A Combinatorial Insertion Algorithm for the Public Vehicle System

Ning Li*, Linghe Kong†*, Jia-liang Lu*, Wei Shu‡*, Min-You Wu*
*Shanghai Jiao Tong University, China
†McGill University, Canada.
‡University of New Mexico, USA
*ningl_sjtu@163.com, *{linghe.kong, jialiang.lu, mwu}@sjtu.edu.cn, ‡shu@ece.unm.edu

***Abstract*—In Intelligent Transport field, the Public Vehicle System is proposed to introduce a concept of a specialized vehicle for public transportation, which can integrate and substitute for vehicles such as taxis, buses and railways. Public Vehicle model has several advantages over earlier models, but the algorithm proposed in the model can't consider all potential solutions when building new paths, resulting in a decrease in performance. We expand the searching method to cover all cases of insertion and achieve a lower cost. Our simulations show that the Combinatorial Insertion Algorithm can have a 5%-11% promotion in total traveling distance.**

I. Introduction

Intelligent Transport System(ITS) is developing fast in recent years, aiming at eliminating the shortcomings of traditional transportation such as serious pollution and congestion by integrating the features of cloud computing and smart devices. In these systems, a natural trend is transforming autonomous ride-sharing process into computations and instructions to guide or manipulate cars in a server-client system which distributes most of the computational work to the server and enables the client to receive instructions. During that interacting process vehicular networks [1] [2] play a key role in exchanging such information. Besides, the client also refers to the mobile devices such as smart phones and collects requests from customers in wide areas. More than just serving customers by satisfying all their demands, these systems would usually head for economical objectives such as lower total traveling distance, better efficiency or smaller fleet size.

Carpool, or ridesharing, is the sharing of car journeys so that more than one person travels in a car [3]. A typical example of static ridesharing is the school bus. As progresses have been made to meet arbitrary demands for traveling in the city, dynamic ridesharing model [4] emerged to supply users with more choices of time and locations. Furthermore, some ridesharing models support singlehop or even multihop [5] for less distances of detour. However, we think adding many transfers into the journey may lead to a rather bad impact on comfort degree.

The Vehicle Routing Problem(VRP) [6] was initially introduced around 1960. Afterwards, with different kinds of constraints added to the problem, it developed into several branches, contributing a lot to intelligent transport study. The VRP with Time Windows(VRPTW) [7] is a branch with soft or hard time constraints, and it's meaningful and practical because waiting time has much to do with the comfort of customers. Nevertheless, in VRPTW, customers or commodities seem to share the same destination. This kind of centralized transportation may not be expanded to serve widely dispersed people.

The Dial-a-Ride Problem(DARP) [8] resembles VRPTW except that every customer should set individual origin and destination. It allows customers to choose destinations freely, and makes the model widely adaptable to many implementations. But it still has several limitations, which we will point out in comparisons with PVS as follows.

Zhu et al. [9] propose a new model called Public Vehicle System(PVS) to further develop the DARP model. We pay more attention to PVS for three reasons:

1) The concept of Public Vehicle keeps up with the trend of autonomous cars and connected vehicles [10], so it may have enormous potentials;
2) In PVS, vehicles don't have to start from a centralized depot, neither do they have to return the starting point, granting the drivers more convenience;
3) No request is necessarily known in advance, so it's completely dynamic in collecting information and scheduling paths.

On the contrary, the DARP Model is not as good as PVS in 1) and 3), and still constrained by centralization.

However, in the algorithm called Precedence Constrained origin-destination Pair Insertion(PCPI) Algorithm proposed in PVS model, we have found a defect that it may miss out the optimum insertion case, causing a low performance. The contribution of our paper is: (1) We analyze the defect of the PCPI algorithm, and improve it with a practical method to be a new Combinatorial Insertion Algorithm. (2) We list and explain the characteristics in comparisons of experimental results between two algorithms through simulations with different scales and PV capacities.

II. Problem Formulation

In Public Vehicle System, assume there are N PVs and M requests at time T. To improve the formulation in Paper [9], we refine two definitions:

Definition II.1. *If a request r is **served** by a PV p, it means that the customer of r has been picked up by p and will be*

978-1-5090-1698-3/16/$31.00 ©2016 IEEE

dropped off as long as p arrives at the destination of r.

Definition II.2. *If a request r is **assigned** to a PV p, it means that the origin and destination of r have been added to the path of p. Maybe p is serving r right now, or will serve it some minutes later.*

Some of these requests denoted by R_s are served by PVs, and some others denoted by R_a have been assigned and are waiting to be served, while the others denoted by R_w haven't been assigned yet. For a request $r \in R_s$, if it's served by a PV p, then p must carry the customer to the destination; if $r \in R_a$, then p must go to pick up the customer at the origin of r before traveling to the destination. The system continues to receive new requests and assigns them to PVs. Let $G = (V, E)$ be the weighted complete graph where vertices in V consists of the current locations of PVs and the origins and destinations of all M requests at time T, and edges in E represent the paths of PVs. The system is constructing as many edges as possible to cover every location once, and the objective is to minimize the total distance of all PVs. We set the denotations used in our formulation in Table. I.

TABLE I
DENOTATIONS IN PVS

T	the current time of the system
P	the set of PVs at time T
N	the number of PVs in P
C	the capacity of PVs, assuming their capacities are the same
$C_{p,i}$	the capacity of p when it leaves from vertex i
R	the set of requests at time T
R_s, R_a and R_w	are defined above
M	the number of requests at time T, assuming $N*C > M$
r_o and r_d	the origin and destination of request r
$S_{p,r}$	is 1 if r is served by p; otherwise, 0
$A_{p,r}$	is 1 if r is assigned to p but hasn't been served yet; otherwise, 0
$I_{p,i}$	is 1 if vertex i is on the path of p; otherwise, 0
$X_{p,i,j}$	is 1 if PV p travels directly from vertex i to j; otherwise, 0
$K_{p,i,j}$	is 1 if PV p must arrive vertex i before j
$W_{i,j}$	the weight of edge (i,j) or (j,i)
L_p	the set of all locations on the path of p
c_p	the current location of p
e_p	the end location of p
t_r	the earliest start time of r
$\tau_{p,i,j}$	the shortest time for p to travel from vertex i to j through its path

The formulation is formed by Eqns(1-12). Eqn(1) is the objective function, and Eqn(2-12) are constraints. Eqn(2) ensures that if r is served by p, p must travel to destination of r. Eqn(3) ensures that if r is assigned to p but hasn't been served, p must travel to the origin of r first, and then to the destination. Eqn(4) implies that any request can be assigned or served by at most one PV at a time. Eqn(5)(6) imply that if p travels from i directly to j, and j is the origin or destination of request r which is served by or assigned to p, then p must pick up or drop off the customer at j, and the number of customers on p will increase or decrease by 1. Eqn(7)(8) show that the path of p is a sequential line, where every node on the path except the first and last node have only one successor and one precursor. Eqn(9) means that if p will visit both i and j, it must decide which to visit firstly, and can not turn back after leaving. Eqn(10) prevents the occurrence of subtours, or cycles in G. Eqn(11) specifies that the number of customers on a PV can not exceed the capacity anytime. Eqn(12) implies that if p should not arrive at the location to pick up a customer before the earliest start time.

$$\textbf{Objective: } \min \sum_{p \in P} \sum_{\{i,j\} \in E} W_{i,j} X_{p,i,j} \tag{1}$$

Subject to:

$$S_{p,r} = 1 \Rightarrow I_{p,r_d} = 1 \tag{2}$$

$$A_{p,r} = 1 \Rightarrow I_{p,r_o} = I_{p,r_d} = 1, K_{p,r_o,r_d} = 1 \tag{3}$$

$$\sum_{p \in P} (S_{p,r} + A_{p,r}) \leq 1, r \in R \tag{4}$$

$$X_{p,i,j} = 1, j = r_o, A_{p,r} = 1 \Rightarrow C_{p,j} = C_{p,i} + 1 \tag{5}$$

$$X_{p,i,j} = 1, j = r_d, S_{p,r} = 1 \text{ or } A_{p,r} = 1 \Rightarrow C_{p,j} = C_{p,i} - 1 \tag{6}$$

$$\sum_{j \in I_p \backslash \{c_p\}, j \neq i} X_{p,i,j} = 1, i \in I_p \backslash \{e_p\} \tag{7}$$

$$\sum_{i \in I_p \bigcup \{e_p\}, i \neq j} X_{p,i,j} = 1, j \in I_p \backslash \{c_p\} \tag{8}$$

$$K_{p,i,j} + K_{p,j,i} \leq 1 \tag{9}$$

$$K_{p,i,j} + K_{p,j,k} + K_{p,k,i} \leq 2 \tag{10}$$

$$C_{p,i} \leq C, i \in L_p \tag{11}$$

$$r \in R_w, A_{p,r} = 1 \Rightarrow T + \tau_{p,c_p,r_o} \geq t_r \tag{12}$$

III. DEFECT ANALYSIS AND THE IMPROVED ALGORITHM

In this section, we analyze the defect of PCPI algorithm and propose a method for improvement.

The PCPI algorithm consists of two sequential segments. The first segment is to schedule an initial path for every vehicle, and the second is to assign requests, and insert the origins and destinations of newly assigned requests to build a new path. The defect lies in the insertion stage where the algorithm chooses to insert the origin and destination separately: origin first, and then destination. We will review and analyze the insertion process as follows.

As the paper [9] defines, the insertion cost(IC) $\pi_{r,p,i,j}$ means the increased distance of the newly constructed path compared with the original path, by inserting the origin of request r at location i and the destination at location j; and the least insertion cost(LIC) $\pi_{r,p}$ means the smallest IC for all possible i and j. In this way, we define $\pi_{r_o,p,i}$ or $\pi_{r_d,p,i}$ as the insertion cost by inserting the origin or destination of r individually at location i.

Let the original path of p be $\{\theta_0, \theta_1, \ldots, \theta_n\}$. The paper provides a concrete formula to calculate $\pi_{r,p,i,j}$ as shown in Eqn. 13. To find the LIC and insert the request r, the algorithm firstly selects one location $\theta_i (1 \leq i \leq n+1)$ on the path of p to insert r_o, then selects another location θ_j behind θ_i, or

$i+1 \leq j \leq n+2$, to insert r_d. It means we must determine the value of i before j by some criterion. For example, a simplest criterion is to find the lowest $\pi_{r_o,p,i}$ for all possible $i \in [1, n+1]$. But no matter what criterion it is, it's not consistent with our final objective to find the lowest IC because it doesn't take the insertion of destination r_d into account. As a result, the most proper locations for insertion may probably not be (θ_i, θ_j), but cases $(\theta_{i_1}, \theta_{j_1})(i_1 \neq i)$ are all skipped. The insertion algorithm in PVS model is described in Alg. 1 .

To overcome this drawback, we propose a Combinatorial Insertion Algorithm to achieve the optimum insertion cost under the condition that the original sequence of locations in the path keeps stable. Every time we perform an insertion operation, we select the location pairs to insert origin and destination together as the meaning of combinatorial number C_n^2, where n is the number of locations in the original path. In dealing with every case in C_n^2 ones, the location ahead is obviously for inserting the origin, and the location behind it for the destination. Moreover, the destination may be inserted right after the origin, so there're C_n^1 more cases. These cases are just all that are valid.

$$\pi_{r,p,i,j} = \begin{cases} d\{\theta_{i-1}, r_o, r_d, \theta_i\} - d\{\theta_{i-1}, \theta_i\}, \\ \qquad \text{if } 1 \leq i \leq n, j = i+1. \\ d\{\theta_n, r_o, r_d\}, \\ \qquad \text{if } i = n+1, j = i+1. \\ d\{\theta_{i-1}, r_o, \theta_i\} + d\{\theta_n, r_d\} - d\{\theta_{i-1}, \theta_i\}, \\ \qquad \text{if } i \leq n, j = n+2. \\ d\{\theta_{i-1}, r_o, \theta_i\} + d\{\theta_{j-1}, r_d, \theta_j\} \\ -d\{\theta_{i-1}, \theta_i\} - d\{\theta_{j-1}, \theta_j\}, \\ \qquad \text{if } i \leq n, j \leq n+1, j \neq i+1. \end{cases} \quad (13)$$

A. Algorithm and Complexity

The improved algorithm should make sure that every possible combination (i, j) is traversed, as steps shown in Alg. 2.

Apparently, Alg. 1 includes two For loops in sequential order, while in Alg. 2 one loop is nested by another, so the complexity of Alg. 1 is $O(n)$ and that of Alg. 2 is $O(n^2)$. But there's a key statement in Alg. 2 whose execution times can be easily reduced. It's the 4th statement to calculate the value of $\pi_{r,p,i,j}$, located in the inner loop. To illustrate the way, we give an improved version of Eqn. 13:

$$\pi_{r,p,i,j} = \begin{cases} d\{\theta_{i-1}, r_o, r_d, \theta_i\} - d\{\theta_{i-1}, \theta_i\}, \\ \qquad \text{if } 1 \leq i \leq n, j = i+1. \\ d\{\theta_n, r_o, r_d\}, \quad \text{if } i = n+1, j = i+1. \\ \pi_{r_o,p,i} + \pi_{r_d,p,j}, \quad \text{if } j \neq i+1. \end{cases} \quad (14)$$

So most calculations in Alg. 2 are the same with Alg. 1 except when r_d is inserted right after r_o. Compared with Alg. 1, the extra executions in Alg. 2 is $O(n)$ operations to calculate the shortest length between two vertices in G, and $O(n^2)$ addition and comparison operations. And the former ones cost much more time than the latter ones, so the complexity of Alg. 2 can be well optimized.

Algorithm 1 Insertion Algorithm in PCPI

Input:
Request $r = (r_o, r_d)$ and Path $p = \{\theta_0, \theta_1, \ldots, \theta_n\}$.
Output:
A new path p' containing r_o, r_d and all elements in p in original sequence.

```
1: Min = ∞
2: Record_i = n + 1
3: for i = 1 to n + 1 do
4:    Figure out π_{r_0,p,i}
5:    if π_{r_0,p,i} < Min then
6:       Min = π_{r_0,p,i}
7:       Record_i = i
8:    end if
9: end for
10: Insert r_o in location Record_i
11: Min = ∞
12: Record_j = n + 2
13: for j = Record_i + 1 to n + 2 do
14:    Figure out π_{r_d,p,j}
15:    if π_{r_d,p,j} < Min then
16:       Min = π_{r_d,p,j}
17:       Record_j = j
18:    end if
19: end for
20: Insert r_d in location Record_j
21: return the newly constructed path p'
```

Algorithm 2 Combinatorial Insertion Algorithm

Input:
Request $r = (r_o, r_d)$ and Path $p = \{\theta_0, \theta_1, \ldots, \theta_n\}$
Output:
A new path p' containing r_o, r_d and all elements in p in original sequence

```
1: Min = ∞
2: for i = 1 to n + 1 do
3:    for j = i + 1 to n + 2 do
4:       Figure out π_{r,p,i,j}
5:       if π_{r,p,i,j} < Min then
6:          Min = π_{r,p,i,j}
7:          Record i and j as i_final and j_final
8:       end if
9:    end for
10: end for
11: Insert r_o and r_d in location i_final and j_final
12: return the newly constructed path p'
```

B. An example

Figure. 1 is a simple example to demonstrate the differences between two algorithms. The broken line from S through A, B, and C to D represents the original path; the vector from Origin

to Des indicates that a request $r = (Origin, Des)$ is assigned to this vehicle, and the Origin and Des will be inserted into the path; The dotted lines connect the Origin and Des with every point in the path; And the numbers clung to each line segment are the shortest distances between their two ends. The distances clung to each dotted line are listed in Table. II.

First we calculate $\pi_{Start,p,i}$ and $\pi_{Des,p,i}$ for $i \in [1, 5]$, listed in Table. III. The first column suggests that inserting Origin at location 4 will bring about the least cost of 8, so if we run Alg. 1, after inserting the Origin at location 4, we will get new path $p' = \{S, A, B, C, Origin, D\}$, and then there are only 2 choices for Des: right after the Origin, or at location 6. According to Eqn. 13, the former equals to $d_{Origin,Des} + d_{Des,D} - d_{Origin,D} = 39 + 60 - 23 = 76$, and the latter corresponds to $\pi_{Des,p,5}$ in Table. III which is 60, so the Des will be inserted at the end and the path will be $p'' = \{S, A, B, C, Origin, D, Des\}$. The IC of r by Alg. 1 is 68.

If Alg. 2 runs for the example, then we have to build another table to illustrate the process. Table. IV calculates all IC for all possible cases. Obviously IC is lowest when the request is inserted at location (1,2).

From the above analysis and the two tables, we can see that Alg. 1 takes only two cases(rows) of Table. IV into consideration, but leaves out the other 13 cases. In contrast, Alg. 2 has considered all feasible cases so as to choose the best one.

TABLE II
DISTANCES CLUNG TO DOTTED LINES

	Origin	Des
S	30	46
A	60	22
B	37	14
C	23	48
D	23	60

TABLE III
$\pi_{Start,p,i}$ AND $\pi_{Des,p,i}$

i \ Loc $\pi_{Loc,p,i}$	Origin	Des
1	31	9
2	63	2
3	21	23
4	8	70
5	23	60

TABLE IV
$\pi_{r,p,i,j}$

i	j	IC	i	j	IC
1	2	32	2	6	123
1	3	33	3	4	85
1	4	54	3	5	91
1	5	101	3	6	81
1	6	91	4	5	84
2	3	79	4	6	68
2	4	86	5	6	62
2	5	133			

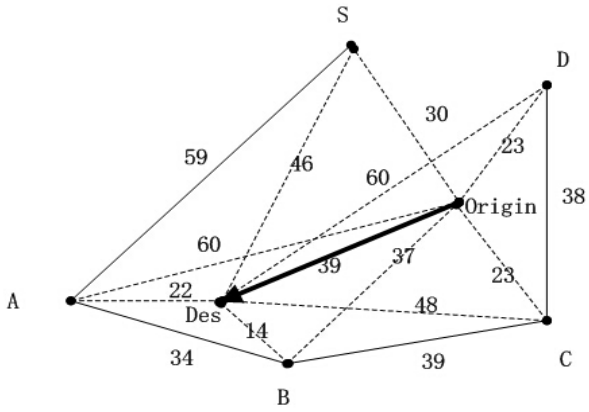

Fig. 1. Figure For Example 1

IV. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the Combinatorial Insertion Algorithm compared with the algorithm in Paper [9].

Our simulation is based on the the same area in Shanghai as Paper [9], but we simplify some settings to highlight the differences between two algorithms. First, we use requests known in advance rather than those generated every few seconds to save execution time. Second, as the time complexity grows, we downsize the number of requests to 1000, and run the simulation several times to test its stability. Third, we set the capacity of PVs as two different values to make out its impact on performance.

Firstly, we use 100 requests and 5 PVs with capacity of 15 to get experimental data of 200 simulations. Each simulation is independent, and information of requests and PVs will be randomly created when a new simulation starts. These data show that the average running time of Alg. 2 is about 1.8 times that of Alg. 1, while the average total distance of the former is 95.4% of that of the latter. But Alg. 2 does not always achieve a better performance: Alg. 1 generates a little less total distance in about 60 simulations.

Then, Table. V shows the total distances and the gross execution time when there are 1000 requests and 50 PVs. The table is divided into two parts: the upper part is the data set with capacity of 15, while the capacity in the lower part is 1000, regarded to be infinite. The column Time holds the durations in millisecond to complete the insertions for 1000 requests, and the column T.D. is short for total distance, whose unit equals to about 13.7 meters. From the upper part, we can see that Alg. 1 performs better in 3 out 10 simulations; The average promotion in total traveling distance is 6%; The average execution time of Alg. 2 is 1.94 times that of Alg. 1. Similarly, in the lower part, the average promotion in T.D. is 11.6%, and the ratio of execution time is 1.89. But in contrast, in lower part the execution time is longer, and the total distances are much smaller. We will explain all these

characteristics as follows.

1) Data indicate that Alg. 2 is not always better than Alg. 1 when the capacity is constrained as a small number. However, if the capacity becomes larger, Alg. 2 is likely to build shorter paths than Alg. 1.
2) In Alg. 2, every insertion step makes sure that IC is optimally small, but it can not forecast its impact on the future insertion steps, so a solution with every step locally optimum is not certainly global optimum.
3) When the capacity is set to be infinite, the performances become much better, and the execution time is longer. It's probably because larger capacity makes it possible to insert at more locations , and the time spent on searching is going up as well.
4) Since the ratio of execution time is still less than 2 even when the number of requests and the capacity both grow to 1000, we can conclude that our optimization for Alg. 2 works well, and its time complexity is close to $O(n)$.

TABLE V
RESULT: 1000 REQUESTS AND 50 PVS

	Alg.1		Alg.2	
No.	Time(ms)	T.D.	Time(ms)	T.D.
1	1927798	145809	3795349	125166
2	1895593	139312	3601380	125995
3	1924272	136200	3728778	121727
4	1894571	127816	3621946	134755
5	1906295	122872	3673107	131439
6	1917893	138998	3988072	122473
7	1924150	134103	3653835	117760
8	1867749	137388	3660761	123686
9	1894474	137204	3702970	122798
10	1902247	124734	3537999	137569
avg.	1905504	134444	3696420	126337
	Alg.1		Alg.2	
No.	Time(ms)	T.D.	Time(ms)	T.D.
1	2628650	43239	5619300	37829
2	3029626	39644	5238024	36113
3	2970041	44157	4767932	40935
4	3144396	43610	7037991	29266
5	2899855	44370	6390028	33393
6	2618378	46789	4360320	45165
7	3064938	36907	5413107	33430
8	2752770	42941	4710389	39906
9	2825366	44059	5378623	42760
10	3082874	41359	5827113	38717
avg.	2901689	42708	5474283	37751

V. CONCLUSION

In this paper, we introduce the model of Public Vehicle, and analyze the defect of its algorithm. To improve its performance, we propose a Combinatorial Insertion Algorithm to select the optimum case for insertion in searching for solutions. Through many small-scale and medium-scale simulations, we find the total distance decreases by a considerable percent, and meanwhile, time complexity of the new algorithm grows not very fast, so it's efficient and practical. But as the results show, the new algorithm is not always better than the original one, so if time complexity is not concerned, it's better to combine the two algorithms together.

ACKNOWLEDGMENT

This work was supported by the Natural Science Foundation of China (NSFC) projects (Nos. 61373155, 91438121, 61303202 and 61373156), the Key Basic Research Project (No. 12JC1405400), China Postdoctoral Science Foundation (2014M560334 and 2015T80433) and the Shanghai Pujiang Program (No. 13PJ1404600) of the Shanghai Municipality.

REFERENCES

[1] L. Kong, X. Chen, X. Liu, and L. Rao, "Fine: Frequency-divided instantaneous neighbors estimation system in vehicular networks," in *Pervasive Computing and Communications (PerCom), 2015 IEEE International Conference on*, pp. 172–177, 2015.
[2] Q. Xiang, X. Chen, L. Kong, L. Rao, and X. Liu, "Data preference matters: A new perspective of safety data dissemination in vehicular ad hoc networks," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pp. 1149–1157, April 2015.
[3] *Carpool - Wikipedia*. "https://en.wikipedia.org/wiki/Carpool.
[4] R. W. Hall and A. Qureshi, "Dynamic ride-sharing: Theory and practice," *Journal of Transportation Engineering*, vol. 123, no. 4, pp. 308–315, 1997.
[5] W. Herbawi and M. Weber, "Modeling the multihop ridematching problem with time windows and solving it using genetic algorithms," in *Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on(Volume:1)*, pp. 89–96, 2012.
[6] P. Toth and D. Vigo, *The vehicle routing problem*. Society for Industrial and Applied Mathematics, 2001.
[7] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations research*, vol. 35, no. 2, pp. 254–265, 1987.
[8] J. W. BAUGH JR, G. K. R. Kakivaya, and J. R. Stone, "Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing," *Engineering Optimization*, vol. 30, no. 2, pp. 91–123, 1998.
[9] M. Zhu, L. Kong, X.-Y. Liu, R. Shen, W. Shu, and M.-Y. Wu, "A public vehicle system with multiple origin-destination pairs on traffic networks,"
[10] *How it works - Google Driverless Car Project*. "http://www.google.com/selfdrivingcar/how.