Application of parallel tabu search in solving Ride-sharing problem with HOV Lanes

Y L Hanifa\textsuperscript{1a}, Y Satria\textsuperscript{1b}, and H Burhan\textsuperscript{1c}

\textsuperscript{1}Department of Mathematics, Faculty of Mathematics and Natural Sciences Universitas Indonesia, Depok, 16424, Indonesia

E-mail: \textsuperscript{a}yulinsa.luthfianur@sci.ui.ac.id, \textsuperscript{b}ysatria@sci.ui.ac.id, \textsuperscript{c}helen.burhan@sci.ui.ac.id

Abstract. Ride-sharing can be defined as a system where travelers can share their vehicles and travel cost with others who have a similar destination and time schedule. Ride-sharing problem with High Occupancy Vehicle (HOV) Lanes is a problem of finding an optimal route to serve customer demand where each demand and vehicle consists of origin and destination point, and some of the route are restricted-use freeways lanes reserved for vehicles with more than a predetermined number of occupants, which called HOV lanes. With the use of ride-sharing and HOV Lanes, it is expected to reduce the number of the vehicles thus will mitigate the congestion. The optimal solution is the one with minimum number of total distance, passenger's ride time, and cost. This study discusses the application of insertion heuristic and parallel tabu search to optimize the use of Ride-sharing with HOV Lanes. Insertion heuristic is applied to obtain an initial solution and parallel tabu search algorithm is applied to improve the initial solution and obtain a better solution. For the case of eight vehicles and eight requests, from insertion heuristic we obtained an initial solution with an objective function value of 425 and with the used of parallel tabu search method we obtained a better solution with an objective function value of 420.

1. Introduction

Along with population growth, the need for transportation is also increasing and it makes people flock to choose which convenient transportation facilities to use. With the increasing use of private vehicles while the road capacity is not increasing, traffic congestion is unavoidable, which certainly has so many effects, such as increasing travel time and travel costs [1].

According to Wang (2016) [1], ride-sharing can be a potential solution to reduce the impact of the congestion. Travelers can share their vehicles with others who have a similar time schedule and destination so it will maximize the use of vehicle capacity and later reduces the volume of vehicles operating on the highway. Besides Ride-sharing, HOV Lanes can also be one solution to mitigate congestion. HOV Lanes themselves can be interpreted as restricted-use freeways lanes reserved for vehicles with more than a predetermined number of occupants [2]. We can say that HOV Lanes is one way to encourage people to do ride-sharing. With the use of ride-sharing and HOV Lanes, it is expected to reduce the number of vehicles operating on the highway thus will reduce the travel time and costs of the travelers.

In this study, based on research conducted by Wang et al (2016) [1], we used the Insertion Heuristic and Parallel Tabu Search to optimize the use of ride-sharing with HOV Lanes. Insertion
Heuristic is applied to obtain an initial solution and Parallel Tabu Search is applied to obtain the optimal solution, which is the one with the lowest total distance, travel time and travel cost.

2. Literature review

2.1. Insertion Heuristic
According to Wang et al (2016) [1], insertion heuristic is a popular method to use to get the initial solution since it is more efficient, easy to implement, and produces good results. To see how this method works, suppose given a set of vehicles and a set of requests. In obtaining a feasible solution, insertion heuristic method begins by selecting the first request that is entered into the route. Then, insert another request one at a time into the route with the minimum distance, time travel and travel cost while the time window and capacity constraint are not violated. The requests are inserted based on the width of their time window in an ascending order [1]. If there are some requests that cannot be fulfilled by the ride-sharing vehicle then a taxi will be used to satisfy these requests.

According to Solomon (1987) [3], to calculate the insertion cost, it can be done using the following formula:
\[
c_{11}(i,u,j) = d_{iu} + d_{uj} - d_{iu}, \mu \geq 0
\]
\[
c_{12}(i,u,j) = v_{ju} - v_j
\]
\[
c_1(i,u,j) = \alpha_1 c_{11}(i,u,j) + \alpha_2 c_{12}(i,u,j), \alpha_1 + \alpha_2 = 1, \alpha_1 \geq 0, \alpha_2 \geq 0
\]
\[
c_2(i,u,j) = \beta_1 R_d(u) + \beta_2 R_t(u), \beta_1 + \beta_2 = 1, \beta_1 \geq 0, \beta_2 > 0
\]
where \(d_{iu}\) is the distance between request \(i\) and request \(u\), \(v_j\) is the current service time of request \(j\), and \(v_{ju}\) is the service time of request \(j\) given request \(u\) is now in the route. \(c_1(i,u,j)\) is used to compute request \(u\)’s best feasible insertion place in the route, \(R_d(u)\) is the total route distance and \(R_t(u)\) is the total travel time if the request \(u\) is inserted in the route, where \(\mu, \alpha_1, \alpha_2, \beta_1, \) and \(\beta_2\) are the parameters.

2.2. Parallel Tabu Search
The Tabu Search method is a meta-heuristic method that was first introduced by Fred Glover in 1986. Tabu search can be interpreted as a metaheuristic method that guides local heuristic to explore the space beyond the local optimum. The basic concept of the Tabu Search algorithm is to guides each stage of the searching process to produce the most optimum value of the objective function without being trapped in a solution that has been found before [4].

The "Tabu" status is the most important thing in Tabu Search that distinguishes it from other methods. During the searching process, the solution that has appeared will be stored in a memory called "Tabu List" and when a solution reappears, the "Tabu" status will appears on the solution.

The first step in using the tabu search method is to get the initial solution with the insertion heuristic method as previously explained. Then we set the obtained initial solution as the current solution \(s\) and also as the best solution \(bs\). After that we form the neighborhood of the current solution and select the best solution from it with the minimum objective function value and save it as \(s'\).

Then we compare the solution \(s'\) with solutions \(s\) and \(bs\). If the objective function of \(s'\) is better than the solution \(bs\), then \(bs\) and the Tabu List will be updated and we set \(s'\) as \(s\), but if not, we need to check whether the solution is in Tabu List. If the solution \(s'\) is already in the Tabu List, then it will be removed from the set of candidate solutions, but if not, the solution will be set as and we put it in the Tabu List to start the searching process with the new current solution. These steps will continue until we fulfill the stopping criterion, which is after some number of iterations without the improvement of the objective function value or after a fixed number of iterations [5].

When the size of the data is huge, we need a method that can be used to speed up the execution time. The implementation of parallel systems in tabu search is very possible. According to Garcia et al
(1994) [6], parallelization in the tabu search method can be done using the master-slave scheme. The master processor is in charge of running the tabu search process and each slave processor has the task of exploring different neighborhood to find their best solution. Furthermore, slave processors will send their best solution to the master processor.

After the master processor receives the best solution from each slave processor, it will choose the best one to be used as the solution and save it in the tabu list. If the solution is better than the current solution, the solution will be updated. Next, the master processor will provide information about the selected solution to the slave processors so that the slave processors can update their neighborhood [6].

3. Model formulation
As a basis to our formulation, we follow Lu and Dessouky’s formula [1]. We defined Ride-sharing problem with HOV Lanes into a directed graph $G = (N, A)$ where $N = \{1, 2, ..., n\}$ is a set of nodes and index $i \in N$ represents the node $i$. $A$ is the arc set where arc $(i, j) \in A$ represents an arc that connects two nodes, node $i$ and node $j$, where time and cost that associated to each arc depends on the number of people in the vehicle. $H = \{1, 2, ..., n\}$ is the request set where $h \in H$ corresponds to the $h$-th request and $S = \{1, 2, ..., m\}$ is the vehicle set where $s \in S$ corresponds to the $s$-th vehicle.

- $i = 1, 2, ..., n$ is request $h$’s start location
- $i = n + 1, n + 2, ..., 2n$ is request $h$’s send location
- $i = 2n + 1, 2n + 3, ..., 2n + 2m - 1$ is vehicle $s$’s start location
- $i = 2n + 2, 2n + 4, ..., 2n + 2m$ is vehicle $s$’s end location

$N_p$ is a set of request’s start location where $N_p = \{1, 2, ..., n\}$

$N_d$ is a set of request’s end location where $N_d = \{n + 1, n + 2, ..., 2n\}$

$N_a$ is a set of vehicle’s start location where $N_a = \{2n + 1, 2n + 3, ..., 2n + 2m - 1\}$

$N_r$ is a set of vehicle’s end location where $N_r = \{2n + 2, 2n + 4, ..., 2n + 2m\}$

**Parameters:**

- $R_h$: number of passengers of request $h$, $h \in H$
- $G_i$: request $i$, $i \in N_p, h = i$
- $R_h$: number of passengers of request $h$, $h \in H$
- $G_i$: request $i$, $i \in N_p, h = i$
- $C_a$: vehicle’s capacity
- $S_{i,n+i}$: travel time from node $i$ to node $n + i$ using taxi
- $E_i$: the earliest time request can be picked up
- $L_i$: the latest time request can be delivered
- $O$: number of people in the vehicle at the vehicle’s start location
- $Y_{ij}$: minimum travel time from node $i$ to node $j$
- $T_{ijk}$: travel time from node $i$ to node $j$ with $k$ number of people in the vehicle
- $D_{ijk}$: distance from node $i$ to node $j$ with $k$ number of people in the vehicle
- $C_{ijk}$: travel cost from node $i$ to node $j$ with $k$ number of people in the vehicle

**Variables:**

- $x_{ij}$: 1, if vehicle travels from node $i$ to $j$
- $u_{ij}$: 1, if node $i$ is serviced by taxi
- $b_{ij}$: 1, if node $i$ is before node $j$ in the vehicle’s route
- $v_i$: the time at which a request is serviced at node $i$
- $z_i$: the number of people in the vehicle after serving node $i$
- $t_{ij}$: the actual travel time from node $i$ to node $j$
- $d_{ij}$: the actual distance from node $i$ to node $j$
\( c_{ij} \): the actual travel cost from node \( i \) to node \( j \)

Based on our aim, namely to minimize the total distance, travel time, and cost, the objective function can be defined as:

\[
\text{Minimize } (\sum_{i \in N_P} \beta \times (v_{n+i} - v_i) + \sum_{(i,j) \in A} (\gamma \times d_{ij} + \mu \times c_{ij}) + \sum_{(i,n+i) \in A} (\lambda \times u_i)) \tag{1}
\]

Subject to:

\[
\sum_{j \in N} x_{ij} + u_i = 1, i \in N\{2n + 2, \ldots, 2n + 2m\} \tag{2}
\]

\[
\sum_{i \in N} x_{ij} + u_j = 1, j \in N\{2n + 1, \ldots, 2n + 2m - 1\} \tag{3}
\]

\[
u_i = u_{n+i}, i \in N_P \tag{4}
\]

\[
b_{kl} \leq b_{kj} + (1 - x_{ij}), (i,j) \in A \{2n + 1, \ldots, 2n + 2m\}, k \in N \{i\} \tag{5}
\]

\[
b_{kj} \leq b_{kl} + (1 - x_{ij}), (i,j) \in A \{2n + 1, \ldots, 2n + 2m\}, k \in N \{i\} \tag{6}
\]

\[
b_{kl} + u_i \leq 1, i \in N \{2n + 1, \ldots, 2n + 2m\}, k \in N \{i\} \tag{7}
\]

\[
b_{lk} + u_i \leq 1, i \in N \{2n + 1, \ldots, 2n + 2m\}, k \in N \{i\} \tag{8}
\]

\[
x_{ij} \leq b_{ij}, (i,j) \in A \tag{9}
\]

The first component of the objective function is represents the total passengers travel time with a weight of \( \beta \). The second component represent the total distance and cost with a weight of \( \gamma \) and \( \mu \). The third component is the total taxi cost with a weight of \( \lambda \). Constraint (2) and (3) ensures that the vehicle moves from node \( i \) to node \( j \) and each request is visited only once, either by the vehicle or the taxi. Constraint (4) ensures that each request that picked up by the taxi will be delivered also by the taxi. Constraint (5) and (6) ensure that if the vehicle moves from node \( i \) to node \( j \), every nodes \( k \) that are before node \( i \) are also before node \( j \), and vice versa. Constraint (7) and (8) remove node \( i \) from the routes if it is visited by the taxi. Constraint (9) ensures that all requests is either visited by the vehicle or the taxi. Constraint (10) forces node \( i \) to be before node \( j \) if the vehicle moves from \( i \) to \( j \).

\[
b_{ii} = 0, i \in N \tag{11}
\]

\[
b_{n+i,l} = 0, i \in N_P \tag{12}
\]

\[
b_{l2n+2m-1,i} = 0, i \in N, m \in S \tag{13}
\]

\[
b_{2n+2m,i} = 0, i \in N, m \in S \tag{14}
\]

\[
z_i = (G_i + O) \times (1 - u_i) + \sum_{l \in N} (b_{ii} \times G_i), i \in N \tag{15}
\]

\[
z_i \leq Ca \tag{16}
\]

\[
\nu_i + S_{i,n+i} \leq v_{n+i} + (1 - u_i) \times M, i \in N_P \tag{17}
\]
\[ v_i + t_{ij} \leq v_j + (1 - x_{ij}) \times M, \ i, j \in N \]  
(18)

\[ E_i \leq v_i \leq L_i, \ i \in N \]  
(19)

\[ t_{ij} \geq T_{ijk} - |z_i - k| \times M - (1 - x_{ij}) \times M, \ (i, j) \in A, k = 1, 2, \ldots, Ca \]  
(20)

\[ d_{ij} \geq D_{ijk} - |z_i - k| \times M - (1 - x_{ij}) \times M, \ (i, j) \in A, k = 1, 2, \ldots, Ca \]  
(21)

\[ c_{ij} \geq C_{ijk} - |z_i - k| \times M - (1 - x_{ij}) \times M, \ (i, j) \in A, k = 1, 2, \ldots, Ca \]  
(22)

Constraint (11) ensures that there is no node before itself on a tour. Constraint (12) ensures that there is no drop off location before its pickup location. Constraint (13) ensures that there is no node before the vehicle’s start location. Constraint (14) ensures that there is no node after the vehicle’s end location.

\[ z_i \] is the number of people in the vehicle after serving node \( i \). \( \sum_{k \in N} (b_{ij} \times G_i) \) is the number of people picked up and have not been dropped off before node \( i \). \( G_i \) is the number of people that need to be served at node \( i \). It will be positive at the pickup node and negative at the drop off node. \( O \) is the number of people in the vehicle at the vehicle’s start location.

Constraint (16) ensures that the capacity constraint is not violated. Constraint (17) ensures the consistency of the time variables if the node is visited by the taxi where \( M \) is a very big number. Constraint (18) ensures the consistency of the time variables if the node is visited by the vehicle. \( (E_i, L_i) \) is the time windows of node \( i \). \( E_i \) is the earliest pick up time and \( L_i \) is the latest drop off time. Constraint (20) forces the actual time travel is larger than the Ride-sharing time travel. Constraint (21) forces the actual distance is larger than the Ride-sharing distance. Constraint (22) forces the actual travel cost is larger than the Ride-sharing travel cost

\[ x_{ij} = \{0, 1\}, \ (i, j) \in A \]  
(23)

\[ u_i = \{0, 1\}, \ i \in N_p \]  
(24)

\[ b_{ij} = \{0, 1\}, \ (i, j) \in A \]  
(25)

\[ z_i \in Z \]  
(26)

\[ v_i \geq 0, \ i \in N \]  
(27)

\[ t_{ij}, d_{ij}, c_{ij} \geq 0, \ (i, j) \in A \]  
(28)

\[ v_{n+i} - v_i \geq Y_{ij}, \ i \in N_p \]  
(29)

\[ b_{k,i} + b_{k,n+i} \geq x_{l,k} + x_{k,i}, \ k \in N \setminus \{i, n + i\}, \ i \in N_p \]  
(30)

\[ b_{l,k} + b_{l+n,k} \geq x_{l+n,k} + x_{k,l+n}, \ k \in N \setminus \{i, n + i\}, \ i \in N_p \]  
(31)

\[ \sum_{k \in N} (b_{ki} - G_k) = 0, \ i \in \{2n + 1, 2n + 2, \ldots, 2n + 2m - 1, 2n + 2m\} \]  
(32)

Constraint (23), (24), and (25) is the binary constraints for the parameters. Constraint (26) sets \( z_i \) to an integer value. Constraint (27) to (28) is the non-negativity constraints. \( Y_{ij} \) is the minimum
passenger’s travel time. The actual travel time from node $i$ to $j$ will always be larger than the minimum travel time. Constraint (30) and (31) is the adjacent prior constraints. Constraint (32) ensures that there is no more passenger left in the vehicle when the vehicle is in its end location.

4. Experimental results

In this study, we use C++ as programming language with the software Code :: Blocks version 17.12 on a hardware with AMD A8-7410 APU processor specifications, AMD Radeon R5 Graphics 2.20 GHz, memory: 4.00 GB RAM and 64-bit Operating System, x64-based processor.

The experiments are run on an undirected graph that contains normal lanes and HOV Lanes. The map is a 10*10 grid with 180 arcs. 125 arcs are the normal lanes and the other 55 arcs are the HOV Lanes. We set the length of every lanes is 1 km. For the normal lanes, the travel time is 2 minutes and it costs $1, whereas for the HOV Lanes, the travel time is 1 minute and it is free. To see how HOV Lane works, we can see the example below. Let the black lines as the normal lanes and the red line as the HOV Lane. Let a vehicle in point A has destination in point B. If there is only one person in the vehicle, it should take the normal lanes for 7 kilometers. But if there are more than two persons in the vehicle, it could take the HOV Lane for only 5 kilometers which means it is shorter than the normal one.

![application_of_hov_lanes.png](attachment:application_of_hov_lanes.png)

**Figure 1**: Application of the HOV Lanes

We used dummy data with 8 vehicles and 8 requests, where all of them have their own time window and end location. The earliest time window of the vehicle starts at 00 and ends at 29 and the latest time windows starts at 10 and ends at 37. For the request, two of them has 2 passengers to be served and the others have only 1 passenger. The earliest request’s time window starts at 00 and ends at 24 whereas the latest time window starts at 10 and ends at 42. The reason why we only used dummy data with 8 vehicles and 8 requests is that it is already quite hard to calculate it manually, thus this program is needed in order to solve the problem.

As previously explained, before we used parallel tabu search method to finds the optimal solution, an initial solution is needed and it can be obtained from the insertion heuristic method. The initial solution is shown in the following table:

| Table 1: Initial Solution |
|---------------------------|
| vehicle[0]: [1+] [5+] [1-] [5-] |
| vehicle[1]: [0+] [7+] [0-] [7-] |
| vehicle[2]: [3+] [2+] [3-] [2-] |
| vehicle[3]: [4+] [4+] |
| vehicle[4]: [6+] [6+] |
| vehicle[5]: - |
| vehicle[6]: - |
| vehicle[7]: - |
After getting the initial solution, the next step is to find the optimal solution using the parallel tabu search method. In this study, the Tabu List contains the value of the objective function of the solutions that have been selected during the process. For the objective function, we used coefficients $\beta = \gamma = \mu = 1$ and $\lambda = 10000$ [1]. The Tabu Tenure that we used is 100 iterations, which means a solution can exit the Tabu List after 100 iterations. We did the process until the objective function value has converged to a value or after 300 iterations. After 41 iterations, the optimal solution that obtained from the parallel tabu search method is shown in the following table:

| Vehicle | Route | Objective Function |
|---------|-------|--------------------|
| Vehicle[0] | [0+] [0-] | 425 |
| Vehicle[1] | [4+] [4-] | 420 |
| Vehicle[2] | [3+] [3-] | 420 |
| Vehicle[3] | [5+] [5-] | 420 |
| Vehicle[4] | [7+] [7-] | 420 |
| Vehicle[5] | [2+] [2-] | 420 |
| Vehicle[6] | [6+] [6-] | 420 |
| Vehicle[7] | [1+] [1-] | 420 |

From the initial solution, the first vehicle serves the second (1) and sixth (5) requests, the second vehicle serves the first (0) and eighth (7) requests, and so on. Then the sixth to eighth vehicles travel privately without sharing. The objective function of the initial solution obtained is 425. After the initial solution is obtained, this solution would be used to start the process of finding the optimal solution using the parallel tabu search method.

In the final solution, each vehicle serves one request. The first vehicle (0) serves the first request (0), the second vehicle (1) serves the fifth request (4), and so on. The objective function is 420 which is obtained after 41 iterations for 0.227 seconds. Because the objective function has been converged to a value, the searching process stopped after 44 iterations with execution time of 0.227 seconds.

It can be seen that in the initial solution there are three vehicles that must travel privately to their destination through the normal lanes. Although this solution is good enough because there are some vehicles and requests that do Ride-sharing, but this condition still allows congestion on the normal lanes. Whereas in the final solution, all vehicles share with each one request. It can be said that this is a very good condition because when all vehicles and requests do ride-sharing, they can pass through the HOV lanes to reach their destination, decrease the total distance, travel time and travel cost and it automatically reduces the congestion. Thus, the solution generated through the parallel tabu search method can be said to be the optimal solution.

5. Conclusions
This study discusses the application of parallel tabu search algorithms in solving ride-sharing problems with HOV lanes. The objective function of this study is to minimize the total distance, travel time, and travel cost as the impact of congestion. Based on the obtained result, in the case of eight vehicles and eight requests, from insertion heuristic we obtained an initial solution with an objective function value of 425 and with the used of parallel tabu search method, we improved the initial solution to obtained a better solution with an objective function value of 420.
Ride-sharing problems with HOV lanes policies can also be implemented into the program using the C programming language. For the case of eight vehicles and eight requests, the best solution is obtained with the value of the objective function of 420 with an execution time of 0.227 seconds.

Acknowledgements
This research was supported by Publikasi Internasional Terindeks untuk Tugas Akhir Mahasiswa (PITTA) research grant of Universitas Indonesia 2018.

References
[1] Wang X, Dessouky M, and Ordonez Fernando 2016 A Pickup and Delivery Problem for Ride-sharing Considering Congestion Transportation Letters 8(5) 259-269.
[2] Menendez, Monica, and Daganzo C F 2017 Effects of HOV Lanes on Freeway Bottlenecks Transportation Research Part B: Methodological 41(8) 809-802.
[3] Solomon M M 1987 Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints Operations Research 35(2) 254-265 (Maryland: INFORMS).
[4] Glover F 1989 Tabu Search-Part I ORSA Journal on Computing 1(3) 190-206.
[5] Cheeneebash J and Na dal C 2010 Using Tabu Search Heuristics in Solving the Vehicle Routing Problem with Time Windows: Application to Mauritian Firm University of Mauritius Research Journal 16.
[6] Garcia B L, Potvin J Y, and Rosseau J M 1994 A Parallel Implementation of the Tabu Search Heuristic for Vehicle Routing Problems with Time Window Constraints Computers Ops Res 21(9) 1025-1033.
[7] Toth P and Vigo D 2002 The Vehicle Routing Problem (United State of America: Society for Industrial and Applied Mathematics).