A Simple Approximation Algorithm for the Modified Bottleneck Assignment Problem in Vector Case

Yuusaku Kamura and Mario Nakamori

***Abstract*—We deal with the modified bottleneck assignment problem in vector case. The problem is to find the complete matching between two vector sets that minimizes the maximum sum of vectors' elements. In scalar case, there is a simple polynomial time algorithm that uses the order of elements' value. We extend this problem to the 2 dimensional vector case and propose a simple approximation algorithm for it. We show the effectiveness of our algorithm by numerical experiments.**

***Index Terms*—Approximation algorithm, assignment problem, balanced assignment problem.**

I. Introduction

Suppose we have *n* suppliers $u_1, u_2, ..., u_n$ (to be denoted by a set *U*) and *n* customers $v_1, v_2, \ldots, v_n$ (to be denoted by *V*). 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 $\pi : U \to V$ under an appropriate objective function. This is the well-known assignment problem.

Since making the one to one correspondence $\pi : U \to V$ is equivalent to permuting the set $\{1, 2, \ldots, n\}$, we hereafter call the one to one correspondence as *permutation*.

The assignment problem has various versions with respect to the objectives [1], [2]. If we are going to minimize the total sum of the cost $\sum_{i=1}^{n} c_{i\pi(i)}$, the problem is the *linear sum assignment problem*, and there have been proposed many efficient algorithms [3].

If we are going to minimize the maximum cost of the corresponded pair, the problem is the *bottleneck assignment problem* [4]. The objective is

$$\min_{\pi} \max_{1 \le i \le n} c_{i\pi(i)} \tag{1}$$

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 one of the corresponded pair, the problem is the *balanced assignment problem* [5]. The objective is

$$\min_{\pi} \left\{ \max_{1 \le i \le n} c_{i\pi(i)} - \min_{1 \le i \le n} c_{i\pi(i)} \right\}. \tag{2}$$

Again, polynomial time algorithms have been proposed for the balanced assignment problem.

In this paper we extend the bottleneck 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 a_i and the customer vector 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 $\mathbf{a}_i = (a_i^{(1)}, a_i^{(2)}, \cdots, a_i^{(m)})$ and each element of *B* by $\mathbf{b}_i = (b_i^{(1)}, b_i^{(2)}, \cdots, b_i^{(m)})$. We assume that $a_i^{(1)}, a_i^{(2)}, \cdots, a_i^{(m)}$ and $b_j^{(1)}, b_j^{(2)}, \cdots, b_j^{(m)}$ are nonnegative and define the sum of vectors $\mathbf{a}_i + \mathbf{b}_j$ as

$$\mathbf{a}_i + \mathbf{b}_j = \left(a_i^{(1)} + b_j^{(1)}, a_i^{(2)} + b_j^{(2)}, \ldots, a_i^{(m)} + b_j^{(m)}\right). \tag{3}$$

We denote it $\mathbf{c}_{ij}$, namely $\mathbf{c}_{ij} = \mathbf{a}_i + \mathbf{b}_j$. Let us consider the following problem: find a permutation π of a set $\{1, 2, \ldots, n\}$ such that

$$T_m(\pi) = \max\left\{ \max_{1 \le i \le n} c_{i\pi(i)}^{(1)}, \max_{1 \le i \le n} c_{i\pi(i)}^{(2)}, \ldots, \max_{1 \le i \le n} c_{i\pi(i)}^{(m)} \right\} \tag{4}$$

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).

It is required to propose a method that minimizes systems' error. That problem is generally formulated to a multi-level assignment problem in vector case. In our former research [6], [7], [8] we have considered the problem of making *n* sets of lenses such that the worst combined error is the minimum.

A multi-level bottleneck assignment problem was introduced in relation to the bus drivers' rostering problem and its NP-completeness has been proved [9]. And for several types of the problem in scalar case, approximation algorithms have been proposed [10]. However, we realized that there has been no study on extending bottleneck assignment problems to the vector case's yet. So we consider the (single-level) bottleneck assignment problem in vector case. In this paper, we show the reformed algorithm that is based on the method in our former research [11].

The problem is formulated as follows:

Manuscript received August 29, 2014; revised November 20, 2014.

Y. Kamura is with Academic Planning Center, Hitotsubashi University, Kunitachi, Tokyo 186-8601, Japan (e-mail: b101440u@r.hit-u.ac.jp).

M. Nakamori was with Tokyo University of Agriculture and Technology, Koganei, Tokyo 184-8588, Japan. He is now with MOC Co, Ltd. Japan (e-mail: nakamori@cc.tuat.ac.jp).

Problem 1: Given 2 sets of n vectors A and B, find the permutation π that minimizes

$$T_m(\pi) = \max_{1\le k\le m}\left\{\max_i c_{i\pi(i)}^{(k)}\right\}, \tag{5}$$

where $c_{i\pi(i)}^{(k)} = a_i^{(k)} + b_{\pi(i)}^{(k)}$.

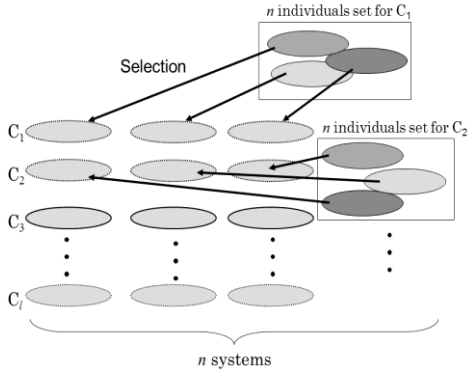

Fig. 1. Construct n lens systems. Each set for the position $C_i(i=1,2,\ldots,l)$ has n individuals.

We deal with the case that vector's dimension is 2, that is, m=2 in Problem 1. In the following we propose an approximation algorithm and show the effectiveness of our proposed method by the numerical experiments.

Remark: In general assignment problems, edges' costs are given independently. However, in our problem they are determined by the vertices' costs. So we call this the 'modified' problem.

II. Formulation as an Integer Programming Problem

Our problem is formulated to the following 0-1 integer programming problem. This is correspondent to the case m=2 in Problem 1.

Problem 2:

$$\text{Minimize } f = t$$

Subject to

$$(a_i^{(k)} + b_j^{(k)})x_{ij} \le t \qquad (i,j=1,2,\ldots,n;k=1,2),$$

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

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

$$x_{ij} \in \{0,1\} \qquad (i,j=1,2,\ldots,n).$$

Unlike the classical assignment problem (i.e., scalar case), no polynomial time algorithm is known to obtain the integer solution of this problem, and unfortunately, the relaxation method is not effective for this type of integer programming problems [12].

III. The Algorithm

Hereafter we assume that each element $a_i^{(1)}$, $a_i^{(2)}; b_j^{(1)}, b_j^{(2)}$ is nonnegative.

In the scalar case, the method that gives the optimal combination is simple. Our proposed algorithm for vector case makes use of the scalar case's combination rule. So first, we show Algorithm 1 that is for the scalar case's.

Algorithm 1

A, B: 2 sets of nonnegative n scalars. $a_i \in A, b_j \in B$.

Step 1:

Sort $a_i, (i=1,2,\ldots,n)$ in ascending order.

Then we assume $0 \le a_1 < a_2 < \cdots < a_{n-1} < a_n$.

Step 2:

Sort $b_j, (j=1,2,\ldots,n)$ in ascending order.

Then we assume $0 \le b_1 < b_2 < \cdots < b_{n-1} < b_n$.

Step 3:

Combine a_i to $b_{n-i+1}, (i=1,2,\ldots,n)$.

Applying the rule in Algorithm 1 to the vector case, we have to order vectors in some way. So in 3.1 and 3.2, we show the method to order vector sequence and combine them.

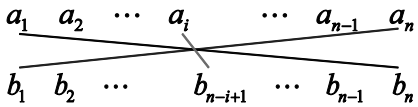

Fig. 2. The optimal combination for the scalar case. This combination minimizes the maximum of $a_i + b_j$.

A. Divide the Cartesian Plane by 2 Lines $y = x$ and $y = -x$

First, we normalize each $a_i^{(1)}$, and we denote it $\tilde{a}_i^{(1)}$, that is,

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

where m_{a_1} and σ_{a_1} are the mean and the variance of $a_i^{(1)}$, $(i=1,2,\ldots,n)$ respectively. By this normalization, the mean and the variance of $\tilde{a}_i^{(1)}$ are 0 and 1, which is also the case for $\tilde{a}_i^{(2)}$. We assume that $\tilde{A}$ is the set of $\tilde{\mathbf{a}}_i = (\tilde{a}_i^{(1)}, \tilde{a}_i^{(2)})$.

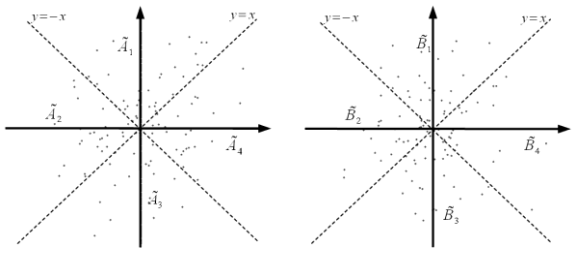

Fig. 3. 2 lines $y = x$ and $y = -x$ divide $\tilde{A}$ to $\tilde{A}_i$, $\tilde{B}$ to $\tilde{B}_j$, $(i, j = 1,2,3,4)$.

Then 2 lines $y = x$ and $y = -x$ divide $\tilde{A}$ into 4 sets $\tilde{A}_1$, $\tilde{A}_2, \tilde{A}_3$ and $\tilde{A}_4$. Where each $\tilde{A}_i$ is the set of $\tilde{\mathbf{a}}_i$ that satisfies the following conditions:

$$\begin{aligned}\tilde{A}_1 &: -\tilde{a}_i^{(2)} < \tilde{a}_i^{(1)} < \tilde{a}_i^{(2)},\\ \tilde{A}_2 &: \tilde{a}_i^{(1)} < \tilde{a}_i^{(2)} < -\tilde{a}_i^{(1)},\\ \tilde{A}_3 &: \tilde{a}_i^{(2)} < \tilde{a}_i^{(1)} < -\tilde{a}_i^{(2)},\\ \tilde{A}_4 &: -\tilde{a}_i^{(1)} < \tilde{a}_i^{(2)} < \tilde{a}_i^{(1)}.\end{aligned}$$

This is also the case for $\mathbf{b}_j = (b_j^{(1)}, b_j^{(2)})$. We define 4 sets $\tilde{B}_1, \tilde{B}_2, \tilde{B}_3$ and $\tilde{B}_4$ in the same way.

Note that $\tilde{a}_i^{(2)} > 0$ and $\tilde{a}_i^{(2)}$ dominates $\tilde{a}_i^{(1)}$ in $\tilde{A}_1$. On the other hand, $\tilde{b}_j^{(2)} < 0$ and $\tilde{b}_j^{(2)}$ dominates $\tilde{b}_j^{(1)}$ in $\tilde{B}_3$. So we can expect that combine $\tilde{\mathbf{a}}_i \in \tilde{A}_1$ to $\tilde{\mathbf{b}}_j \in \tilde{B}_3$ compensate the differences each other. It is the same for the combination $\tilde{\mathbf{a}}_i \in \tilde{A}_2$ to $\tilde{\mathbf{b}}_j \in \tilde{B}_4$, $\tilde{\mathbf{a}}_i \in \tilde{A}_3$ to $\tilde{\mathbf{b}}_j \in \tilde{B}_1$, and $\tilde{\mathbf{a}}_i \in \tilde{A}_4$ to $\tilde{\mathbf{b}}_j \in \tilde{B}_2$. For original $\tilde{\mathbf{a}}_i$ and $\tilde{\mathbf{b}}_j$, we can say the same scheme. We can anticipate that these rules lead to minimize the maximum sum.

B. Make Sequences and Combinations

Here we show the method to make the ordered vectors' sequence and combine them. This is the main part of our algorithm.

Outline of our idea is as follows: (1) apply the rotational transformation $R(-\pi/4)$ at origin to $\tilde{\mathbf{a}}_i$ and $\tilde{\mathbf{b}}_j$, (2) select a point from each domain and combine them, recurrently.

We show the way in the following 3 steps.

1) Step 1: Transformation $R(-\pi/4)$

Apply $R(-\pi/4)$ at origin to $\tilde{\mathbf{a}}_i = (\tilde{a}_i^{(1)}, \tilde{a}_i^{(2)})$, then $\tilde{A}_1$, that is the set of $\tilde{\mathbf{a}}_i$ satisfies $-\tilde{a}_i^{(2)} < \tilde{a}_i^{(1)} < \tilde{a}_i^{(2)}$, is transformed to on the first quadrant. Also $\tilde{A}_2$ is transformed to on the second quadrant, $\tilde{A}_3$ is to the third, $\tilde{A}_4$ is to the fourth.

By this transformation, we can select the point $\tilde{\mathbf{a}}_i$ easily. Hereafter again we denote the sets $\tilde{A}_i, (i = 1,2,3,4)$ that is transformed.

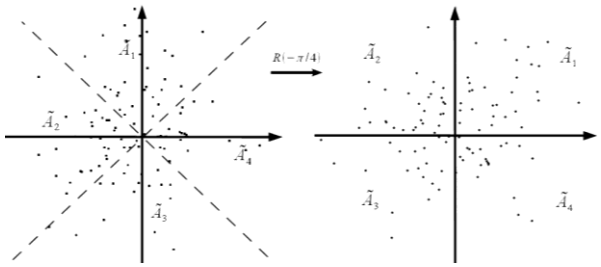

Fig. 4. By $R(-\pi/4)$, each $\tilde{A}_i$ is to be corresponding to i-th quadrant in the Cartesian plane.

Also to $\tilde{\mathbf{b}}_j$ we apply $R(-\pi/4)$.

2) Step 2 (1st time): Make sequences for the largest $\tilde{\mathbf{a}}_i$ and $\tilde{\mathbf{b}}_j$

For $\tilde{\mathbf{a}}_i$:

1) Find i_1 such that $\tilde{\mathbf{a}}_{i_1}$ has the largest distance from the origin in $\tilde{A}_1$. Find i_2, i_3, i_4 that $\tilde{\mathbf{a}}_{i_2}, \tilde{\mathbf{a}}_{i_3}, \tilde{\mathbf{a}}_{i_4}$ have the same property in $\tilde{A}_2, \tilde{A}_3, \tilde{A}_4$ respectively.
2) Make the ordered sequence $\{\mathbf{a}_{i_1}, \mathbf{a}_{i_2}, \mathbf{a}_{i_3}, \mathbf{a}_{i_4}\}$.

For $\tilde{\mathbf{b}}_j$:

1) Find j_1 such that $\tilde{\mathbf{b}}_{j_1}$ has the largest distance from the origin in $\tilde{B}_1$. Find j_2, j_3, j_4 that $\tilde{\mathbf{b}}_{j_2}, \tilde{\mathbf{b}}_{j_3}, \tilde{\mathbf{b}}_{j_4}$ have the same property in $\tilde{B}_2, \tilde{B}_3, \tilde{B}_4$ respectively.
2) Make the ordered sequence $\{\tilde{\mathbf{b}}_{j_2}, \tilde{\mathbf{b}}_{j_1}, \tilde{\mathbf{b}}_{j_4}, \tilde{\mathbf{b}}_{j_3}\}$.

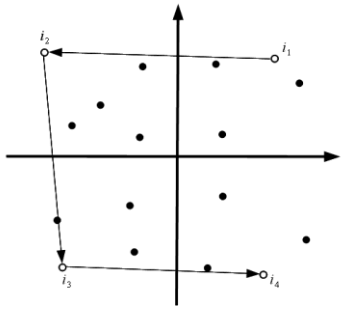

Fig. 5. Find i_1, i_2, i_3, i_4 that $\tilde{\mathbf{a}}_{i_k}$ has the largest distance from the origin in each domain.

3) Step 3(1st time): Combination

Combine $\mathbf{a}_{i_1}$ to $\mathbf{b}_{j_3}$, $\mathbf{a}_{i_2}$ to $\mathbf{b}_{j_4}$, $\mathbf{a}_{i_3}$ to $\mathbf{b}_{j_1}$, $\mathbf{a}_{i_4}$ to $\mathbf{b}_{j_2}$.

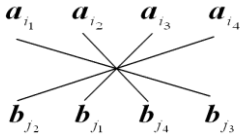

Fig. 6. Combination.

4) Step 2 (2nd time): Make sequences for the second largest $\tilde{\mathbf{a}}_i$ and $\tilde{\mathbf{b}}_j$

For $\tilde{\mathbf{a}}_i$:

1) Find i_5 such that $\tilde{\mathbf{a}}_{i_5}$ has the second largest distance from the origin in $\tilde{A}_1$. Find i_6, i_7, i_8 that $\tilde{\mathbf{a}}_{i_6}, \tilde{\mathbf{a}}_{i_7}, \tilde{\mathbf{a}}_{i_8}$ have the same property in $\tilde{A}_2, \tilde{A}_3, \tilde{A}_4$ respectively.

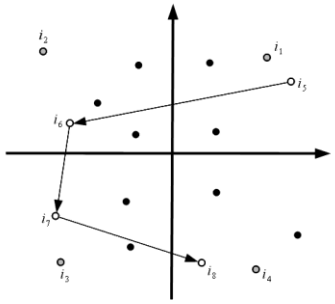

Fig. 7. Find i_5, i_6, i_7, i_8 that $\tilde{\mathbf{a}}_{i_k}$ has the second largest distance from the origin in each domain.

2) Make the ordered sequence $\{\mathbf{a}_{i_5}, \mathbf{a}_{i_6}, \mathbf{a}_{i_7}, \mathbf{a}_{i_8}\}$.

For $\tilde{\mathbf{b}}_j$:

1) Find j_5 such that $\tilde{\mathbf{b}}_{j_5}$ has the largest distance from the origin in $\tilde{B}_1$. Find j_6, j_7, j_8 that $\tilde{\mathbf{b}}_{j_6}, \tilde{\mathbf{b}}_{j_7}, \tilde{\mathbf{b}}_{j_8}$ have the same property in $\tilde{B}_2, \tilde{B}_3, \tilde{B}_4$ respectively.
2) Make the ordered sequence $\{\tilde{\mathbf{b}}_{j_6}, \tilde{\mathbf{b}}_{j_5}, \tilde{\mathbf{b}}_{j_8}, \tilde{\mathbf{b}}_{j_7}\}$.

5) Step 3(2nd time): Combination

Combine $\mathbf{a}_{i_5}$ to $\mathbf{b}_{j_7}$, $\mathbf{a}_{i_6}$ to $\mathbf{b}_{j_8}$, $\mathbf{a}_{i_7}$ to $\mathbf{b}_{j_5}$, $\mathbf{a}_{i_8}$ to $\mathbf{b}_{j_6}$.

Repeat Step 2 and 3 $\lfloor n/4 \rfloor$ times. If $\tilde{A}_k$ has no element $\tilde{\mathbf{a}}_i$ while the procedure, then skip to select $\tilde{\mathbf{a}}_i$ from $\tilde{A}_k$ and proceed to the next set $\tilde{A}_{k+1}$. We make the sequence that has 4 elements in each repetition except for the last one.

Here we summarize our method in the following Algorithm 2.

Algorithm 2

Let $A = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n\}$, $\mathbf{a}_i = (a_i^{(1)}, a_i^{(2)}) \geq \mathbf{0}$ and $B = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$, $\mathbf{b}_i = (b_j^{(1)}, b_j^{(2)}) \geq \mathbf{0}$.

Normalize $a_i^{(1)}, a_i^{(2)}; b_j^{(1)}, b_j^{(2)}$, $(i, j = 1,2,\ldots,n)$.

$k \leftarrow 1$.

Step 1:

Apply the rotational transformation $R(-\pi/4)$ to each vector $\mathbf{a}_i = (a_i^{(1)}, a_i^{(2)})$ and $\mathbf{b}_j = (b_j^{(1)}, b_j^{(2)})$.

Step 2:

Make the suffixes' sequences $i_k, i_{k+1}, i_{k+2}, i_{k+3}$ and $j_k, j_{k+1}, j_{k+2}, j_{k+3}$ by the method illustrated in Fig.5.

Step 3:

Combine $\mathbf{a}_{i_k}$ to $\mathbf{b}_{j_{k+2}}$, $\mathbf{a}_{i_{k+1}}$ to $\mathbf{b}_{j_{k+3}}$, $\mathbf{a}_{i_{k+2}}$ to $\mathbf{b}_{j_k}$, $\mathbf{a}_{i_{k+3}}$ to $\mathbf{b}_{j_{k+1}}$. $k \leftarrow k+1$.

Repeat Step 2 and 3 $\lfloor n/4 \rfloor$ times.

IV. Numerical Experiment

We prepared the test data for the numerical experiments as follows: For vectors $\mathbf{a}_i = (a_i^{(1)}, a_i^{(2)})$ and $\mathbf{b}_j = (b_j^{(1)}, b_j^{(2)})$, $(i, j = 1,2,\ldots,n)$, let $a_i^{(1)}, a_i^{(2)}$; $b_j^{(1)}, b_j^{(2)}$ follow the normal distribution that the mean is 10 and the variance is 1, respectively.

Here ρ_a is the correlation coefficient of $a_i^{(1)}$ and $a_i^{(2)}$, similarly ρ_b is that of $b_j^{(1)}$ and $b_j^{(2)}$. We varied ρ_a and ρ_b to 0.5, 0.6, 0.7, 0.8 and 0.9, respectively.

For each pair of ρ_a and ρ_b, we prepared data sets for n = 30,80,50,150. For each data set we solved Problem 2 by Algorithm 2, then compared to the exact solution. Exact solutions are found by using the solver SCIP[1].

Table I-IV show the results for each n. In most cases, we can find that the relative error is in 5% and the average is about 3%.

[1] SCIP (Solving Constraint Integer Programs) is a non-commercial MIP solver developed at Zuse Institute Berlin. http://scip.zib.de/

TABLE I: N = 30

ρ_a	ρ_b	Exact	Algorithm 2	Rel. err.
0.5	0.5	21.240022	21.892769	0.03073
	0.6	20.877908	21.634428	0.03624
	0.7	21.733145	21.792718	0.00274
	0.8	20.854514	21.351487	0.02383
	0.9	21.318328	22.031924	0.03347
0.6	0.6	21.422296	21.591739	0.00791
	0.7	20.782263	21.975371	0.05741
	0.8	21.146058	21.361803	0.01020
	0.9	21.320370	21.683439	0.01703
0.7	0.7	20.839286	20.946184	0.00513
	0.8	21.385878	21.698508	0.01462
	0.9	20.672497	21.588316	0.04430
0.8	0.8	20.453943	21.305041	0.04161
	0.9	21.354388	21.354388	0.00000
0.9	0.9	21.022683	22.071167	0.04987
Avg.				0.02501

TABLE II: N = 50

ρ_a	ρ_b	Exact	Algorithm 2	Rel. err.
0.5	0.5	21.065313	21.630872	0.02685
	0.6	20.679222	21.202660	0.02531
	0.7	21.127199	21.205217	0.00369
	0.8	21.552706	21.744116	0.00888
	0.9	21.016692	21.173828	0.00748
0.6	0.6	20.768893	21.781906	0.04878
	0.7	20.927895	21.315981	0.01854
	0.8	21.051215	21.258797	0.00986
	0.9	21.336829	21.554520	0.01020
0.7	0.7	20.479623	22.155545	0.08183
	0.8	20.944688	21.366895	0.02016
	0.9	20.872997	20.956433	0.00400
0.8	0.8	20.721295	21.600856	0.04245
	0.9	20.504707	21.166976	0.03230
0.9	0.9	20.546791	20.886970	0.01656
Avg.				0.02379

TABLE III: N = 80

ρ_a	ρ_b	Exact	Algorithm 2	Rel. err.
0.5	0.5	20.615041	22.247895	0.07921
	0.6	20.263159	21.398213	0.05602
	0.7	21.059577	21.270711	0.01003
	0.8	21.916284	22.006118	0.00410
	0.9	21.777466	21.791620	0.00065
0.6	0.6	21.366608	21.726156	0.01683
	0.7	20.613449	21.115830	0.02437
	0.8	21.219204	21.457670	0.01124
	0.9	20.349369	21.586816	0.06081
0.7	0.7	20.974483	21.547970	0.02734
	0.8	21.154642	21.707206	0.02612
	0.9	20.622246	21.506437	0.04288
0.8	0.8	20.798744	21.209092	0.01973
	0.9	20.420639	20.980290	0.02741
0.9	0.9	20.701731	21.095293	0.01901
Avg.				0.02838

TABLE IV: N = 150

ρ_a	ρ_b	Exact	Algorithm 2	Rel. err.
0.5	0.5	20.962094	21.600369	0.03045
	0.6	21.158003	21.890947	0.03464
	0.7	21.499656	21.601543	0.00474
	0.8	21.198603	21.399110	0.00946
	0.9	20.835847	21.832972	0.04786
0.6	0.6	20.806331	21.851099	0.05021
	0.7	20.666572	21.534317	0.04199
	0.8	20.841770	21.504302	0.03179
	0.9	20.879549	21.480433	0.02878
0.7	0.7	20.601341	21.490196	0.04315
	0.8	20.841926	21.091256	0.01196
	0.9	21.067726	21.361331	0.01394
0.8	0.8	20.412167	20.962446	0.02696
	0.9	20.719152	21.049624	0.01595
0.9	0.9	20.666116	21.106759	0.02132
Avg.				0.02755

V. Conclusions

In this paper, we considered a permutation that minimizes the maximum value in 2*n* sums given by 2 dimensional vectors.

We first formulated this problem as an integer programming problem. Then we proposed the approximation algorithm Algorithm 2 for the problem based on the method for the scalar case's. Similar approach will be found in [13], [14].

And we presented the results from computational experiments using our algorithm and compared to the exact solutions. It gave sufficient good approximate solutions.

For further research, we will consider the case that vector's dimension is much higher and the vector case's multi-level problem.

References

[1] R. E. Burkard, "Selected topics on assignment problems," *Discrete Appl. Math*., vol. 123, pp. 257-302, 2002.

[2] D. W. Pentico, "Assignment problems: A golden anniversary survey," *European J. Oper. Res.*, vol. 176, pp. 774-793, 2007.

[3] D.-Z. Du and P. M. Pardalos, *Handbook of Combinatorial Optimization*, 1999.

[4] H. N. Gabow and R. E. Tarjan, "Algorithms for two bottleneck optimization problems," *J. of Algorithms*, vol. 9, pp. 411-417, 1988.

[5] S. Martello, W. R. Pulleyblank, P. Toth, and D. de Werra, "Balanced optimization problems," *Oper. Res. Lett*., vol. 3, pp. 275-278 1984.

[6] Y. Kamura and M. Nakamori, "Combining imperfect components to minimize the system's error," in *Proc. International Conf. on Parallel and Distributed Processing Techniques and Applications*, 2001, pp. 1277-1283.

[7] Y. Kamura, M. Nakamori, and Y. Shinano, "Combining imperfect components (II) — The case of multidimensional error," in *Proc. International Conf. on Parallel and Distributed Processing Techniques and Applications*, 2002, pp. 228-232.

[8] Y. Kamura and M. Nakamori, "Combining imperfect components (III) — Minimax optimization of multidimensional cost error," in *Proc. International Conf. on Parallel and Distributed Processing Techniques and Applications*, 2004, pp. 311-316.

[9] P. Carraresi and G. Gallo, "A multi-level bottleneck assignment approach to the bus drivers' rostering problem," *European J. Oper. Res.*, vol. 16, pp. 163-173, 1984.

[10] T. Dokka, A. Kouvela *et al*., "Approximating the multi-level bottleneck assignment problem," *Oper. Res. Lett*., vol. 40, pp. 163-173, 1984.

[11] Y. Kamura and M. Nakamori, "Modified balanced assignment problem in vector case: System construction problem," in *Proc. the International Conf. on Computational Sci. & Computational Intelligence*, 2014, vol. II, pp. 52-56.

[12] M. Mori and T. Matsui, *Operations Research (in Japanese)*, Japan: Asakura Publishing, 2004.

[13] C. Mcdiarmid and T. Müller, "On the chromatic number of random geometric graphs," *Combinatorica*, vol. 31, no. 4, pp. 423-488, 2011.

[14] Y. Shang, "Improper coloring of random geometric graphs," *J. of Advanced Research in Appl. Math*., vol. 4, no. 1, pp. 1-9, 2012.

Yuusaku Kamura was born in Osaka, Japan. He received his M.Sc. degree in mathematics from Tokyo University of Science in 1993. He was a doctoral student in Tokyo University of Agriculture and Technology. He is studying combinatorial algorithms.

Now he is an assistant in Hitotsubashi University. And also he has lectures of numerical analysis in Tokyo Gakugei University.

Mario Nakamori was born in 1948 in Fukuoka, Japan. He received the Dr. Eng. degree in mathematical engineering and instrumentation physics from the University of Tokyo in 1977.

Since then he was an associate professor of Tokyo University of Agriculture and Technology (TUAT) until 1991 and a professor until the retirement from TUAT in 2014. His major research is in mathematical programming and algorithms.

Prof. Nakamori is now a professor emeritus of TUAT and also an adviser of MOC Co., Limited. He is a member of ACM, fellow of ORSJ, fellow of IPSJ, and a member of JSIAM.