Smart Ride Share with Flexible Route Matching

Chung-Min Chen*, David Shallcross*, Yung-Chien Shih†, Yen-Ching Wu†, Sheng-Po Kuo†, Yuan-Ying Hsu†, Yuhsiang Holderby†, William Chou‡

*Telcordia Technologies, Piscataway, NJ 08854, USA

†Telcordia Applied Research Center Taiwan, Taipei, Taiwan

‡Dmobile System, Taipei, Taiwan

{cchen2, yshih, ywu, skuo, ihsu, davids}@telcordia.com, william.chou@dmobile.cc

Abstract— **The concept of providing an on-line car pooling service to facilitate matching of drivers and riders has been around for a while. A number of systems or trials have also been built for use in university campuses or on-line communities. These systems however are quite restrictive in terms of the quality of matches and flexibility of pick-up/drop-off places, routes and schedules. Capitalizing on the ubiquitous wireless networks and GPS- enabled mobile devices, we have developed a smart ride-share system with an efficient scheduling algorithm for ride sharing, which can potentially achieve better vehicle utilization, energy consumption and user convenience. This paper describes the algorithms adopted in the system and presents a performance evaluation. The results show that with the proposed algorithm, ride-share scheduling can be achieved efficiently with large size of fleets.**

Keywords— **telematics, ITS, carpool, fleet scheduling**

I. INTRODUCTION

Carpooling has been around for many years, mostly realized as a community-driven service or through small affinity groups, where people arrange to share the rides, for example, between homes and work places. The immediate benefits of carpooling to the participants include shared driving load, saving in gas expenses, and reduction in personal vehicle mileages and thus vehicle depreciation.

Lately, the concept of carpooling has a new appeal to the government policy makers as well as concerned citizens in light of the global climate change. Carpooling, in its various formats, has the potential of saving energy consumption and reducing traffic congestion. The former helps to achieve a sustainable and everlasting human being life style; the latter contributes to improved quality of individuals' transportation experience.

Despite of its eminent benefits to the human society, carpooling in its traditional embodiment has not attracted popular user adoption. There are many reasons for this: some people are not feeling safe or comfortable riding with strangers; lack of automated tools for publishing and searching for carpool rides, inflexibility of pick-up/drop-off locations and times that cause inconvenience to potential users.

Most of the concerns can be mitigated with the modern information and communication technologies including the ubiquitous wireless networks, GPS-enabled mobile devices and social networks. Indeed, there are initiatives and standard work that re-address the carpool applications with modern and improved formats [1,2,3]. In this paper, we describe a Smart Ride-Share (SmartRide for short) system that we built and deployed in partnership with a number of LBS (Location-Based Service) vendors. SmartRide provides two major ride share applications: para-transit service and peer-to-peer (P2P) ride matching. This paper focuses on the para-transit service.

Para-transit is a type of transportation service that fills the market gap between the traditional fixed route, fixed schedule city bus service and the ad hoc taxi service. Traditionally, the para-transit service is operated by a service provider contracted by the city government which manages a fleet of cars, vans or small buses. Most of them provide transportation services to special groups of users such as senior citizens or the handicapped. Recently, para-transit has also been considered as a less expensive alternative to regular bus service in providing transportation to people living in the rural areas.

In para-transit, the users need to reserve the ride in advance by specifying pick-up location, time and drop-off location. The challenge for the service provide is how to schedule the rider pick-up/drop-off sequences while not compromising user satisfaction by introducing intolerable delay due to ride sharing. In the core of SmartRide we developed a scheduling algorithm that, given the daily reservation data, will minimize the number of vehicles needed with minimum total distance travelled, constrained by a threshold of user-experienced delay.

This paper proposes an efficient algorithm for the above ride-share scheduling problem for para-transit.

The rest of the paper is organized as follows: The Subsection below provide review of related work. Section II provides a system overview of SmartRide. In Section III we define the scheduling problem for para-transit ride sharing and describe the algorithm we proposed and developed. We examine the preliminary performance results of the scheduling algorithm in Section IV and provide conclusions in Section V.

A. Related Work

Para-transit fleet dispatch and dynamic ride-sharing are identified in ISO TC-204 standardization [4] as ITS services under the public transport service groups. An early work [1] describes the software functionality and architecture of a service called "dial-a-ride" that is similar to the concept of para-transit. A follow-up study [5] evaluates the cost and

benefits of the dial-a-ride transit service concept. A dynamic carpool prototype and related survey are reported in [2]. The work in [6] describes a method that uses short-range communications to discover available rides for share among requesting riders and vehicles. A grass-root initiative called SmartJitney [3] promotes the development of a nation-wide ride sharing platform using the modern information and communication technologies.

None of the above work proposes or describes any algorithm for shared ride scheduling. In contrast, our work proposes a fleet dispatch scheduling algorithm for ride-sharing.

II. System Overview

We have developed a telematics product bundled with various telematics and ITS applications. SmartRide is one of the applications which we offer to para-transit fleet operators. The service is offered in partnership with a number of partners including Dmobile System (a mobile device vendor), OLEmap (an on-line map service provider), MactionTech (a navigation software vendor) and CTS (a wheelchair van/bus vendor). Figure 1 provides an architecture overview of the SmartRide system.

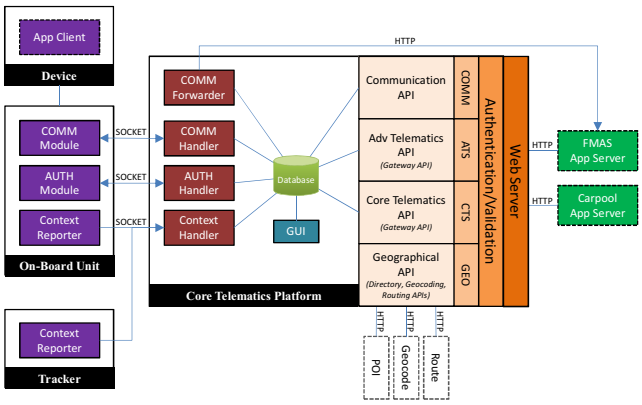

Figure 1. Sytem overview of SmartRide

The vehicles are equipped with GPS-equipped on-board-unit (OBU), which on one side may connects to user devices and on the other side connects to the Core Telematics Platform of the SmartRide system. The OBU contains data and voice communication capability to facilitate dispatch and tracking. Alternatively, the vehicle may simply use a less expensive one-way GPS+GSM/GPRS tracker without user interface. The driver will then communicate with the dispatch center through a regular cellular phone.

The Core Telematics Platform collects location data from the vehicles through cellular network and stores them in a database. Various applications can be developed on top of the platform which retrieves vehicle information including location through a Web Service interface. The platform also offers other needed common functionality such as user and vehicle authentication. The FMAS (Fleet Management Application Server) provides monitoring and tracking capabilities, whereas the Carpool Application Server hosts the ride sharing application logic and user interface.

III. Algorithmic Design

First we define the ride-share scheduling problem for para-transit. We then discuss the simpler case of no ride sharing and map the problem to the classical bi-partite matching problem which has found efficient algorithmic solutions. We then propose a 2-phase algorithm that combines a greedy method and the bi-partite matching algorithm, to solve the original ride-share scheduling problem.

A. Problem Statement

In this application the users are required to reserve their rides ahead of time. We call each reservation a *task request,* or simply *task*. A task is a triplet $X=(X_p, X_s, X_d)$, where X_p is the pick-up time, X_s is the pick-up location (source) and X_d is the drop-off location (destination). We assume the shortest time travelling route from location X to location Y can be computed by an underlying function and let $t(X,Y)$ be the travelling time. There are existed algorithms and software that can compute $t(X,Y)$ effectively.

If each vehicle can only carry a passenger at a time (i.e. no ride sharing), then the travel time for task A is $t(A_s, A_d)$. However in the case of ride sharing, user A may have to wait for extra time for the pick-up and the actual riding time may also be increased due to detour to pick up or drop off other co-riders. This delay must be bounded so as not to compromise user satisfaction. We set a system variable θ which serves as a threshold for the delay which requires, for any task A:

$$t_d(A) - A_p \leq \mathrm{t}(A_s, A_d) + \theta, \qquad \textbf{(Constraint I)}$$

where $t_d(A)$ is the actual drop-off time of A. This constraint means that the extra delay experienced by user A due to ride share should not exceed θ. Note in the above A_p is the reserved pick-up time so $t_d(\mathrm{A}) - A_p$ is effectively the actual travel time (including waiting and riding time) experienced by the user.

Now suppose we have a fleet of M vehicles, each of which may accommodate more than one passenger, and let $T = \{A, B, C, \ldots\}$ be the set of booked *tasks* for a given day, our objective is to assign the tasks to the vehicles using as few vehicles as possible, subject to Constraint I above. When there are more than one possible ways for the assignment, we also want to minimize the total distance travelled by the vehicles. Note the schedule of a vehicle may be interleaved with pick-up and drop-off of different riders due to ride sharing.

B. Scheduling without Ride-Share

We will first take a look at a simpler version of the original problem by considering no ride share and with the objective to minimize the number of vehicles needed to carry out a given set of tasks. We will relax these assumptions later.

It is not hard to see that a vehicle can serve tasks A and B in sequence if the following condition holds:

$$A_e + t(A_d, B_s) \leq B_p,$$

where $A_e = A_p + t(A_s, A_d)$ is the drop-off time of task A. Given a set of tasks T we can construct a directed graph G, whose nodes are the tasks, and there exists a directed edge from node

A to node B, denoted (A,B), if a single vehicle can serve task B after task A. The graph is *transitive*, in that if edges (A, B) and (B,C) are present, then edge (A,C) is present. Figure 2 shows an example of five tasks.

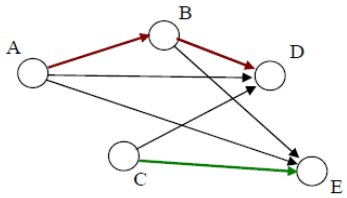

Figure 2. An example graph G for five tasks

Each path in the graph corresponds to a schedule that can be carried out by a single vehicle. Note a path with only one node and no edges corresponds to a schedule that serves only the task corresponding to the node. Given a graph with N nodes, in the worst case we need N vehicles to carry out the tasks if all N tasks are overlapped in time. But in most cases some of the tasks are non-overlapped and can be scheduled in sequence. Our goal therefore is to find the minimum number of disjoint paths that can cover all nodes in the graph. In Figure 2, the answer constitutes two disjoint paths: $A \rightarrow B \rightarrow D$ and $C \rightarrow E$. That is, the five tasks can be carried out by two vehicles.

To solve the problem, we map it to a 'maximum cardinality bipartite matching problem' [7]. We create, as follows in Figure 3, a new bipartite graph G' in which the nodes on the left side correspond to the drop-off time and the nodes on the right side correspond to the pick-up time of the tasks in T. In G', a directed edge from node A_E to node B_P exists if we can follow task A by task B on a single vehicle, that is, if in the earlier graph G, there was an edge from A to B.

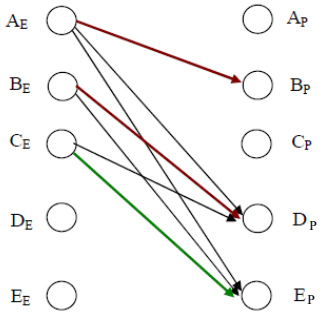

Figure 3. Bipartite graph G'

A disjoint set of directed paths in G is a set of edges such that each node of G has at most one incoming edge, and at most one outgoing edge. If G is covered by V disjoint directed paths, then V nodes (the first nodes in each path) have no incoming edge in the set, and the rest of the nodes have one incoming edge (likewise V nodes have no outgoing edge in the set, and the rest of the nodes have one outgoing edge). The number of edges in the set of disjoint directed paths covering G is therefore $(N\text{-}V)$. Such a cover of G by V disjoint directed paths corresponds to a set of links in the bipartite G', where each node A_E has at most one edge in the set, and each node B_P has at most one edge in the set. That is, it is a 'matching' [7] in G', consisting of $(N\text{-}V)$ edges. Thus, minimizing V in graph G is equivalent to maximizing the number of edges in the matching in graph G'. Hence, the problem of covering the nodes in G by a minimum number of disjoint directed paths is equivalent to the problem of finding a maximum cardinality matching in the bipartite graph G'.

This is a well-studied problem, and has a fairly simple solution algorithm with complexity $O(nm)$ [7], where n is the number of nodes in G', and m is the number of edges in G'. In the example of Figure 3, n is $2N$ and m is at most $N(N\text{-}1)$. The $O(nm)$ algorithm, attributed to van der Waerden and Kınig, is outlined below. We would apply this algorithm to G', taking input U as the set of left-hand side nodes X_E from G', and input W as the set of right-hand side nodes X_P from G'.

Input: a bipartite graph G with node sets U and W, and edges E.
Output: a maximum cardinality matching M^* of G.
Initialize $M = \varnothing$ (the empty set), as a subset of E.
Repeat steps 1-4 below until the loop exits in Step 3
1. Construct a directed graph D_M by re-orienting each edge $e=\{u,w\}$ of G (where $u \in U$, $w \in W$) as follows:
   a. if e is in M, orient e from w to u,
   b. if e is not in M, orient e from u to w.
2. Let U_M and W_M be the sets of nodes in U and W missed by M.
3. Find a directed path P in D_M from U_M to W_M, by, for example, breadth-first search. This will alternate reorientations of edges in M with edges not in M. If no such path is found, exit the loop.
4. Remove from M edges in $P \cap M$, and add to M edges in $P - M$. This will increase the size of M by 1.

Return the final value of M as M^*.

Minimizing Total Travel Distance

It may occur that G can be covered by V disjoint directed paths in more than one way. If we associate a cost C_{AB} for each edge (A, B) in G, we can choose a solution of minimum total cost. Such a cost might reflect total distance travelled, fuel consumption, or tolls. We can map this problem to that of finding a `maximum-weight matching' of size $(N\text{-}V)$ of G', i.e., of finding a matching of cardinality $(N\text{-}V)$ that has maximum weight among all matchings of size $(N\text{-}V)$. To do this, we just assign to each edge (A_E, B_P) in G' the cost C_{AB}.

The problem of finding a maximum-weight matching of a specified cardinality in a bipartite graph is again well-studied. It has a simple $O(n^2m)$ algorithm, and a more complicated $O(n((m+n) \log n))$ algorithm [8], where again n is the number of nodes in G', and m the number of edges in G'. We adopted the simple algorithm which is attributed to Kuhn and very similar to the algorithm for maximal cardinality bipartite matching. The algorithm is outlined below.

Input: a bipartite graph G with node sets U and W, and edges E, edge weights c_e for each edge e in E, and size k.
Output: a maximum weight matching M^* of G among all matchings of G of size k.
Initialize $M = \varnothing$, as a subset of E.
Repeat steps 1-4 below until M contains k edges:
1. Construct a directed graph D_M by re-orienting each edge $e = \{u,w\}$ of G (where $u \in U$, $w \in W$) as follows:
   a. if e is in M, orient e from w to u, and assign length $l_e = c_e$,
   b. if e is not in M, orient e from u to w, and assign length $l_e = -c_e$.
2. Let U_M and W_M be the sets of nodes in U and W missed by M.
3. Find a shortest (according to the edge lengths l_e) directed path P in D_M from U_M to W_M (this can be done by, for example, a breadth-first search). This will alternate reorientations of edges in M with edges not in M. If no such path is found, there is no matching of the required size.
4. Remove from M edges in $P \cap M$, and add to M edges in $P - M$. This will increase the size of M by 1.

Return the final value of M as M^*.

In our implementation we first run the maximal cardinality bipartite matching algorithm to find k – the minimum of vehicles needed. Then run the maximum-weight matching algorithm using k as an input to find the schedule with the lowest travel distance among all candidates with size k.

C. Greedy Method for Ride Sharing

The algorithms described above do not consider ride share, i.e. a vehicle can carry at most one passenger at a time. Adding this dimension to the problem makes it much harder to solve. We propose a 2-phase approach solution which turns out to be quite effective in practice for ride-share scheduling. In the first phase, we use a greedy method to identify tasks that can share on the same ride and merge them into a new task. Effectively this will produce a new set of tasks some of which are merged from the original tasks. We then take the new set of tasks as input to the bipartite algorithms as described above to generate the final schedule. We describe the greedy method in the following.

To simplify the discussion and without loss of generalization, we assume each vehicle can carry at most two passengers at any given time. This assumption can be easily relaxed in our algorithm. The greedy method will merge two tasks A and B into a new one if they overlap and will not violate Constraint I, i.e., the extra delay experienced by either passenger due to ride sharing should not exceed a pre-defined system threshold θ. We outline below the greedy method algorithm that takes as input a set of tasks T and produces a consolidated set of tasks T' by merging some of the overlapping tasks, subject to Constraint I.

Greedy Method for Ride Share
Input: ***T***
Output: ***T'*** (initially empty)
For each task *A* in *T* do
 For each task *B* in *T* do
  If (*A and B can be combined without violating Constraint I*)
  then {
   Combine *A* and *B* into a new task *C* and add *C* into *T'*;
   Remove *A* and *B* from *T*;
  }
 Endfor;
Endfor;
Move all remaining tasks in *T* to *T'*.

We describe next how to check whether two tasks A and B can be combined for ride share without violating Constraint I. Let us assume the following notations:

Xp: pick-up location of task X
Xd: drop-off location of task X
$t(X_p)$: reserved pick-up time of task X
$T(X,Y)$: travelling time from location X to location Y

Without loss of generality assuming $t(A_p) < t(B_p)$, then there are two possible vehicle routing scenarios if A and B share the ride:

Case A. Route $A_p \rightarrow B_p \rightarrow A_d \rightarrow B_d$ as shown in Figure 4.

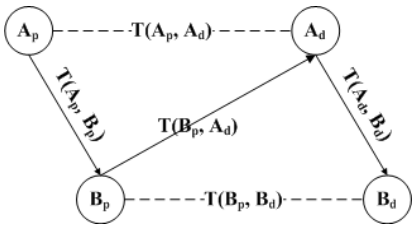

Figure 4. A scenario of ride share routing – Case A

In this case tasks A and B can share ride without violating Constraint I only if the following conditions are satisfied:

$$[T(A_p, B_p) + T_{A\_wait\_B} + T(B_p, A_d)] - T(A_p, A_d) < \theta$$
$$[T_{B\_wait\_A} + T(B_p, A_d) + T(A_d, B_d)] - T(B_p, B_d) < \theta.$$

In the above, $T_{A\_wait\_B}$ and $T_{B\_wait\_A}$ are defined as follows:

$$T_{A\_wait\_B} = \max\,(t(B_p) - t(A_p) - T(A_p, B_p)\,,\, 0)$$
$$T_{B\_wait\_A} = \max\,(t(A_p) + T(A_p, B_p) - T(B_p),\, 0)$$

Case B. Route $A_p \rightarrow B_p \rightarrow B_d \rightarrow A_d$ as shown in Figure 5.

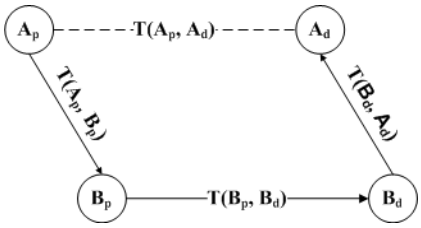

Figure 5. A scenario of ride share routing – Case B

In this case tasks *A* and *B* can share ride without violating Constraint I only if the following conditions are satisfied:

$$[T(A_p, B_p) + T_{A\_wait\_B} + T(B_p, B_d) + T(B_d, A_d)] - T(A_p, A_d) < \theta$$
$$T_{B\_wait\_A} < \theta,$$

where $T_{A\_wait\_B}$ and $T_{B\_wait\_A}$ are computed as before.

IV. PERFORMANCE EVALUATION

We have implemented the greedy ride sharing algorithms in our SmartRide para-transit application. The application is serving the handicap vans operators in Taipei for ride share scheduling. However as the current number of operating vehicles and rides are small (less than 50 vehicles), we opt to evaluate the performance of the algorithms using a simulated task generator. This will help us assess the scalability of the algorithm as the size of fleet grows to a large number.

In the experiments, we generated the tasks by randomly selecting pick-up and drop-off locations from a 100km x 100km area. The pick-up time for each task is also randomly generated. The vehicle speed is set to 60km per hour.

Figure 6 shows the result where we depict the algorithm computation time (*y*-axis) as a function of the number of tasks (*x*-axis). We also varied the number of vehicles, ranging from 50 to 200, each corresponding to a separate curve in the figure.

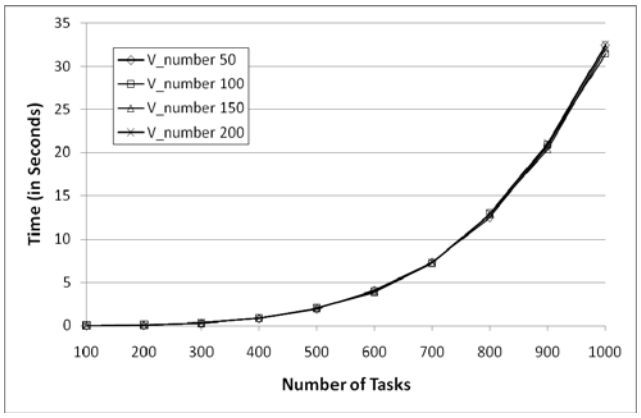

Figure 6. Performance results of the ride-share scheduling algorithm

The results show that the computation time rises as the number of tasks increases, at a rate close to $O(n^2)$ where n is the number of tasks. The time spent by the greedy method and by the bipartite matching algorithm is about 1:60 for 1000 tasks. The number of vehicles does not have much impact on the performance as the four curves do part from each other noticeably, though our algorithm computation time grows in $O(m)$ where *m* is the number of vehicles.

The algorithm spends about 32 seconds for scheduling 1000 tasks. This is quite efficient as the experiment is run on a Dell PC with Intel Core2 Duo 3.00GHz and 2GB RAM.

V. CONCLUSIONS

Para-transit has an appeal to government policy makers from the consideration of reducing energy consumption, air pollution, traffic congestion, and saving of public transportation cost. We have proposed an ride-share scheduling algorithm that combines a greedy method and bipartite matching algorithms to minimize the number of vehicles needed and total travel distance for an input set of dispatch tasks. We have built it into our SmartRide system which we offer to para-transit service providers in Taipei. The performance study shows that the scheduling algorithm is quite efficient in finding good fleet utilization schedule, taking only 32 seconds to complete for up to 1000 tasks.

ACKNOWLEDGMENT

We would like to thank Komandur Krishnan for discussions with us on many aspects of the scheduling problem and providing helpful directions.

REFERENCES

[1] R. B. Dial, "Autonomous dial-a-ride transit: Software functionality and architecture overview", *Transportation Research*, Vol. 3, No. 5, 1995, pp. 261-275.

[2] D. W. Massaro, B. Chaney, S. Bigler, J. Lancaster, S. Iyer, M. Gawade, M. Eccleston, E. Gurrola, and A. Lopez, "Carpool Now: just in-time carpooling without elaborate preplanning", in *Proc. 5th Int. Conf. on Web Information Systems and Technologies*, Lisbon, Portugal,, March 23-26, 2009.

[3] R. Content, "The SmartJitney: rapid, realistic, transit reinvention", Community Solutions, April 2009, www.communitysolution.org/rideshare.html.

[4] ISO TC 204 Standard TS14813-1:2007, "Intelligent transport systems – Reference model architecture(s) for the ITS sector, Part 1: ITS Service Domains, Service Groups and Services".

[5] S. W. Lau, "Autonomous dial-a-ride transit: benefits-cost evaluation", Volpe National Transportation Systems Center, U.S. Department of Transportation, August 1998.

[6] S. Winter and S. Nittel, "Ad-hoc shared-ride trip planning by mobile geosensor networks", *International Journal of Geographical Information Science*, Vol. 20, July 2006, pp. 899-916.

[7] http://en.wikipedia.org/wiki/Matching_(graph_theory)

[8] Kuhn, H.W, The Hungarian method for the assignment problem. Naval Research Logistics Quarterly. v2. 83-97.