Research Paper

An efficient exact algorithm for the Minimum Latency Problem

Ha BANG BAN[1], Kien NGUYEN[2], Manh CUONG NGO[3] and Duc NGHIA NGUYEN[4]
[1,3,4] Hanoi University of Science and Technology
[2] National Institute of Informatics

ABSTRACT
The Minimum Latency Problem (MLP) is a class of combinational optimization problems that has many practical applications. In the general case, the MLP is proved to be NP-hard. One of the approaches to solve the problem is using exact algorithms. However, the algorithms which were recently proposed are applied only to the problems with small size, i.e., 26 vertices. In this paper, we present a new exact algorithm to solve the MLPs with a larger size. Our algorithm is based on the branch and bound method and it has two new rules that improve the pruning technique. We have evaluated the algorithm on several data sets. The results show that the problems up to 40 vertices can be solved exactly.

KEYWORDS
Minimum Latency Problem (MLP), exact algorithm, branch and bound method

1 Introduction

The minimum latency problem is also known in the literature as the delivery man problem or the traveling repairman problem. In the general case, the problem was described as NP-hard, and unless $P = NP$, a polynomial time approximation scheme is unlikely to exist [12]. However, the reduction from the general MLP to the problem in a metric case can be done by a simple transformation as in [15]. The metric case reflects a condition in which a complete graph with distances between vertices satisfying the triangle inequality. In this paper, we consider the problem in the metric case, and formulate the MLP as follows:
Given a complete graph K_n with the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and a symmetric distance matrix $C = \{c(v_i, v_j) \mid i, j = 1, 2, \ldots, n\}$, where $c(v_i, v_j)$ is the distance between two vertices v_i and v_j. Suppose that $T = \{v_1, \ldots, v_k, \ldots, v_n\}$ is a tour in K_n. Denote by $P(v_1, v_k)$ the path from v_1 to v_k on this tour and by $l(P(v_1, v_k))$ its length. The latency of a vertex $v_k (1 < k \leq n)$ on T is the length of the path from starting vertex v_1 to v_k:

$$l(P(v_1, v_k)) = \sum_{i=1}^{k-1} c(v_i, v_{i+1}).$$

The total latency of the tour T is defined as the sum of latencies of all vertices

$$L(T) = \sum_{k=2}^{n} l(P(v_1, v_k))$$

The minimum latency problem asks for a minimum latency tour, which starts at a given vertex v_1 and visits each vertex in the graph exactly once.

Minimizing $L(T)$ arises in many practical situations because whenever a server (i.e., a repairman or a disk head) has to accommodate a set of requests with their minimal total (or average) waiting time [6], [12]. In the scope of our paper, we are interested in finding the minimum latency in a tour other than a cycle. In this case, the repairman need not to return v_1. This variant can be seen in [2]–[4], [6], [7], [9], [14].

The MLP can be solved in polynomial time in several cases, for example when the graph of the problem is a path [1], [8], an edge-unweighted tree [11], a tree

Received June 25, 2012; Revised October 1, 2012; Accepted December 4, 2012.
1) *bangbh@soict.hut.edu.vn*, 2) *kienng@nii.ac.jp*, 3) *kirimaru148@gmail.com*, 4) *nghiand@soict.hut.edu.vn*
DOI: 10.2201/NiiPi.2013.10.10

with diameter 3 [6], a tree with a constant number of leaves [10](for example with constant k, there exists an exact algorithm with complexity $O(n^k)$). In the general case, the problems can be solved by using approximation algorithms or exact algorithms. In order to describe related works we denote an approximation algorithm as p-approximation when the algorithm finds the solution at most p times worse than the optimal solution. Here p is an approximation ratio with constant value. Blum et al. [6] provided a 144-approximation in the metric case and an 8-approximation for weighted trees. Goemans et al. [9] presented a 21.55-approximation in the metric case and a 3.59-approximation in the tree case. Arora et al. [4] then gave a 17.24-approximation in the metric case. The approximation algorithm of Archer et al. [2] obtained a 7.18-approximation in the metric case and a 3.01-approximation on the tested Euclidean instances. Kamalika et al. [7] presented a 3.59-approximation for the metric case. Recently, Archer et al. [3] improved an approximation ratio for trees to 3.03.

Beside the approximation algorithms, several exact algorithms have been proposed for solving the problems. In [14], Wu presented a dynamic programming algorithm for the MLP, but the algorithm was very time-consuming. Wu et al. then improved the algorithm and proposed a more efficient one [15]. In the new algorithm, they have designed a lower bound and utilized a combination of the dynamic programming and a branch and bound method. The experimental results showed that the latter algorithm in [15] is much more efficient than the previous one [14]. However, the better algorithm was evaluated only on instances with small size, i.e, 26 vertices. Since there was a lack of experiments on larger instances, the real efficiency of their algorithm cannot be evaluated. We choose the algorithm in [15] as a baseline in our research, and call it Wu et al.'s.

In this paper, we propose a new exact algorithm based on the branch and bound method. In our algorithm, we improve a pruning technique by introducing two new rules. The efficiency of the algorithm was extensively evaluated on both random test data and real test data. The results indicate that our algorithm exactly solves the problems up to 40 vertices. Moreover the algorithm was also compared with Wu et al.'s in the case of real test data with small size. The comparision results show that our algorithm consumes less time than Wu et. al.'s.

The rest of this paper is organized as follows: section 2 presents the proposed algorithm. The experimental results are reported in section 3, and finally section 4 concludes the paper.

2 The proposed algorithm

On the tour $T = (v_1, v_2, v_3, ..., v_n)$, we realize that the first arc (v_1, v_2), the second arc (v_2, v_3),..., and the last arc (v_{n-1}, v_n) gives a contribution $(n-1) \times c(v_1, v_2)$, $(n-2) \times c(v_2, v_3)$,..., $1 \times c(v_{n-1}, v_n)$, respectively to the latency $L(T)$. Therefore, the latency of T can be rewritten as follows:

$$L(T) = \sum_{k=1}^{n-1} (n-k)c(v_k, v_{k+1})$$

We denote $F = (v_1, v_2, \ldots, v_k)$ as a prefix subtour and $B = (v_{k+1}, v_{k+2}, \ldots, v_n)$ as a suffix subtour of the tour T. Then we describe several lemmas in order to construct the algorithm. The lemmas are proven by contradiction.

Lemma 1. *Let $T = (v_1, v_2, \ldots, v_{k-1}, v_k, \ldots, v_n)$ be an optimal tour. If the following condition*

$$l(P(v_{k-1}, v_k)) = l(P(v_1, v_k)) - l(P(v_1, v_{k-1}))$$

holds, then $l(P(v_{k-1}, v_k))$ is the length of the shortest path from v_{k-1} to v_k.

Proof. Assume that $l(P(v_{k-1}, v_k)) > m(v_{k-1}, v_k)$, where $m(v_{k-1}, v_k)$ is the length of the shortest path from v_{k-1} to v_k. We have

$$L(T) > \sum_{j=2, j\neq k}^{n} l(P(v_1, v_j)) + l(P(v_1, v_{k-1})) + m(v_{k-1}, v_k)$$

We denote $l(P'(v_1, v_k)) = l(P(v_1, v_{k-1}) + m(v_{k-1}, v_k)$. It is clear that there exists a tour T' which has

$$L(T') = \sum_{j=2, j\neq k}^{n} l(P(v_1, v_j)) + l(P'(v_1, v_k)).$$

Therefore, $L(T) > L(T')$. This implies that T is not the optimal tour. □

Lemma 2. *Let $V = (v_1, v_2, ..., v_k, ..., v_n)$ be the vertex set and $F = (v_1, v_2, \ldots, v_j, v_{j+1}, \ldots, v_k)$ be a prefix subtour. If there exists a vertex $v_p \in V' = V \setminus F$ $(k+1 \le p \le n)$ and an index j $(1 < j < k)$ such that the following condition*

$$(n-j)c(v_j, v_{j+1}) > (n-j)c(v_j, v_p) + (n-j-1)c(v_p, v_{j+1}) \quad (1)$$

holds, then F cannot be extended to an optimal tour.

Proof. Assume that $T = (v_1, v_2, \ldots, v_j, v_{j+1}, \ldots, v_k, v_{k+1}, v_{k+2}, v_{k+3}, v_{k+4}, \ldots, v_n)$ is an optimal tour. We have $F = (v_1, v_2, \ldots, v_j, v_{j+1}, \ldots, v_k)$ and $B = (v_{k+1}, v_{k+2}, v_{k+3}, v_{k+4}, ..., v_n)$. We insert $v_p \in B$ $(p = k+2)$

between $(v_j, v_{j+1}) \in F$ to obtain $T^{'} = (v_1, v_2, \ldots, v_j, v_p, v_{j+1}, \ldots, v_k, v_{k+1}, v_{k+3}, v_{k+4}, \ldots, v_n)$. The latency of tour T, $T^{'}$ can be rewritten as follows:

$$L(T) = \sum_{k=1}^{n-1}(n-k)c(v_k, v_{k+1})$$

$$\begin{aligned} L(T^{'}) = & \sum_{k=1}^{j-1}(n-k)c(v_k, v_{k+1}) + (n-j)c(v_j, v_p) \\ & + (n-j-1)c(v_p, v_{j+1}) \\ & + \sum_{k=j+1}^{p-2}(n-k-1)c(v_k, v_{k+1}) \\ & + (n-k-2)c(v_{p-1}, v_{p+1}) \\ & + \sum_{k=p+1}^{n-1}(n-k)c(v_k, v_{k+1}) \end{aligned}$$

If we denote $\triangle T = L(T) - L(T^{'})$, we obtain

$$\begin{aligned} \triangle T = & (n-j)c(v_j, v_{j+1}) - (n-j)c(v_j, v_{k+2}) \\ & - (n-j-1)c(v_{k+2}, v_{j+1}) + c(v_{j+1}, v_{j+2}) \\ & + c(v_{j+2}, v_{j+3}) + \ldots + c(v_k, v_{k+1}) + (n-k \\ & - 1)c(v_{k+1}, v_{k+2}) + (n-k-2)c(v_{k+2}, v_{k+3}) \\ & - (n-k-2)c(v_{k+1}, v_{k+3}). \end{aligned} \tag{2}$$

On the other hand, by the triangle inequality and assuming that $c(v_{k+1}, v_{k+2}) > 0$, we have

$$(n-k-1)c(v_{k+1}, v_{k+2}) + (n-k-2)c(v_{k+2}, v_{k+3}) - (n-k-2)c(v_{k+1}, v_{k+3}) > 0$$

So if condition (1) is satisfied, then $\triangle T > 0$. This implies that T is not the optimal tour. Similar arguments hold for the case when p is not equal to $k+2$ □

Lemma 3. *Let $V = (v_1, v_2, ..., v_k, ..., v_n)$ be the vertex set and $F = (v_1, v_2, ..., v_j, v_{j+1}, ..., v_k)$ be a prefix subtour. If there exists a vertex $v_p \in V^{'} = V \setminus F$ $(k+1 \leq p \leq n)$ and an index j $(1 < j < k)$ such that the following condition*

$$\begin{aligned} & (n-j)c(v_j, v_{j+1}) + l(P(v_{j+1}, v_k)) + c(v_k, v_p) \\ & > (n-j)c(v_j, v_p) + (n-j-1)c(v_p, v_{j+1}) \end{aligned} \tag{3}$$

holds, then F cannot be extended to an optimal tour.

Proof. Assume that $T = (v_1, v_2, ..., v_j, v_{j+1}, ..., v_k, v_{k+1}, v_{k+2}, v_{k+3}, v_{k+4}, ..., v_n)$ is an optimal tour. We define $F = (v_1, v_2, ..., v_j, v_{j+1}, ..., v_k)$ and $B = (v_{k+1}, v_{k+2}, v_{k+3}, v_{k+4}, ..., v_n)$ is a prefix, and a suffix tour of T, respectively. If we insert v_p $(p = k+2) \in B$ between $(v_j, v_{j+1}) \in F$, we have the new tour $T^{'} = (v_1, \ldots, v_j, v_p, v_{j+1}, \ldots, v_k, v_{k+1}, v_{k+3}, v_{k+4}, ..., v_n)$. According to the proof of lemma 2, we obtain a similar $\triangle T$:

$$\begin{aligned} \triangle T = & (n-j)c(v_j, v_{j+1}) - (n-j)c(v_j, v_{k+2}) \\ & - (n-j-1)c(v_{k+2}, v_{j+1}) + l(P(v_{j+1}, v_k)) \\ & + c(v_k, v_{k+1}) + c(v_{k+1}, v_{k+2}) + (n-k-2) \\ & \times c(v_{k+1}, v_{k+2}) + (n-k-2)c(v_{k+2}, v_{k+3}) \\ & - (n-k-2)c(v_{k+1}, v_{k+3}) \end{aligned}$$

By the triangle inequality, we also get

$$\begin{aligned} & (n-k-2)c(v_{k+1}, v_{k+2}) + (n-k-2)c(v_{k+2}, v_{k+3}) \\ & \geq (n-k-2)c(v_{k+1}, v_{k+3}). \end{aligned}$$

and

$$c(v_k, v_{k+1}) + c(v_{k+1}, v_{k+2}) \geq c(v_k, v_{k+2}).$$

Therefore,

$$\begin{aligned} \triangle T \geq & (n-j)c(v_j, v_{j+1}) + l(P(v_{j+1}, v_k)) \\ & + c(v_k, v_{k+2}) - (n-j)c(v_j, v_{k+2}) \\ & - (n-j-1)c(v_{k+2}, v_{j+1}) \end{aligned}$$

If condition (3) is satisfied, then $\triangle T > 0$. This implies that T is not the optimal tour. Similar arguments hold for the case when p is not equal to $k+2$ □

Algorithm The Proposed Algorithm

Input: The starting vertex v_1.
Output: The optimal tour T.

1: **Initiate** UB;
2: **Try**$(v_1, 1, 0, 0)$;
3: **return** UB;

In the following part, we describe our proposed algorithm in detail.

The pseudo-code of our algorithm is shown in the Algorithm. The algorithm always records an upper bound (UB) of an optimal solution, which may be a solution of a feasible tour. In the initial stage, the value of UB is derived from either the nearest neighbour method in [13] or the GA algorithm in [5]. In the following stage, the algorithm invokes Procedure 1, which is a recursive function. In Procedure 1, in the trivial case a subtour (e.g., F) is pruned if its latency (denoted as $L(F)$) is not less than UB. In the other case, the lower bound of F (LB), which is an underestimation of any complete tour containing F as a prefix, is computed. When the LB is larger than the UB, the subtour is also pruned. As a result, the efficiency of the algorithm depends on how the UB and LB are computed. Therefore, we need to estimate the lower bound of a complete tour containing subtour F as a prefix. Assume that $E = \{e_1, e_2, \ldots, e_{n-k}\}$

Procedure 1 Try $(u, k, l(P(v_1, u)), L(F))$

Input: u is the vertex in k-th position of the tour, $l(P(v_1, u))$, $L(F)$.

```
if (L(F) >= UB)//L(F) is latency of subtour F then
    Exit(); //prune
end if
if (LB(F) >= UB)//LB(F) is calculated by (5) then
    Exit(); //prune
end if
if (k == n) then
    UB = L(F);
    Update the best tour as F;
end if
for v ∈ V do
    if v is not visited then
        select v;
        if (PruningRules(F, v)==true) then
            Exit(); //prune
        end if
        F = F ∪ v; // Add v
        Try(v, k + 1, l(P(v1, u)) + c(u, v), L(F) + l(P(v1, u)) +
        c(u, v));
        F = F \ v; //remove v
    end if
end for
```

is a set of the cheapest costs in matrix C such that $e_1 < e_2 < \ldots < e_{n-k}$. Note that $F = (v_1, v_2, \ldots, v_k)$, $L(v_i)$ is the latency of v_i and $m(v_{k-1}, v_k)$ is the length of the shortest path from v_{k-1} to v_k. Since K_n is the complete graph in the metric case, $m(v_1, v_k)$ is equal to $c(v_1, v_k)$. Then we have the following observation:

$$
\begin{aligned}
L(v_{k+1}) &= L(v_k) + m(v_k, v_{k+1}) \\
&= l(P(v_1, v_k)) + c(v_k, v_{k+1}) \\
&\geq l(P(v_1, v_k)) + e_1. \\
L(v_{k+2}) &= L(v_{k+1}) + m(v_{k+1}, v_{k+2}) \\
&= L(v_k) + m(v_k, v_{k+1}) + m(v_{k+1}, v_{k+2}) \\
&= L(v_k) + c(v_k, v_{k+1}) + c(v_{k+1}, v_{k+2}) \\
&\geq l(P(v_1, v_k)) + e_1 + e_2. \\
&\quad \ldots\ldots \\
L(v_n) &= L(v_k) + m(v_k, v_{k+1}) + m(v_{k+1}, v_{k+2}) \\
&\quad + \ldots + m(v_{n-1}, v_n) \\
&\geq l(P(v_1, v_k)) + e_1 + e_2 + \ldots + e_{n-k}.
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
L(T) &= L(F) + L(v_{k+1}) + L(v_{k+2}) + \ldots + L(v_n) \\
&\geq L(F) + (n-k)l(P(v_1, v_k)) + (n-k)e_1 \\
&\quad + (n-k-1)e_2 + \ldots + e_{n-k} \qquad (4)
\end{aligned}
$$

Procedure 2 PruningRules(F, v)

Input: F, v is the prefix subtour, and an unvisited vertex, respectively

Output: The variable *pruned* is true or false.

```
pruned = false;
for (i = 2; i < F.length; i + +) do
    w1 = (n − i)c(F[i], F[i + 1]);
    w2 = (n − i)c(F[i], v);
    w3 = (n − i − 1)c(v, F[i + 1]);
    if (w1 > w2 + w3) then
        pruned = true; //rule in lemma 2
        break;
    end if
    w4 = l(P(F[i + 1], F[F.length]));
    if ((w1 + w4 + c(F[F.length], v) > w2 + w3)) then
        pruned = true; //rule in lemma 3
        break;
    end if
end for
return pruned;
```

From (4), we obtain an estimation function for the lower bound:

$$
\begin{aligned}
LB &= L(F) + (n-k)l(P(v_1, v_k)) + (n-k)e_1 \\
&\quad + (n-k-1)e_2 + \ldots + e_{n-k}. \qquad (5)
\end{aligned}
$$

Beside that, when an unvisited vertex is added to F, the rules in Lemma 2 and 3 will be applied by invoking Procedure 2. If one of the rules is satisfied then F is pruned. The global variables used in the algorithm include UB, K_n and C_{ij}. In all procedures, each tour is represented by a list of n vertices $(v_1, v_2, \ldots, v_k, \ldots, v_n)$, where v_k is the k-th vertex to be visited in the tour and takes the k-th position in the list. The function *length* returns the number of elements in the list.

3 Experimental results

We have implemented the algorithm in C language to evaluate its performance. The experiments were conducted on a personal computer, which is equipped with an Intel Pentium IV 2.4 GHz CPU and 256 M bytes memory. The input data of the experiments includes two random and one real test data. In the experiments, our algorithm was implemented with two upper bounds. The first upper bound UB_1 was calculated by the nearest neighbour algorithm in [13]. The second one UB_2 was the solution of the genetic algorithm in [5]. We also use experimental results to evaluate efficiency of the algorithm in comparison against Wu et al.'s algorithm [15].

The results are shown in the last page. We denote EA_1, EA_2 as our algorithm with the different values of upper bound UB_1 and UB_2, respectively. BA is used for

Table 1 The results of the algorithms in the random test data 1.

Group	Group one			Group two			Group three		
Instance	test1	test2	test3	test4	test5	test6	test7	test8	test9
n	30	35	40	30	35	40	30	35	40
OS	2697	3209	3028	3172	4104	4070	4169	2659	2375
UB_1	4017	5126	4327	5028	5663	6382	6619	7767	3162
UB_2	2697	3209	3128	3172	4104	4070	4179	2759	2380
Time(EA_1)	3.4	24	125	4.0	25	121	3.4	26	118
Time(EA_2)	2.1	19	100	2.2	18	106	2.2	20	98

Table 2 The results of the algorithms in the random test data 2.

Group	Group one			Group two			Group three		
Instance	test1	test2	test3	test4	test5	test6	test7	test8	test9
n	30	35	40	30	35	40	30	35	40
OS	3625	3897	4521	3751	4012	4621	3451	4101	4652
UB_1	4321	5764	6354	5121	5641	5954	4232	5841	5865
UB_2	3625	3912	4521	3751	4065	4696	3451	4101	4695
Time(EA_1)	3.5	25	120	5.0	35	140	4.6	35	135
Time(EA_2)	2.0	20	100	3.0	30	120	2.0	30	120

Table 3 The results of the algorithms in Group1 of the partial instances.

Group	Eil51			St70			Eil76			Rat195		
Instance	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10	test11	test12
n	30	35	40	30	35	40	30	35	40	30	35	40
OS	4049	5165	5006	3435	6731	6574	3676	4247	4947	7759	9203	11224
UB_1	6126	7723	7485	5101	9423	9992	5588	6349	7396	16953	18688	25122
UB_2	4097	5165	5101	3495	6798	6574	3676	4247	5001	11309	12451	16741
Time(EA_1)	2.5	4.0	80	1.5	4.2	82	1.6	4.0	80	1.8	4.1	80
Time(EA_2)	2.0	3.0	62	2.5	3.1	68	2.0	3.5	65	2.1	3.2	62

Wu et al.'s algorithm in [15]. In Tables 1 to 5, the values in the third row is the size of the instance. The fourth row gives the total latency $L(T)$ for the optimal solution (OS). The fifth and sixth rows give the value of UB_1 and UB_2, respectively. The seventh and eighth rows give the running time of the EA_1 and EA_2 algorithms in minutes.

3.1 Experiment for random test data

Two random test data comprises non-Euclidean and Euclidean instances, and was named as random test data 1 and random test data 2, respectively. In the former one, the instances were generated artificially with an arc cost drawn from a uniform distribution. The values of the arc costs were integers between 1 and 100. In the latter one, the Euclidean distance between two vertices was calculated. The coordinates of the vertices were randomly generated according to a uniform distribution in a 200×200 square. We chose the number of vertices as n = 30, 35 and 40. For each value of n, we generated three different instances. Hence, each random test data included nine instances that were divided equally among three groups.

Each instance was tested ten times, and the results are illustrated in Table 1 and 2. The average value of the running times are shown in Fig. 1 and Fig. 2. We can conclude that a better upper bound makes the algorithm prune the bad branches more quickly. Since the upper bound is the solution of the GA algorithm, the average running times are shown significantly in comparison with the nearest neighbour upper bound method.

3.2 Experiment for real test data

The data instances chosen include Ulysses22, Fir26, and Gr24 from TSPLIB [16](where 22, 26, 24 are the number of vertices). Besides that, we added more real

Table 4 The results of the algorithms in Group2 of the partial instances.

Group	KroA100			KroB100			KroC100			Berlin52		
Instance	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10	test11	test12
n	30	35	40	30	35	40	30	35	40	30	35	40
OS	163478	183985	202756	142148	198334	235091	141581	207946	177070	71427	86252	91091
UB_1	246852	274138	304379	212099	301468	345584	208124	303601	265605	107141	130241	134815
UB_2	164512	187541	208451	142148	198652	251214	141598	212114	177985	71498	86541	91451
Time(EA_1)	5.1	14	90	5.0	15	90	4.5	16	90	4.0	14	88
Time(EA_2)	3.5	6.0	72	3.0	7.5	68	3.5	5.0	70	3.0	8.0	65

Table 5 The results of the algorithms in Group3 of the partial instances.

Group	Tsp225			Tss225			Pr76			Lin105		
Instance	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10	test11	test12
n	30	35	40	30	35	40	30	35	40	30	35	40
OS	10143	13549	17588	267200	373280	469965	873523	910679	959268	116606	114670	146662
UB_1	15012	20324	26382	403472	563653	690849	1284079	1375125	1448495	172577	169712	219993
UB_2	10143	13651	17784	268741	374151	470121	873598	910698	959298	116641	114700	146671
Time(EA_1)	5.0	15	91	5.5	14	92	4.6	14	91	4.5	14	87
Time(EA_2)	3.5	6.0	72	3.5	7.0	70	3.7	5.5	74	3.0	8	68

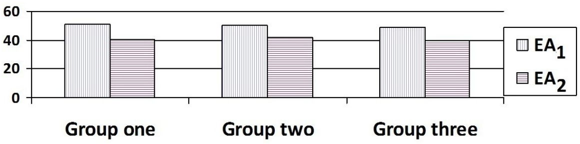

Fig. 1 The running time for the random test data 1.

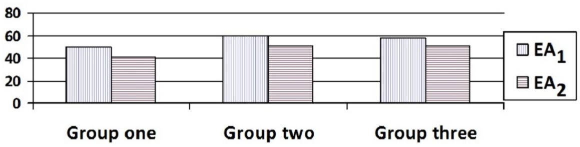

Fig. 2 The running time for the random test data 2.

instances by randomly choosing partial data from the larger instances in TSPLIB. The number of vertices of each partial instance is between thirty to forty. We divided the partial instances into three groups based on the following method: Suppose that X_{max}, X_{min} is the max, min abscissa of an instance, and Y_{max}, Y_{min} is the max, min ordinate of an instance, respectively. We denote $\triangle x = \frac{X_{max}-X_{min}}{n}$ and $\triangle y = \frac{Y_{max}-Y_{min}}{n}$. We have analyzed the data of TSPLIB and found that instances mostly belong to one of the following three groups. Group one with $\triangle x, \triangle y \leq 3$ where vertices are concentrated; group two, $\triangle x$, $\triangle y \geq 9$ where vertices are scattered; or group three where vertices are spaced in a special way such as along a line or evenly distributed. Specifically, group one includes instances extracted from Eil51, St70, Eil76, and Rat195. In group two, the instances are chosen from KroA100, KroB100, KroC100, and Berlin52. In the last group, the instances are from Tsp225, Tss225, Pr76, and Lin105.

In the experiment, the instances are also tested ten times. We show the results in Table 3 to 6 and Fig. 3 and 4. In Table 6, the second to fourth rows give the running time of the EA_1, EA_2 and BA algorithms in seconds, respectively.

Figure 3 shows the average running time of the EA_1 and EA_2 algorithms. We can see that if the upper bound is computed by the GA algorithm instead of the nearest neighbour method, the running time becomes much better. Figure 3 also shows that the running time of the algorithm in group one is much better than the one in the other groups. Obviously, our estimation function in (5) gives a better lower bound for the instances in group one. Hence, the algorithm is more efficient for the instances where vertices are concentrated. According to Fig. 1 to 3, our algorithm works for the real data much better than for the random data. The reason is that the real data is more structured. The results also indicate that the GA algorithm in [5] produces nearly optimal solutions. This implies that the GA algorithm is a promising approach for solving the problems.

In Fig. 4, we show the running time of our algorithm against the BA in case of solving the MLP with smaller size. The running time of the BA algorithm in Table 6 is derived from the experimental results in [15]. In this

Table 6 The results of the algorithms in the small size instances from TSPLIB.

Group	Ulysses22	Gr24	Gr26
Time(EA_1)	4.00	25.24	20.00
Time(EA_2)	3.00	22.12	15.00
Time(BA)	3.40	30.23	27.41

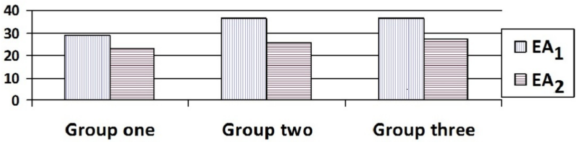

Fig. 3 The running time for the partial test data.

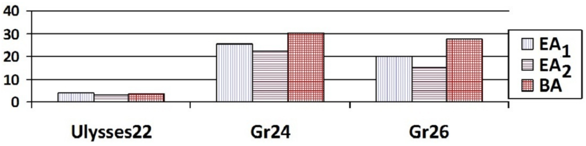

Fig. 4 The running time for the small size real data.

case, we also obtain that the EA_1 consumes more time than the EA_2, but the EA_1 consumes less time than the BA with all smaller instances.

4 Conclusion

In the paper, we have proposed a new exact algorithm based on the branch and bound method for solving the MLP problem in the metric case. Two new rules are applied to improve the pruning technique. The experimental results on the random data and the real data indicate that the algorithm exactly solve the problems up to 40 vertices. Additionally, the running time of the algorithm is superior to that of the Wu et al.'s algorithm in the case of solving the MLP problem with smaller size. However, our algorithm only applies for the problems up to 40 vertices. The upper limit depends on the efficiency of the algorithm, and with our algorithm, the result is generic for all computers. Enhancing the limitation using the branch and bound approach is still a challenge. This is our aim in future research.

References

[1] F. Afrati, S. Cosmadakis, C. Papadimitriou, G. Papageorgiou, and N. Papakostantinou, "The complexity of the traveling repairman problem," *J. ITA*, vol.20, no.1, pp.79–87, 1986.

[2] A. Archer, A. Levin, and D. Williamson, "A faster, better approximation algorithm for the minimum latency problem," *J. SIAM*, vol.37, no.1, pp.1472–1498, 2007.

[3] A. Archer and Anna Blasiak, "Improved approximation algorithms for the minimum latency problem via prize-collecting stroll," *Proc. ACM-SIAM SODA*, pp.429–447, 2010.

[4] S. Arora and G. Karakostas, "Approximation schemes for minimum latency problems," *Proc. ACM STOC*, pp.688–693, 1999.

[5] H.B. Ban and D.N. Nguyen, "Improved genetic algorithm for minimum latency problem," *Proc. ACM SOICT*, pp.9–15, 2010.

[6] A. Blum, P. Chalasani, D. Coppersmith, W. Pulleyblank, P. Raghavan, and M. Sudan, "The minimum latency problem," *Proc. ACM STOC*, pp.163–171, 1994.

[7] K. Chaudhuri, B. Goldfrey, S. Rao, and K. Talwar, "Path, tree and minimum latency tour," *Proc. IEEE FOCS*, pp.36–45, 2003.

[8] A. Garcia, P. Jodr, and J. Tejel, "A note on the traveling repairmen problem," *J. Networks*, vol.40, no.1, pp.27–31, 2002.

[9] M. Goemans and J. Kleinberg, "An improved approximation ratio for the minimum latency problem," *Proc. ACM-SIAM SODA*, pp.152–158, 1996.

[10] E. Koutsoupias, C. Papadimitriou, and M. Yannakakis, "Searching a fixed graph," *Proc. ICALP, LNCS*, vol.1099, pp.280–289, 1996.

[11] E. Minieka, "The delivery man problem on a tree network," *J. AOR*, vol.18, no.1, pp.261–266, 1989.

[12] S. Sahni and T. Gonzalez, "P-complete approximation problem," *J. ACM*, vol.23, no.3, pp.555–565, 1976.

[13] D.S. Johnson and L.A. McGeoch, "The traveling salesman problem: A case study in local optimization," *Local Search in Combinatorial Optimization*, John Wiley and Sons, Ltd., pp.215–310, 1997.

[14] B.Y. Wu, "Polynomial time algorithms for some minimum latency problems," *Inform. Proc. Letters*, vol.75, no.5, pp.225–229, 2000.

[15] B.Y. Wu, Z.-N. Huang, and F.-J. Zhan, "Exact algorithms for the minimum latency problem," *Inform. Proc. Letters*, vol.92, no.6, pp.303–309, 2004.

[16] http://www.iwr.uniheidelberg.de/groups/comopt/software/TSPLIB95

Ha BANG BAN

Ha Bang Ban is a phd student at Hanoi University of Science and Technology, Vietnam. He received the Engineer and Master of Science degree from Hanoi University of Science and Technology in 2006 and 2008, respectively. During March to November, 2011, he was an internship student in National Institute of Informatics. His main research interests include combinatorial optimization, algorithm design and graphs.

Kien NGUYEN

Kien Nguyen received a B.Eng. from Hanoi University of Technology in 2004, and Ph.D. from the Graduate University for Advanced Studies in 2012. He is currently a postdoctoral researcher at National Institute of Informatics, Japan. His research interests include quality of service provisioning in wired and wireless networks, network protocols, and network virtualization.

Manh CUONG NGO

Manh Cuong Ngo received the Engineer degree from Hanoi University of Science and Technology in 2011. Currently, he has studied about combinatorial optimization at Fondation Mathematique Jacques Hadamard, France. His main research interests include combinatorial optimization and graphs.

Duc NGHIA NGUYEN

Duc Nghia Nguyen is an associate professor of Department of Computer Science, Hanoi University of Science and Technology. He received Ph.D. from Belarusian State University. His main research interests include combinatorial and global optimization, algorithm design and implementation. Recently he is interested in developing metaheuristic algorithms for some NP-hard graph optimization problems.