On Pre-Processing for Least-Cost Carpooling Routing in a Transportation Network

Qifeng Lu
MacroSys LLC.
Arlington, USA
qilu1@vt.edu

Abstract— **Within a location based social network, carpooling is becoming more and more preferable among workers both living and working near each other due to the continuous increase in gasoline price and air pollution, and is more desirable when working places are far away from homes. To find the least-cost path for carpooling means to find the optimal order to traverse the homes and workplaces to pick up passengers and to drop off them. However, before the order searching is performed, in a large transportation network, it is prevalent to first obtain the least-cost path between any two locations, with one as a work place and the other as a home, to reduce computation time. In fact, for carpool, since only a few people can stay within a vehicle, it is fast to obtain the best pickup and drop-off order. Therefore, the bottleneck to obtain the least-cost route is indeed the calculation of the least-cost paths between any such two locations. The two existing dominant approaches to pre-compute these least-cost paths are Dijkstra's algorithm and A*, where Dijkstra's algorithm is used to compute single-origin multiple-destination least-cost paths while A* is used to compute the least-cost path between an origin-destination pair. In this paper, LU, a best first search algorithm and framework recently proposed to compute single-origin multiple-destination least-cost routes, is adopted in this pre-processing step to retrieve the optimal path to traverse the carpool-based pickup and drop-off locations. Its performance is compared with A* and Dijkstra's algorithm through a set of experiments in a large transportation network. The results demonstrate that LU is significantly faster than Dijkstra's algorithm and much better than A*.**

Keywords- LU, A*, Dijkstra's algorithm, Best First Search, Single-Origin Multiple-Destination, Carpool, Location Based Social Network

I. INTRODUCTION

Within a location based social network, carpooling is becoming more and more preferable among workers both living and working near each other due to the continuous increase in gasoline price and air pollution, and is more desirable when working places are far away from homes. In general, to carpool, a worker is assigned to start from him/her home to pick up the other workers from their homes and drop off them at their workplaces and to end at his/her workplace. This carpool-based least-cost route processing is similar to a Traveling Salesman Problem (TSP) that asks for a least-cost route traversing a set of non-ordered points of interest [1][2]. Compared to TSP, it has an additional constraint in the sense that a worker must be picked up at his/her home before he/she is dropped off at his/her workplace, i.e., an order is imposed on any origin-destination pair. In addition, compared to general TSPs, there are two distinct characters for carpooling routing. First, origins and destinations are likely to cluster. Second, the number of destinations for a ride is not large.

In a large transportation network, prior to obtaining the least-cost path for carpooling, which means to find the optimal order to traverse the homes and workplaces to pick up passengers and drop off them, similar to TSP, it is prevalent to obtain the least-cost path between any two locations, with one as a work place and the other as a home, to reduce computation time [1] [2]. Otherwise, a partial optimal traversal order may have to be evaluated at each vertex along a minimum-cost route between any two locations in a transportation network, which is computation-intensive. In fact, for carpool, since only a few people can stay within a vehicle, given the pre-computed least-cost routes, it takes no time for a computer to calculate the best pickup and drop-off order. The only issue is to guarantee that the pickup location of a person is always traversed before his/her drop-off location during the optimal traversal order searching. Therefore, the dominant factor of the computation cost to obtain the least-cost route is indeed the cost to calculate the least-cost paths between any such two locations, which is the major challenge to obtain the least cost route for carpooling.

A set of web sites are available to provide people the access to carpooling and even vanpooling. erideshare.com helps diverse people groups with different trip purposes organize carpooling. Vpsiinc.com provides vanpooling with more people sharing a ride than carpooling. However, till now, none of these websites provide a complete solution for carpooling or vanpooling. In other words, no site provides services to help organize carpooling or vanpooling and calculate the optimal routes for each ride.

Three fundamental and prevalent algorithms exist to process single-origin multiple-destination routes in a graph: Dijkstra's algorithm [3], Bellman-Ford algorithm [4], and LU [5]. Compared to Bellman-Ford algorithm, Dijkstra's algorithm is more efficient but only applicable to graphs with non-negative cost edges, while Bellman-Ford can be used in graphs with negative cost edges but still cannot handle cases with negative-cost cycles. Compared to LU, Dijkstra's algorithm is less efficient when the number of destinations is relatively small compared to the total number of vertices in a transportation network [5], which is exactly the case In a carpooling scenario where there are only a small number of

 ISBN: 978-1-61208-118-2

pickup and drop-off locations in a large transportation network.

The A* algorithm is a generalization of Dijkstra's algorithm for one origin and one destination routing [6]. It takes additional information from the problem domain into account to provide a lower bound on the "distance" from a generated state to the goal. It is best-first since the vertex chosen to be expanded in a graph is the one appearing to have the shortest path from the origin to the goal.

The performance of LU has been studied against Dijkstra's algorithm [5]. However, its performance against A* has not been explored. In this paper, LU is adopted to help calculate the least cost route for carpooling with clustered destinations more efficiently when compared to existing approaches, and extensive experiments and result analysis are performed to explore the performance differences between A* and LU for a set of carpooling scenarios in a large urban transportation network. The results demonstrate that LU is significantly faster than Dijkstra's algorithm and much better than A* when the number of locations is not larger than 12.

The paper is organized as follows. First, related work is presented in Section II. Next in Section III, the algorithm LU is introduced, an example is provided to illustrate how the algorithm works, and its characteristics are discussed and compared with A* and Dijkstra's algorithm. Section IV presents the experiment and result analysis, followed by conclusions in Section V. Future research is discussed in Section VI.

II. RELATED WORK

Dijkstra's algorithm is an algorithm to retrieve the shortest paths from a single source vertex to multiple destination vertices in a weighted, directed graph [3]. All weights must be nonnegative. Dijkstra's algorithm works as follows. First, Dijkstra expands the origin and generates its children, or states, and the cost from the origin to each generated state is assigned to that state. Thereafter, Dijkstra continues to expand the state with the least cost, and generate its children, assigning the corresponding cost to each generated child. This process continues until all the destinations are reached or no state can be expanded. Dijkstra's algorithm is used widely for routing in computer network, transportation network, etc. For example, for computer network routing, Dijkstra's algorithm is the prevalent working principle behind link-state routing protocols such as OSPF and IS-IS [7][8][9]. In routing assignment in transportation, Dijkstra's algorithm plays the key role [10].

Unlike Dijkstra's algorithm, the Bellman-Ford algorithm [4] can be used on graphs with negative edge weights, as long as the graph does not contain any negative cycle reachable from the source vertex. Compared to Dijkstra's algorithm in a graph with nonnegative cost edges, Bellman-Ford requires more time to retrieve the optimal solutions.

LU, a best first search algorithm, is another approach to retrieve single-origin multiple-destination least-cost routes [5]. It uses a heuristic, *h_LU*, estimated based on destinations yet-to-be-reached to expedite the search process following a best first way. In nature, LU is a framework that can adopt different heuristics to provide optimal, optimally efficient, and sub-optimal solutions [5]. As a result, the capability of best first search was first extended to process multiple-destination queries in a graph. LU is significantly faster than Dijkstra's algorithm when the number of destinations is much smaller than the total number of vertices in a transportation network and can perform worse when the number of destinations is comparable to the total number of vertices due to the additional time to compute the heuristics.

The A* algorithm is a generalization of Dijkstra's algorithm for one origin and one destination routing [6]. It takes additional information from the problem domain into account to provide a lower bound on the "distance" from a generated state to the goal. It is best-first since the vertex chosen to be expanded in a graph is the one appearing to be the closest to the goal. A* uses a distance-plus-cost heuristic function as *f(n)* to determine the order in which the search visits vertices in the graph [6]. *f(n)* is the sum of two functions: *g(n)*, the path cost function of the path from the origin to the current vertex *n*, and *h(n)*, the heuristic estimate of the distance from the current vertex *n* to the goal.

III. LU: A BEST FIRST SEARCH ALGORITHM TO RETRIEVE SINGLE-ORIGIN MULTIPLE-DESTINATION ROUTES IN A GRAPH

In this section, the details of the recently proposed algorithm LU to retrieve single-origin multiple-destination least cost routes [5] are presented. As a best first search, LU follows a vertex generation and expansion search process. It evaluates the promise, the closeness, of each generated vertex towards the destinations yet-to-be-reached. Starting from the origin, every time LU expands the most promising vertex based on some rule and generates its children, until all the destinations are expanded, or reached [5]. It takes advantage of useful information from a problem domain to expedite the search process in a graph without negative cost edges.

A. Algorithm

LU uses the following evaluation function, *f(n)*, to evaluate the promise of a vertex, or a state, n.

$$f(n)=g(n)+h\_LU(n) \quad (1)$$

where

f(n) is the estimate to the promise of a state *n* to be expanded,

g(n) is the cost from the origin state to the state *n*, and

h_LU(n) is the minimum estimate to the actual cost from the state *n* to any unclosed destination state.

h_LU(n) is evaluated through equation (2).

$$h\_LU(n)=\min(h_i(n)) \ (1<=i<=K) \quad (2)$$

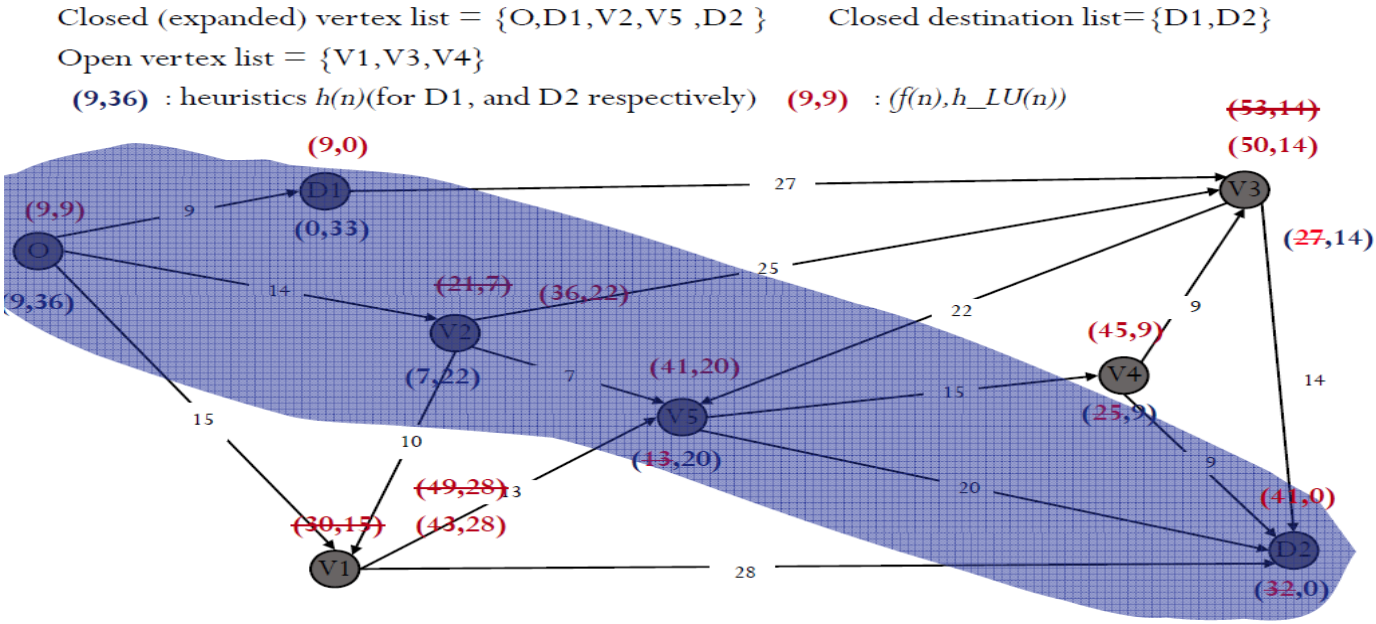

Figure 1. The LU search Process

where K is the number of unclosed, i.e., unreached, destination, and

$h_i(n)$ is the heuristic between n and the ith unclosed destination.

LU incrementally searches all paths leading from the starting vertex until it finds the path of minimum cost to any destination. It first takes the partial paths most likely to lead towards the unclosed destination appearing to have the least cost.

Similar to A*, LU also maintains a *closed list* where expanded states are stored, and a set of *partial paths*, unexpanded leaf states of expanded states. These partial paths are stored in an *open list*, also called a *priority queue*, a modified queue that outputs the one with the highest priority. Differently from existing best first searches, 1) LU has an array that marks a destination as closed after its path is found by LU, and a destination is *unclosed*, or *open*, when a route for the destination is not found yet; and 2) one additional variable, *Dest_H*, is adopted as one component of a generated state, *gs*, to indicate the unclosed destination appearing to be the one to which *gs* is the closest. In other words, *Dest_H* is used to indicate the unclosed destination towards which the heuristic is equal to *h_LU(gs)*.

Whenever an equal *f(n)* occurs, one is randomly selected to expand.

To retrieve least-cost routes for N destinations, LU first generates the origin and puts it into the open list. Next, for all the states in the open list, LU expands the state with the lowest *f(n)* value, and its children states are generated and their *f* values are evaluated based on all the unclosed destinations. The process continues until all destination states are reached or no solution is found, i.e., at least no route for one destination is found. Once a destination, *DS*, is reached, the algorithm will output the obtained path, mark the destination as closed, reevaluate the generated but unexpanded states whose *Dest_H* is equal to *DS*, and update their *f(n)*s in the open list by removing *DS* from consideration to calculate *h_LU(n)*, which results in equal or larger *f(n)*s.

LU is optimal, i.e., the retrieved routes are all optimal, when $h_i(n)$ is never larger than its corresponding actual cost, i.e., the actual least-cost between n and the ith unclosed destination.

B. An Example

Figure 1 presents the state snapshot when LU finishes its search for a problem asking for shortest routes from O to D_1 and D_2 respectively. For each vertex, ($cost_1$,$cost_2$) in blue represents the cost estimations, or heuristics, where $cost_1$ represents the estimated cost, or heuristic, to D_1 and $cost_2$ represents the estimated cost to D_2. For a vertex n, ($cost_3$,$cost_4$) in dark red represents the cost estimations where $cost_3$ represents f(n), and $cost_4$ represents *h_LU(n)*.The cross line in ~~($cost_3$,$cost_4$)~~ indicates that a state is generated first and then either discarded directly or put into the open list and later removed from the open list.

The search starts with the generation of state O in the open list. Its cost to D_1 and $_{D2}$ are evaluated and the corresponding *f(O)* and *h_LU(O)* are calculated. Next, since O is the only state in the open list, O is expanded, and its three children states, V_1, V_2, and D_1, are generated and their costs to the destinations are evaluated. Now D_1 has a minimum *f(n)* value, so it is expanded. A state for V_3 is generated and put into the open list. Since D_1 is a destination, D_1 is put into the closed destination list, and the state for V_1 and V_2 are re-evaluated only based on the cost estimation to D_2. Accordingly, *(30,13)* for V_1 is changed to *(43,28)*, and *(21,7)* for V_2 is changed to *(36,22)*. Then V_2 is selected because it has a smallest *f(n)* among V_1, V_2, and V_3. It generates its children states for V_1, V_3, and V_5 and put them

into the open list. Since the original states for V_1 and V_3 have smaller *f(n)*s, their newly generated states are discarded. Next, since V_5 has the lowest *f(n)* in the open list, it is expanded, and its children states for V_4 and D_2 are generated and put into the open list. Next, since D_2 has the lowest *f(n)*, it is expanded. Since it is a destination, it is put into the closed list. Now the closed list contains all the destinations, so the search stops. Therefore, the optimal routes obtained are $O->_{D1}$, and $O->V_2->V_5->D_2$.

C. Discussion

LU can be used for both route planning and dynamic routing. When a traffic event occurs, the costs of the corresponding street links can be adjusted accordingly. For example, the costs for closed streets can be considered as infinite and for congested links can be high so that an alternative route not traversing the closed streets can be chosen as the best route. Under these situations, only the graph representing the street network needs update, and no need to change the algorithm LU.

Compared to Dijkstra's algorithm, LU gains performance through reduced state generations. However, to retrieve *N*-destination least-cost paths, LU may not outperform *N* sequentially-running A*s when their corresponding expanded states do not significantly overlap. The major reason is that compared to an A* search process, LU must maintain a longer closed list and a longer open list. Consequently, it requires more time to update and re-order the open list and search the closed list. This additional cost will be dominant if the expanded states between different A* processes do not significantly overlap. Therefore, when the destinations are far away from the origin and not clustered, for example, uniformly distributed in a transportation network, LU may not outperform A*. When carpooling, generally workers are both living closely and working closely. In other words, both the origins and the destinations are likely to cluster, which indicates LU may perform much better than multiple sequentially-running A*s.

Since a pickup location, represented by *pkloc*, for a person is always traversed before his/her drop-off location, represented by *drl*oc, during the least-cost route calculation, no need to calculate any route from *drloc* to *pkloc*. In LU and Dijkstra's algorithm, this can be achieved by neglecting the corresponding *pkloc* as a destination when a route starts with a *drloc*.

IV. EXPERIMENT AND RESULT ANALYSIS

To investigate the performance of LU for route pre-processing to retrieve a carpool-based least-cost route traversing a set of pickup and drop-off locations, a set of experiments is performed, and Dijkstra's algorithm and multiple sequentially-running A*s are used as the baselines. Their performance is studied using network distance in a large dense urban transportation road network. In the experiment, each problem sample is to ask for a set of shortest routes, each of which is between two locations within the pickup homes and drop-off workplaces.

The Euclidean distance is used as the basis to calculate the heuristic *h_LU(n)* for each generated vertex *n* in LU. Since a Euclidean distance between two vertices in transportation network is never larger than the actual network distance, LU is optimal.

The experiment uses one large dense urban transportation network, the road network of Fairfax City and Fairfax County, US that contains 35,435 vertices and 82,926 directed edges, as shown in Figure 2. Both origins and destinations are clustered. In practice people use a van or a car for carpooling, so the number of pickup and drop-off locations, *N*, may not be larger than 12.

Figure 2. The road network of Fairfax county and Fairfax city, VA, US

Three data sets are generated. As shown in Figure 3, in each data set, an origin, represented by a green dot in Figure 3, is generated first, and then a destination around a specified Euclidean distance, *ED* (in mile), is generated. Thereafter, the other origins, represented by green dots in Figure 3, within a selected radius, *R*, of the origin and the other destinations, represented by red dots in Figure 3, within the same radius of the destination are generated. The origin number is either equal to or 1 larger than the destination number.

Data set I is used to investigate the impact of the number of pickup and drop-off locations on LU, Dijkstra's algorithm, and A*. It varies *N* from 3 to 12. *ED* is 10 miles and *R* is 0.5 mile. For each *N*, the number of problem samples, *PS*, is 30.

 ISBN: 978-1-61208-118-2

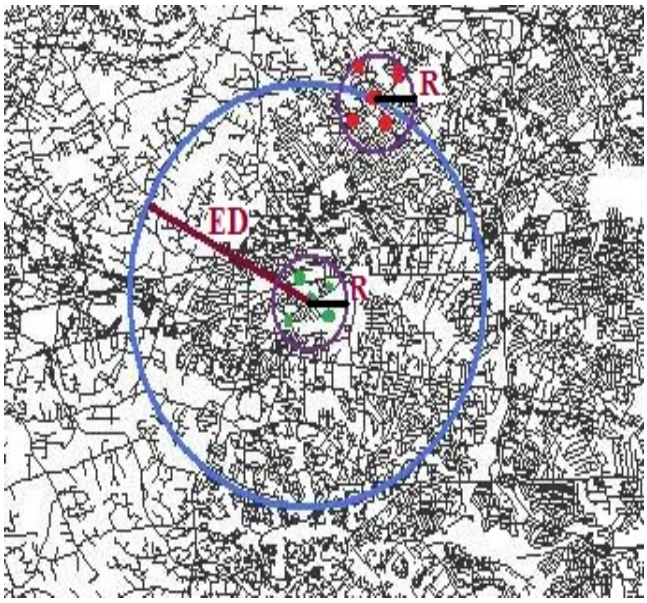

Figure 3. Data set generation

Data set II is generated to investigate the impact of the distance between homes and workplaces on LU and A*. It varies *ED* from 8 miles to 18 miles, with a fixed 2-mile interval. *N* is 3, and *R* is 0.5 mile. For each *ED*, *PS* is 30.

Data set III is adopted to investigate the impact of the radius size for carpool on LU and A*. It varies *R* from 0.5 mile to 1.0 mile. *ED* is 10 miles and *N* is 3. For each *R*, *PS* is 30.

A*, Dijkstra's algorithm, and LU are implemented with C#. The experiments are performed on a Toshiba Satellite A215 Laptop with 2.0GB memory (RAM), AMD Turion™ 64*2 Mobile Technology TL-56 1.80HZ processors, and Windows Vista™ Home Premium operating system.

A. Performance Measures

The following measures are used to analyze the performance of LU, Dijkstra's algorithm, and A*.

Average Shortest Distance (ASD): the average sum of shortest route distances obtained over all runs, in mile;

Average Number of States Expanded (ANSE): the average number of expanded states obtained over all runs;

Maximum Additional Number of States expanded by Dijkstra's algorithm or A (MaxANS):* For each run, compared to LU, obtain the additional number of states expanded by Dijkstra's algorithm or A*, and the measure is the maximum among all runs;

Minimum Additional Number of States expanded by Dijkstra's algorithm or A (MinANS):* For each run, compared to LU, obtain the additional number of states expanded by Dijkstra's algorithm or A*, and the measure is the minimum among all runs;

Average Process Time (APT): the time required to return the solution for a query, in second;

Maximum Additional Cost by Dijkstra's algorithm or A (MaxAC):* For each run, compared to LU, obtain the additional time cost required by Dijkstra's algorithm or A*, and the measure is the maximum among all runs;

Minimum Additional Cost by Dijkstra's algorithm or A (MinAC):* For each run, compared to LU, obtain the additional time cost required by Dijkstra's algorithm or A*, and the measure is the maximum among all runs;

Average Relative Number of States expanded (ARNS): the ratio of the number of states expanded by Dijkstra's algorithm or A* over by LU; and

Average Relative Process Time (ARPT): the ratio of the time processed by Dijkstra's algorithm or A* over by LU.

B. Results

The results are provided in Table I through Table VI. Dijk represents Dijkstra's algorithm. The minimum and maximum are highlighted in bold for *ASD, MaxAC, MinAC, ARPT, MaxANS, MinANS,* and *ARNS*. "-" represents a value is not available due to the high computation cost.

It is observed that all ASDs obtained from LU are the same as from Dijkstra's algorithm and from A*, which is direct evidence showing that LU retrieves optimal solutions with Euclidean distance as the basis to calculate its heuristics.

TABLE I. THE PERFORMANCE ON AVERAGE SHORTEST DISTANCE AND PROCESS TIME FOR DATA SET I

N	ASD	APT			ARPT	
		LU	*A**	*Dijk*	*A**	*Dijk*
3	49.5	12.1	13.1	733.4	1.1	60.6
4	100.4	26.5	37.9	1060.1	1.4	40.0
5	153.8	38.3	66.7	1567.7	1.7	41.2
6	223.6	16.9	39.9	1750.7	2.4	103.5
7	313.3	48.9	118.0	1746.4	2.4	35.7
8	394.9	36.5	75.1	2093.2	2.1	57.3
9	438.8	39.1	97.8	-	2.5	-
10	640.9	45.6	171.4	-	3.8	-
11	775.9	59.7	205.4	-	3.4	-
12	931.8	73.0	264.7	-	3.6	-

TABLE II. THE PERFORMANCE ON EXPANDED STATES FOR DATA SET I

N	ASD	ANSE			ARNS	
		LU	*A**	*Dijk*	*A**	*Dijk*
3	49.5	9972	76348	59372	7.6	5.9
4	100.4	14802	226009	82399	15.2	5.5
5	153.8	20328	593254	109821	29.1	5.4
6	223.6	19649	952359	124700	48.4	6.3
7	313.3	31809	2544339	133529	79.9	4.2
8	394.9	28206	2633455	150217	93.3	5.3
9	438.8	25504	5893107	-	231.0	-
10	640.9	43738	9211019	-	210.5	-
11	775.9	51828	14425335	-	278.3	-
12	931.8	57312	19725022	-	344.1	-

 ISBN: 978-1-61208-118-2

TABLE III. THE PERFORMANCE ON AVERAGE SHORTEST DISTANCE AND PROCESS TIME FOR DATA SET II

ED	ASD	APT		MaxAC	MinAC	ARPT
		LU	*A**			
8	40.5	4.2	5.2	1.8	-0.1	1.2
10	49.5	12.1	13.0	4.9	-1.2	1.1
12	59.4	13.1	15.4	5.9	-0.3	1.2
14	64.5	19.9	23.2	9.8	-0.5	1.2
16	70.1	96.9	146.8	246.9	0.4	1.5
18	83.3	110.9	139.8	119.6	-15.1	1.3

TABLE IV. THE PERFORMANCE ON EXPANDED STATES FOR DATA SET II

ED	ANSE		MaxANS	MinANS	ARNS
	LU	*A**			
8	6587	51362	59519	31877	7.8
10	9972	76348	126260	26931	7.7
12	12438	97819	132983	52870	7.9
14	14863	114118	143007	50346	7.7
16	21728	206193	301551	43721	9.5
18	24109	196415	294018	87167	8.1

TABLE V. THE PERFORMANCE ON AVERAGE SHORTEST DISTANCE AND PROCESS TIME FOR DATA SET III

R	ASD	APT		MaxAC	MinAC	ARPT
		LU	*A**			
0.5	49.5	12.1	13.0	4.9	-1.2	1.1
0.6	50.5	14.2	17.7	10.2	0.7	1.3
0.7	50.0	11.7	13.7	2.8	1.2	1.2
0.8	50.1	11.7	14.8	4.8	2.2	1.3
0.9	49.6	12.1	13.1	4.9	-1.2	1.1
1.0	49.7	11.8	14.5	5.0	1.1	1.2

TABLE VI. THE PERFORMANCE ON EXPANDED STATES FOR DATA SET III

R	ANSE		MaxANS	MinANS	ARNS
	LU	*A**			
0.5	9972	76348	126260	26931	7.7
0.6	11736	96549	112879	48974	8.2
0.7	11261	91033	99623	54530	8.1
0.8	11174	91257	99623	54530	8.2
0.9	9972	76348	126260	26931	7.7
1.0	10950	89573	99623	54530	8.2

Based on Table I through Table VI, the following conclusions can be drawn.

Compared to A* and Dijkstra's algorithm, based on MinANS values in Table II, Table IV, and Table VI, it is clear that LU always expands the least number of states. This is because LU is more informed than Dijkstra's algorithm and do not have to re-expand states, which occur when multiple sequentially-running A*s are used to retrieve all shortest pair-wise distances.

In practice, for carpooling, both pickup locations and drop-off locations likely cluster to reduce trip cost, gasoline usage, and emission. Compared to of A*, According to the ARPT values in Table I, Table III, and Table V, on average LU is about 0.1 to 2.8 times faster than A*. Especially, based on ARPT in Table I, when *N* increases, LU is increasingly faster than A*. According to ARPT in Table 1, it is clear that when the number of pickup and drop-off locations increase, Lu increasingly outperforms A* because more states are re-expanded by sequentially-running A*s.

It is clear that when *N* is small, LU significantly outperforms Dijkstra's algorithm in terms of computation efficiency. Based on Dijkstra's ARPT values in Table I, LU can outperform Dijkstra's algorithm by 2 magnitudes.

Based on ARPT values in Table III, when *ED* increases, generally LU is increasingly faster than A*.

Based on ARPT values in Table V, compared to of A*, the performance of LU does not have a clear relation to *R*.

Based on MinAC values in Table III and Table V, in some rare cases, LU may still be less efficient than A*.

V. CONCLUSION

Within a location based social network, carpooling is becoming more and more preferable among workers both living and working near each other due to the continuous increase in gasoline price and air pollution, and is more desirable when working places are far away from homes. Consequently, it is highly desirable to obtain the optimal traversal order to pick up carpool participants and drop off them to retrieve the least-cost carpooling route. However, in a large network, it is desirable to first compute least-cost pair-wise distances among the pickup and drop-off locations to reduce the computation complexity to retrieve the optimal route for carpooling.

In this paper, LU, a fundamental best first search algorithm and framework, is adopted to pre-compute all least-cost pairwise routes. In a carpooling scenario, both pickup locations and drop-off locations are likely to cluster to get most out of carpooling in terms of reductions in trip cost, emission, and gasoline usage. Accordingly, compared to the two existing prevalent algorithms, A* and Dijkstra's algorithm, LU is more appropriate to compute all least-cost pairwise network distances. A set of experiments is performed, and the results demonstrate that LU expands the least number of states when compared to A* and Dijkstra's algorithm, and 2) on average LU is much more efficient than A* and significantly faster than Dijkstra's algorithm when the number of pickup and drop-off locations are not larger than 12. On average, LU is 0.1~2.8 times faster than A* and outperforms Dijkstra's algorithm by 2 magnitudes.

VI. FUTURE RESEARCH

Even though LU significantly reduces overlapped states expanded by multiple sequentially-running A*s, LU may still be less efficient. One major reason is that unnecessary states having been used to search for the routes to the closed destinations but not helpful for searching the routes to the

 ISBN: 978-1-61208-118-2

remaining unclosed destinations are not removed timely in the current implementation of LU. Future research can be performed to reduce unnecessary states stored in the open list and the closed list whenever a destination is closed to expedite the search and update operations performed on both lists in LU, and thus to further improve the efficiency of LU.

REFERENCES

[1] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. The Traveling Salesman Problem: A Computational Study. Springer, 2007.

[2] S. Arora. Approximation Schemes for NP-hard Geometric Optimization Problems: A survey. Mathematical Programming, Springer, 97 (2003) pp. 43-69.

[3] E. W. Dijkstra: A Note on Two Problems in Connexion with graphs. In Numerische Mathematik, 1 (1959), S. pp. 269–271.

[4] Richard Bellman. On a Routing Problem, in Quarterly of Applied Mathematics, 16(1), pp. 87-90, 1958.

[5] Q. Lu. LU: A Best First Search to Process Single-Origin Multiple-Destination Route Query in a Graph. Proceedings of the 2010 Second International Conference on Advanced Geographic Information Systems, Applications, and Services, (2010) pp. 137-142.

[6] P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics SSC4 (2) (1968) pp. 100–107.

[7] Grout, V., (2003), Towards an Optimal Routing Strategy, Proceedings of IADIS WWW/Internet 2003, Algarve, Portugal, 5 th. - 8th. November, pp. 903-906

[8] Pierre A. Humblet. An adaptive distributed dijkstra shortest path algorithm. Technical Report CICS-P-60, Center for Intelligent Control Systems, MIT, May 1988

[9] Baruch Awerbuch. Shortest Paths and Loop-Free Routing in Dynamic Networks. SIGCOMM '90 Proceedings of the ACM symposium on Communications architectures & protocols: pp. 177-187

[1] Dafermos, Stella. C. and F.T. Sparrow. The Traffic Assignment Problem for a General Network." J. of Res. of the National Bureau of Standards, 73B, pp. 91-118. 1969.

 ISBN: 978-1-61208-118-2