Multimodal Transportation with Ridesharing of Personal Vehicles

Qian-Ping Gu
School of Computing Science, Simon Fraser University, Burnaby, Canada

JiaJian Liang
School of Computing Science, Simon Fraser University, Burnaby, Canada

Abstract

Many public transportation systems are unable to keep up with growing passenger demand as the population grows in urban areas. The slow or lack of improvement for public transportation pushes people to use private transportation modes, such as carpooling and ridesharing. However, the occupancy rate of personal vehicles has been dropping in many cities. In this paper, we describe a centralized transit system that integrates public transit and ridesharing, which matches drivers and transit riders such that the riders would result in shorter travel time using both transit and ridesharing. The optimization goal of the system is to assign as many riders to drivers as possible for ridesharing. We give an exact approach and approximation algorithms to achieve the optimization goal. As a case study, we conduct an extensive computational study to show the effectiveness of the transit system for different approximation algorithms, based on the real-world traffic data in Chicago City; the data sets include both public transit and ridesharing trip information. The experiment results show that our system is able to assign more than 60% of riders to drivers, leading to a substantial increase in occupancy rate of personal vehicles and reducing riders' travel time.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Applied computing → Transportation

Keywords and phrases Multimodal transportation, ridesharing, approximation algorithms, computational study

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2021.39

Related Version *Full Version*: `https://arxiv.org/abs/2106.00232` [13]

Funding This work was partially supported by Canada NSERC Discovery Grant 253500.

Acknowledgements The authors thank anonymous reviewers for their constructive comments.

1 Introduction

As the population grows in urban areas, commuting between and within large cities is time-consuming and resource-demanding. Due to growing passenger demand, the number of vehicles on the road for both public and private transportation has increased to handle the demand. Public transportation systems are unable to keep up with the demand in terms of service quality. This pushes people to use personal vehicles for work commute. In the United States, personal vehicles are the main transportation mode [6]. However, the occupancy rate of personal vehicles in the U.S. is 1.6 persons per vehicle in 2011 [12, 24] (and decreased to 1.5 persons per vehicle in 2017 [6]), which can be a major cause for congestion and pollution. This is the reason municipal governments encourage the use of public transit; the major drawback of public transit is the inconvenience of last mile and/or first mile transportation compared to personal vehicles [28]. With the increasing popularity in ridesharing/ridehailing service, there may be potential in integrating private and public transportation. From the research report of [9], it is recommended that public transit agencies should build on mobility innovations to allow public-private engagement in ridesharing because the use of shared modes increases the likelihood of using public transit. As pointed out by Ma et al. [20], some basic form of collaboration between MoD (mobility-on-demand) services and public transit already exists (for first and last mile transportation). There is an increasing interest for collaboration between private companies and public sector entities [22].

The spareness of transit networks usually is the main cause of the inconvenience in public transit. Such transit networks have infrequent transit schedule and can cause customers to have multiple transfers. In this paper, we investigate the potential effectiveness of integrating public transit with ridesharing to increase ridership in such sparse transit networks and reduce traffic congestion for work commute (not very short trips). For example, people who drive their vehicles to work can pick-up *riders*, who use public transit regularly, at designated locations and drop-off them at some transit stops, and then those riders can take public transit to their destinations. In this way, riders are presented with a cheaper alternative than ridesharing for the entire trip, and it is more convenient than using public transit only. The transit system also gets a higher ridership, which matches the recommendation of [9] for a more sustainable transportation system. Our research focuses on a centralized system that is capable of matching drivers and riders satisfying their trips' requirements while achieving some optimization goal; the requirements of a trip may include an origin and a destination, time constraints, capacity of a vehicle, and so on. When a rider is assigned a driver, we call this *ridesharing route*, and it is compared with the fastest *public transit route* for this rider which uses only public transit. If the ridesharing route is faster than the public transit route, the ridesharing route is provided to both the rider and driver. To increase the number of rider participants, our system-wide optimization goal is to maximize the number of riders, each of whom is assigned a ridesharing route. We call this the *maximization problem* (formal definition in Section 2).

In the literature, there are many papers about standalone ridesharing/carpooling, from theoretical to empirical studies (e.g., [1, 4, 14, 29]). For literature reviews on ridesharing, readers are referred to [2, 10, 21, 27]. On the other hand, there are only few papers study the integration of public transit with dynamic ridesharing. Aissat and Varone [3] and Huang et al. [17] proposed approaches which find a route with ridesharing that substitutes part of a public transit route for each rider in the first-come first-serve basis (system-wide optimization goal is not considered). Ma [19] and Stiglic et al. [26] proposed models to integrate ridesharing and public transit as graph matching problems to achieve system-wide optimization goals; their approaches are similar, except the work in [26] supports more rideshare match types. The graph matching problems in [19, 26] are formulated as integer linear program (ILP) and solved by standard branch and bound (CPLEX). The optimization goal in [19] is to minimize the cost related to waiting time and travel time, but ridesharing routes are not guarantee to be better than transit route. Although the optimization goal in [26] aligns with ours, there are some limitations in their approach; they limit at most two riders for each rideshare match, each rider must travel to the transit stop that is closest to the rider's destination, and more importantly, ridesharing routes assigned to riders can be longer than public transit routes.

In this paper, we use a similar model as in [19, 26]. We extend the work in [26] to eliminate the limitations described above and give approximation algorithms for the optimization problem to ensure solution quality. Our discrete algorithms allow to control the trade-off between quality and computational time. Our main contributions are summarized as follows:

1. We give an exact algorithm approach (an ILP formulation based on a hypergraph representation) for integrating public transit and ridesharing.
2. We prove our maximization problem is NP-hard and give a 2-approximation algorithm for the problem. We show that previous $O(k)$-approximation algorithms [5, 7] for the k-set packing problem are 2-approximation algorithms for our maximization problem. Our algorithm is more time and space efficient than previous algorithms.

3. As a case study, we conduct an extensive numerical study based on real-life data in Chicago City to evaluate the potential of having an integrated transit system and the effectiveness of different approximation algorithms.

The rest of the paper is organized as follows. In Section 2, we give the preliminaries of the paper, describe a centralized system that integrates public transit and ridesharing, and define the maximization problem. In Section 3, we describe our exact algorithm approach. We then propose approximation algorithms in Section 4. We discuss our numerical experiments and results in Section 5. Finally, Section 6 concludes the paper.

2 Problem definition and preliminaries

In the problem *multimodal transportation with ridesharing* (MTR), we have a centralized system, and for every fixed time interval, the system receives a set $\mathcal{A} = D \cup R$ of trips with $D \cap R = \emptyset$, where D is the set of driver trips and R is the set of rider trips. Each trip is expressed by an integer label i and consists of an individual, a vehicle (for driver trip) and some requirements. A connected public transit network with a fixed timetable T is given. We assume that for any source o and destination d in the public transit network, T gives the fastest travel time from o to d. A *ridesharing route* π_i for a rider $i \in R$ is a travel plan using a combination of public transportation and ridesharing to reach i's destination satisfying i's requirements, whereas a *public transit route* $\hat{\pi}_i$ for a rider i is a travel plan using only public transportation. The multimodal transportation with ridesharing problem asks to provide at least one feasible route (π_i or $\hat{\pi}_i$) for every rider $i \in R$. We denote an instance of multimodal transportation with ridesharing problem by $(N, \mathcal{A}, T)$, where N is an edge-weighted directed graph (network) for both private and public transportation. We call a public transit station or stop just *station*. The terms rider and passenger are used interchangeably (although passenger emphasizes a rider has been provided with a ridesharing route).

The requirements of each trip i in $\mathcal{A}$ are specified by i's parameters submitted by the individual. The parameters of a trip i contain an origin location o_i, a destination location d_i, an earliest departure time α_i, a latest arrival time β_i and a maximum trip time γ_i. A driver trip i also contains a capacity n_i of the vehicle, a limit δ_i on the number of stops a driver wants to make to pick-up/drop-off passengers, and an optional path to reach its destination. The maximum trip time γ_i of a driver i includes a travel time from o_i to d_i and a detour time limit i can spend for offering ridesharing service. A rider trip i also contains an acceptance rate θ_i for a ridesharing route π_i, that is, π_i is given to rider i if $t(\pi_i) \leq \theta_i \cdot t(\hat{\pi}_i)$ for every public transit route $\hat{\pi}_i$ and $0 < \theta_i \leq 1$, where $t(\cdot)$ is the travel time. Such a route π_i is called an *acceptable ridesharing route* (acceptable route for brevity). For example, suppose the best public transit route $\hat{\pi}_i$ takes 100 minutes for i and $\theta_i = 0.9$. An acceptable route π_i implies that $t(\pi_i) \leq \theta_i \cdot t(\hat{\pi}_i) = 90$ minutes. We consider two match types for practical reasons.

- **Type 1 (rideshare-transit)**: a driver may make multiple stops to pick-up different passengers, but makes only one stop to drop-off all passengers. In this case, the *pick-up locations* are the passengers' origin locations, and the *drop-off location* is a public station.
- **Type 2 (transit-rideshare)**: a driver makes only one stop to pick-up passengers and may make multiple stops to drop-off all passengers. In this case, the *pick-up location* is a public station and the *drop-off locations* are the passengers' destination locations.

Riders and drivers specify one of the match types to participate in; they are allowed to choose both in hope to increase the chance being selected, but the system will assign them only one of the match types such that the optimization goal of the MTR problem is achieved, which is to assign acceptable routes to as many riders as possible. Formally, the **maximization problem** we consider is to maximize the number of passengers, each of whom is assigned an acceptable route π_i for every $i \in R$.

For a driver i and a set $J \subseteq R$ of riders, $\sigma(i) = \{i\} \cup J$ is called a *feasible match* if the routes for all trips of $\sigma(i)$ satisfy the requirements (constraints) specified by the parameters of the trips collectively as listed below (a summary of notation and constraints can be found in [13], Section 3.2):

1. *Ridesharing route constraint*: for $J = \{j_1, \dots, j_k\}$, there is a path $(o_i, o_{j_1}, \dots, o_{j_k}, s, d_i)$ in N, where s is the drop-off location for Type 1 match; or there is a path $(o_i, s, d_{j_1}, ..., d_{j_k}, d_i)$ in N, where s is the pick-up location for Type 2 match.
2. *Capacity constraint*: $1 \leq |J| \leq n_i$.
3. *Acceptable constraint*: each passenger $j \in J$ is given an acceptable route π_j offered by i.
4. *Travel time constraint*: each trip $j \in \sigma(i)$ departs from o_j no earlier than α_j, arrives at d_j no later than β_j, and the total travel duration of j is at most γ_j.
5. *Stop constraint*: the number of unique locations visited by driver i to pick-up (for Type 1) or drop-off (for Type 2) all passengers of $\sigma(i)$ is at most δ_i.

Two feasible matches $\sigma(i), \sigma(i')$ are *disjoint* if $\sigma(i) \cap \sigma(i') = \emptyset$. Then, the maximization problem considered is to find a set of pairwise disjoint feasible matches such that the number of passengers included in the feasible matches is maximized.

Intuitively, a rideshare-transit (Type 1) feasible match $\sigma(i)$ is that all passengers in $\sigma(i)$ are picked-up at their origins and dropped-off at a station, and then i drives to destination d_i while each passenger j of $\sigma(i)$ takes transit to destination d_j. A transit-rideshare (Type 2) feasible match $\sigma(i)$ is that all passengers in $\sigma(i)$ are picked-up at a station and dropped-off at their destinations, and then i drives to destination d_i after dropping the last passenger. We give algorithms to find pairwise disjoint feasible matches to maximize the number of passengers included in the matches. We describe our algorithms for Type 1 only. Algorithms for Type 2 can be described with the constraints on the drop-off location and pick-up location of a driver exchanged, and we omit the description. Further, it is not difficult to extend to other match types, such as rideshare only and park-and-ride, as described in [26].

3 Exact algorithm

An exact algorithm for the maximization problem is presented in this section, which is similar to the matching approach described in [4, 23] for ridesharing and in [19, 26] for MTR. The exact algorithm is summarized as follows. First, we compute all feasible matches for each driver i. Then, we create a bipartite (hyper)graph $H(D, R, E)$, where $D(H)$ is the set of drivers, and $R(H)$ is the set of riders. There is a hyperedge $e = (i, J)$ in $E(H)$ between $i \in D(H)$ and a non-empty subset $J \subseteq R(H)$ if $\{i\} \cup J$ is a feasible match, denoted by $\sigma_J(i)$, for driver i. An example is given in Figure 1. Any driver i and rider j with no feasible match is removed from $D(H)$ and $R(H)$ respectively, namely, no isolated vertex (such riders must use public transit routes). For an edge $e = (i, J)$, let $A(e) = \{i\} \cup J$ and $p(e) = |J|$ be the

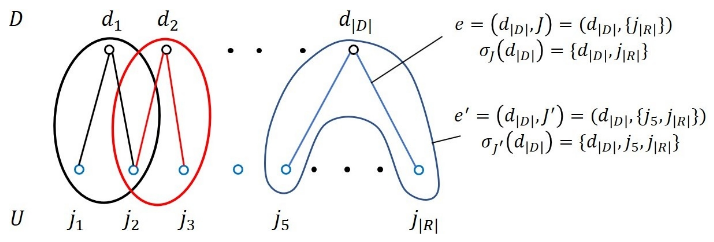

D
d_1
d_2
$d_{|D|}$
$e = (d_{|D|}, J) = (d_{|D|}, \{j_{|R|}\})$
$\sigma_J(d_{|D|}) = \{d_{|D|}, j_{|R|}\}$
$e' = (d_{|D|}, J') = (d_{|D|}, \{j_5, j_{|R|}\})$
$\sigma_{J'}(d_{|D|}) = \{d_{|D|}, j_5, j_{|R|}\}$
U
j_1
j_2
j_3
j_5
$j_{|R|}$

Figure 1 A bipartite hypergraph for all possible matches of an instance $(N, \mathcal{A}, T)$.

number of riders represented by e. For a trip $j \in \mathcal{A}$, define $E_j = \{e \in E \mid j \in A(e)\}$ to be the set of edges in E associated with j. To solve the maximization problem, we give an integer program (ILP) formulation:

$$\text{maximize} \quad \sum_{e \in E(H)} p(e) \cdot x_e \tag{1}$$

$$\text{subject to} \quad \sum_{e \in E_j} x_e \leq 1, \qquad \forall\, j \in \mathcal{A} \tag{2}$$

$$x_e \in \{0,1\}, \qquad \forall\, e \in E(H) \tag{3}$$

The binary variable x_e indicates whether the edge $e = (i, J)$ is in the solution ($x_e = 1$) or not ($x_e = 0$). If $x_e = 1$, it means that all passengers in J are served by i. Inequality (2) in the ILP formulation guarantees that each driver serves at most one feasible set of passengers and each passenger is served by one driver. Note that the ILP (1)-(3) is similar to a set packing formulation. An advantage of this ILP formulation is that the number of constraints is substantially decreased, compared to traditional ridesharing formulation. From Observation 1 in [25], it is not difficult to see that the following result holds (a proof of Theorem 1 is given in [13], Sections 3.1).

▶ **Theorem 1.** *Given a bipartite graph $H(D, R, E)$ representing an instance of the multimodal transportation with ridesharing maximization problem, an optimal solution to the ILP (1)-(3) is an optimal solution to the maximization problem and vice versa.*

Computing feasible matches. Let i be a driver in D and n_i be the capacity of i (maximum number of riders i can serve). The maximum number of feasible matches for i is $\sum_{p=1}^{n_i} \binom{|R|}{p}$. Assuming the capacity n_i is a small constant (which is reasonable in practice), the above summation is polynomial in R, that is, $O((|R|+1)^{n_i})$. Let $K = \max_{i \in D} n_i$ be the maximum capacity among all vehicles (driver trips). Then, in the worst case, $|E(H)| = O(|D| \cdot (|R|+1)^K)$. We compute all feasible matches for each trip in two phases. In phase one, for each driver i, we find all feasible matches $\sigma(i) = \{i, j\}$ with one rider j. In phase two, for each driver i, we compute all feasible matches $\sigma(i) = \{i, j_1, .., j_p\}$ with p riders, based on the feasible matches $\sigma(i)$ with $p - 1$ riders computed previously, for $p = 2$ and upto the number of passengers i can serve. Complete description and algorithms (Algorithm 1 for phase one and Algorithm 2 for phase two) for computing the feasible matches can be found in Sections 3.2.1 and 3.2.2 of [13]. We make two simplifications in our algorithms:

- Given a source station s_o and a destination d_i of trip i with departure time t at s_o, we use a simplified transit system in our experiments to calculate the fastest public transit route from s_o to d_i.
- We use a simplified model for the transit waiting time and ridesharing service time (time it takes to pick-up and drop-off riders, walking time between locations and stations).

As shown in [13](Section 3.2.2), we compute a feasible path with minimum travel time for driver i to pick-up p passengers in each feasible match $\sigma(i)$.

4 NP-hardness and approximation algorithms

We show that the maximization problem is NP-hard and give approximation algorithms for the problem. When every edge in $H(D, R, E)$ consists of only two vertices (one driver and one passenger), the maximization problem is equivalent to the maximum matching, which can be solved in polynomial time (e.g., [16]). However, if the edges consist of more than two vertices, they become hyperedges. In this case, the ILP (1)–(3) becomes a formulation of the maximum weighted set packing problem (MWSP), which is NP-hard [11, 18]. In fact, ILP (1)–(3) formulation gives a special case of MWSP (due to the structure of $H(D, R, E)$). We prove that this special case is also NP-hard, and by Theorem 1, the maximization problem is NP-hard (a proof of Theorem 2 is in [13], Section 4.1).

▶ **Theorem 2.** *The maximization problem is NP-hard.*

Next, we describe approximation algorithms for the maximization problem. For consistency, we follow the convention in [5, 7] that a ρ-approximation algorithm for a maximization problem is defined as $\rho \cdot w(\mathcal{C}) \geq OPT$ for $\rho > 1$, where $w(\mathcal{C})$ and OPT are the values of approximation and optimal solutions respectively.

4.1 2-Approximation algorithm

We first give a simple 2-approximation algorithm for our maximization problem. For a maximization problem instance $H(D, R, E)$, we use Γ to denote a current partial solution, which consists of a set of matches represented by the hyperedges in $E(H)$. Let $P(\Gamma) = \bigcup_{e \in \Gamma} J_e$ (called *covered passengers*). Initially, $\Gamma = \emptyset$. In each iteration, we add a match with the most number of uncovered passengers to Γ, that is, select an edge $e = (i, J_e)$ such that $|J_e \setminus P(\Gamma)|$ is maximum, and then add e to Γ. Remove $E_e = \cup_{j \in A(e)} E_j$ from $E(H)$ (E_j is defined in Section 3). Repeat until $P(\Gamma) = R$ or $|\Gamma| = |D|$. The pseudo code of ImpGreedy is shown in Algorithm 3. In the ImpGreedy algorithm, when an edge e is added to Γ, E_e is removed from $E(H)$, so Property 3 holds for Γ.

▶ **Property 3.** *For every $i \in D$, at most one edge e from E_i can be selected in any solution.*

Algorithm 3 ImpGreedy Algorithm.

```
input  : The hypergraph $H(D, R, E)$ for problem instance $(N, \mathcal{A}, T)$
output : A solution $\Gamma$ to $(N, \mathcal{A}, T)$ with 2-approximation ratio
1 $\Gamma = \emptyset$; $P(\Gamma) = \emptyset$;
2 while ($P(\Gamma) \neq R$ and $|V(\Gamma)| < |D|$) do
3     compute $e = \text{argmax}_{e \in E(H)} |J_e \setminus P(\Gamma)|$, $\Gamma = \Gamma \cup \{e\}$, and remove $E_e$ from $E(H)$;
4     update $P(\Gamma)$;
5 end
```

4.1.1 Analysis of ImpGreedy Algorithm

Let $\Gamma = \{x_1, x_2, \ldots, x_a\}$ be a solution found by Algorithm 3, where x_i is the i^{th} edge added to Γ. Throughout the analysis, we use OPT to denote an optimal solution, that is, $P(OPT) \geq P(\Gamma)$. Further, $\Gamma_i = \bigcup_{1 \leq b \leq i} x_b$ for $1 \leq i \leq a$, $\Gamma_0 = \emptyset$ and $\Gamma_a = \Gamma$. The driver of match x_i is denoted by $d(x_i)$. The main idea of our analysis is to add up the maximum difference between the number of covered passengers by selecting x_i in Γ and not selecting x_i in OPT. For each $x_i \in \Gamma$, by Property 3, there is at most one $y \in OPT$ with $d(y) = d(x_i)$. We order OPT and introduce dummy edges to OPT such that $d(y_i) = d(x_i)$ for $1 \leq i \leq a$. Formally, for $1 \leq i \leq a$, define

$$OPT(i) = \{y_1, \ldots, y_i \mid 1 \leq b \leq i, d(y_b) = d(x_b) \text{ if } y_b \in OPT, \text{ otherwise } y_b \text{ a dummy edge}\}.$$

A dummy edge $y_b \in OPT(i)$ is defined as $d(y_b) = d(x_b)$ with $J_{y_b} = \emptyset$. The gap of an edge $x_i \in \Gamma$ is defined as

$$\text{gap}(x_i) = |J_{y_i}| - |J_{x_i} \setminus P(\Gamma_{i-1})| + |J'_{x_i}|,$$

where $J'_{x_i} = (J_{x_i} \setminus P(\Gamma_{i-1})) \cap P(OPT \setminus \Gamma)$ is the maximum subset of passengers in $J_{x_i} \setminus P(\Gamma_{i-1})$ that are also covered in $OPT \setminus \Gamma$. The intuition is that the sum of gap(x_i) for all $x_i \in \Gamma$ states the maximum possible number of passengers may not be covered by Γ. Let $P(OPT(i)) = \bigcup_{1 \le b \le i} J_{y_b}$ and $P(OPT'(i)) = \bigcup_{1 \le b \le i} J'_{x_b}$ for any $i \in [1, \dots, a]$. Then the maximum gap between Γ and OPT can be calculated as $\sum_{x \in \Gamma_a} \text{gap}(x) = |P(OPT(a))| + |P(OPT'(a))| - |P(\Gamma_a)|$. First, we show that $P(OPT) = P(OPT(a)) \cup P(OPT'(a))$.

▶ **Proposition 4.** *Let $\Gamma = \{x_1, \dots, x_a\}$, $P(OPT(a)) = \bigcup_{1 \le i \le a} J_{y_i}$ and $P(OPT'(a)) = \bigcup_{1 \le i \le a} J'_{x_i}$. Then, $P(OPT) = P(OPT(a)) \cup P(OPT'(a))$.*

Proof. By definition, $P(OPT) = P(OPT(a)) \cup P(OPT \setminus OPT(a))$. For any z in $OPT \setminus OPT(a)$, $d(z) \neq d(x)$ for every $x \in \Gamma$. If $J_z \setminus P(\Gamma) \neq \emptyset$, then z would have been found and added to Γ by Algorithm 3. Hence, $J_z \setminus P(\Gamma) = \emptyset$, implying $J_z \subseteq P(OPT'(a))$ and $P(OPT \setminus OPT(a)) \subseteq P(OPT'(a))$. ◀

▶ **Lemma 5.** *Let OPT be an optimal solution and $\Gamma = \{x_1, x_2, \dots, x_a\}$ be a solution found by the algorithm. For any $1 \le i \le a$, $\sum_{x \in \Gamma_i} \text{gap}(x) = |P(OPT(i))| - |P(\Gamma_i)| + |P(OPT'(i))| \le |P(\Gamma_i)|$.*

Proof. Recall that $OPT(i) = \{y_1, \dots, y_i\}$ as defined above. For $y_b \in OPT(i), 1 \le b \le i, d(y_b) = d(x_b)$. We prove the lemma by induction on i. Base case $i = 1$: $|P(OPT(1))| - |P(\Gamma_1)| + |P(OPT'(1))| \le |P(\Gamma_1)|$. By definition, gap$(x_1) = |J_{y_1}| - |J_{x_1} \setminus \Gamma_0| + |J'_{x_1}|$. Since x_1 is selected by the algorithm, it must be that $|J_{x_1}| \ge |J_u|$ for all $u \in V(G')$, so $|J_{y_1}| \le |J_{x_1}|$. Thus,

$$\begin{aligned}\text{gap}(x_1) &= |J_{y_1}| - |J_{x_1} \setminus \Gamma_0| + |J'_{x_1}| \\ &\le |J'_{x_1}| \le |J_{x_1}|.\end{aligned}$$

Assume the statement is true for $i-1 \ge 1$, that is, $\sum_{x \in \Gamma_{i-1}} \text{gap}(x) \le |P(\Gamma_{i-1})|$, and we prove for $i \le a$. By the induction hypothesis, both $P(OPT(i-1))$ and $P(OPT'(i-1))$ are included in the calculation of $\sum_{x \in \Gamma_{i-1}} \text{gap}(x)$. More precisely, $\sum_{x \in \Gamma_{i-1}} \text{gap}(x) = |P(OPT(i-1))| - |P(\Gamma_{i-1})| + |P(OPT'(i-1))| \le |P(\Gamma_{i-1})|$. If $|J_{y_i}| \le |J_{x_i} \setminus P(\Gamma_{i-1})|$, the lemma is true since we can assume $|J'_{x_i}| \le |J_{x_i}|$. Suppose $|J_{y_i}| > |J_{x_i} \setminus P(\Gamma_{i-1})|$. Before x_i is selected, the algorithm must have considered y_i and found that $|J_{x_i} \setminus P(\Gamma_{i-1})| \ge |J_{y_i} \setminus P(\Gamma_{i-1})|$. Then, $|J_{y_i}| > |J_{x_i} \setminus P(\Gamma_{i-1})| \ge |J_{y_i} \setminus P(\Gamma_{i-1})|$, implying $J_{y_i} \cap P(\Gamma_{i-1}) \neq \emptyset$. We have

$$|J_{x_i} \setminus P(\Gamma_{i-1})| + |J_{y_i} \cap P(\Gamma_{i-1})| \ge |J_{y_i} \setminus P(\Gamma_{i-1})| + |J_{y_i} \cap P(\Gamma_{i-1})| = |J_{y_i}|. \tag{4}$$

Let $J''_{y_i} \subseteq (J_{y_i} \cap P(\Gamma_{i-1}))$ be the set of passengers covered by $P(OPT(i-1)) \cup P(OPT'(i-1))$, namely $J''_{y_i} \subseteq (P(OPT(i-1)) \cup P(OPT'(i-1)))$. Then by the induction hypothesis,

$$\sum_{x \in \Gamma_{i-1}} \text{gap}(x) \le P(\Gamma_{i-1}) - |J_{y_i} \cap P(\Gamma_{i-1})| + |J''_{y_i}|. \tag{5}$$

Adding $\sum_{x \in \Gamma_{i-1}} \text{gap}(x)$ and gap(x_i) together:

$$\begin{aligned}
&(\sum_{x\in\Gamma_{i-1}} \text{gap}(x)) + (\text{gap}(x_i))\\
&= |P(OPT(i-1))| - |P(\Gamma_{i-1})| + |P(OPT'(i-1))| + |J_{y_i} \setminus J''_{y_i}| - |J_{x_i} \setminus P(\Gamma_{i-1})| + |J'_{x_i}|\\
&\le (|P(\Gamma_{i-1})| - |J_{y_i} \cap P(\Gamma_{i-1})| + |J''_{y_i}|) + |J_{y_i} \setminus J''_{y_i}| - |J_{x_i} \setminus P(\Gamma_{i-1})| + |J'_{x_i}| && \text{by (5)}\\
&= |P(\Gamma_{i-1})| - |J_{y_i} \cap P(\Gamma_{i-1})| + |J_{y_i}| - |J_{x_i} \setminus P(\Gamma_{i-1})| + |J'_{x_i}|\\
&\le |P(\Gamma_{i-1})| - |J_{y_i} \cap P(\Gamma_{i-1})| + |J_{y_i} \cap P(\Gamma_{i-1})| + |J'_{x_i}| && \text{by (4)}\\
&= |P(\Gamma_{i-1})| + |J'_{x_i}| \le |P(\Gamma_{i-1})| + |J_{x_i} \setminus P(\Gamma_{i-1})| && \text{by defintion of } J'_{x_i}\\
&= P(\Gamma_i)
\end{aligned}$$

Therefore, by the property of induction, the lemma holds. ◀

▶ **Theorem 6.** *Given the hypergraph instance $H(D,R,E)$. Algorithm 3 computes a solution Γ to H such that $2|P(\Gamma)| \ge |P(OPT)|$, where OPT is an optimal solution, with running time $O(|D|\cdot|E|)$ and $|E| \le |D|\cdot(|R|+1)^K$.*

Proof. Let $\Gamma = \{x_1,\dots,x_a\}$, $P(OPT(a)) = \bigcup_{1\le i\le a} J_{y_i}$ and $P(OPT'(a)) = \bigcup_{1\le i\le a} J'_{x_i}$. By Proposition 4, $P(OPT) = P(OPT(a)) \cup P(OPT'(a))$, and by Lemma 5, $|P(OPT(a))| + |P(OPT'(a))| - |P(\Gamma_a)| \le |P(\Gamma_a)|$. We have

$$|P(OPT)| \le |P(OPT(a))| + |P(OPT'(a))| \le 2|P(\Gamma)|.$$

In each iteration of the while-loop, it takes $O(E)$ to find an edge x with maximum $|J_x \setminus P(\Gamma)|$, and there are at most $|D|$ iterations. Hence, Algorithm 3 runs in $O(|D|\cdot|E|)$ time. ◀

4.2 Approximation algorithms for maximum weighted set packing

We briefly explain the algorithms for the maximum weighted set packing problem, which solve our maximization problem. Given a universe $\mathcal{U}$ and a family $\mathcal{S}$ of subsets of $\mathcal{U}$, a *packing* is a subfamily $\mathcal{C} \subseteq \mathcal{S}$ of sets such that all sets in $\mathcal{C}$ are pairwise disjoint. Every subset $S \in \mathcal{S}$ has at most k elements and is given a real weight. The maximum weighted k-set packing problem (MWSP) asks to find a packing $\mathcal{C}$ with the largest total weight. We can see that the maximization problem on $H(D,R,E)$ is a special case of the maximum weighted k-set packing problem, where the trips of $D \cup R$ is the universe $\mathcal{U}$ and $E(H)$ is the family $\mathcal{S}$ of subsets, and every $e \in E(H)$ represents at most $k = K+1$ trips (K is the maximum capacity of all vehicles). Hence, solving MWSP also solves our maximization problem. Hazan et al. [15] showed that the k-set packing problem cannot be approximated to within $O(\frac{k}{\ln k})$ in general unless P = NP. Chandra and Halldórsson [7] presented a $\frac{2(k+1)}{3}$-approximation and a $\frac{2(2k+1)}{5}$-approximation algorithms (refer to as *BestImp* and *AnyImp* respectively), and Berman [5] presented a $(\frac{k+1}{2}+\epsilon)$-approximation algorithm (refer to as *SquareImp*) for the weighted k-set packing problem (here, $k = K+1$), where the latter still has the best approximation ratio. These three algorithms in [5, 7] (AnyImp, BestImp and SquareImp) solve the weighted k-set packing problem by first transferring it into a weighted independent set problem, which consists of a vertex weighted graph $G(V,E)$ and asks to find a maximum weighted independent set in $G(V,E)$. These algorithms use a greedy algorithm (refer to as Greedy) to find an initial independent set solution I and then use local searches to improve the weight of the solution. Algorithm Greedy can be summarized as follows: Select a vertex $u \in V(G)$ with largest weight and add u to I. Eliminate u and all u's neighbors from being selected. Repeatedly select the largest weight vertex until all vertices are eliminated from G.

To apply these algorithms to our maximization problem, we need to convert the bipartite hypergraph $H(D,R,E)$ to a weighted independent set instance $G(V,E)$. The details of the conversion can be found in [13], Section 4.3. Notice that Algorithm 3 is a simplified version of algorithm Greedy, and Greedy is used to get an initial solution in algorithms AnyImp, BestImp and SquareImp. From Theorem 6, we have Corollary 7.

▶ **Corollary 7.** *Greedy, AnyImp, BestImp and SquareImp algorithms compute a solution to $H(D,R,E)$ with 2-approximation ratio.*

Since Algorithm 3 finds a solution directly on $H(D,R,E)$ without converting it to $G(V,E)$ and solving the independent set problem of $G(V,E)$, it is more time and space efficient than the algorithms for MWSP. In the rest of this paper, Algorithm 3 is referred to as ImpGreedy.

5 Numerical experiments

We create a simulation environment consists of a centralized system that integrates public transit and ridesharing. We implement our proposed approximation algorithm (ImpGreedy) and Greedy, AnyImp and BestImp algorithms for the k-set packing problem to evaluate the benefits of having an integrated transportation system supporting public transit and ridesharing. The exact algorithm, ILP (1)-(3), is not evaluated because it takes too long to complete for the instances in our study. The results of SquareImp are not discussed because its performance is same as AnyImp; this is due to the implementation of the search/enumeration order of the vertices and edges in the independent set instance $G(V,E)$ being fixed, and each vertex in $V(G)$ has integer weight. We use a simplified transit network of Chicago to simulate the public transit and ridesharing.

5.1 Description and characteristics of datasets

We built a simplified transit network of Chicago to simulate practical scenarios of public transit and ridesharing. The roadmap data of Chicago is retrieved from OpenStreetMap[1]. We used the GraphHopper[2] library to construct the logical graph data structure of the roadmap. The Chicago city is divided into 77 officially community areas, each of which is assigned an area code. We examined two different dataset in Chicago to reveal some basic traffic pattern (the datasets are provided by the Chicago Data Portal (CDP) and Chicago Transit Authority (CTA)[3], maintained by the City of Chicago). The first dataset is bus and rail ridership, which shows the monthly averages and monthly totals for all CTA bus routes and train station entries. We denote this dataset as *PTR, public transit ridership*. The PTR dataset contains data for the month June, 2019. The second dataset is rideshare trips reported by Transportation Network Providers (sometimes called rideshare companies) to the City of Chicago. We denote this dataset as *TNP*. The TNP dataset range is chosen from June 3rd, 2019 to June 30th, 2019, total of 4 weeks of data. Table 1 and Table 2 show some basic stats of both datasets.

We examined the 12 busiest bus routes based on the total ridership and selected 7 out of the 12 routes as listed in Table 1 to build the transit network. We also selected all major trains/metro lines within Chicago. Each record in the TNP dataset describes a passenger trip

[1] Planet OSM. `https://planet.osm.org`
[2] GraphHopper 1.0. `https://www.graphhopper.com`
[3] CDP. `https://data.cityofchicago.org`. CTA. `https://www.transitchicago.com`

Table 1 Basic stats of the PTR dataset.

Total Bus Ridership	20,300,416
Total Rail Ridership	19,282,992
12 busiest bus routes	3, 4, 8, 9, 22, 49, 53, 66, 77, 79, 82, 151
The busiest bus routes selected	4, 9, 49, 53, 77, 79, 82

Table 2 Basic stats of the TNP dataset.

# of original records	8,820,037
# of records considered	7,427,716
# of shared trips	1,015,329
# of non-shared trips	6,412,387
The most visited community areas selected	1, 4, 5, 7, 22, 23, 25, 32, 41, 64, 76

served by a driver who provides the rideshare service; a trip record consists of the pick-up and drop-off time and the pick-up and drop-off community area of the trip, and exact locations are provided sometimes. We selected 11 of the 20 most visited areas as listed in Table 2 (area 32 is Chicago downtown, areas 64 and 76 are airports) to build the transit network for our simulation. From the selected bus routes, trains and community areas, we create a simplified public transit network connecting the selected areas, depicted in Figure 2. More details on selecting areas and the public transit network are included in [13] (Section 5.1).

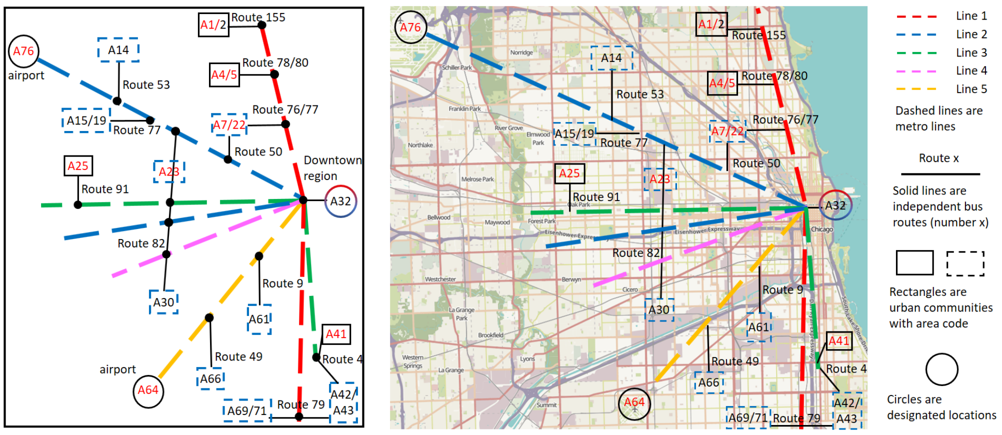
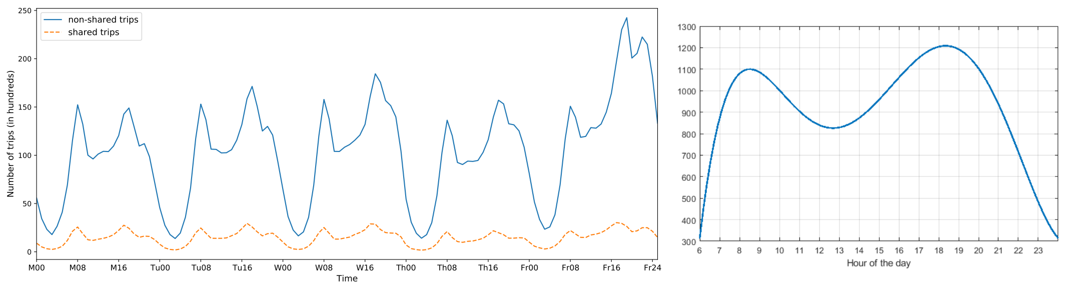


Figure 2 Simplified public transit network of Chicago with 13 urban communities and 3 designated locations. Figure on the right has the Chicago city map overlay for scale.

The travel time between two locations uses the fastest/shortest route computed by GraphHopper, based on personal cars. The shortest paths are **computed in real-time**, unlike many previous simulations where the shortest paths are precomputed and stored. As stated in Section 3, waiting time and service time are considered in a simplified model; we multiply a small constant $\epsilon > 1$ to the fastest route to mimic waiting time and service time.

5.2 Generating instances

In our simulation, we partition each day from 6:00 to 23:59 into 72 time intervals (each has 15 minutes), and we only focus on weekdays. To see ridesharing traffic pattern, we calculated the average number of served trips per hour for each day of the week using the TNP dataset. The dashed (orange) line and solid (blue) line of the plot in Figure (3a) represent shared trips and non-shared trips respectively. A set of trips are called *shared trips* if this set of trips are matched for the same vehicle consecutively such that their trips may potentially overlap (one or more passengers are in the same vehicle). For all other trips, we call them *non-shared trips*. The number of trips generated for each interval is plotted in Figure (3b), which is a



(a) Average numbers of shared and non-shared trips in TNP dataset.

(b) Total number of driver and rider trips generated for each time interval.

Figure 3 Plots for the number of trips for every hour from data and generated.

scaled down and smoothed version of the TNP dataset for weekdays. The ratio between the number of drivers and riders generated is roughly 1:3 (1 driver and 3 riders) for each interval. Such a ratio is chosen because it should reflect the system's potential as capacity of 3 is common for most vehicles. For each time interval, we first generate a set R of riders and then a set D of drivers. We do not generate a trip where its origin and destination are close.

The main idea of generating rider trips is described as follows. Each day is divided into 6 different consecutive time periods (each consists of multiple time intervals): morning rush, morning normal, noon, afternoon normal, afternoon rush, and evening time periods. Each time period determines the probability and distribution of origins and destinations. Based on the PTR dataset and Rail Capacity Study [8], many riders are going into downtown in the morning and leaving downtown in the afternoon. To generate a rider trip j during a time period, we first select a *pick-up area* and a *drop-off area* randomly following the probability distribution for the time period (e.g., downtown is selected with higher probability as drop-off area for the morning rush period). The origin o_j and destination d_j are random points within the pick-up and drop-off areas respectively. The above is repeated until a_t riders are generated, where $a_t + a_t/3$ (riders + drivers) is the total number of trips for time interval t shown in Figure (3b). The probability distribution for each time period and detailed description of generating rider trips can be found in Section 5.2 of [13].

The main idea of generating driver trips is described as follows. We examined the TNP dataset to create a grid heatmap (Figure 8 in [13]) for traffic for each hour. Each cell (c, r) in the heatmap represents the the average number of trips per hour originated from area c to destination area r in the transit network (Figure 2). Let $d(c, r, h)$ be the value at the cell (c, r) for origin c, destination r and hour h in the heatmap. Let $P(c, h) = \sum_r d(c, r, h)$ be the sum of the values of the whole column c for hour h. Given a time interval t in hour h, let c_t be the number of generated riders with origin in area c; and for each area c, we generate $c_t/3$ drivers such that each driver i has origin $o_i = c$ and destination $d_i = r$ with probability $d(c, r, h)/P(c, h)$. The probability of selecting an airport as destination is fixed at 5%.

After the origin and destination of a rider/driver trip have been determined, we decide other parameters of the trip (e.g., the vehicle capacity of a driver i is at most 6). Details of these parameters are specified in Section 5.2 of [13].

When the number of trips increases, the running time for Algorithm 2 and the time needed to construct the k-set packing instance also increase. In a practical setup, we may restrict the number of feasible matches a driver can have. Each match produced by Algorithm 1 is called a *base match*. To make the simulation feasible, we heuristically limit the numbers of base matches for each driver and each rider and the number of total feasible matches for each driver. We use $(x\%, y, z)$, called *reduction configuration* (*Config* for short), to denote that for each driver i, the number of base matches of i is reduced to x percentage and at most y total feasible matches are computed for i; and for each rider j, at most z base matches containing j are used. Details of the reduction procedure can be found in the end of Section 5.2 of [13].

5.3 Computational results

We use the same transit network and same set of generated trip data for all algorithms. All experiments were implemented in Java and conducted on Intel Core i7-2600 processor with 1333 MHz of 8 GB RAM available to JVM. Since the optimization goal is to assign feasible acceptable routes to as many riders as possible, the performance measure is focused on the number of riders served by ridesharing routes, followed by the total time saved for the riders as a whole. The base case instance uses the parameter setting described in Section 5.2 and Config (30%, 600, 20). The experiment results are shown in Table 3. The results of

Table 3 Base case solution comparison between the approximation algorithms.

	ImpGreedy	Greedy	AnyImp	BestImp
Total number of riders served	27413	27413	28248	28258
Avg number of riders served per interval	380.736	380.736	392.333	392.472
Total time saved of all riders (minute)	354568.2	354568.2	365860.6	365945.8
Avg time saved of riders per interval (minute)	4924.56	4924.56	5,081.40	5082.58
Total number of riders and public transit duration		45314 and 1383743.97 minutes		

ImpGreedy and Greedy are aligned since they are essentially the same algorithm – 60.5% of total passengers are assigned ridesharing routes and 25.6% of total time are saved. The results of AnyImp and BestImp are similar because of the density of the graph $G(V, E)$. For AnyImp and BestImp, roughly 62.4% of total passengers are assigned ridesharing routes and 26.4% of total time are saved. On average, passengers are able to reduce their travel duration from 30.5 minutes to 22.5 minutes by using public transit plus ridesharing. The results of these four algorithms are not too far apart. However, it takes too long for AnyImp and BestImp to run to completion. A 10-second limit is set for both algorithms in each iteration for finding an independent set improvement. With this time limit, AnyImp and BestImp run to completion within 15 minutes for almost all intervals. We also recorded the mean occupancy rate of drivers. The mean occupancy rate is calculated as, in each interval, the number of passengers served divided by the number of drivers who serve them. The results are depicted in Figure 9 in [13], which show that mean occupancy rate of a personal vehicle is 2.9–3 (including the driver) on average. Further discussions can be found in [13] (Section 5.3).

Another major component of the experiment is to measure the computational time of the algorithms, which is highly affected by the base match reduction configurations. By reducing more matches, we are able to improve the running time of AnyImp and BestImp significantly, but sacrifice performance slightly. We tested 12 different Configs:

- *Small1* (20%,300,10), *Small2* (20%,600,10), *Small3* (20%,300,20), *Small4-10* (20%,600,20), *Medium1* (30%,300,10), *Medium2* (30%,600,10), *Medium3* (30%,300,20), *Medium4-10* (30%,600,20), *Large1* (40%,300,10), *Large2* (40%,600,10), *Large3-10* (40%,300,20), and *Large4-10* (40%,600,20).

Configs with label "-10" have a 10-second limit to find an independent set improvement, and all other Configs have 20-second limit. Notice that all 12 Configs have the same sets of driver/rider trips and base match sets but generate different feasible match sets. The

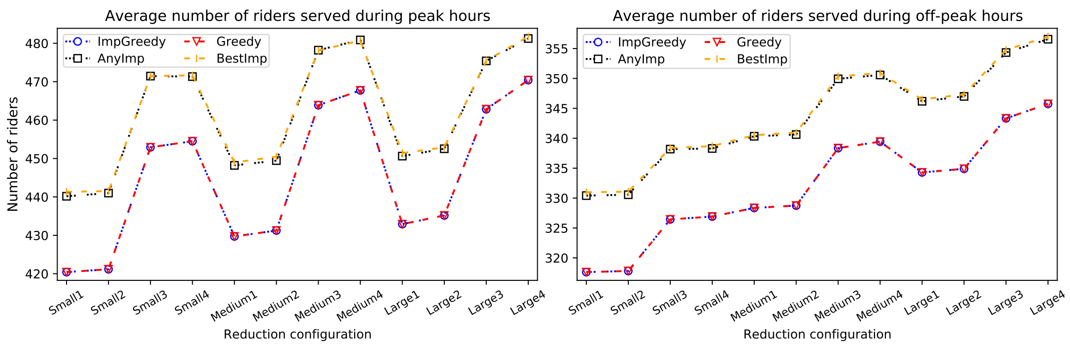


Figure 4 Average performance of peak and off-peak hours for different configurations.

performance and running time results of all 12 Configs are depicted in Figures 4 and 5 respectively. The results are divided into peak and off-peak hours for each Config (averaging all intervals of peak hours and off-peak hours). The running time of ImpGreedy and Greedy are within seconds for all Configs for each interval. On the other hand, it may not be practical to use AnyImp and BestImp for peak hours since they require around 15 minutes for most Configs. Since AnyImp and BestImp provide better performance than ImpGreedy/Greedy when each Config is compared side-by-side, one can use ImpGreedy/Greedy for peak hours and AnyImp/BestImp for off-peak hours so that it becomes practical. The increase in

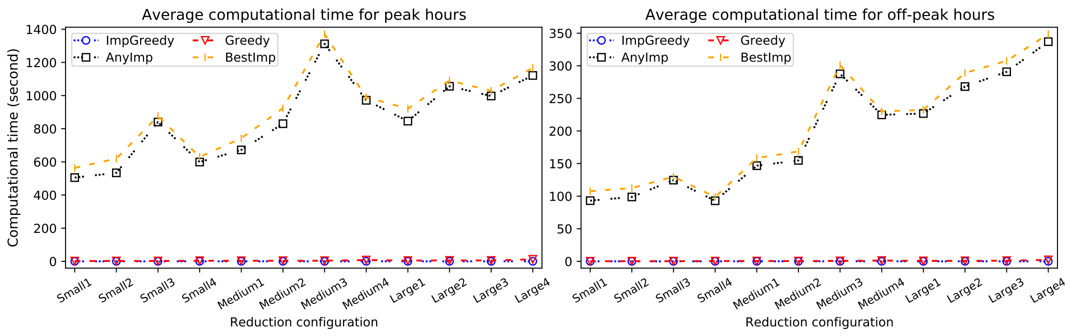


Figure 5 Average running time of peak and off-peak hours for different configurations.

performance from Small1 to Small3 is much larger than that from Small1 to Small2 (same for Medium and Large), implying any parameter in a Config should not be too small. The increase in performance from Large1 to Large4 is higher than that from Medium1 to Medium4 (similarly for Small). Therefore, a balanced configuration is more important than a configuration emphasizes only one or two parameters.

Because ImpGreedy does not create the independent set instance, it runs quicker than Greedy. More importantly, ImpGreedy uses less memory space than Greedy does. We tested ImpGreedy and Greedy with the following Configs: *Huge1* (100%,600,10), *Huge2* (100%,2500,20) and *Huge3* (100%,10000,30) (these Configs have the same sets of driver/rider trips and base match sets as those in the previous 12 Configs). The focus of these Configs is to see if Greedy can handle large number of feasible matches. The results are shown in Table 4. Greedy cannot run to completion for all Configs because in many intervals, the whole graph $G(V, E)$ of the independent set instance is too large to hold in memory. The average number of edges for afternoon peak hours is 0.02, 0.38 and 5.47 billion for Huge1, Huge2 and Huge3 respectively. Further, the time it takes to create $G(V, E)$ excess practicality. Hence, using Greedy (AnyImp/BestImp) for large instances may not be practical. In addition, the performance of ImpGreedy with Huge3 is better than that of AnyImp/BestImp with Large4.

Table 4 The results of ImpGreedy and Greedy using Unlimited reduction configurations.

ImpGreedy	Huge1	Huge2	Huge3
Avg running time for peak/off-peak hours (sec)	0.08 / 0.03	0.43 / 0.12	1.2 / 0.29
Avg number of riders served for peak/off-peak hours	406.9 / 339.0	458.8 / 355.4	484.1 / 361.9
Avg time saved of riders per interval (sec)	284891.8	302774.1	310636.9
Greedy	Huge1	Huge2	Huge3
Avg running time	N/A	N/A	N/A
Avg instance size $G(V, E)$ of afternoon peak ($\lvert E(G)\rvert$)	0.02 billion	0.38 billion	5.47 billion
Avg time creating $G(V, E)$ of afternoon peak (sec)	14.6	320.9	3726.79

We also looked at the total running times of the approximation algorithms including the time for computing feasible matches (Algorithms 1 and 2). The running time of Algorithm 1 solely depends on computing the shortest paths between the trips and stations. Table 5 shows that Algorithm 1 runs to completion within 500 seconds for each interval on average for peak hours. As for Algorithm 2, when many trips' origins/destinations are concentrated in one area, the running time increases significantly, especially for drivers with high capacity. Combining the results of this and previous (Table 4) experiments, ImpGreedy is capable of handling large instances while providing quality solution compared to other approximation algorithms.

Table 5 Average computational time (in seconds) of peak hours for all algorithms.

	Alg1	Alg2	ImpGreedy	Greedy	AnyImp	BestImp	Total computational time			
							ImpGreedy	Greedy	AnyImp	BestImp
Small3	485.2	26.8	0.021	2.0	840.5	876.4	512.1	514.1	1352.5	1388.5
Small4	485.2	28.2	0.029	3.6	599.1	629.9	513.4	517.0	1112.5	1143.3
Medium3	485.2	43.6	0.031	3.7	1312.1	1371.0	532.5	543.0	1840.9	1899.9
Medium4	485.2	50.1	0.048	7.7	971.5	990.0	535.3	543.0	1506.8	1525.3
Large4	485.2	72.0	0.076	12.2	1121.3	1167.2	557.3	569.5	1678.6	1724.4
Huge3	485.2	339.4	1.2	N/A	N/A	N/A	825.8	N/A	N/A	N/A

From the experiment results in Figures 4 and 5, it is beneficial to dynamically select different algorithms and reduction configurations for each interval depending on the number of trips. With large problem instances, previous approximation algorithms are not efficient (time and memory consuming), so they require aggressive reduction to reduce the instance size. On the other hand, ImpGreedy is much faster and capable of handling large instances. The running time of ImpGreedy can also be an advantage to improve the quality of solutions. For example, as shown in Figures 4 and 5, for the same set of drivers and riders, ImpGreedy assigns more riders when taking Meduim/Medium4 as inputs than AnyImp/BestImp on Small1/Small2, and uses less time than AnyImp/BestImp. When the size of an instance is not small and a solution must be computed within some time-limit, ImpGreedy has a distinct advantage over the previous approximation algorithms.

6 Conclusion

Based on real-world transit datasets in Chicago, our study has shown that integrating public and private transportation can benefit the transit system as a whole, more than 60% of the passenger are assigned ridesharing routes and able to save 25% of travel time. Majority of the drivers are matched with at least one passenger, and vehicle occupancy rate has improved close to 3 (including the driver) on average. These results suggest that ridesharing can be a complement to public transit. Our experiments show that the whole system is capable of handling more than 1100 trip requests in real-time using ordinary computer hardware. The performance results of ImpGreedy may be further improved by extending it with the local search strategy. Perhaps the biggest challenge for scalability comes from computing the base matches (Algorithm 1) since it has to compute many shortest paths; it may be worth to apply heuristics to speed-up Algorithm 1. To better understand practicality, a more sophisticated simulation incorporating traffic and transit schedule and demand may be needed.

References

1 Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Dynamic ride-sharing: A simulation study in metro atlanta. *Transportation Research Part B: Methodological*, 45(9):1450–1464, 2011. doi:10.1016/j.trb.2011.05.017.

2 Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295–303, 2012. doi:10.1016/j.ejor.2012.05.028.

3 Kamel Aissat and Sacha Varone. Carpooling as complement to multi-modal transportation. In *Enterprise Information Systems*, pages 236–255, January 2015. doi:10.1007/978-3-319-29133-8_12.

4 Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017. doi:10.1073/pnas.1611675114.

5 Piotr Berman. A d/2 approximation for maximum weight independent set in d-claw free graphs. In *Algorithm Theory - SWAT 2000*, pages 214–219. Springer Berlin Heidelberg, 2000. doi:10.1007/3-540-44985-X_19.

6 Center for Sustainable Systems, University of Michigan. Personal transportation factsheet, 2020. URL: http://css.umich.edu/factsheets/personal-transportation-factsheet.

7 Barun Chandra and Magnús M Halldórsson. Greedy local improvement and weighted set packing approximation. *Journal of Algorithms*, 39(2):223–240, 2001. doi:10.1006/jagm.2000.1155.

8 Chicago Transit Authority. System-wide rail capacity study. Published June 2017; Revised February 2019. URL: https://www.transitchicago.com/assets/1/6/RP_CDMSMITH_RCM_Task2AExecutiveSummary_20170628_FINAL.pdf.

9 Sharon Feigon and Colin Murphy. Shared mobility and the transformation of public transit. TCRP Research Report, Transportation Research Board, 2016. doi:10.17226/23578.

10 Masabumi Furuhata, Maged Dessouky, Fernando Ordóñez, Marc-Etienne Brunet, Xiaoqing Wang, and Sven Koenig. Ridesharing: The state-of-the-art and future directions. *Transportation Research Part B: Methodological*, 57:28–46, 2013. doi:10.1016/j.trb.2013.08.012.

11 Michael R. Garey and David S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* W.H. Freeman and Company, 1979.

12 Keivan Ghoseiri, Ali Haghani, and Masoud Hamedi. Real-time rideshare matching problem. Final Report of UMD-2009-05, U.S. Department of Transportation, 2011.

13 Qian-Ping Gu and Jiajian Leo Liang. Multimodal transportation with ridesharing of personal vehicles, 2021. arXiv:2106.00232.

14 Qian-Ping Gu, Jiajian Leo Liang, and Guochuan Zhang. Approximate ridesharing of personal vehicles problem. *Theoretical Computer Science*, 871:30–50, 2021. doi:10.1016/j.tcs.2021.04.009.

15 Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-set packing. *Computational Complexity*, 15(1):20–39, 2006. doi:10.1007/s00037-006-0205-6.

16 John Hopcroft and Richard Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.

17 Haosheng Huang, Dominik Bucher, Julian Kissling, Robert Weibel, and Martin Raubal. Multimodal route planning with public transport and carpooling. *IEEE Transactions on Intelligent Transportation Systems*, 20(9):3513–3525, 2019. doi:10.1109/TITS.2018.2876570.

18 Richard M. Karp. *Reducibility among combinatorial problems*, pages 85–103. Springer US, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.

19 Tai-Yu Ma. On-demand dynamic bi-/multi-modal ride-sharing using optimal passenger-vehicle assignments. In *2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I CPS Europe)*, pages 1–5, 2017. doi:10.1109/EEEIC.2017.7977646.

20 Tai-Yu Ma, Saeid Rasulkhani, Joseph Y.J. Chow, and Sylvain Klein. A dynamic ridesharing dispatch and idle vehicle repositioning strategy with integrated transit transfers. *Transportation Research Part E: Logistics and Transportation Review*, 128:417–442, 2019. doi:10.1016/j.tre.2019.07.002.

21 Abood Mourad, Jakob Puchinger, and Chengbin Chu. A survey of models and algorithms for optimizing shared mobility. *Transportation Research Part B: Methodological*, 123:323–346, 2019. doi:10.1016/j.trb.2019.02.003.

22 Arvind U Raghunathan, David Bergman, John Hooker, Thiago Serra, and Shingo Kobori. Seamless multimodal transportation scheduling, 2018. arXiv:1807.09676.

23 Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven H. Strogatz, and Carlo Ratti. Quantifying the benefits of vehicle pooling with shareability networks. *Proceedings of the National Academy of Sciences*, 111(37):13290–13294, 2014. doi:10.1073/pnas.1403657111.

24 A. Santos, N. McGuckin, H.Y. Nakamoto, D. Gray, and S. Liss. Summary of travel trends: 2009 national household travel survey. Technical report, US Department of Transportation Federal Highway Administration, 2011.

25 Mitja Stiglic, Niels Agatz, Martin Savelsbergh, and Mirko Gradisar. The benefits of meeting points in ride-sharing systems. *Transportation Research Part B: Methodological*, 82:36–53, 2015. doi:10.1016/j.trb.2015.07.025.

26 Mitja Stiglic, Niels Agatz, Martin Savelsbergh, and Mirko Gradisar. Enhancing urban mobility: Integrating ride-sharing and public transit. *Computers & Operations Research*, 90:12–21, 2018. doi:10.1016/j.cor.2017.08.016.

27 Amirmahdi Tafreshian, Neda Masoud, and Yafeng Yin. Frontiers in service science: Ride matching for reer-to-reer ride sharing: A review and future directions. *Service Science*, 12(2-3):41–60, 2020. doi:10.1287/serv.2020.0258.

28 Hai Wang and Amedeo Odoni. Approximating the performance of a "last mile" transportation system. *Transportation Science*, 50(2):659–675, 2014. doi:10.1287/trsc.2014.0553.

29 Zhengtian Xu, Yafeng Yin, and Jieping Ye. On the supply curve of ride-hailing systems. *Transportation Research Part B: Methodological*, 132:29–43, 2020. 23rd International Symposium on Transportation and Traffic Theory (ISTTT 23). doi:10.1016/j.trb.2019.02.011.