.4762
-1.0	0.0	21.6211	18.1571	3.4640	21.5892	17.4137	4.1755
-1.0	0.5	21.6832	17.9907	3.6925	22.1686	17.5900	4.5786
-1.0	1.0	23.1470	17.2507	5.8963	23.1036	16.5707	6.5329
-0.5	-0.5	21.1391	18.1347	3.0044	21.1575	19.1685	1.9890
-0.5	0.0	21.7274	17.8041	3.9233	20.9602	18.4671	2.4931
-0.5	0.5	22.3257	18.2207	4.1050	21.3936	17.8851	3.5085
-0.5	1.0	22.3949	17.8192	4.5757	22.0393	18.0347	4.0046
0.0	0.0	21.2206	18.8002	2.4204	21.0204	18.6712	2.3492
0.0	0.5	21.1755	18.8423	2.3332	21.5932	19.4143	2.1789
0.0	1.0	21.6214	18.0310	3.5904	22.5547	17.9871	4.5676
0.5	0.5	20.9677	19.3208	1.6469	21.2763	19.2066	2.0697
0.5	1.0	21.0103	18.4634	2.5469	21.3980	17.8165	3.5815
1.0	1.0	20.3449	18.9761	1.3688	20.3802	19.6933	0.6869