Analysis and evaluation of the slugging form of ridesharing

Article · November 2013

DOI: 10.1145/2525314.2525365

CITATIONS
45

READS
338

2 authors:

Shuo Ma
University of Illinois at Chicago
8 PUBLICATIONS **708** CITATIONS

SEE PROFILE

Ouri Wolfson
University of Illinois at Chicago
269 PUBLICATIONS **9,832** CITATIONS

SEE PROFILE

Analysis and Evaluation of the Slugging Form of Ridesharing*

Shuo Ma
Department of Compute Science
University of Illinois at Chicago
Chicago, U.S.A.
sma21@uic.edu

Ouri Wolfson
Department of Compute Science
University of Illinois at Chicago
Chicago, U.S.A.
wolfson@cs.uic.edu

ABSTRACT

Ridesharing is a promising method to address transportation problems such as traffic jams and parking. Although traditional carpooling and taxi ridesharing have been investigated by many, slugging, as a simple yet effective form of ridesharing, has not been well-studied. In this paper, we formally define the slugging problem and its generalization. We provide proofs of their computational time complexity. For the variants of the slugging problem that are constrained by the vehicle capacity and travel time delay, we prove NP-completeness and also propose some effective heuristics. In addition, we discuss the dynamic slugging problem. We conducted experiments using a GPS trajectory data set containing 60 thousand trips. The experimental results show that our proposed heuristics can achieve close-to-optimal performances, which means as much as 59% saving in vehicle travel distance.

Categories and Subject Descriptors

J.m [**Computer Applications**]: Miscellaneous

General Terms

Algorithms

Keywords

ridesharing, slugging, NP-completeness, heuristics.

1 INTRODUCTION

Transportation problems, such as traffic jams, finding parking slots, hailing a taxi during rush hours, are long-existing headaches in cities, especially those with a large population. These problems negatively affect the environment, the economy, and more importantly average peoples' daily lives.

*This research was supported in part by the U.S. Department of Transportation National University Rail Center (NURAIL), Illinois Department of Transportation (METSI), National Science Foundation grants IIS-1213013, CCF-1216096, DGE-0549489.

SIGSPATIAL'13, November 05 - 08 2013, Orlando, FL, USA
Copyright 2013 ACM 978-1-4503-2521-9/13/11 $15.00.
http://dx.doi.org/10.1145/2525314.2525365

Different methods have been mainly proposed to tackle these problems separately. For example, extending the road network is one common approach to tackle traffic jams; sensors which can detect the availability of parking spaces [1] are installed to help drivers find open parking slots more quickly. However, those solutions often require additional construction or new equipment added to the existing infrastructures and thus are often expensive to implement. In addition, their benefits are usually limited to the specific corresponding problem.

One reason for the above transportation problems is that the passenger seats of vehicles are under-utilized. Thus, we study ridesharing as a promising means to improve the utilization of vehicle ridership and thus reduce the number of cars on the road.

Ridesharing practices have a variety of characteristics. For example, ridesharing can be either dynamic or static. Dynamic ridesharing arranges trips on a very short notice. By contrast, static ridesharing arranges trips that are known in advance, usually hours or a day or two before the departure time. Ridesharing can arrange either recurring or ad-hoc trips. Also, ridesharing can either change or keep the route of the original trips of drivers. (In case routes are kept, riders need to get on and off the driver's car at the origin and destination locations of the driver instead of their own.) Riders may share the cost with the driver or not. Table 1 summarizes the characteristics of some of the most common ridesharing applications.

Table 1 Characteristics of some most common ridesharing applications

Ridesharing Applications	Characteristics			
	Dynamic	Recurring Trip	Route Change	Cost Sharing
taxi ridesharing	yes	no	yes	yes
hitchhiking	yes	no	no	no
carpooling	no	yes/no	yes	yes
slugging [6]	yes/no	yes/no	no	no/very-low

In this paper we are interested in one particular ridesharing form, i.e. slugging. In slugging a passenger walks to the driver's origin, boards at the driver's departure time, alights at the driver's destination, then walks from there to the passenger's own destination. Thus slugging involves two modes of transportation, car and walking. Since slugging does not change any spatio-temporal aspect of the drivers' original trips, slugging is the simplest form of ridesharing in the sense of bringing minimum disruptions to the drivers. Thus it can be offered at minimum or no-cost to the riders. Compared to other forms of ridesharing where route change is allowed, e.g. taxi ridesharing [14], slugging avoids unnecessary complications such as complex fare mechanism or ridesharing-incurred travel time delay for drivers (e.g. due to unexpected congestion encountered on the way to some pickup). Thanks to its simplicity, slugging

has already become a common transport mode in some of the busiest traffic areas in the North America, e.g. auxiliary interstate highways around urban areas such as Washington D.C., Bay area, Houston, and other cities [2, 3].

Though currently slugging is mainly used for regular commute trips, we envision that it can also be applied to ridesharing scenarios that involve mostly one-time casual trips. For example, consider a ridesharing website where travelers post their trips scheduled in the near future. When posting their trip, travelers may announce their roles in ridesharing: drivers, passengers, or both (i.e. travelers who have a car can leave the role to be determined by the website). The website will compute a slugging plan to group these travelers and decide the driver and passengers for each group. The only attached string for a passenger is that she needs to walk to the origin location of the driver's trip before the driver departs, and she needs to walk from her driver's destination to her own destination. Drivers are willing to accept such a ride for a various reasons, such as environmental-friendliness, companionship, the privilege of driving on HOV lanes, reduced or waived toll on highways, small payment, etc.

The increasing popularity of bike sharing programs indicates that people are open to alternative modes of transportation, particularly the ones like slugging that involve physical activity (i.e. walking). The motor industry is also actively promoting shared services like slugging, as stated in the "Blueprint for Mobility" vision recently released by Ford company.

To the best of our knowledge, our work is the first one to study slugging from a computational perspective. We define and study the basic slugging problem and its variants that are constrained by the vehicle capacity and travel time delay. We also discuss the dynamic version of the slugging problem. The experimental results show that our proposed heuristics achieve 59% saving in vehicle travel distance. Given the size of our real data set is 39 thousand trips and the average distance of a trip in the data set is 6.3 kilometers, the saving equals to 144,963 kilometers, which means the reduction of over 4.5 thousand gallons of gasoline and 71 tons of carbon dioxide emission.

In summary, the contributions of the paper include:

- We formalize the slugging problem using a graph abstraction. We propose a quadratic algorithm to solve the slugging problem.
- We define a generalization of the slugging problem and prove its NP-completeness.
- For the variants of the slugging problem that are constrained by the vehicle capacity and travel time delay, we prove their NP-completeness and propose effective heuristics. Via extensive experiments, we demonstrate that the proposed heuristics have near-optimal performance in terms of the saving in vehicle travel distance.
- We also consider the dynamic slugging problem and evaluate it via experiments; in the dynamic problem the trips are announced incrementally.

The remainder of the paper is organized as follows. In Section 2, we review existing literature related to our work. Section 3 formally defines and studies the slugging problem, its generalization, its constrained variants, its dynamic version, and heuristics for the intractable variants. We evaluate the proposed heuristics in Section 4.

2 RELATED WORKS

In this section we review existing works on three problems that are relevant to slugging, i.e. taxi-ridesharing, carpooling and the dial-a-ride. Similar to slugging, all these problems are transportation problems that involve pickups and drop-offs. Unlike slugging where passengers change their origin and destinations in order to join the trip of drivers, in all three problems, drivers change their route in order to pick up and deliver the passengers. Both taxi ridesharing and carpooling are specific forms of ridesharing. The difference is that each driver in carpooling usually is associated with her own trip, while in taxi ridesharing this is not the case. Also taxi ridesharing usually needs appropriate pricing mechanisms to incite taxi drivers. The dial-a-ride problem slightly differs from carpooling as all vehicles start a trip and return to the same location called the depot.

2.1 Taxi Ridesharing

There have been a number of works on the taxi ridesharing application [14, 15, 16, 17]. These works modeled the taxi ridesharing problem by considering different constraints. In contrast to slugging, the routes of driver trips, i.e. taxis in this case, change to accommodate passengers. Among these works, some (see [17]) only considered vehicle capacity constraints, while the rest also considered time window constraints, i.e. travelers need to depart and arrive in given time intervals. [15] is the only paper that models monetary constraints, which are used to guarantee monetary incentives for both taxi drivers and taxi riders. These works on taxi ridesharing mainly concern the efficiency and scalability of ridesharing, i.e. how fast a query can be answered and how many queries the system can handle. In contrast, we focus on the effectiveness of slugging as a whole, e.g. the saving in vehicle travel distance, while the existing works on taxi ridesharing often consider the effectiveness of ridesharing from the perspective of a single request, e.g. reducing the increase in vehicle travel distance for every new request [14].

2.2 Carpooling

There have been many works on modeling and analyzing the traditional carpooling problem where drivers need to change their routes due to ridesharing. In [7], the authors modeled a carpooling problem and proposed an exact method based on Lagrangean column generation to solve it optimally. Since the carpooling problem is NP-hard, the exact approach practically only works for small instances of the carpooling problem, where there are at most a few hundred trips. For large instances with hundreds of thousands trips, many heuristics have been proposed [4, 18]. These heuristics are applied to compute the best route of a vehicle for a given set of requests, since the route of drivers is allowed to change. As such route changes do not occur in slugging, these heuristics are not applicable.

Despite being a sibling of the carpooling problem, the slugging problem has so far drawn little attention from researchers. There have been some reports on the current state of slugging operations (see [8]). But our work is the first formal study of slugging from a computational viewpoint.

2.3 Dial-A-Ride Problem (DARP)

The *Dial-A-Ride Problem (DARP)* [5], a.k.a. the Vehicle Routing Problem with Time Windows in the operation research literature, is closely relevant to the carpooling problem. The *DARP* can be considered the carpooling problem with additional

restrictions (e.g. all vehicles are required to start any trip from a depot location and return to the depot after the trip). In contrast to slugging, vehicle routes are manipulated to accommodate passengers' origin and destination locations. *DARP* is proved to be NP-hard. Cordeau et al. summarizes the state-of-the-art heuristics for *DARP* [9].

3 SLUGGING

We introduce the concept of slugging in Sec. 3.1. We formally define the basic slugging problem in Sec. 3.2. Next we introduce and discuss the vehicle-capacity constrained slugging problem in Sec. 3.3, and the delay bounded slugging problem in Sec. 3.4. Then we describe the slugging problem with both constraints and propose heuristics for it in Sec. 3.5. Finally, we discuss the dynamic slugging problem and its parameters in Sec. 3.6.

3.1 Preliminaries

In slugging, some travelers abandon their original trips and join the trip of other travellers, the drivers, without asking the drivers to change their route or their departure time. To be more specific, consider two travellers A and B, and their respective trips T_A and T_B, each of which is described by an origin destination pair and a start time at which the traveller intends to depart. Assume that traveller A abandons her trip and joins B's trip. In this case we say that T_A is *merged into* T_B. More specifically, traveller A executes her new trip as follows: at the start time of T_A she walks to the origin location of trip T_B, then she waits until the start time of T_B (if A arrives later than the start time of T_B then she cannot join T_B), she shares the ride with B, she alights at the destination of T_B and finally she walks from there to her own destination. Clearly, the only impact that traveller A has on trip T_B is the occupation of one seat in B's vehicle. In other words, there is no disruption to any spatio-temporal aspect of T_B.

In the above example, there is only one traveler associated with each trip. In general, each trip can be associated with a party of multiple travelers who cannot be separated during the trip (assuming that the size of the party is always smaller than the number of seats in a vehicle).

As shown in the above example, one necessary condition for trip T_i to be able to be merged into trip T_j is that the travellers of trip T_i can walk from the origin of T_i at the start time of T_i and arrive at the origin of trip T_j before the start time of T_j (assuming a constant walking speed and taking the shortest path). Consider a set of trips $S_T = \{T_1, T_2, \dots, T_m\}$ where the travelers of each trip T_i announce their willingness to serve as: driver, or passenger, or both. Then for each trip pair T_i and T_j, where the travelers of T_i have announced their willingness to be passengers, and the travelers of T_j have announced their willingness to be drivers, we can compute whether or not T_i can be merged into T_j. To do that, a preprocessing stage is performed. At this stage, a map is used to compute the shortest path between the respective origins. Specifically, for such a trip pair (T_i, T_j), the shortest path between the origins of the two trips is computed. Based on the calculated shortest path, a presumed walking speed, and the start times of T_i and T_j, we can readily determine whether or not trip T_i can be merged into T_j. If so, we say that pair (T_i, T_j) is a *mergable pair* where T_i is a *passenger trip* and T_j is a *driver trip*. For a mergeable pair (T_i, T_j), the shortest path between the destinations of T_i and T_j is also calculated in order to determine the travel time delay for the passenger trip T_i. The travel time delay for passenger trips imposes a natural constraint on the slugging problem, which will be discussed further in Sec. 3.4 and 3.5.

Now that we have defined a mergeable pair, for a given set of trips, consider the set of all mergeable pairs represented as a graph S. Assuming that the trip start-times are distinct, we observe that S possesses the following two properties.

First, S is *acyclic*. Suppose there exists a cycle of mergeable pairs (T_{i_1}, T_{i_2}), (T_{i_2}, T_{i_3}), …, (T_{i_n}, T_{i_1}) in S. Mergeable pair (T_{i_n}, T_{i_1}) means that the start time of T_{i_n} is smaller than that of T_{i_1}. However, the first n-1 pairs of the cycle collectively tell us that the start time of T_{i_1} should be smaller than that of T_{i_n}. Contradiction. In other words, S is acyclic because the start-times of the trips on a path in S are increasing.

Second, S is *transitive* (i.e. if $(T_i,\ T_j) \in S$ and $(T_j,\ T_k) \in S$), then $(T_i,\ T_k) \in S$. If the travelers of T_i can arrive at the origin of T_j before the start time of T_j, and the travelers of T_j can arrive at the origin of T_k before the start time of T_k, then the travelers of T_i definitely can arrive at the origin of T_k before the start time of T_k as well by: first arriving at the origin of T_j and then taking the same path used by the travelers of T_j to the origin of T_k; this assumes that all travelers have the same walking speed.

3.2 Basic Slugging Problem

Slugging is a graph problem. We formulate it as follows.

Definition 1 *A slugging graph $G = (V, E)$, is a directed acyclic graph where $V = \{T_1, T_2, \dots, T_m\}$ is a set of trips and E is set of directed edges between nodes that is transitive, i.e. if $(T_i, T_j) \in E$ and $(T_j, T_k) \in E$, then $(T_i, T_k) \in E$.*

Note that a node in a slugging graph may not have any incident edges. A node with no incident edge can exist as it represents a trip that cannot be merged into any other trip, or into which no other trip can be merged. For example, a trip geographically bounded in the northeastern corner of a city may become such a disconnected node if all other trips are bounded in the southwestern corner of the city, and they all start at approximately the same time.

A slugging graph indicates which trips can be merged into others. However, although a trip can be merged into multiple other trips, in a concrete slugging plan it is merged into only one other trip. In other words, a slugging graph gives the possible pairs of trips that can be combined, whereas a slugging plan gives an actual combination that will be executed in practice. So, based on a slugging graph, a slugging plan can be constructed. Intuitively, a slugging plan is a subgraph of the slugging graph that gives the driver and the passengers of each car.

Definition 2 *Given a slugging graph $G = (V, E)$, a slugging plan $G_S = (V, E_S), E_S \subseteq E$, is a subgraph of G that satisfies the following conditions: (i) $\forall (T_i, T_j) \in E_S$, there is no $k \neq j$ such that $(T_i, T_k) \in E_S$; and (ii) $\forall (T_i, T_j) \in E_S$, there does not exist k such that $(T_k, T_i) \in E_S$.*

Intuitively, condition (i) states that any trip T_i can be merged into at most one other trip. Condition (ii) states that a trip T_i can be merged into another trip T_j only if there is no other trip T_k that has been merged into T_i. These constraints precisely reflect the nature of the slugging problem: each trip is either a ridesharing provider, i.e. providing a car to be shared with other

riders, or a ridesharing consumer, i.e. taking exactly one ride provided by a provider.

Fig. 1 gives an illustrative example of slugging plans. Subfigure (a) shows a slugging graph of four trips. Subfigures (b) (c) (d) (e) show all slugging plans that are maximal, i.e. cannot include more edges. For instance, consider the slugging plan shown in subfigure (b). Given that (T_4, T_3) already exists, neither edge (T_4, T_1) nor edge (T_4, T_2) can be added because the addition violates Condition (i), and neither edge (T_3, T_2) nor edge (T_3, T_1) can be added because either addition violates Condition (ii).

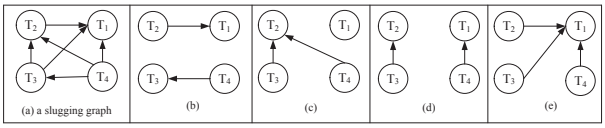


Fig. 1 An illustrative example of slugging plans

A mergable pair (T_i, T_j) in a slugging plan means that T_i is merged into T_j. That is to say, T_i is simply eliminated while there is no change to T_j other than the fact that the number of passengers in T_j 's vehicle is increased. Therefore the benefit of merging T_i into T_j only depends on the passenger trip T_i and thus can be measured by some attribute of T_i, e.g. the vehicle travel distance that is saved. In other words, the benefit of merging T_i into another trip is independent of the other trip. The implication is that if an edge is labeled by the benefit of merging the two trips at its endpoints, then all the edges exiting a node have the same benefit. Formally, we define the benefit of a slugging graph as follows.

Definition 3 *A slugging graph $G = (V, E)$ is called benefit-labeled if each edge $(T_i, T_j) \in E$ is associated with a label $B(T_i, T_j) \in \mathbb{R}^+$, referred to as the benefit of edge (T_i, T_j), and the benefits of all edges outgoing of the same node are identical, i.e. $\forall T_i, T_j, T_k$ such that $(T_i, T_j) \in E$ and $(T_i, T_k) \in E$, $B(T_i, T_j) = B(T_i, T_k)$.*

A straightforward example of a benefit function is the constant function $B(T_i, T_j) = 1$ for any mergeable pair (T_i, T_j). Intuitively, this benefit function measures the number of trips saved by ridesharing. Another example of a benefit function is: $B(T_i, T_j)$ equals to the vehicle travel distance of trip T_i. Intuitively, this benefit function measures the saving in vehicle travel distance.

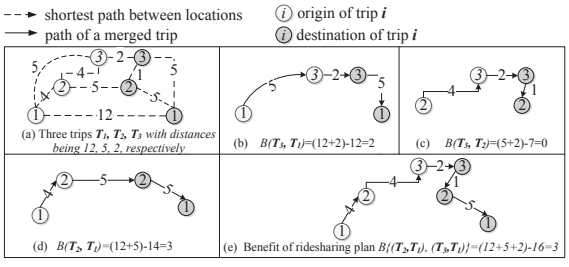


Fig 2 An example of a benefit function for a ridesharing form in which driver trips are changed

Definition 3 essentially says that the benefit of a mergeable pair is independent of the driver trip. Note this characteristic is unique to slugging and is not applicable to other ridesharing forms. For example, if we consider a ridesharing form where the route of driver trips can be changed, such as taxi ridesharing, then a benefit function B that measures the saving in the total travel distance is dependent on the driver trip. Fig 2 shows an illustrative example of this case. Fig 2 (a) shows three trips with their travel distances, and the distances between the origins and destinations of these trips. Fig 2 (b) and (c) show a trip after merging T_3 into trip T_1 and T_2, respectively, resulting $B(T_3, T_1) = 2$ and $B(T_3, T_2) = 0$. In other words, since the passenger is picked up at her origin and dropped off at her destination, the total saving in travel distance depends on the driver's origin and destination.

The next definition gives the benefit of a ride-sharing plan as the total benefit of its edges.

Definition 4 *Given a slugging graph $G = (V, E)$ that is benefit-labeled, the benefit of a slugging plan $G_S = (V, E_S)$, denoted by $B(G_S)$, is the sum of the benefits of the edges in E_S. That is to say, $B(G_S) = \sum_{(T_i, T_j) \in E_S} B(T_i, T_j)$.*

Definition 4 is also applicable to slugging only, but not to other ridesharing forms. To illustrate this point, consider again the example shown by Fig 2. The benefit of merging T_2 into T_1 is 3, as shown by Fig 2 (d); and the benefit of merging T_3 into T_1 is 2, as shown by Fig 2 (b). However, as shown by Fig 2 (e), the benefit of slugging plan $\{(T_2, T_1), (T_3, T_1)\}$ is 3 rather than 5, which is the sum of the benefit of the two pairs in the plan.

Problem 1 *Given a slugging graph $G = (V, E)$ that is benefit-labeled, find a subgraph $G_S = (V, E_S), E_S \subseteq E$ that is a slugging plan and has the maximum benefit. We refer to this as the Slugging Problem (SP).*

Theorem 1 *SP can be solved in $O(|V|^2)$ time.*

Proof A trip $T_i \in V$ is called a *sink trip* if its node has no outgoing edges. Due to the fact that G is acyclic and transitive, for each non-sink trip T_i, there exists at least one sink trip T_s such that $(T_i, T_s) \in E$.

Now we can construct the optimal slugging plan for *SP* using the algorithm as shown by Fig. 3. The G_S in Fig. 3 merges each trip T_i that is not a sink trip into any sink trip T_k such that $(T_i, T_k) \in E$.

Algorithm 1: Quadratic Algorithm for *SP*

Data: the slugging graph $G = (V, E)$ that is benefit-labeled
Result: the ridesharing plan G_S with the maximum benefit

```
passengerTrips ⟵ ∅
driverTrips ⟵ ∅
for edge (T_i, T_j) ∈ E do
    passengerTrips ⟵ passengerTrips ∪ {T_i}
    driverTrips ⟵ driverTrips ∪ {T_j}
sinks ⟵ driverTrips − passengerTrips
G_S ⟵ ∅
for trip T_i ∈ V do                    /* iterate all trips in the set */
    if T_i ∉ sinks then
        Pick any sink trip T_k ∈ sinks such that (T_i, T_k) ∈ E and add edge
        (T_i, T_k) to G_S
return G_S
```

Fig. 3 Quadratic algorithm for *SP*

It is not hard to see that the constructed $G_S = (V, E_S)$ is indeed optimal. First G_S is constructed such that each passenger trip has been merged into some driver trip. And since that the benefit of merging a passenger trip is the same regardless which driver trip the passenger trip is merged into, therefore, the benefit of G_S is maximum

Let us consider the time complexity of Algorithm 1. As shown by Line 1~6, trips that are sinks can be identified in $O(|E|)$ time. From Line 8~10, the slugging plan is calculated. Since there are at most $O(|V|)$ non-sink trips, and for each non-sink trip it takes at most $O(|V|)$ time to find a sink trip into which the non-sink trip can be merged, then the time complexity of Line 8~11 is $O(|V|^2)$ as well. Since $|E|$ is $O(|V|^2)$, the time complexity of Algorithm 1 is $O(|V|^2)$. □

The transitivity of the slugging graph relies on the assumption that travelers walk at the same speed. If we relax this assumption, then the slugging graph is no longer transitive. This relaxation leads to a generalization of *SP* in which the graph is only acyclic. We prove in the appendix that this generalization of *SP* is NP-complete.

3.3 Capacitated Slugging

The basic slugging problem may work well for the case where vehicles have a large number of seats, such as (mini)buses. The reason is that the problem does not constrain the number of passengers that a driver can take. The problem becomes more general if we consider a vehicle capacity constraint, given the fact that private vehicles usually have a few seats. Thus we introduce the slugging problem with the capacity constraint.

As mentioned in Sec. 3.1, a trip can be associated with multiple travelers who ride together. In other words, these travelers have the same origin, destination, and start time. Therefore, each passenger trip in the graph should be tagged with a label which represents the number of travelers associated with the trip. We do so as follows.

Definition 5 *A slugging graph $G = (V, E)$ is called no-of-travelers-labeled if each node $T_i \in V$ that represents a passenger trip (i.e. has outgoing edges) is associated with a number $T_i.s$, referred to as the size of node T_i.*

Each driver trip also has a number of seats available for passengers. In other words, each driver trip is associated with a number of travelers. However, it may still have available seats in the car to take slugging passengers. This availability is represented in the slugging graph as follows.

Definition 6 *A slugging graph G=(V, E) is called no-of-available-seats labeled if each node $T_i \in V$ that represents a driver trip (i.e. has incoming edges) is associated with a label $T_i.c$, referred to as the capacity of node T_i.*

Definition 7 *A slugging plan $G_S = (V, E_S)$ is capacitated if each driver trip in G_S takes at most $T_j.c$ additional passengers, i.e. $\forall T_j \in V, \Sigma_{(T_i,T_j)\in E_S} T_i.s \leq T_j.c$.*

Now we define the *Capacitated Slugging Problem* as follows.

Problem 2 *Given a slugging graph $G = (V, E)$ that is no-of-travelers-labeled, no-of-available-seats-labeled and benefit-labeled, and a number $R \in \mathbb{R}^+$, find a subgraph $G_S = (V, E_S), E_S \subseteq E$ that is a capacitated slugging plan with the benefit at least R. We refer to this as the Capacitated Slugging Problem (CSP).*

Theorem 2 *CSP is NP-Complete.*

Proof: First, it is easy to see *CSP* is in NP. That is, given a subgraph of G, denoted by G_S, we can verify whether G_S is a capacitated slugging plan and if so, whether its benefit is at least R. Now, we prove *CSP* is NP-hard by reducing the *0/1 Knapsack Problem* to *CSP*.

The *0/1 Knapsack Problem* is known to be NP-hard [11]. The decision version of the problem is defined as follows: given a set of n items, $\{p_1, p_2, \dots, p_n\}$ and a knapsack of capacity W. Each item p_i has a size w_i and a value v_i. The question is whether or not we can pack items worth at least R into the knapsack without exceeding its capacity and without splitting items.

Given an instance of *0/1 Knapsack Problem*, we can build an instance of *CSP* as follows. First we construct a slugging graph $G = (V, E)$ as follows. Let the node set $V = \{T_{p_1}, T_{p_2}, \dots T_{p_n}, T_d\}$. Construct the edge set E as follows. For each node T_{p_i}, $i = 1,2, \dots n$, we add an edge (T_{p_i}, T_d) to E. Note that the edge set $E = \{(T_{p_i}, T_d)\}$ is indeed transitive and acyclic. Therefore, G is a slugging graph. Next we label the nodes of G. Each node T_{p_i} is labeled with a size equals to w_i. The capacity of node T_{p_i} does not matter since they can only be passenger trips. Node T_d is labeled with a size equals to 1 and a capacity equals to $W + 1$. Next we label the edges of G with a benefit. Each edge (T_{p_i}, T_d) is label with a benefit $B(T_{p_i}, T_d)$ equals to v_i, for $i = 1,2, \dots n$. Now G is slugging graph that is no-of-travelers-labeled, no-of-available-seats-labeled and benefit-labeled.

It can readily be shown that the constructed instance of *CSP* has a capacitated slugging plan with a benefit of R if and only if the instance of *0/1 Knapsack Problem* can pack items worth at least R into the knapsack.□

3.3.1 A special case of CSP

A special case of *CSP* where the capacity of each car is 2, and all trips are associated with only one traveler, is polynomial-time solvable. We prove it formally as follows.

Problem 3 *Given a slugging graph $G = (V, E)$ that is no-of-travelers-labeled where the size of each passenger node is 1, and no-of-available-seats-labeled where the capacity of each driver node is 1, and benefit-labeled, find a subgraph $G_S = (V, E_S), E_S \subseteq E$ that is a capacitated slugging plan with the maximum benefit. We refer to this as the 1-traveler-1-availability Capacitated Slugging Problem (1t1CSP).*

Theorem 3 *The 1t1CSP is solvable in $O(|V||E|\log|V|)$ time.*

Proof We will show that the *1t1CSP* is equivalent to the maximum weighted matching problem. A matching of a graph is a set of pairwise vertex-disjoint edges. The maximum weighted matching problem is defined as: given an edge-weighted undirected graph $G_M = (V_M, E_M)$, find the matching where the sum of the weight of the edges in it is maximum.

Given an slugging graph $G = (V, E)$ of the *1t1CSP*, we construct a weighted undirected graph G_M as follows: $V_M = V$, $E_M = E$, the weight of an edge $e \in E_M$ equals to the benefit of e. Since E is acyclic, G_M contains no self-loops. Thus, each matching M of G_M is a legitimate capacitated slugging plan and the sum of the weight of edges in M equals to the benefit of the slugging plan.

Since the maximum weighted matching problem is solvable in polynomial time [10], we can also solve the *1t1CSP* in

polynomial time using the same algorithm. The running time of this algorithm is $O(|V||E|\log|V|)$. □

3.4 Delay-Bounded Slugging

In addition to the vehicle capacity constraint, it is also natural to constrain *SP* by a bounded travel time delay. As mentioned in Sec 3.1, in the preprocessing stage (that uses a map), for each mergeable pair (T_i, T_j), we compute the travel time delay for the passenger trip T_i, denoted by $\Delta_{i\to j}$. Intuitively, $\Delta_{i\to j}$ is the delay incurred by T_i due to the fact that T_i needs to walk to/from T_j's origin/destination, and possibly wait for T_j to start. More specifically, the delay equals to the difference: (the arrival time of T_i when the passengers ride with T_j (i.e. walk to/from origin/destination of T_j)) – (the arrival time of T_i when the passengers ride in their own vehicle from their origin directly to their destination). Now we define the travel time delay representation in the graph.

Definition 8 *A slugging graph $G=(V,E)$ is called delay-labeled if each edge $(T_i,T_j)\in E$ is associated with a label $\Delta_{i\to j}$, which represents the travel time delay of T_i with respect to (T_i,T_j).*

The travelers of each trip T_i can specify a threshold which represents their maximum tolerable travel time delay. We define the delay threshold representation in the graph.

Definition 9 *A slugging graph $G=(V,E)$ is called delay-threshold-labeled if each node $T_i\in V$ is associated with a label $T_i.\delta$, referred to as the delay threshold of node T_i.*

The travel time delay constraint means that all edges outgoing of a node with a travel time delay exceeding the delay threshold of the node need to be filtered out. Thus we define a delay-bounded-slugging-graph that satisfies this property.

Definition 10 *Given a slugging graph $G=(V,E)$ that is delay-labeled and delay-threshold labeled, the delay-bounded slugging graph $G_\delta=(V,E_\delta), E_\delta\subseteq E$, is a subgraph of G where $\forall(T_i,T_j)\in E_\delta, \Delta_{i\to j}\le T_i.\delta$.*

Now we introduce the *delay-bounded slugging problem.*

Problem 4 *Given the delay-bounded slugging graph G_δ that is benefit-labeled, and a number $R\in\mathbb{R}^+$, find a subgraph G_S of G_δ that is a slugging plan with a benefit of at least R.*

3.5 Delay Bounded and Capacitated Slugging and Its Heuristics

In practice, both the capacity constraint and the travel time delay threshold constraint are important. Thus, we combine them to form the following problem.

Problem 5 *Given the delay-bounded slugging graph $G_\delta=(V,E_\delta)$ that is no-of-travelers-labeled, no-of-available-seats-labeled, and benefit-labeled, and a number $R\in\mathbb{R}^+$, find a subgraph $G_S=(V,E_S), E_S\subseteq E_\delta$ that is a capacitated slugging plan with a benefit of at least R. We refer to this as the Delay Bounded and Capacitated Slugging Problem (DBCSP).*

Theorem 4 *The DBCSP is NP-Complete.*

Proof Obvious, since *DBCSP* is a generalization of *CSP*. □

Since the *DBCSP* is NP-Complete, we propose two greedy heuristics for the *DBCSP,* namely *Greedy-Benefit* and *Greedy-AVG-Benefit*. Both heuristics work in an iterative way. That is, each heuristic greedily chooses one driver trip T_d based on certain criteria. Intuitively, *Greedy-Benefit* chooses the driver trip that collects the maximum benefit of its incoming edges, and *Greedy-AVG-Benefit* chooses the driver trip that collects the maximum average benefit of its incoming edges.

To compute the maximum benefit and the maximum average benefit, we need to solve an instance of the 0/1 knapsack problem for each driver trip. Each driver trip T_d and all its passenger trips T_p where $(T_p,T_d)\in E_\delta$, form an instance of the 0/1 knapsack problem (see proof of Theorem 2). That is, trip T_d is considered the knapsack with a capacity equals to $T_d.c$ and each T_p is considered an item with a value equal to $B(T_p,T_d)$ and a size equal to $T_p.s$.

Since the 0/1 knapsack program is NP-complete, we employ an approximation algorithm called *Efficiency Greedy (EG)* approximation algorithm [11]. The *EG* algorithm outputs the larger between the following two numbers: (i) the total value when packing items into the knapsack in non-increasing order of their efficiencies (i.e. the ratio of value to size); (ii) the value of the single item which is most valuable among all items. It is known that the *EG* algorithm has a worst-case performance bound of 2 [11].

Intuitively, at each iteration the *Greedy-Benefit* heuristic applies the *EG* algorithm to each driver trip, and selects the driver trip with the maximum benefit computed by *EG*. This trip and its passengers are eliminated from the slugging graph, and then the next iteration is started. The *Greedy-AVG-Benefit* is identical, except that the driver trip selected is the one with the (maximum benefit / number of passenger trips selected by *EG*).

Algorithm 2: Heuristics Algorithms of *DBCSP*

Data: the delay-bounded slugging graph $G_\delta=(V,E_\delta)$ that is no-of-travelers-labeled, no-of-available-seats-labeled and benefit-labeled
Result: a ridesharing plan G_S

```
G_S ⟵ ∅
while E_δ ≠ ∅ do
    /* calculate driver trips                                          */
    driverTrips ⟵ ∅
    for edge (T_i, T_j) ∈ E_δ do
        driverTrips ⟵ driverTrips ∪ {T_j}
    /* select a driver trip                                            */
    for T ∈ driverTrips do
        Calculate B_appr(T) using the EG algorithm
    pick driver trip T_d with the maximum B_appr(T) /* GREEDY-BENEFIT
        heuristic                                                      */
    /* replace B_appr(T) with B_appr(T)/n when using the
        GREEDY-AVG-BENEFIT heuristic                                   */
    S_p ⟵ {T_p | T_p is a passenger trip such that (T_p, T_d) ∈ E_δ and selected
    by the EG algorithm }
    /* update ridesharing plan G_S                                     */
    for T_p ∈ V_δ do
        add (T_p, T_d) to G_S
    /* update the E_δ                                                  */
    S_chosen ⟵ {T_d} ∪ S_p
    for trip T ∈ S_chosen do
        remove T and all its incident edges from graph G_δ;
return G_S
```

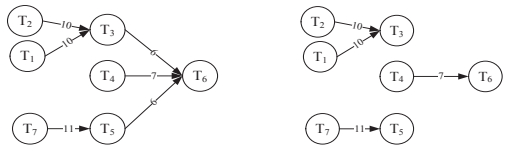
Fig. 4 Heuristics for the *DBCSP*

More precisely, denote by $B_{appr}(T_d)$ the result of the instance of the 0/1 knapsack program formed for trip T_d output by the *EG* algorithm. Denote by n the number of passenger trips that are selected for driver trip T_d by the *EG* algorithm. Then, in each iteration, the *Greedy-Benefit* and *Greedy-AVG-Benefit* heuristic select the driver trip with the maximum $B_{appr}(T_d)$ and $B_{appr}(T_d)/n$, respectively. Once a driver trip T_d is picked, the set of passenger trips that are merged into T_d are also determined by the *EG* algorithm. Next the delay-bounded slugging graph is updated by removing the nodes of the driver and its passengers, and the edges that touch upon them. This

update completes an iteration, and a new iteration then starts. The algorithm terminates when the slugging graph becomes empty.

Fig. 4 summarizes the algorithm for the proposed heuristics. Lines 6~7 calculates $B_{appr}(T_d)$ for every driver trip T_d in an iteration. Since there are $O(|V|)$ driver trips in an iteration, and the computation of $B_{appr}(T_d)$ for each T_d by the EG algorithm runs in $O(|V|log|V|)$, therefore, the selection of the driver trip in an iteration runs in $O(|V|^2 \log |V|)$. The updating process takes $O(|V|)$ for each selected trip and takes $O(|V|^2)$ in total, as there are at most $|V|$ trips selected in an iteration. Since there are $O(|V|)$ iterations, the time complexity of the greedy heuristics is $O(|V|^3 \log |V|)$.

To illustrate the *Greedy-Benefit* and *Greedy-AVG-Benefit* heuristics, please consider the delay-bounded slugging graph shown in Fig. 5 (a). The number on each edge represents its benefit. Assume that the capacity and the size of each node is 4 and 1, respectively. The optimal slugging plan for this simple example is shown in Fig. 5 (b), with a benefit of 38.



(a) a delay-bounded slugging graph (b) the corresponding optimal slugging plan

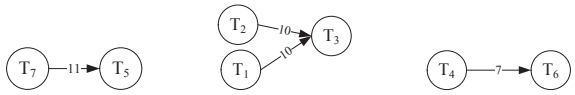
Fig. 5 An example of delay-bounded slugging graph

Now we generate a slugging plan using the greedy-based heuristics. Let us first consider the *Greedy-Benefit* heuristic. In the first iteration, T_3 is chosen as the driver trip because the maximum benefit that it can collect from its incoming edges is the largest among all driver trips. Then the graph is updated by deleting all edges associating with any of T_1, T_2, T_3. In the next iteration T_6 is chosen as the driver trip. The graph then updates again and becomes empty of edges. The edges selected in each step are shown in Fig. 6 and the benefit of the resulting slugging plan is 33.

(a) edges selected in 1st step (b) edges selected in 2nd step

Fig. 6 Running example of the Greedy-Benefit heuristic

In contrast, *Greedy-AVG-Benefit* chooses T_5 in the first iteration because T_5 has the largest average benefit of passenger trips. Then T_3 is selected in the second iteration and T_6 is selected in the third iteration. Fig. 7 shows edges selected in each iteration and the final slugging plan has a total benefit of 38.



(a) 1st step (b) 2nd step (c) 3rd step

Fig. 7 Running example of Greedy-AVG-Benefit heuristic

For the example shown in Fig. 5, *Greedy-AVG-Benefit* is coincidentally optimal. But the greedy heuristics cannot always guarantee the optimal solution. For example, neither heuristics is optimal for the example shown in Fig. 8, assuming that the size of each node is 1.

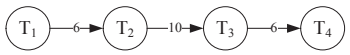


Fig. 8 An example for which heuristics are sub-optimal

3.6 Dynamic Slugging

The basic and constrained slugging problems that we have discussed are presented in a static context where all trips are known before the calculation of the slugging plan and the slugging plan is only calculated once. In this section, we discuss how to deal with the slugging problem in a dynamic context where the computation of a slugging plan is performed many times on the fly as the announcements of trips are continuously arriving.

Fig. 9 illustrates an instance of the dynamic slugging problem that involves five trips. The announcement of each trip is depicted by a circle and the start time of each trip is depicted by a diamond. On the one hand, it is necessary that there exists a temporal gap between the announcement and the start time for each trip; otherwise (i.e. if trips start at the same time when they are announced) there will be no room for ridesharing. On the other hand, such a temporal gap may be small (a few minutes) since these trips are dynamically generated. In the extreme case where these temporal gaps are huge (e.g. hours or even a day), the dynamic problem then degenerates to the static problem. Here we assume that the temporal gap between the announcement and trip start time is the same for all trips and denote this number by G. In other words, each trip is announced G time units before its start-time.

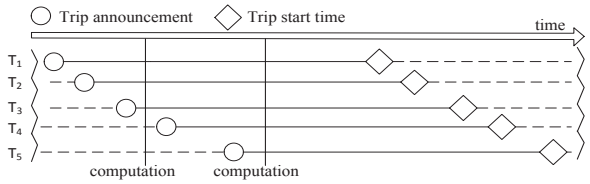


Fig. 9 The dynamic slugging problem

As in the static case, the objective of the dynamic problem is maximizing the total benefit. In the dynamic slugging problem, the slugging plan is computed and executed every f seconds as depicted by the vertical lines in Fig. 9, where f is referred to as the *decision interval*. Now we describe how the overall benefit of ridesharing is calculated for the dynamic problem. Once a trip is announced, it remains in the input set of the slugging plan computation until either of the following events happens: (i) the trip is included in the slugging plan as a result of a computation; (ii) the start time of the trip is reached (i.e. the trip starts without any ridesharing). The aggregate slugging plan (of all trips) is simply the union of all slugging plans calculated in each decision time point, and thus the overall benefit is computed based on the aggregate plan.

Table 2 An example of the dynamic slugging problem

	The Set of Trips As the Input	Calculated Slugging plan	Benefit
First computation	$\{T_1, T_2, T_3\}$	$\{(T_1, T_2)\}$	1
Second computation	$\{T_3, T_4, T_5\}$	$\{(T_3, T_5), (T_4, T_5)\}$	2
Aggregate	$\{T_1, T_2, T_3, T_4, T_5\}$	$\{(T_1, T_2), (T_3, T_5), (T_4, T_5)\}$	3

Table 2 gives a running example of how the benefit is computed in the dynamic context for the example given in Fig. 9. Suppose

that the first computation outputs a slugging plan $\{(T_1, T_2)\}$. Since T_3 is not included in the plan and it has not reach its start time, T_3 remains in the input set. Suppose that the second slugging plan computation yields $\{(T_3, T_5), (T_4, T_5)\}$. Therefore, the aggregate slugging plan is $\{(T_1, T_2), (T_3, T_5), (T_4, T_5)\}$ and the aggregate benefit is 3, assuming that the benefit of each merging is 1.

The value of f should be tuned carefully in order to maximize the benefit of ridesharing. We evaluate the optimal value of f experimentally and present the results in the next section.

4 EVALUATION

4.1 Setting

We conducted experiments using a taxi GPS trajectory data set [13]. The dataset contains real traces from more than five thousand taxis in Shanghai during a single day. These taxis have been equipped with GPS receivers (one for each). The GPS receivers periodically report their current states to a data center via GPRS links. Each record has a format *<TAXI_ID, TIMESTAMP, LONGITUDE, LATITUDE, OCCUPIED>*. Intuitively, each sequence of consecutive records where the *OCCUPIED* field constantly equals to 1 is an occupied trip. Fig. 10 shows a *TAXI_ID* and *TIMESTAMP* ordered snippet of a GPS trajectory data file and the blue rectangle represents an occupied trip.

```
105,2007-02-20 17:05:42,121.532800,31.231500,0
105,2007-02-20 17:06:02,121.532100,31.231100,1
105,2007-02-20 17:07:02,121.534100,31.225100,1
105,2007-02-20 17:11:06,121.537000,31.214300,1
105,2007-02-20 17:12:07,121.545800,31.212600,1
105,2007-02-20 17:14:09,121.551500,31.206100,1
105,2007-02-20 17:15:10,121.551500,31.206100,1
105,2007-02-20 17:16:11,121.551500,31.205800,1
105,2007-02-20 17:17:12,121.554800,31.198000,1
105,2007-02-20 17:17:45,121.556500,31.197800,0
```

Fig. 10 A snippet of taxi trajectory data that defines a trip

For our experiments, each such occupied trip defines a trip T as follows: the time stamp and the GPS point of the first record in the sequence defines the start time and origin of T, respectively; the time stamp and the GPS point of the last record in the sequence defines the end time and destination of T, respectively; the travel time of T then equals to the start time minus the end time; the travel distance is the road network distance between the origin and destination as obtained via the Google Map API. Out of 60 thousand occupied trips extracted from the data set, we selected 39 thousand trips which last over 5 minutes. This constituted our experimental pool of trips. The average travel time and travel distance of these trips is 12.3 minutes and 6.3 kilometers, respectively.

We evaluate the *DBCSP* in all experiments. The benefit of ridesharing is measured by the saving in vehicle travel distance. To compute the edge set of the slugging graph, we assume that all travelers walk at the same speed, denoted by W, and always walk along the shortest road path between two locations.

Table 3 Parameter setting in the experiments

Notation	Definition	Default Value
W	travelers' walking speed	5 km/h
δ	travel time delay threshold	20 minute
C	vehicle capacity	3
G	temporal gap between the announcement and trip start time	15 minute

In all the experiments, we assume that each trip is associated with only one traveler and she is willing to be either a passenger or driver in the slugging plan. For simplicity, we assume that all trips have the same travel time delay threshold, denoted by δ; and all cars have the same number of seats, denoted by C. Table 3 lists default values for the parameters used in our experiments.

4.2 Upper Bound on the DBCSP

Algorithm 3: Calculate an upper bound on the benefit of the ridesharing plan for the *DBCSP*

Data: the delay-bounded slugging graph $G_\delta = (V, E_\delta)$ that is no-of-travelers labeled, no-of-available-seats labeled and benefit-labeled
Result: an upper bound B_{upp} on the benefit of a ridesharing plan

```
1  passengerTrips ⟵ ∅
2  driverTrips ⟵ ∅
3  for edge (T_i, T_j) ∈ E_δ do
4      passengerTrips ⟵ passengerTrips ∪ {T_i}
5      driverTrips ⟵ driverTrips ∪ {T_j}
6  sum_1 ⟵ 0
7  sum_2 ⟵ 0
8  for trip T_i ∈ V do
9      if T_i ∈ passengerTrips then
           /* the upper bound if all passenger trips are merged */
10         sum_1 = sum_1 + B(T_i, T_d) , d is any number such that (T_i, T_d) ∈ E_δ
11     if T_i ∈ driverTrips then
           /* the upper bound if all driver trips collect maximum benefit */
12         L ⟵ top C − 1 passenger trips of the driver trip T_i
13         for trip T_p ∈ L do
14             sum_2 = sum_2 + B(T_p, T_i)
15 B_upp ⟵ min(sum_1, sum_2)
16 return B_upp
```

Fig. 11 An upper bound of the *DBCSP*

What is the maximum benefit that can be obtained by slugging? To answer this question, we obtain an upper bound on the benefit of a slugging plan for the *DBCSP* by relaxing either one of the two constraints imposed by the definition of slugging plan (see Def. 2). Relaxing Condition (ii) and the capacity constraint, we get an upper bound, denoted by B^1_{upp}, by merging each passenger trip T_i into some driver trip T_j regardless whether or not T_j has been merged into some other trip and regardless whether or not T_j has any available seat left. Relaxing Condition (i), we get another upper bound, denoted by B^2_{upp}, by making each driver trip T_d collect the maximum benefit regardless whether or not any its passenger trip T_p has been merged into any driver trip other than T_d. The smaller value of B^1_{upp} and B^2_{upp} is used as the final upper bound. Fig. 11 shows the algorithm that outputs this bound. It is easy to see that the time complexity of the algorithm is $O(|V|^2)$.

4.3 DBCSP With Varying Travel Delay

First we evaluate the proposed greedy-based heuristics by fixing the vehicle capacity and varying the travel time delay threshold. Experiments are performed for various thresholds of travel time delay $\delta \in [5, 20]$ minutes with an increment of 5 minutes.

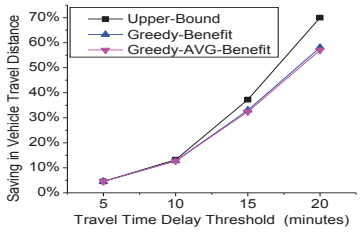
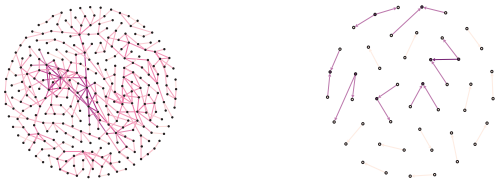


Fig. 12 *DBCSP* with varying delay thresholds

As Fig. 12 shows, when the travel time delay threshold is large, both the *Greedy-Benefit* and the *Greedy-AVG-Benefit* perform

consistently close to the upper bound. When $\delta = 20$, these heuristics have a 59% saving while the upper bound is 70%. Given the average distance of these trips is 6.3 kilometers and the size of our data set is 39 thousand, the 59% saving in vehicle travel distance is 144, 963 kilometers which means the reduction of over 4.5 thousand gallons of gasoline and 71 tons of carbon dioxide emission.

When the travel time delay threshold δ is small, there is no significant difference between the greedy-based heuristics and the upper bound. This is because, with a small δ, slugging opportunities are so rare that the slugging graph is extremely sparse. As a result, the graph admits very few possible slugging plans.



(a) δ= 20 minutes (b) δ = 5 minutes

Fig. 13 Visualization of a delay-bounded slugging graph

For example, Fig. 13 (a) and (b) visualize the delay-bounded slugging graph of a subset of trips when δ is 20 and 5 minutes, respectively. The darker the node's color is, the larger the node's in-degree (i.e. the number of incoming edges) is. When δ is 20 minutes, the graph is weakly connected. When δ is 5 minutes, most edges disappear and the graph is scattered into many disconnected components, each of which comprises of at most four nodes. In this case, it is clear that different algorithms will make little difference in the resulting slugging plan.

4.4 DBCSP with Varying Vehicle Capacity

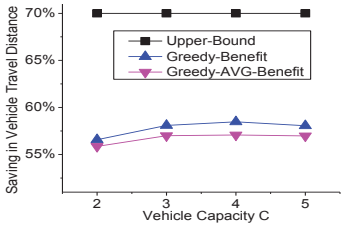


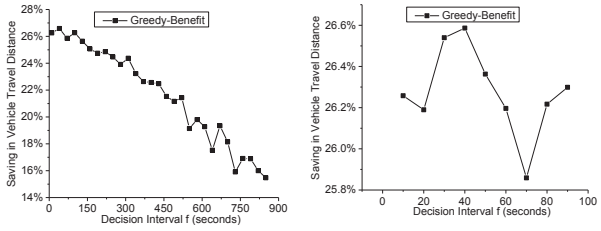
Fig. 14 *DBCSP* with varying vehicle capacities

In this experiment, the vehicle capacity varies, i.e. $C \in [2, 5]$ while the travel time delay threshold is fixed. Fig. 14 shows the performance of different heuristics. The result is consistent to that of Fig. 12, i.e., the greedy heuristics perform relatively close to the upper bound consistently. In addition, it is shown that the saving percentage saturates as the capacity increases, given the travel time delay is bounded. This is because the average number of incoming edges for each driver trip in the input graph is small, which can also be observed from Fig. 13. Even when $\delta = 20$ minutes, as shown in Fig. 13 (a), most driver trips have only one or two passenger trips that can be merged into them, and the average number of passenger trips for a driver trip is 1.38.

4.5 Dynamic DBCSP

In this experiment, we evaluate the *Greedy-Benefit* heuristic in a dynamic context. C is set to be 3 and δ is set to be 15 minutes. We set f to be smaller than G, otherwise many trips will start without encountering any computation of a slugging plan.

Since G is 15 minutes, we set the range of f to [10, 880] second with an increment of 30 seconds. Fig. 15 (a) shows a clear trend of decrease in benefit as the value of f increases. But the figure also clearly shows that the benefit fluctuates locally. To see the fluctuation more clearly, we further fine tune the value of f within a relative small range. Fig. 15 (b) shows the benefit fluctuating as decision interval f increase from 10 seconds to 100 seconds with an increment of 10 seconds. The saving rate reaches the maximum when f equals to 40 seconds.



(a) the decrease trending (b) fine tune of f

Fig. 15 Impact of the decision interval

Fig. 15 can be explained by two conflicting factors. On the one hand, as f increases, the computation of slugging plans becomes less frequent, so travelers of passenger trips cannot start walking until the time of the next computation. Therefore, the wasted time costs many ridesharing opportunities and thus decreases the benefit. On the other hand, as f increases, the input pool of trips for each computation of the slugging plan becomes larger and thus the benefit may increase. Fig. 15(a) suggests that the first factor wins the tug-of-war, so we have an overall decreasing trend with local fluctuations. It is also revealed from Fig. 15 that the saving rate at its peak (i.e. f=40 seconds) is about 26.6%. In contrast, the saving rate of the corresponding static problem with the same capacity and delay parameters (i.e. C=3 and $\delta = 15$ minutes) is 33%.

5 SUMMARY AND CONCLUSIONS

In this paper, we have analyzed slugging, an increasingly popular form of ridesharing, probably due to its simplicity. Specifically, we have formalized and studied the slugging problem. For the unconstrained slugging problem, we have proposed a quadratic algorithm to solve it optimally. We prove the NP-completeness of its constrained variants. For the constrained variant, we proposed heuristics and evaluated them on a data set consisting of tens of thousands of real trajectories of taxi cabs in Shanghai. The heuristics achieved near-optimal travel distance savings. The experimental results suggest that the saving in travel distance can reach as much as 59% whereas the optimal slugging plan achieves at most 70% savings. Given the size of our data set is 39K and the average distance of trips in the data set is 6.3 kilometers, the saving equals to 144,963 kilometers, which means the reduction of over 4.5 thousand gallons of gasoline and 71 tons of carbon dioxide emission. In addition, for the dynamic slugging problem, we evaluated the impact of the decision interval, i.e. how frequently to run the slugging algorithm, on the overall benefit.

In the future, we are going to refine this work towards a working system by considering and modeling other practical individual preferences such as riders' social preferences [12]. For the dynamic slugging problem, we are going to improve the plan calculation algorithm by considering a probabilistic model

which looks ahead, i.e. predicts the future announcement of trips. The objective function here optimized the overall benefit. It is possible to consider individual trips and minimize the walking distance and/or travel time delay of an average passenger trip. These extensions will be considered in future work.

6 REFERENCES

[1] San francisco parking. http://sfpark.org/.

[2] Slugging. http://en.wikipedia.org/wiki/slugging.

[3] Map of slugging sites in washington d.c. *slug-lines.com, Forel Publishing Company, LLC* (June 2010).

[4] AGATZ, N., E. A. S. M. W. X. Sustainable passenger transportation: Dynamic ride-sharing. Tech. rep., Erasmus Research Inst. of Management (ERIM), Erasmus Uni., Rotterdam, 2010.

[5] ATTANASIO, A., CORDEAU, J.-F., GHIANI, G., AND LAPORTE, G. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Comput. 30*, 3 (Mar. 2004), 377–387.

[6] BADGER, E. Slugging the people transit. *Miller-McCune* (2011).

[7] BALDACCI, R., MANIEZZO, V., AND MINGOZZI, A. An exact method for the car pooling problem based on lagrangean column generation. vol.52, INFORMS, pp. 422–439.

[8] CHAN, N.D., AND SHAHEEN, S.A. Ridesharing in north america: Past, present, and future. *Transport Reviews 32*, 1 (2012), 93–112.

[9] CORDEAU, J.-F., AND LAPORTE, G. The dial-a-ride problem: models and algorithms. *Annals of Operations Research 153* (2007), 29–46.

[10] GALIL, Z. Efficient algorithms for finding maximal matching in graphs. In *CAAP'83*, G.Ausiello and M.Protasi, Eds., vol.159 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1983, pp. 90–113.

[11] GAREY, M.R., AND JOHNSON, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman Co., New York, NY, USA, 1979.

[12] GIDÓFALVI, G. Instant Social Ride-Sharing. *ITS World, 8p, Transportation Society of America* (2008), 1–8.

[13] LIU, S., LIU, Y., NI, L. M., FAN, J., AND LI, M. Towards mobility-based clustering. In *Proc. of KDD'10*.

[14] MA, S., ZHENG, Y., AND WOLFSON, O. T-share: A large-scale dynamic ridesharing service. In *Proc. of ICDE'*13.

[15] MA, S., ZHENG, Y., AND WOLFSON, O. Real-time taxi-sharing with smart phones.

[16] P. D'OREY, R. FERNANDES, M. F. Empirical evaluation of a dynamic and distributed taxi-sharing system. In *IEEE Conf. on Intelligent Transportation Systems* (sept. 2010).

[17] TAO, C.-C. Dynamic taxi-sharing service using intelligent transportation system technologies. In *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007*.

[18] TSUBOUCHI, K., HIEKATA, K., AND YAMATO, H. Scheduling algorithm for on-demand bus system. In *Information Technology: New Generations, 2009. ITNG '09*.

Appendix

Problem: *Given a directed acyclic graph $G = (V, E)$ where V is a set of trips and E is the set of edges that is benefit-labeled, and a number $R \in \mathbb{R}^+$, find a subgraph $G_S = (V, E_S), E_S \subseteq E$ that is a slugging plan with the benefit at least R. We refer to this problem as the Generalized Slugging Problem (GSP).*

Theorem: *GSP is NP-complete.*

Proof: First, it is easy to see *GSP* is in NP. Now, we prove *GSP* is NP-hard by reducing the set cover problem to *GSP*.

The set cover problem is well-known NP-hard. It is defined as follows: given a set U of n elements, a family of subsets of U, $\{S_1, S_2, \dots, S_m\}$ and a integer k, the question is whether there exists a set of at most k of these subsets whose union equals to U. If the answer is yes, the problem has a set covering of size k.

Given an instance of set covering problem, i.e. a universe $U = \{1,2, \dots n\}$ and a family of m subsets $S_1, S_2, \dots, S_m$ of U, we can build an instance of *GSP* as follows. First we construct the graph $G = (V, E)$ as follows: We define $V = \{T_1, T_2, \dots, T_n, T_{S_1}, T_{S_2}, \dots, T_{S_m}, T_{sink}\}$ and construct E as follows. If $i \in S_j$, add an edge (T_i, T_{S_j}) to E for all $i = 1,2, \dots, n$ and $j = 1,2, \dots, m$, and add an edge $\left(T_{S_j}, T_{sink}\right)$ to E for all $j = 1,2, \dots m$. Note that E is indeed acyclic. We define benefit function B as a constant function $B\left(T_i, T_j\right) = 1$ for all $\left(T_i, T_j\right) \in E$. We also define benefit threshold $R = n + m - k$. Now we show that the set cover problem has a set covering of size k iff there is a subgraph of G that is a slugging plan and has a benefit as least of R.

First, assume that the set cover instance admits a set covering of size k, denoted by $\mathbb{C}$, we will now construct a subgraph of G, denoted by G_S, that is a slugging plan and has a benefit as least of R, i.e. $n + m - k$, as follows: start with an empty subgraph G_S; for each node $T_i, i \in U$, choose another node T_{S_j} such that $i \in S_j$ and $S_j \in \mathbb{C}$, then add edge (T_i, T_{S_j}) to the edge set of G_S. Since $|U| = n$, the benefit of RP increases by 1 for n times. For each set $S_j \notin \mathbb{C}$, add edge (T_{S_j}, T_{sink}) to the edge set of G_S, so $B(G_S)$ increases by 1 for at least $m - k$ times. G_S is a legitimate slugging plan since each $T_i, i \in U$ and each $T_{S_j}, S_j \notin \mathbb{C}$ is chosen to be passenger trips only while each $T_{S_j}, S_j \in \mathbb{C}$ and T_{sink} is chosen to be driver trips only, and no trip is merged into more than one other trips. The benefit of G_S is at least $n + m - k$.

Conversely, assume that there is a slugging plan $G_S = (V, E_S), E_S \subseteq E$ of benefit at least $n + m - k$, we now prove that there is a set covering of size k by proof of contradiction. Suppose there is no set covering of size k. Since for each node $T_i, i \in U$, it can contribute at most 1 to the benefit of G_S, and all n T_i's collectively contribute at most n to the benefit of G_S. Let us first assume that all T_i's are contributing. Denote by $\mathbb{C}$ the set of S_j's such that edge $(T_{S_j}, T_{sink}) \notin G_S$. Clearly $\mathbb{C}$ is a set cover. Since any cover size is larger than k thus $|\mathbb{C}| > k$, then there are less than $m - k$ nodes T_{S_j} are free to merged into node T_{sink}, i.e. contribute 1 to the benefit of G_S. Thus $B(G_S) < n + m - k$. Contradiction. Now assume that not all n T_i's contribute to G_S, say edge (T_i, T_{S_j}) is removed from G_S, the removal may or may not set T_{S_j} free, i.e. allow (T_{S_j}, T_{sink}) add to G_S. Even (T_{S_j}, T_{sink}) is added to G_S, since (T_{S_j}, T_{sink}) is previously removed from G_S, and thus the benefit of G_S will not increase. Contradiction remains. Therefore, there must be a set cover of size of k.□