Journal of Computer Sciences

Original Research Paper

A Successive Admissible Cell Method for Solving Large Scale Linear Assignment Problem with Dense Cost Matrix

Warut Boonphakdee and Peerayuth Charnsethikul

Industrial Engineering, Kasetsart University, Bangkok, Thailand

Article history
Received: 25-04-2016
Revised: 30-10-2016
Accepted: 16-11-2016

Corresponding Author:
Warut Boonphakdee
Industrial Engineering,
Kasetsart University, Bangkok,
Thailand
Email: warutboon@yahoo.com

Abstract: Normally, the Linear Assignment Problem (LAP) has been solved by successful algorithms such as Lapjv and Munkres programmed as MATLAB codes. This study presented an improved algorithm for solving large scale LAP. A preprocessing (PP) algorithm was proposed to apply for constructing the kth transferred reduced cost matrix then this matrix was solved by Lapjvalgorithm. Performances of PP-Lapjvalgorithm were faster than theoriginal Lapjv about 1.90-8.20% when problem sizes are expanded from problem sizes 18000 to 34000 on integer number range [1,10] and [1,1000]. On the other hand, PP-Lapjvalgorithm was inefficient on integer number range [1, 1000000] due to more time-consuming for executing Lapjv.m file in PP-Lapjvalgorithm. The enlargement of number ranges is influenced to the average computation time of Lapjvalgorithm raised about 53.63% and 32.05% when the range is expanded from [1,1000] to [1,1000000] on problem sizes 18000 and 30000,respectively. The limitations of problem size were determined by virtual memory of the tested computer that both algorithms enabled to solve at the maximum problem size of 34000.

Keywords: Large Scale Linear Assignment Problem, Complementary Slackness Conditions, Shortest Augmenting Path Method, Preprocessing Algorithm, Lapjv Code

Introduction

LAP is one of the most famous problems in linear programming and in combinatorial optimization. In the present, the scale of problems are expanded to large scale problems as the road network equilibrium traffic (LeBlanc *et al*., 1985), a fleet scheduling (Hane *et al*., 1995) and the computing utility (Zhu *et al*., 2004) etc. The large scale problem appeared in the management science has been growing rapidly up to now. However, the large scale problem can be time-consuming in order to solve for the optimal solution. Algorithms for solving LAP can be divided into six groups; primal-dual, simplex-based, primal (non-simplex), dual simplex-based, dual (non-simplex) algorithms and parallel algorithms (Burkard and Cela, 1999).

Sixty years ago, Harold H. Kuhn proposed a primal-dual algorithm as the Hungarian method, the first polynomial-time method for the assignment problem, can solve the real world problem easily. After that the new research area has been studying today known as the combinatorial optimization. The Hungarian method is improved by James R. Munkres who developed the Hungarian's algorithm for solving the rectangular cost matrix. The Hungarian algorithm was written in the first computer code by R. Silver in 1960 using ALGOL language (Hung and Rom, 1980).

Another primal-dual approach, is called the shortest augmenting path algorithm. Tomizawa's shortest augmenting path was the alternative algorithm involved under the complementary slackness property and using the shortest path technique from Dijkstra's algorithm for searching the optimal solution.

Jonker and Volgenant developed a code to solve the LAP in PASCAL while G.Carpaneto, S.Martello and P.Tothcode was in FORTRAN. The auction algorithm was proposed by D.P. Bertsekas in1981. This algorithm being modeled using admissible transformation of the primal and dual solutions can be both updated simultaneously. In the Hungarian method, the value of dual objective function expands after all primal and dual solutions update, whereas in the auction algorithm it is certain that this objective function does not diminish. Other primal-dual algorithms are formed as a minimum cost flow problem called a pseudo-flow algorithm. It is a flow which completes the capacity constraints. However, it does not accomplish the flow conversation constraints. This algorithm works with ε-relaxations of the minimum cost flow problem that ε is decreased and the procedure is iterated until ε converges to $1/n$. At this point, an optimal solution of the ε-relaxed flow problem is also an optimal solution for the minimum cost flow problem. The pseudo-flow algorithm applies ε-complementary conditions whereas the auction algorithm applies cost scaling. The family of pseudo-flow algorithms were developed by J.B.Orlin and R.K.Ahuja in 1987, A.V.Goldberg, S.A.Plotkin and P.Vaidya in 1993, A.V.Goldberg and R.Kennedy in 1995 (Burkard *et al.*, 2009).

Normally, the traditional Hungarian, shortest augmenting path methods are written as source codes for solving LAP in FORTRAN and PASCAL. The Hungarian method was written in FORTRAN by E.L.Lawler in 1976, by G.Carpaneto and P.Toth in 1980, by G.Carpaneto, S.Martello and P.Toth in 1988 for both dense and sparse matrices. The shortest augmenting path method was written in FORTRAN by N.Tomizawa in 1971 found in the book by R.E.Burkard and U.Derigs in 1980, in PASCAL by R.Jonker and A.T.Volgenant in 1986 (Burkard and Cela, 1999). However, the format of FORTRAN and PASCAL language are inefficient for constructing a large scale cost matrix. While the MATLAB software are designed for supporting large scale matrix which can be applied to write source code in almost large scale problems with both fully dense and sparse matrices.

Nowadays, the large scale problems involve widely in decision making such as crew assignment problem in airline industry that need weekly large scale flight scheduling. The problem size should expand rapidly until the traditional algorithms are unable to solve them.

This study intends to introduce the admissible cell generation that uses the successive complementary slackness condition for searching the admissible cells to replace the fully dense cells for solving large scale LAP. Comparative performances between the traditional algorithm and the proposed algorithm consist of the Hungarian, shortest augmenting path and the pre-processing algorithms.

Material and Methods

Mathematical Model

Burkard *et al.* (2009) stated that consider a problem of matching n persons to n tasks where for each person i and for each task j. There is an associated cost c_{ij} of assigning person i to task j. The LAP is the problem of matching n persons to n tasks in order to minimize the total cost. The standard integer programming of the LAP is defined as follows:

$$Let\ x_{ij} = 1\ if\ \text{person}\ i\ is\ assigned\ to\ task\ j, otherwise\ x_{ij} = 0$$

$$Minimize \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{1}$$

Subject to:

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = 1,2,...,n \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j = 1,2,...n \tag{3}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j = 1,2,...,n \tag{4}$$

Complementary Slackness Condition

Burkard *et al.* (2009) stated that the dual problem is associated with dual variables u_i and v_j with assignment constraints (2) and (3) respectively as follows:

$$Maximize \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j \tag{5}$$

Subject to:

$$u_i + v_j \le c_{ij} \qquad \forall i,j = 1,2,...,n \tag{6}$$

Rearrange Equation 6:

$$c_{ij} - u_i - v_j \ge 0 \qquad \forall i,j = 1,2,...,n \tag{7}$$

$$c_{ij} - (u_i + v_j) \ge 0 \tag{8}$$

$$c_{ij} - w_{ij} \ge 0 \tag{9}$$

From Equation 7:

$$\bar{c}_{ij} = c_{ij} - u_i - v_j \tag{10}$$

where, $\bar{c}_{ij}$ is called the reduced cost.

Theorem 1 (Complementary slackness theorem).

Let $x = [x_{ij}]$ be a primal feasible solution and $w = [w_{ij}]$ be a dual feasible solution to a symmetric pair of linear programs. Then x and w become an optimal solution pair if and only if the following complementary slackness conditions are satisfied (Fang and Puthenpura, 1993)

Multiply Equation 9 by:

$$x_{ij}; x_{ij}(c_{ij} - w_{ij}) \geq 0 \tag{11}$$

Case a: if x_{ij} = 0 and c_{ij}-u_i-v_j ≠ 0 then there exists at least a violation in inequality for some i and j; therefore, this condition has an opportunity for improving solution.

Case b: if x_{ij} = 1 and c_{ij}-u_i-v_j ≥ then they satisfy Equation 4 and inequality (7) for all i and j; therefore, x_{ij} and u_i, v_j can be an optimal solution.

Both cases can be applied to the corresponding linear programming model both in primal and dual forms of the LAP.

Basic Preprocessing Algorithm

Burkard *et al*. (2009) introduced the algorithm for solving LAP that adopt a Basic Preprocessing (BPP) phase to find a feasible dual solution and a partial primal solution (which it to be less than n rows) satisfying the complementary slackness conditions. A BPP algorithm for finding feasible dual solution and partial primal solution can be written as this Pseudocode 1.

```
Pseudocode 1% BPP algorithm
% dual variables
for $i$ = 1:$n$;
$u_i = \min\{c_{ij}: j = 1,2,\ldots,n\}$;
end for;
for $j$ = 1: $n$;
$v_j = \min\{c_{ij}: u_i: i = 1,2,\ldots,n\}$
end for;
% partial feasible solution
for $j$ = 1 to $n$;
        row($j$) = 0;
        for $i$ = 1 to $n$;
                if row($j$) = 0 and $c_{ij} - u_i - v_j = 0$ then
                $x_{ij}$ = 1 and row($j$)=$i$;
                break;
                end if;
        end for;
end for;
```

The partial primal solution from a BPP algorithm can be allocated in φ as:

$$\varphi(i) = \begin{cases} j & \text{if row}(i) \text{ is assigned to column } (j) \\ 0 & \text{otherwise} \end{cases} \quad \forall i, j = 1,2,\ldots,n \tag{12}$$

The u_i and v_j values are determined by the left hand side statements of Equation 7. The x_{ij} values achieve satisfying the complementary slackness conditions in Equation 11.

The Proposed Preprocessing Algorithm

The preprocessing (PP) algorithm is applied from the BPP algorithm that it makes many admissible cells on each row or column allocated in the kth reduced cost matrix. This matrix is transferred all admissible cells to be their unit costs in the kth transferred reduced cost matrix without assigning partial primal solution as the BPP algorithm. The kth transferred reduced cost matrix is solved by some traditional algorithms to provide the assignment solution and total cost. Then repeating the new u-v dual variables from the kth reduced cost matrix, the new kth transferred reduced cost matrix to be constructed and solved by traditional algorithms again. Until the new total cost is certainly unchangeable and the computation goes to terminate; therefore, the last total cost and the last assignment solution are the optimal solution. The difference between the BPP algorithm and the PP algorithm includes that the BPP provides the partial assignment solution, whereas the PP algorithm arranges the admissible cells for searching the optimal solution later.

PP algorithm uses conditions on both cases "a" and "b" to provide the admissible cells as: if c_{ij}-v_j≤ 0 then x_{ij} has an opportunity for improving solution. The procedure of the PP algorithm can be explained as follows:

```
Pseudocode2 % PP algorithm
z_1 = 0;
for k = 1:$n$;
 % dual variables
for $i$ = 1:$n$;
$u_i = \min\{c_{ij} : j = 1,2,...,n\}$;
end for;
for $j$ = 1:$n$;
$v_j = \min\{c_{ij} - u_i : i = 1,2,...,n\}$;
end for;
 % admissible cells
for $j$ = 1 to $n$;
        for $i$ = 1 to $n$;
                if $c_{ij} - u_i - v_j \leq 0$ then
                        $\bar{c}_{ij} = c_{ij}$; % admissible cell
                else
                        $\bar{c}_{ij}$ = 'NaN ;
        end if;
        end for;
end for;
 % assignment solution (a) and total cost (z)
 [a,z] = Lapjv($\bar{c}_{ij}$) or Munkres($\bar{c}_{ij}$);
 % convert the reduced cost to the equivalent unit cost
$c_{ij} = \bar{c}_{ij}$ ;
% loop controller (z)
        if z-z_1 = 0 then
```

```
                break;
        else
z_1 = z;
        end if;
end for k; %
```

Applied the Preprocessing Algorithm

Determine the admissible cells from the PP algorithm; consequently, applying the PP algorithm creates the reduced cost matrix and executes it by some traditional algorithms.

The PP algorithm is operated to search the admissible cells that is transformed them to construct a transferred reduced cost matrix. This matrix consists of a unit cost value of each admissible cell and NaN symbol in place of zero values each non-admissible cell. It is solved by some traditional algorithms for searching the current total cost and current assignment solution and is turned to iterate searching the new the admissible cells and the new total cost and assignment solution. If total cost to be unchangeable then this total cost is optimal and terminates the iteration, otherwise turn to find the new admissible cells. This approach is called the k-preprocessing (k-PP) algorithm that can be illustrated by its procedure as below:

- Generate a random unit cost matrix
- Find the *u*-*v* dual variables
- Construct the kth reduced cost matrix computing $\bar{c}_{ij} = c_{ij} - u_i - v_j$ for all c_{ij}.

Transfer the reduced cost matrix in these criteria:

- If the reduced cost value of any cells is equal to zero or being less than zero, allocate its unit cost on a cell
- Otherwise, allocate a 'NaN' symbol on its cell

Solve the kth transferred reduced cost matrix by Lapjv/Murkres source codes.

Check the following conditions:

- If total cost value is unchangeable then the current total cost and assignment solution are optimal and terminates the iteration
- Otherwise, turn to step 2

Procedure depicts that if the current total cost has been unchangeable then it is certainly optimal. Moreover, the k-time of iterations are involved with particular of each number range whenthe optimal solution already achieves.

This k-PP algorithm must execute Lapjv or Munkres source code with time of iterations together; consequently, it operates inefficiently on the running time. To solve this disadvantage, the k-PP algorithm can be improved by firstly providing the kth transferred reduced cost matrix and secondly, this matrix can be solved by Lapjv or Munkres source code once only. When executing Lapjv or Murkres with one iteration, it can diminish obviously the overall running time. While the kth iteration of providing the kth transferred reduced cost matrix can find from testing the k-PP algorithm with the same number range. The improved algorithm referred as the preprocessing (PP) algorithm that can be explained as the following procedure.

Steps 1-4 follow the procedure previously described in the k-PP algorithm:

- Perform steps 2-4 k iterations
- Solve the kth transferred reduced cost matrix by Lapjv or Murkres source code
- The obtained total cost and assignment solution from step 6 are optimal and terminate the iteration

PP algorithm must operate using k time of iterations from k-PP algorithm with the same number range. If number range expands sharply, it can be also influenced to increase the value of k.

This successive admissible cell method can be applied in the primal simplex-based algorithm such as the column generation method (Boonphakdee and Charnsethikul, 2014). This method uses some heuristics to search the initial basic solution then constructing the Restricted Master Problem (RMP). This RMP is merely allocated with 2*n*-1 basic variables which utilizes less memory space used to appropriately solving large scale LAP. Primal simplex algorithm is conducted to solve RMP in order to search its dual variables for computing all reduced costs. The minimal negative reduced cost is selected to generate an admissible cell which to be attached to the sparse RMP matrix. Return to solve iteratively on this sparse matrix using the primal simplex method until all reduced costs are nonnegative then terminates iteration. Therefore, the solution of the last iteration is an optimal solution. Advantage of this method can solve larger problem size due to less memory space used; however, it must use more time-consuming for searching a required admissible cells one at a time.

Computational Experiments

This study intends to concentrate the performance of the proposed algorithm comparing with the successfully algorithms such as Hungarian (Munkres.m) and shortest augmenting (Lapjv.m) algorithms. The proposed algorithms consist of the applied preprocessing algorithm for searching the initial basic feasible solution matrix and solved by the faster algorithm. MATLAB 2011a is implemented to write source codes for all algorithms because it can construct large scale matrix efficiently.

PP-Lapjvalgorithm applies the PP algorithm to create the kth reduced cost matrix then using the Shortest augmenting path or Hungarian algorithm searches an optimal assignment solution. Hungarian [Munkres.m version 2.3] and shortest augmenting path [Lapjv.m version 3.0] algorithms have been applied for MATLAB source code written by Yi Cao at Cranfield University (Cao, 2011; 2013). The Lapjv (Jonker-Volgenant) algorithm is faster than the famous Hungarian algorithm. Cao's source code modified from the original C++ code which was made by Roy Jonker, one of the inventors of this algorithm. Both algorithms were compared the performance and tested the exact solution with a simple test problem from OR- library (Beasley, 1990) in 100×100 and 300×300 problem sizes. The faster algorithm was interesting to improve the performance for solving large scale LAP later. The effect of integer uniformly randomly generated in the range parameter K (with $K = 10, 10^3, 10^6$) are tested within 10 substances of both the improved algorithm and the original algorithm.

This study was interested in the computation time and the maximum problem size solving by the proposed algorithm and the effect of related input integer number data on the computation time. The proposed algorithm used the integer uniformly randomly generated numbers for a fully dense cost matrix. All developed programs were performed using a HP "Pavilion" personal computer with processor: Intel ® core ™i3-530 CPU@2.93GHz 12 GB usable RAM and the operating system was Windows 7 64 bit.

Results and Discussion

Verified Test

To verify all source codes, both the successful algorithms were tested using data from Beasley (1990) such as A100.txt and A300.txt. The result of testing can be illustrated in Table 1.

In Table 1, the performances of Lapjv code can execute to be faster than those of Munkres code about 64.71 and 56.00% for A100.txt and A300.txt sample, respectively. Amico and Toth indicated that the time complexity of Lapjvalgorithm was $O(n^3)$ whereas Burkard *et al*. (2009) stated that the time complexity of Munkres algorithms was $O(n^4)$. Therefore, this study intended to improve Lapjvalgorithm for solving large-scale LAP. The PP algorithm was applied for constructing the initial basic solution sparse matrix then it was solved by Lapjv code. PP-Lapjvalgorithm was tested and compared with Lapjvalgorithm from A100.txt and A300.txt problems shown in Table 2.

In Table 2, the average computation time of PP-Lapjv is faster than the original Lapjvalgorithm for operating A300.txt about 10.30%. On the other hand, PP-Lapjv is slower for running A100.txt about 50.00%. This result shows that if the number of tasks is expanded to a large scale size, Table 2 depicts that trend of computation time curve of PP-Lapjv can be the faster one. The total cost solution of A100.txt to be displayed "305" on workspace window of MATLAB shown as Fig. 1a and 1b.

Computing k Times of PP Algorithm

PP algorithm need k-times for computing the transferred reduced cost matrix on the kth iteration. The k-PP algorithm determines k-time on different number ranges shown in Table 3. These k can be appropriately difference values depend upon problem structure of c_{ij}.

The Average Computation Time for Solving Large-Scale AP

When the large-scale LAP was started from problem size 2000 to problem size 26000 and range of unit cost [1,25], the computation time enlarged rapidly. The comparative performance between Lapjv and PP-Lapjv can be illustrated in Fig. 2.

Table 1. Total cost (Z) and the average computation time (s.) of Lapjv and Munkres codes

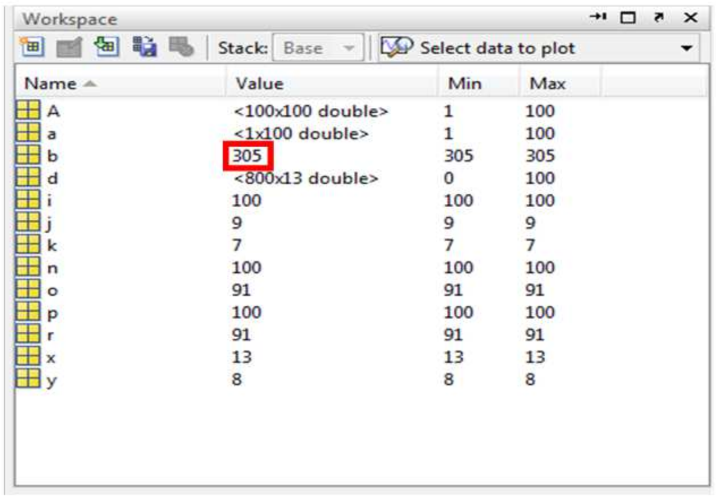
Data	Lapjv	Munkres
A100.txt.(Z)	305.000	305.000
A300.txt.(Z)	626.000	626.000
A100.txt.(s.)	0.018	0.051
A300.txt.(s.)	0.165	0.375
No. of samples	10.000	10.000

Table 2. Total cost (Z) and the average computation time (s.) of Lapjv and PP-Lapjv code

Data	Lapjv	PP-Lapjv
A100.txt (Z)	305.000	305.000
A300.txt (Z)	626.000	626.000
A100.txt (s.)	0.018	0.027
A300.txt(s.)	0.165	0.148
No. of samples	10.000	10.000

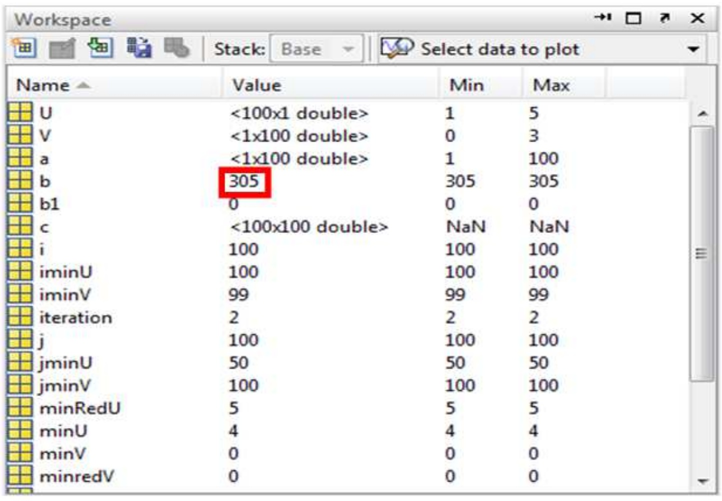
Table 3. k-Time for constructing the transferred reduced cost matrix

Number range	[1,10]	[1,25]	[1,1000]	[1,1700000]
k-time	2	2	2	7



Name	Value	Min	Max
A	<100x100 double>	1	100
a	<1x100 double>	1	100
b	305	305	305
d	<800x13 double>	0	100
i	100	100	100
j	9	9	9
k	7	7	7
n	100	100	100
o	91	91	91
p	100	100	100
r	91	91	91
x	13	13	13
y	8	8	8

(a)



Name	Value	Min	Max
U	<100x1 double>	1	5
V	<1x100 double>	0	3
a	<1x100 double>	1	100
b	305	305	305
b1	0	0	0
c	<100x100 double>	NaN	NaN
i	100	100	100
iminU	100	100	100
iminV	99	99	99
iteration	2	2	2
j	100	100	100
jminU	50	50	50
jminV	100	100	100
minRedU	5	5	5
minU	4	4	4
minV	0	0	0
minredV	0	0	0

(b)

Fig. 1. (a) Window of MATLAB workspace when Lapjvalgorithm operates with A100.txt (b) Window of MATLAB workspace when PP-Lapjvalgorithm operates with A100.txt

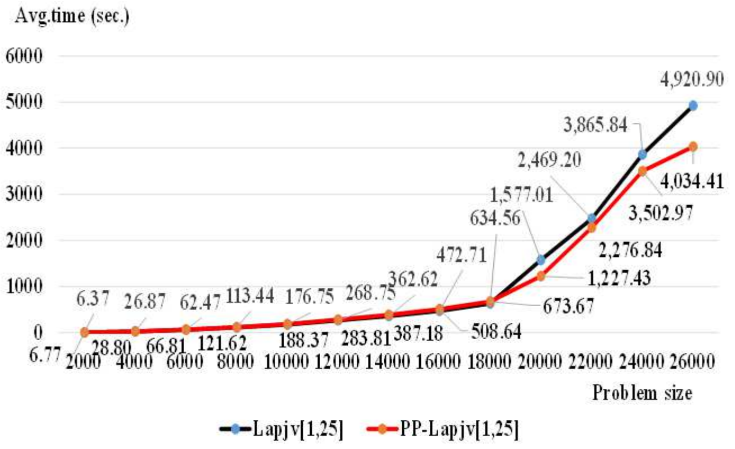


Fig. 2. The average running time of Lapjv.m file in Lapjvand PP-Lapjv algorithms on integer range [1,25]

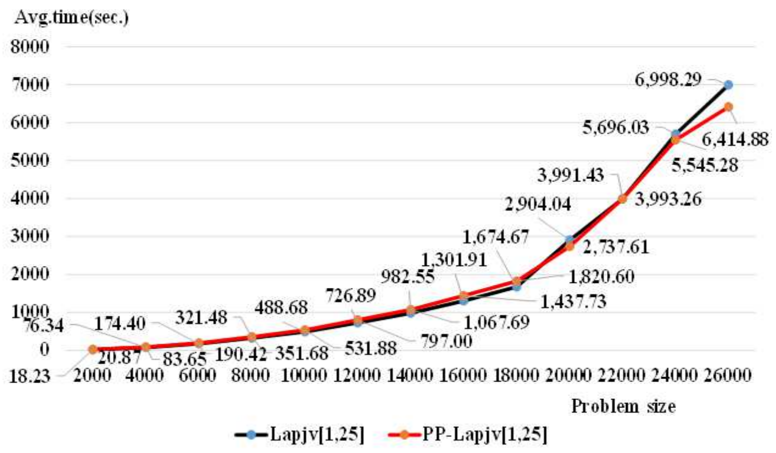


Fig. 3. The average total computation time of Lapjv and PP-Lapjv algorithms on integer range [1,25]

In the above figure, at n>18000, the trend of the average computation time in Lapjv.m file of PP-Lapjvalgorithm begins to improve at n = 20000, 22000, 24000 and 26000 about 22.17, 7.79, 9.39 and 18.01%, respectively. On the other hand, at n <4000, the average time solved by Lapjv.m file of PP-Lapjvalgorithm are slower than the direct Lapjvalgorithm. This result can be supported from the fact that PP-algorithm provided the kth reduced cost matrix using the Lapjv.m file to find the optimal solution from the admissible cells in the sparse kth transferred reduced cost only whereas Lapjv.m file of Lapjvalgorithm must start to operate with a fully dense cost matrix.

The running time of PP-Lapjv in Fig. 2 combines with time for constructing the kth transferred reduced cost matrix and time for generating random unit cost matrix, the result in Fig. 3 depicts the average total computation time of PP-Lapjvalgorithm on number range [1,25] to be faster than Lapjvalgorithm about 5.73, 2.65 and 8.34% at problem sizes 20000, 24000 and 26000, respectively.

Shortest augmenting path algorithm usually consists of two part: Firstly, compute a primal partial solution and a dual feasible solution which satisfy the complementary slackness conditions and for the second part, the primal solution is added one row-column assignment at a time till the current primal-dual solution is iterated in order to hold the complementary slackness conditions and become feasible solution. PP algorithm also provides the first part for searching the admissible cells which are allocated in the transferred reduced cost matrix (step 2-4 of PP-Lapjvalgorithm) to replace a fully dense cost matrix.

Effect of Integer Number Ranges

Integer number of a dense cost matrix is interesting in this experiment. Performance of Lapjvalgorithm is measured and divided into three parts as time for generating random unit cost matrix, time for running Lapjv.m file and total running time whereas the performance of PP-Lapjvalgorithm to be divided into four parts which include three parts as the same time for running Lapjvalgorithm and one part as time for constructing the kth transferred reduced cost matrix. However, this study is interested in time for running Lapjv.m file and its total running time which are illustrated as Table 4 and 5.

Table 5 depicts that the average total time of both algorithms trend to expand continuously when extending the upper bound of range cost; therefore, number range impacts significantly for running time performances. This behavior of PP-Lapjvalgorithm can be explained in Fig. 4.

In Fig. 4,theaverage total time of Lapjvalgorithm at problem sizes 18,000 and 30,000 using range [1, 10^6] are greater than range [1, 10^3] about 53.51 and 36.27%, respectively and the average total time of PP-Lapjvalgorithm at problem sizes 18,000 and 30,000 using range [1, 10^6] are also greater than range [1, 10^3] about 63.51 and 51.33%, respectively. The enlargement of the average total time corresponds with time of iteration. The unit cost of PP-Lapjvalgorithm on range [1, 10^6] must be executed with 7 iterations for constructing the 7th reduced cost matrix whereas the unit cost of Lapjvalgorithm on number range [1, 10^3] to be merely operated in two iterations.

Comparative Study

Performances of both algorithms can be compared in order to identify better algorithm for solving large scale problem sizes. From results of the last section, it indicates that the unit cost affects to the running time; hence, analyzing on various number range is significant certainly. Figures 5-10 depict comparison of both algorithms at number ranges [1,10], [1, 10^3], [1, 10^6] as follows.

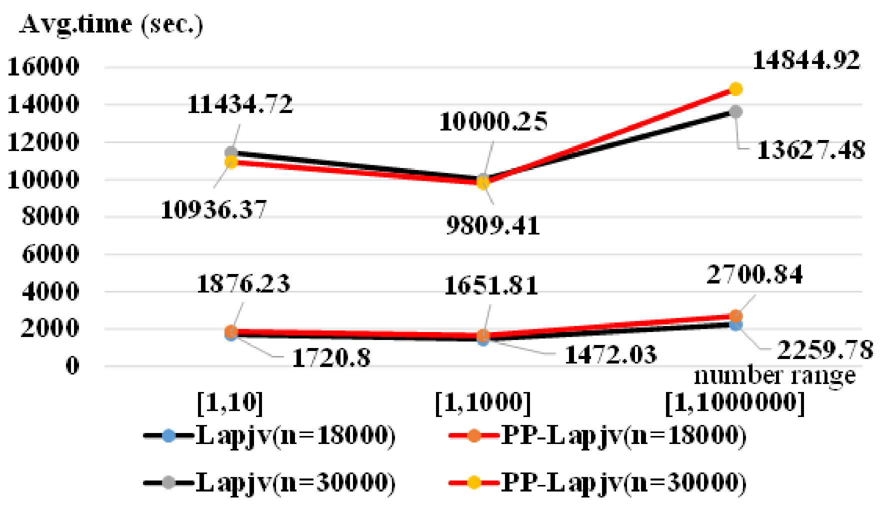


Fig. 4. The average total running time of Lapjv and PP-Lapjvalgorithms at problem sizes 18,000 and 30,000

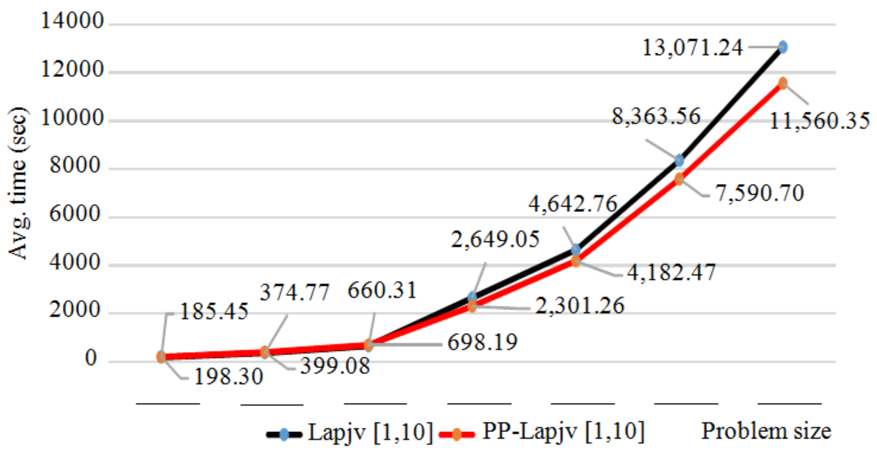


Fig. 5. The average running time of Lapjv.m file in Lapjv and PP-Lapjv algorithms on integer range [1,10]

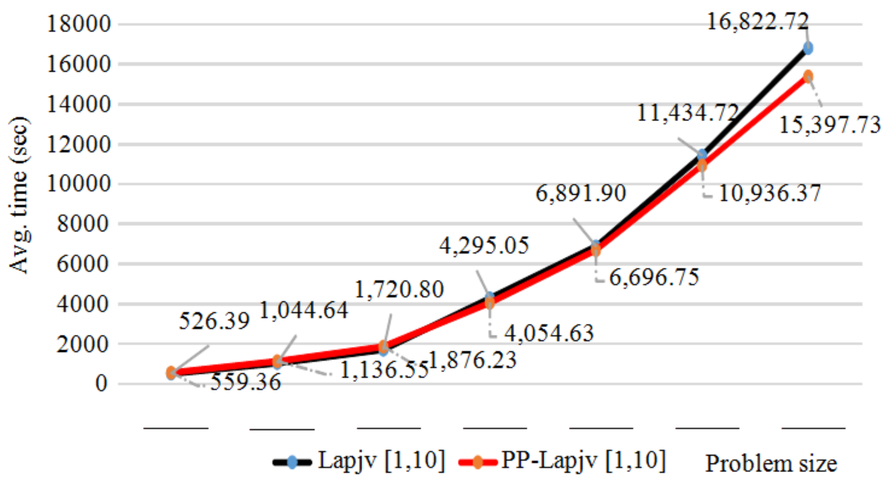


Fig. 6. The average total computation time of Lapjv and PP-Lapjv algorithms on range [1,10]

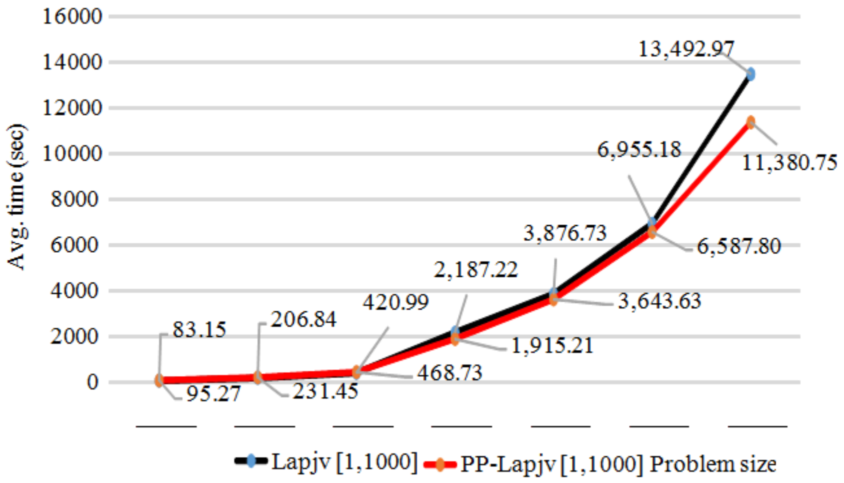


Fig. 7. The average running time of Lapjv.m file in Lapjv and PP-Lapjv algorithms on integer range [1, 10^3]

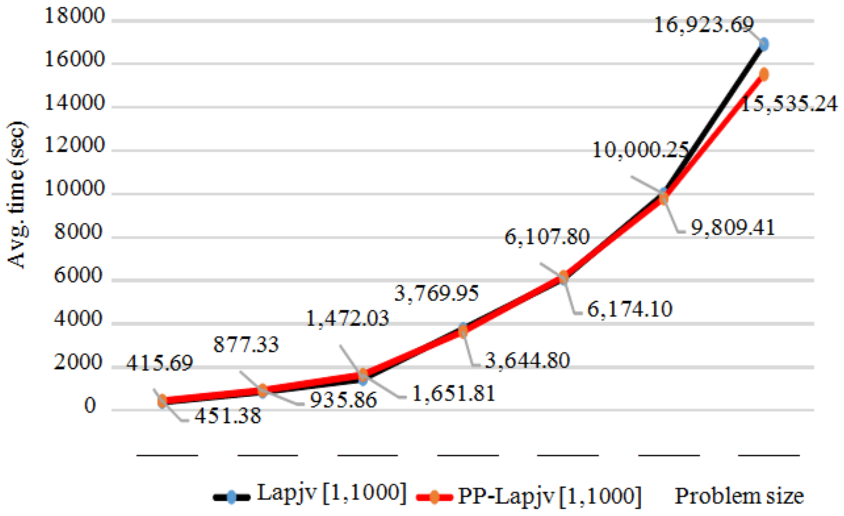


Fig. 8. The average total computation time of Lapjvand PP-Lapjv algorithms on range [1, 10^3]

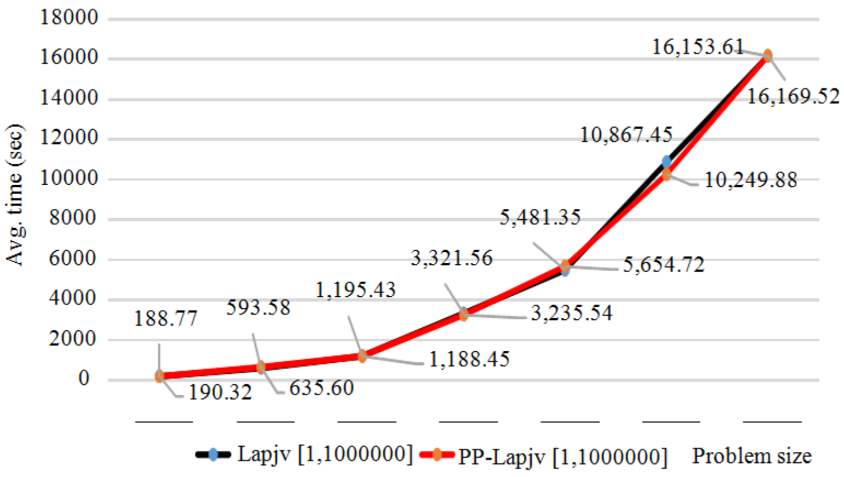


Fig. 9. The average running time of Lapjv.m file in Lapjv and PP-Lapjv algorithms on range [1, 10^3]

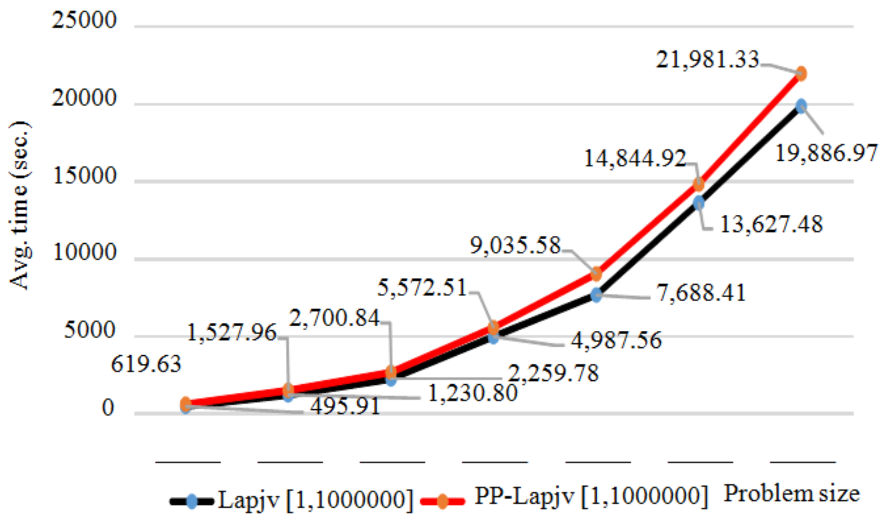


Fig. 10. The average total computation time of Lapjv and PP-Lapjv algorithms on range [1, 10^3]

Table 4. The average computation time of Lapjv.m file in Lapjv and PP-Lapjvalgorithms on range [1,10], [1, 10^3], [1, 10^6]

Algorithm	Lapjv			PP-Lapjv		
Number range	[1,10]	[1, 10^3]	[1, 10^6]	[1,10]	[1, 10^3]	[1, 10^6]
n = 10,000	185.45	83.14	188.77	198.3	95.27	190.32
14,000	374.77	206.84	593.58	399.08	231.45	635.60
18,000	660.31	420.99	1195.43	698.19	468.73	1188.45
22,000	2649.05	2187.22	3321.56	2301.26	1915.21	3235.54
26,000	4642.76	3876.73	5481.35	4182.47	3643.63	5654.72
30,000	8363.56	6955.18	10867.45	7590.69	6587.80	10249.88
34,000	13071.24	13492.97	16153.61	11560.35	11380.75	16169.52
No. of samples	10.00	10.00	10.00	10.00	10.00	10.00

Table 5. The average total computation time of Lapjv and PP- Lapjvalgorithms on range [1,10], [1, 10^3], [1, 10^6]

Algorithm	Lapjv			PP-Lapjv		
Number range	[1,10]	[1, 10^3]	[1, 10^6]	[1,10]	[1, 10^3]	[1, 10^6]
n = 10,000	526.39	415.69	495.91	559.36	451.38	619.63
14,000	1044.64	877.33	1230.80	1136.55	935.86	1527.961
18,000	1720.80	1472.03	2259.78	1876.23	1651.81	2700.84
22,000	4295.05	3769.95	4987.56	4054.63	3644.80	5572.51
26,000	6891.90	6107.80	7688.41	6696.75	6174.10	9035.58
30,000	11434.72	10000.25	13627.48	10936.37	9809.41	14844.92
34,000	16822.72	16923.69	19886.97	15397.73	15535.24	21981.33
No. of samples	10	10	10	10	10	10
The kth reduced cost	-	-	-	2	2	7

Figure 5 depicts that the running time of Lapjv.m file in PP-Lapjvalgorithm to be faster than the direct Lapjvalgorithm about 13.13, 9.91, 9.24 and 11.56% at problem sizes 22000, 26000, 30000 and 34000, respectively. When n>18000, the running time of Lapjv.m file in PP-Lapjvalgorithm can be improved efficiently.

The running time of PP-Lapjv in Fig. 5 combines with time for constructing the kth transferred reduced cost matrix and time for generating random unit cost matrix, whereas Fig. 6 depicts the average total computation time of PP-Lapjvalgorithm on number range [1,10] to be faster than Lapjvalgorithm about 5.60, 2.83, 4.36 and 8.47% at problem sizes 22000, 26000, 30000 and 34000, respectively.

In Fig. 7, the running time of Lapjv.m file in PP-Lapjvalgorithm to be faster than Lapjvalgorithm about 12.44, 6.01, 5.28 and 15.65% at problem sizes 22000, 26000, 30000 and 34000, respectively. Therefore, the running time of Lapjv.m file in PP-Lapjv can be improved efficiently.

In Fig. 8, the average total computation time of PP-Lapjvalgorithm on number range [1, 10^3] to be faster than Lapjvalgorithm about 3.32, 1.91 and 8.20% at problems sizes 22000, 30000 and 34000, respectively.

Figure 9 depicts that the running time of Lapjv.m file in PP-Lapjvalgorithm to be faster than Lapjvalgorithm about 0.58, 2.59 and 5.68% at problem sizes 18000, 22000 and 30000, respectively. However, the running time of Lapjv.m file in PP-Lapjvalgorithm cannot be improved significantly.

In Fig. 10, PP-Lapjvalgorithm cannot solve efficiently at all problem sizes with range [1,10^6] since the running time of Lapjv.m file of PP-Lapjvalgorithm (Fig. 9) overcomes Lapjvalgorithm slightly. So, the average total running time of PP-Lapjvalgorithm is still slower.

From Fig. 5-10, PP-Lapjvalgorithm can enhance better performances for solving large scale LAP when the problem size is expanded over 18000 and its unit cost matrix to be in range [1,1000].

Maximum Problem Size Solving

Large scale LAP can be solved with problem size expanded until MATLAB command window shows "out of memory". Memory spaced used by MATLAB can be displayed on command window by 'memory' command. For running Lapjv and PP-Lapjvalgorithms, actual memory space used by MATLAB to be illustrated in Fig. 11.

In Fig. 11, memory used for running Lapjv and PP-Lapjvalgorithms differ slightly. When the problem size n is expanded over the value of 34000, its memory space used by MATLAB closes to the maximum problem size. Expanding n experimentally, until MATLAB command window displays 'out of memory' as shown in Fig. 12.

When the problem size is expanded to 35000, Lapjvalgorithm fails to operate due to out of memory space used. Therefore, the maximum problem size was possible on 34,000. Figure 12 depicts that actual memory space used by MATLAB should be less than 9,824 MB.

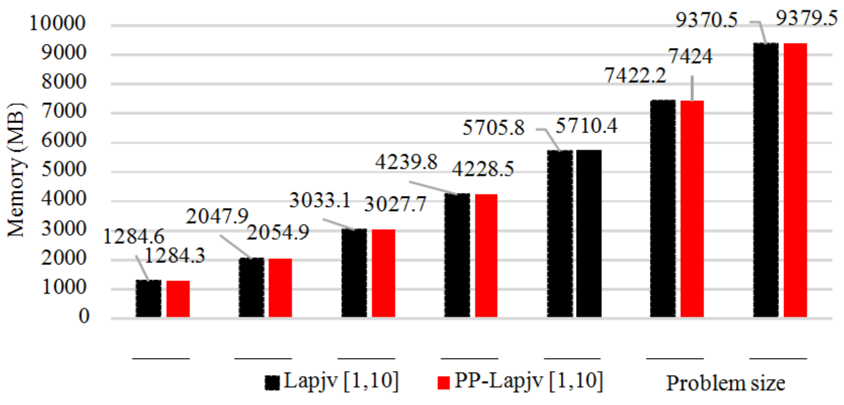
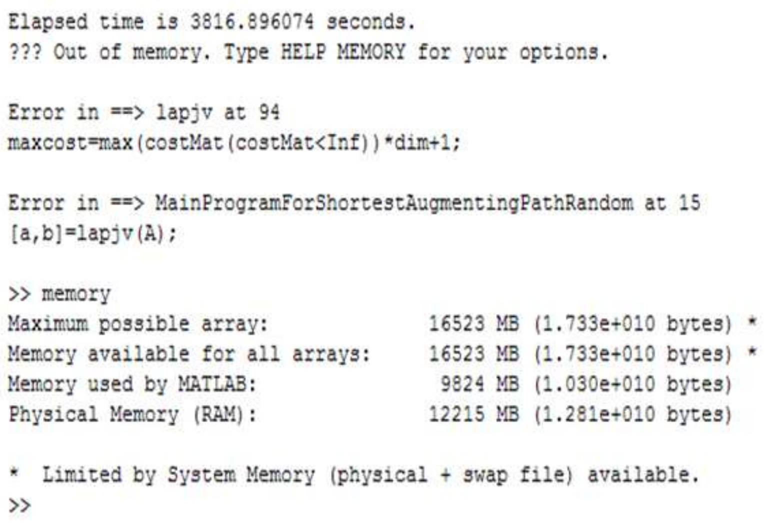


Fig. 11. Actual memory space used by MATLAB of Lapjv and PP-Lapjv algorithms



```
Elapsed time is 3816.896074 seconds.
??? Out of memory. Type HELP MEMORY for your options.

Error in ==> lapjv at 94
maxcost=max(costMat(costMat<Inf))*dim+1;

Error in ==> MainProgramForShortestAugmentingPathRandom at 15
[a,b]=lapjv(A);

>> memory
Maximum possible array:              16523 MB (1.733e+010 bytes) *
Memory available for all arrays:     16523 MB (1.733e+010 bytes) *
Memory used by MATLAB:                9824 MB (1.030e+010 bytes)
Physical Memory (RAM):               12215 MB (1.281e+010 bytes)

*  Limited by System Memory (physical + swap file) available.
>>
```

Fig. 12. Memory space used by MATLAB at problem size 35,000

Limitation of Lapjv code in this experiment cannot solve large scale LAP on a dense cost matrix with problem size over 34,000. This limitation can be improved by using the preprocessing algorithm for constructing an admissible sparse matrix. This sparse matrix will be solved efficiently by a sparse Lapjv code which can solve with less memory space used. Therefore, this sparse code could be solve large scale LAP with more problem size. However, a sparse Lapjv code was just written in FORTRAN and PASCAL language (Jonker and Volgenan, 1987).

Normally, Lapjvalgorithm was designed to solve efficiently with a fully dense matrix. PP-Lapjvalgorithm can generate the admissible cells matrix which is transferred to a sparse cost matrix. If Lapjv code is also improved to solve a sparse matrix, this code will execute with both less time-consuming and the higher maximum problem size.

Authors are interested in applying the PP algorithm to improve the performance of both primal-dual and simplex-based algorithms for solving transportation and shortest path problems.

Conclusion

Nearly thirty years ago, LAP has been efficiently solved by the successful shortest augmenting path algorithm; nevertheless, this study proposes the PP algorithm to improve the performance of the successful algorithm. The problem size was extended, the PP algorithm generated the successive admissible cells to allocate in a sparse cost matrix replacing a fully dense cost matrix in order to solve with both less time-consuming and less memory space used. On the other hand, Lapjvalgorithm executed on a fully dense cost matrix with more time-consuming. However, if the integer number range is expanded, the PP algorithm will be executed a more k iterations leading to an increase on running time. In this case, performances of PP-Lapjvalgorithm is inefficient for solving LAP on a large related number range. The problem size and memory space used are related reversely. The proposed algorithm can utilize with much less memory space used and it can solve the problem with larger sizes as compared to the direct approach.

Acknowledgement

The authors would like to thank the staffs of department of industrial engineering in Kasetsart University for supporting this work.

Author's Contributions

Warut Boonphakdee: Designed, collected and checked the analyzed data and wrote the manuscript.

Peerayuth Charnsethikul: Designed the research plan and organized the study and reviewed this manuscript.

Ethics

The authors announce that this is an original research and confirm that no ethical issues are involved.

References

Beasley, J.E., 1990. OR-Library: Distributing test problems by webpage.

Boonphakdee, W. and P. Charnsethikul, 2014. Solving the linear programming model of large-scale transportation and assignment problems using the column generation technique. J. Operat. Res., 2: 10-21.

Burkard, R.E. and E. Cela, 1999. Linear Assignment Problem and Extensions. In: Handbook of Combinational, Du, D.Z. and P.M. Pardalos (Eds.), Kluwer Acadamic Publishers, Dordrecht, ISBN-10: 978-1-4419-4813-7, pp: 75-149.

Burkard, R.E., M.D. Amico and S. Martello, 2009. Linear sum assignment problem in Assignment Problem. 1st Edn., Society for Industrial and Applied Mathematices, Philadelphia, ISBN-13: 978-0898716634.

Cao, Y., 2011. Distributing code by webpage.

Cao, Y., 2013. Distributing codes by webpage.

Fang, S. and S. Puthenpura, 1993. Linear Optimization and Extension: Thoery and Algorithm. 1st Edn., Prentice Hall College Div, New Jersey, ISBN-10: 0139152652, pp: 62.

Hane, C., C. Barnhart, E.L. Johnson, R. Marsten and G.L. Nemhauser *et al*., 1995. The fleet assignment problem: Solving a large-scale integer program. Math. Programm., 70: 211-232. DOI: 10.1007/BF01585938

Hung, M.S. and W.O. Rom, 1980. Solving the assignment problem by relaxation. Operat. Res., 28: 969-982. DOI: 10.1287/opre.28.4.969

Jonker, R. and A. Volgenant, 1987. Distributing codes by webpage.

LeBlanc, L.J., R.V. Helgason and D.E. Boyce, 1985. Improved efficiency of the frank-wolfe algorithm for convex network programs. Transport. Sci., 19: 445-462. DOI: 10.1287/trsc.19.4.445

Zhu, X., C. Santos, J. Ward, D. Beyer and S. Singhal, 2004. Resource assignment for large-scale computing utilities using mathematical programming. Internet systems and storage laboratory and HP laboratory and HP laboratories Palo Alto.