An efficient exact algorithm for the Minimum Latency Problem

Ha BANG BAN¹, Kien NGUYEN², Manh CUONG NGO³ and Duc NGHIA NGUYEN⁴
¹,²,³,⁴Hanoi University of Science and Technology
²National 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 \( T = \{ v_1, v_2, \ldots, v_n \} \) 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 \( v_k \):

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

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
with diameter 3 [6], a tree with a constant number of leaves [10] (for example with constant \( k \) with diameter 3 [6], a tree with a constant number of leaves) gives a contribution \((n - 1) \times c(v_1, v_2), (n - 2) \times c(v_2, v_3), \ldots, 1 \times c(v_{n-1}, v_n)\), 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 \( T = (v_1, v_2, \ldots, v_k, v_{k+1}, \ldots, v_n) \) 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

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

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

\[
L(T') = \sum_{j=2}^{n} (l(P(v_1, v_j)) + l(P(v_1, v_{k-1})).
\]

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

**Lemma 2.** Let \( V = (v_1, v_2, \ldots, v_k, v_{k+1}, \ldots, v_n) \) 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 \( v_p \in V \setminus F \) \((k+1 \leq p \leq n)\) and an index \( j (1 \leq j \leq k) \) such that the following condition

\[
(n - j)c(v_j, v_{j+1}) > (n - f)c(v_j, v_p) + (n - j - 1)c(v_p, v_{j+1})
\]

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 \( B = (v_{k+1}, v_{k+2}, v_{k+3}, v_{k+4}, \ldots, v_n) \). We insert \( v_p \in B (p = k + 2) \).
Lemma 3. Let \( V = (v_1, v_2, ..., v_n) \) be the vertex set and \( F = (v_1, v_2, ..., v_j, v_{j+1}, v_{j+2}, ..., v_k) \) be a prefix subtour. If there exists a vertex \( v_p \in V \setminus F \) \((k + 1 \leq p \leq n)\) and an index \( j \) \((1 < j < k)\) such that the following condition holds, then \( F \) cannot be extended to an optimal tour.

\[
(n - j)c(v_j, v_{j+1}) + l(P(v_{j+1}, v_k)) + c(v_k, v_p) > (n - j)c(v_j, v_p) + (n - j - 1)c(v_p, v_{j+1})
\]

Proof. Assume that \( T = (v_1, v_2, ..., v_j, v_{j+1}, v_{j+2}, ..., v_k) \) is an optimal tour. We define \( \bar{T} = (v_1, v_2, ..., v_j, v_{j+1}, ..., v_k) \) and \( B = (v_{k+1}, v_{k+2}, ..., v_n) \) is a prefix and a suffix tour of \( T \), respectively. If we insert \( v_p \) \((p = k + 2)\) into \( B \) between \( (v_j, v_{j+1}) \in F \), we have the new tour \( T' = (v_1, v_2, ..., v_j, v_{j+1}, v_p, v_{j+2}, ..., v_k) \). The latency of tour \( T, T' \) can be rewritten as follows:

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

\[
L(T') = \sum_{k=1}^{j-1} (n - k)c(v_k, v_{k+1}) + (n - j)c(v_j, v_p) + (n - j - 1)c(v_p, v_{j+1}) + \sum_{k=j+1}^{p-1} (n - k)c(v_k, v_{k+1}) + (n - k - 2)c(v_{j+1}, v_{j+2}) + \sum_{k=p+1}^{n-1} (n - k)c(v_k, v_{k+1})
\]

If we denote \( \Delta T = L(T) - L(T') \), we obtain

\[
\Delta T = (n - j)c(v_j, v_{j+1}) - (n - j)c(v_j, v_{k+2}) - (n - j - 1)c(v_{k+2}, v_{j+1}) + c(v_{j+2}, v_{j+3}) + ... + c(v_{k+1}, v_{k+2}) + (n - k - 2)c(v_{k+1}, v_{k+2}) - (n - k - 2)c(v_{k+1}, v_{k+2})
\]

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

\[
(n - k - 1)c(v_{k+1}, v_{k+2}) + (n - k - 2)c(v_{k+2}, v_{k+3}) - (n - k - 2)c(v_{k+1}, v_{k+3}) \geq 0
\]

So if condition (1) is satisfied, then \( \Delta T \geq 0 \). This implies that \( T \) is not the optimal tour. Similar arguments hold for the case when \( p \) is not equal to \( k + 2 \). □
is a set of the cheapest costs in matrix \( C \) such that \( e_1 \geq e_2 \geq \ldots \geq e_{n-k} \). 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 \( K_0 \) is the complete graph in the metric case, \( m(v_1, v_j) \) is equal to \( c(v_1, v_j) \). Then we have the following observation:

\[
L(v_{k+1}) = L(v_k) + m(v_k, v_{k+1}) \\
= l(P(v_1, v_k)) + c(v_1, v_{k+1}) \\
\geq l(P(v_1, v_k)) + e_1.
\]

\[
L(v_{k+2}) = L(v_{k+1}) + m(v_{k+1}, v_{k+2}) \\
= L(v_k) + m(v_k, v_{k+1}) + m(v_{k+1}, v_{k+2}) \\
= L(v_k) + c(v_1, v_{k+1}) + c(v_{k+1}, v_{k+2}) \\
\geq l(P(v_1, v_k)) + e_1 + e_2.
\]

\[
\ldots
\]

\[
L(v_k) = L(v_{k-1}) + m(v_{k-1}, v_k) + m(v_{k+1}, v_{k+2}) + \ldots + m(v_{n-1}, v_n) \\
\geq l(P(v_1, v_k)) + e_1 + e_2 + \ldots + e_{n-k}.
\]

Therefore,

\[
L(T) = L(F) + L(v_{k+1}) + L(v_{k+2}) + \ldots + L(v_n) \\
\geq L(F) + (n-k)l(P(v_1, v_k)) + (n-k)e_1 \\
+ (n-k-1)e_2 + \ldots + e_{n-k} \tag{4}
\]

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

\[
LB = L(F) + (n-k)l(P(v_1, v_k)) + (n-k)e_1 \\
+ (n-k-1)e_2 + \ldots + e_{n-k}.
\] \tag{5}

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 \( UB, K_0 \) and \( C_2 \). 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.

<table>
<thead>
<tr>
<th>Group</th>
<th>Group one</th>
<th>Group two</th>
<th>Group three</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>test1</td>
<td>test2</td>
<td>test3</td>
</tr>
<tr>
<td>n</td>
<td>30</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>OS</td>
<td>2697</td>
<td>3209</td>
<td>3028</td>
</tr>
<tr>
<td>UB</td>
<td>4017</td>
<td>5126</td>
<td>4327</td>
</tr>
<tr>
<td>Time(EA1)</td>
<td>3.4</td>
<td>24</td>
<td>125</td>
</tr>
<tr>
<td>Time(EA2)</td>
<td>2.1</td>
<td>19</td>
<td>100</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Group</th>
<th>Group one</th>
<th>Group two</th>
<th>Group three</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>test1</td>
<td>test2</td>
<td>test3</td>
</tr>
<tr>
<td>n</td>
<td>30</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>OS</td>
<td>3625</td>
<td>3897</td>
<td>4521</td>
</tr>
<tr>
<td>UB</td>
<td>4321</td>
<td>5764</td>
<td>6354</td>
</tr>
<tr>
<td>Time(EA1)</td>
<td>3.5</td>
<td>25</td>
<td>120</td>
</tr>
<tr>
<td>Time(EA2)</td>
<td>2.0</td>
<td>20</td>
<td>100</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Group</th>
<th>Eil51</th>
<th>St70</th>
<th>Eil76</th>
<th>Rat195</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>test1</td>
<td>test2</td>
<td>test3</td>
<td>test4</td>
</tr>
<tr>
<td>n</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>OS</td>
<td>4049</td>
<td>5165</td>
<td>5006</td>
<td>3435</td>
</tr>
<tr>
<td>UB</td>
<td>6126</td>
<td>7723</td>
<td>7485</td>
<td>5101</td>
</tr>
<tr>
<td>Time(EA1)</td>
<td>2.5</td>
<td>4.0</td>
<td>80</td>
<td>1.5</td>
</tr>
<tr>
<td>Time(EA2)</td>
<td>2.0</td>
<td>3.0</td>
<td>62</td>
<td>2.5</td>
</tr>
</tbody>
</table>

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 \( \sum L(T) \) for the optimal solution \( \text{OS} \). 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
We denote $X_{min}$, $X_{max}$ is the max, min abscissa of an instance, respectively. $Y_{min}$, $Y_{max}$ is the max, min ordinate of an instance, respectively.

We divided the partial instances into three groups based on the following method: Suppose that each partial instance is between thirty to forty. We have analyzed the data of TSPLIB and found that instances mostly belong to one of the following three groups. Group one includes instances where vertices are concentrated; 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 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. 1 to 3. Our algorithm works for the real data in every case of solving the MLP with smaller size. The running time of the BA algorithm in Table 6 is computed by the GA algorithm instead of the near 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 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 algorithm. 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.
Table 6 The results of the algorithms in the small size instances from TSPLIB.

<table>
<thead>
<tr>
<th>Group</th>
<th>Ulysses22</th>
<th>Gr24</th>
<th>Gr26</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time($E_1$)</td>
<td>4.00</td>
<td>25.24</td>
<td>20.00</td>
</tr>
<tr>
<td>Time($E_2$)</td>
<td>3.00</td>
<td>22.12</td>
<td>15.00</td>
</tr>
<tr>
<td>Time($BA$)</td>
<td>3.40</td>
<td>30.23</td>
<td>27.41</td>
</tr>
</tbody>
</table>

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 $E_1$ consumes more time than the $E_2$, but the $E_1$ 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

[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.