Bus frequency optimization: When waiting time matters in user satisfaction

Songsong MO

Zhifeng BAO

Baihua ZHENG
Singapore Management University, bhzheng@smu.edu.sg

Zhiyong PENG

Bus Frequency Optimization: When Waiting Time Matters in User Satisfaction

Songsong Mo[1], Zhifeng Bao[2], Baihua Zheng[3], and Zhiyong Peng[1]⋆

[1] School of Computer Science,Wuhan University, Hubei, China
{songsong945, peng}@whu.edu.cn
[2] RMIT University, Melbourne, Australia
zhifeng.bao@rmit.edu.au
[3] Singapore Management University, Singapore, Singapore
bhzheng@smu.edu.sg

Abstract. Reorganizing bus frequency to cater for the actual travel demand can save the cost of the public transport system significantly. Many, if not all, existing studies formulate this as a bus frequency optimization problem which tries to minimize passengers' average waiting time. However, many investigations have confirmed that the user satisfaction drops faster as the waiting time increases. Consequently, this paper studies the bus frequency optimization problem considering the user satisfaction. Specifically, for the first time to our best knowledge, we study how to schedule the buses such that the total number of passengers who could receive their bus services within the waiting time threshold is maximized. We prove that this problem is NP-hard, and present an index-based algorithm with $(1-1/e)$ approximation ratio. By exploiting the locality property of routes in a bus network, we propose a partition-based greedy method which achieves a $(1-\rho)(1-1/e)$ approximation ratio. Then we propose a progressive partition-based greedy method to further improve the efficiency while achieving a $(1-\rho)(1-1/e-\varepsilon)$ approximation ratio. Experiments on a real city-wide bus dataset in Singapore verify the efficiency, effectiveness, and scalability of our methods.

Keywords: bus frequency scheduling optimization · user waiting time minimization · approximate algorithm.

1 Introduction

Public transport and the services delivered by buses are essential to our daily life. Bus services provide us with the capability to move around, which shapes where we can work and live, where we shop and how we spend our leisure time. In this paper, we focus on bus frequency design which plays a very important role in urban public transport systems, as reorganizing bus frequencies to meet the actual travel demands is expected to achieve significant savings in cost. Taking New York City as an example, the cost of each bus is around \$550,000 and the operating cost of transit agencies reaches \$215 per hour[4]. If we re-organize the

⋆ Zhiyong Peng is the corresponding author.
[4] https://www.liveabout.com/bus-cost-to-purchase-and-operate-2798845

bus frequencies based on real travel demands and save 10% bus departures, we can save \$20 operating costs per hour and \$55,000 per vehicle.

In the literature, there are many studies focusing on the problem of bus frequency optimization. Most of them share a common objective, which is to minimize the *average* travel cost (in terms of waiting time) of passengers [13,3,5,10,9]. Moreover, their solutions are usually heuristic rather than approximate (with theoretical guarantees). However, most, if not all, existing works ignore an important aspect, the *user satisfaction*. Many studies have confirmed that the user satisfaction drops faster as the waiting time increases [1,8]. Motivated by this finding, we aim to schedule the buses in a way to serve more passengers within a given waiting time threshold θ but not to minimize the average waiting time. In addition, our algorithms are adaptive to cater for different settings of θ.

We call this novel problem as Satis<u>FA</u>ction-Boo<u>ST</u> Bus Scheduling (FAST). Given a bus database $\mathcal{B}$, a bus route database $\mathcal{R}$, a passenger database $\mathcal{P}$, and a vector $\mathcal{N}$ $\langle n_1, n_2, \cdots, n_i, \cdots, n_{|\mathcal{R}|}\rangle$ that specifies the expected number of bus departures for each bus route, it chooses n_i buses for each route $r_i \in \mathcal{R}$ such that the whole bus system is able to satisfy the most passengers. The analysis shows that the objective function of FAST is submodular and FAST is NP-hard.

To resolve the FAST problem, we develop a range of approximate algorithms with non-trivial theoretical guarantees. First, we propose an index-based greedy method (Greedy), which can provide $(1 - 1/e)$ approximation factor for FAST as the baseline, and two enhanced versions, namely PartGreedy and ProPartGreedy. PartGreedy is inspired from [18] and by the fact that a bus network is designed to cover different parts of the city and it tries to avoid unnecessary overlapping among routes [4,16]. It adopts a partitioning algorithm to divide the bus network into several disjoint partitions. Accordingly, it invokes local greedy search within each partition, which effectively reduces the computation cost of the original greedy algorithm. On the other hand, ProPartGreedy adopts a different strategy to address the efficiency issue. Instead of finding one bus that contributes the most to the objective function in each iteration of the local greedy search, it fetches multiple buses in each iteration of the local greedy search to cut down the total number of iterations required. Meanwhile, ProPartGreedy has a tunable parameter that could determine roughly how many buses could be fetched in each iteration and hence provide a trade-off between efficiency and effectiveness.

In summary, we make the following contributions.

- We propose and study the FAST problem. To the best of our knowledge, this is the first study on bus frequency optimization that considers user satisfaction. We prove that the objective function of FAST is monotone and submodular, and FAST is NP-hard.
- We propose an index-based greedy method (Greedy), a partition-based greedy method (PartGreedy) and a progressive partition-based greedy method (ProPartGreedy) to solve the FAST problem efficiently. They can achieve an approximation ratio of $(1 - \frac{1}{e})$, $(1 - \rho)(1 - \frac{1}{e})$, and $(1 - \rho)(1 - \frac{1}{e} - \varepsilon)$ respectively, where ρ and ε are the user-defined parameters.

– We conduct extensive experiments on real-world bus route and bus touch-on/touch-off records in Singapore (396 routes, 28 million trip records of one week) to demonstrate the effectiveness, efficiency and scalability of our methods.

2 Related Work

In this section, we will review existing related work and report the difference between this work and existing ones.

We divide the literature into two categories based on the overall optimization objective. One is called the travel time driven bus frequency optimization problem (Travel-BFO), which aims to minimize the average/total travel time of passengers for either one bus route or a bus route network, based on passenger demands. It treats each ride as a new trip. Another is called the transfer time driven bus frequency optimization problem (Transfer-BFO), which aims to minimize the total transfer time of the transfer passengers.

Travel-BFO. Here, the passenger demands are usually abstracted as origin-destination (OD) pairs. The model proposed in [13] treats the travel time of passengers as an aggregation of the walking time, the waiting time, and the on-board travel time. The problem is usually formulated as a nonconvex objective function with linear or convex constraints. In [3], it is modeled as a nonlinear bilevel problem: the upper level represents the planner who wants to ensure minimal total travel time under fleet size constraints; the lower level represents the users who act by minimizing the travel time. In [5], a multi-objective model is proposed, seeking to minimize the overall travel time of the users and the operational cost of the operators (assumed to be linearly proportional to the frequencies). Martínez et al. [10] study the transit frequency optimization problem to determine the time interval between subsequent buses for a set of bus lines. They propose a mixed-integer linear programming (MILP) formulation for an existing bilevel model [3], and present a metaheuristic method. A new model considering user behavior is proposed in [9]. It assigns a user's trip to three stages (pre-trip, on-board and end-trip) and aims to minimize users total travel costs of the objective bus line.

Differences. Although different bus frequency optimization models have been proposed, they share a very similar optimization objective, i.e., minimizing the average/total travel cost of passengers. Different from the above literature, we aim to improve the *overall passenger satisfaction* by scheduling the buses such that they can serve more passengers within the given waiting time threshold. Our work is mainly motivated by the following two findings. *First*, waiting time has a direct impact on the user satisfaction, as evident by many studies [8,1]. *Second*, the waiting time threshold is tunable, hence the bus company can adjust thresholds to cater to various concerns on budget, government needs, passengers' tolerance of waiting, etc.

Transfer-BFO. Transfer time driven bus frequency optimization problem is an extension of single bus route timetabling. It determines the departure time of each trip of all lines in the bus network with the consideration of passenger transfer activities at transfer stations [6].

This problem is modeled by mixed integer programming models to maximize the number of synchronized bus arrivals at transfer nodes [2]. Ibarra-Rojas et al. [7] extend the work of Ceder et al. [2] to address a flexible Transfer-BFO problem with almost evenly spaced departures and preventing bus bunching. The model proposed in [14] tries to minimize the total transfer time experienced by passengers. Parbo et al. [12] studied a bi-level bus timetabling problem to minimize the weighted transfer waiting time of passengers, and a Tabu Search algorithm was applied to solve the bilevel model. Recently a non-linear mixed integer-programming model is proposed to maximize the number of total transferring passengers with small excess transfer time [17].
Differences. The above studies on the Transfer-BFO problem mainly focus on minimizing the total transfer cost for passengers on transfer, which can only improve the satisfaction of the transfer passengers. In contrast, our problem aims to improve overall passenger satisfaction by serving them within a given time threshold.

For all the above work in both categories, despite the difference, all existing approaches only propose heuristic methods without theoretical guarantees, while we propose algorithms with non-trivial theoretical guarantees.

3 Problem Formulation

In a bus route database $\mathcal{R}$, a route r is a sequence of bus stations (s_1, s_2, $\cdots$, s_i, $\cdots$, s_m), where s_i is a bus station represented by (latitude, longitude). In a passenger database $\mathcal{P}$, a passenger $p \in \mathcal{P}$ is in form of a tuple $\{s_b, s_e, t\}$, where s_b denotes the boarding station, s_e denotes the alighting station, and t denotes the time when p reaches s_b. A bus b_{ij} is in form of a tuple $\{r_i, dt_j\}$, where r_i and dt_j denote the bus service route and the departure time from $r_i.s_1$ respectively.

Definition 1. *We define that a bus b_{ij} can serve a passenger p, if r_i contains $p.s_b$ and $p.s_e$ in order, and $0 \leq dt_j + T(r_i.s_1, p.s_b) - t \leq \theta$, where $T(r_i.s_1, p.s_b)$ denotes the travel time required by bus b_{ij} from $r_i.s_1$ to $p.s_b$ via the bus route r_i, and θ is a given waiting time threshold.*

There are multiple ways available to approximate $T(r_i.s_1, p.s_b)$. In this paper, we utilize the historical average travel time from $r_i.s_1$ to $p.s_b$ via the route r_i to compute $T(s_1, s_b)$. Based on Definition 1, we formally introduce $\mathcal{S}(b_{ij}, p_k)$ to denote the service of b_{ij} to p_k, as presented in Equation (1).

$$\mathcal{S}(b_{ij}, p_k) = \begin{cases} 1 \text{ if } b_{ij} \text{ can serve } p_k \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

Next, we introduce the concept of bus service frequency in Definition 2. Let the bus service frequency $\mathcal{F}$ for $\mathcal{R}$ be a set, with each element $f_i \in \mathcal{F}$ corresponding to a bus route $r_i \in \mathcal{R}$, i.e., $\mathcal{F} = \{\cup_{\forall r_i \in \mathcal{R}} f_i\}$. Then, the service of $\mathcal{F}$ to a passenger p_k can be computed by Equation (2). Note $\mathcal{S}(\mathcal{F}, p_k) = 1$ as long as any $b_{ij} \in \mathcal{F}$ can serve p_k; otherwise, $\mathcal{S}(\mathcal{F}, p_k) = 0$.

$$\mathcal{S}(\mathcal{F}, p_k) = 1 - \prod\nolimits_{b_{ij} \in \mathcal{F}} (1 - \mathcal{S}(b_{ij}, p_k)) \tag{2}$$

Definition 2. *A bus service frequency (f_i) for r_i refers to a set of buses (b_{i1}, b_{i2}, $\cdots$, b_{in_i}) that serve the route r_i, where n_i ($n_i \geq 1$) denotes the total number of bus departures corresponding to the route r_i within a day.*

Next, we formulate our problem in Definition 3 and show its NP-hardness. Note that we ignore the passenger capacity of the bus in our problem definition.

Definition 3 (Satis<u>FA</u>ction-Boo<u>ST</u> Bus Scheduling (FAST)). *Given a bus route database $\mathcal{R}$, a passenger database $\mathcal{P}$, a waiting time threshold θ, and a vector $\mathcal{N}\langle n_1, n_2, \cdots, n_i, \cdots, n_{|\mathcal{R}|}\rangle$ where n_i (≥ 1) denotes the total number of bus departures of bus route $r_i \in \mathcal{R}$, we output a bus service frequency $\mathcal{F}$ which can maximize $\mathcal{G}(\mathcal{F}) = \sum_{p_k \in \mathcal{P}} \mathcal{S}(\mathcal{F}, p_k)$, where $\mathcal{G}(\mathcal{F})$ denotes the total number of passengers served by $\mathcal{F}$.*

Theorem 1. *The objective function $\mathcal{G}$ of FAST is monotone and submodular.*

Proof. We skip the proof of the monotonicity of $\mathcal{G}$ as it is straightforward. In the following, we prove that $\mathcal{G}$ is submodular. Let $V \subseteq T \subset \mathcal{B}$, where $\mathcal{B}$ denotes the universe of buses, and b refers to a bus in $\mathcal{B}\backslash T$. According to [11], $\mathcal{G}(V)$ is submodular if it satisfies: $\mathcal{G}(V \cup b) - \mathcal{G}(V) \geq \mathcal{G}(T \cup b) - \mathcal{G}(T)$. To facilitate the proof, we define $V_b = V \cup b$ and $\mathcal{G}_b(V) = \mathcal{G}(V \cup b) - \mathcal{G}(V)$. Then, we have:

$$\begin{aligned}\mathcal{G}_b(V) - \mathcal{G}_b(T) = &(\sum_{p_k \in \mathcal{P}} \mathcal{S}(V_b, p_k) - \sum_{p_k \in \mathcal{P}} \mathcal{S}(V, p_k)) \\ &-(\sum_{p_k \in \mathcal{P}} \mathcal{S}(T_b, p_k) - \sum_{p_k \in \mathcal{P}} \mathcal{S}(T, p_k)) \\ &= \sum_{p_k \in \mathcal{P}} (\mathcal{S}(V_b, p_k) - \mathcal{S}(V, p_k) - \mathcal{S}(T_b, p_k) + \mathcal{S}(T, p_k)).\end{aligned} \tag{3}$$

To show the submodularity of $\mathcal{G}$, we first prove Inequality (4).

$$\mathcal{S}(V_b, p_k) - \mathcal{S}(V, p_k) - \mathcal{S}(T_b, p_k) + \mathcal{S}(T, p_k) \geq 0 \tag{4}$$

According to whether p_k can be served by buses in V or buses in $T\backslash V$ or bus b, there are in total four cases corresponding to Inequality (4). <u>Case 1</u>: p_k can be served by a bus $b_0 \in V$. Then we have $\mathcal{S}(V, p_k) = \mathcal{S}(V_b, p_k) = \mathcal{S}(T, p_k) = \mathcal{S}(T_b, p_k) = 1$, because $V \subset V_b$ and $V \subseteq T \subset T_b$. Thus, $\mathcal{S}(V_b, p_k) - \mathcal{S}(V, p_k) - \mathcal{S}(T_b, p_k) + \mathcal{S}(T, p_k) = 0$. <u>Case 2</u>: p_k cannot be served by any bus $b_0 \in V$ but it can be served by a bus $b_1 \in T\backslash V$. Then we have $\mathcal{S}(V, p_k) = 0$, $\mathcal{S}(V_b, p_k) \geq 0$ and $\mathcal{S}(T, p_k) = \mathcal{S}(T_b, p_k) = 1$. Thus, $\mathcal{S}(V_b, p_k) - \mathcal{S}(V, p_k) - \mathcal{S}(T_b, p_k) + \mathcal{S}(T, p_k) \geq 0$. <u>Case 3</u>: p_k cannot be served by any bus $b_0 \in T$ and can be served by the bus b. Then we have $\mathcal{S}(V, p_k) = \mathcal{S}(T, p_k) = 0$ and $\mathcal{S}(V_b, p_k) = \mathcal{S}(T_b, p_k) = 1$. Thus, $\mathcal{S}(V_b, p_k) - \mathcal{S}(V, p_k) - \mathcal{S}(T_b, p_k) + \mathcal{S}(T, p_k) = 0$. <u>Case 4</u>: p_k cannot be served by any bus $b_0 \in T$ or the bus b. Then we have $\mathcal{S}(V, p_k)$=$\mathcal{S}(V_b, p_k)$=$\mathcal{S}(T, p_k) = \mathcal{S}(T_b, p_k) = 0$. Thus, $\mathcal{S}(V_b, p_k) - \mathcal{S}(V, p_k) - \mathcal{S}(T_b, p_k) + \mathcal{S}(T, p_k) = 0$. The above shows the correctness of Inequality (4). Based on Equation (3) and Inequality (4), we have $\mathcal{G}_b(V) - \mathcal{G}_b(T) \geq 0$ and hence $\mathcal{G}$ is a submodular function. ■

Theorem 2. *The FAST problem is NP-hard.*

Algorithm 1: Greedy $(\mathcal{B}, \mathcal{R}, \mathcal{P}, \mathcal{N})$

```
1.1  Input: a bus database $\mathcal{B}$, a bus route database $\mathcal{R}$, a passenger database $\mathcal{P}$, and a
     vector $\mathcal{N}$ $\langle n_1, n_2, \cdots, n_{|\mathcal{R}|} \rangle$
1.2  Output: a bus service frequency $\mathcal{F}$
1.3  Initialize $\mathcal{F} \leftarrow \phi$, $n \leftarrow \sum_{i=1}^{|\mathcal{N}|} n_i$
1.4  Initialize a $|\mathcal{N}|$-dimension vector $\langle k_1, k_2, \cdots, k_{|\mathcal{N}|} \rangle$ with zero
1.5  for $i \leftarrow 1$ to $n$ do
1.6      Select a bus $b_{jl} \leftarrow \arg\max_{b \in \mathcal{B} \setminus \mathcal{F}} (\mathcal{G}(\mathcal{F} \cup b) - \mathcal{G}(\mathcal{F}))$
1.7      $k_j++$
1.8      if $k_j \leq n_j$ then
1.9          $\mathcal{F} \leftarrow \mathcal{F} \cup b_{jl}$
1.10     if $k_j \geq n_j$ then
1.11         remove all the buses serving the route $j$ from $\mathcal{B}$
1.12 return $\mathcal{F}$
```

Proof. It is worth noting that the minimum unit of time is second in daily life. Therefore, $\mathcal{B}$ is a finite set. Based on this, we prove it by reducing the Set Cover problem to the FAST problem. In the Set Cover problem, given a collection of subsets $S_1, \cdots, S_i, \cdots, S_j$ of a universe of elements U, we wish to know whether there exist k of the subsets whose union is equal to U. We map each element in U in the Set Cover problem to each passenger in $\mathcal{P}$, and map each subset S_i to the set of passengers server by a bus $b \in \mathcal{B}$. Consequently, if all passengers in U are served by S, the total number of passengers served by S is $|U|$. Subsequently, $n = \sum_{i=1}^{|\mathcal{R}|} n_i$ is set to k (selecting k buses). The Set Cover problem is equivalent to deciding if there is a k-bus set with the maximum served passenger number U in FAST. As the Set Cover problem is NP-complete, the decision problem of FAST is NP-complete, and the optimization problem is NP-hard. ■

4 Basic Greedy Method

To address FAST, we first present a baseline which extends the basic greedy method for the problem of submodular function maximization. To accelerate the marginal gain computation, we propose a mapping structure to index the bus and passenger database. The basic greedy method is guaranteed to achieve (1 - $1/e$)-approximation, as proved by Nemhauser et al. [11].

4.1 A Basic Greedy Method

The pseudo-code of the greedy method is listed in Algorithm 1. In each iteration, it selects a bus $b_{jl} \in \mathcal{B} \setminus \mathcal{F}$ with the largest marginal gain, such that $b_{jl} = \arg\max_{b \in \mathcal{B} \setminus \mathcal{F}} (\mathcal{G}(\mathcal{F} \cup b) - \mathcal{G}(\mathcal{F}))$, and inserts it to the current service frequency $\mathcal{F}$. In lines 1.8-1.11, it checks whether the number of bus departures of route j, which b_{jl} serves, has reached the total number of bus departures required by this route. If so, it removes all buses serving the route j from $\mathcal{B}$. Such an iteration is repeated n times, with n being the total number of bus departures required by all the bus routes. Finally, it returns $\mathcal{F}$ as the solution.

Bus List	$N_{ToBeServed}$	L_P
b_1	3	$p_1, p_3, p_{\|\mathcal{P}\|}$
b_2	2	p_1, p_2
b_3	1	p_3
$\cdots$	$\cdots$	$\cdots$
$b_{\|\mathcal{B}\|}$	1	p_2

Fig. 1. Forward list

Passenger List	$IsServed$	Optional Buses
p_1	$false$	b_1, b_2
p_2	$false$	$b_2, b_{\|\mathcal{B}\|}$
p_3	$false$	b_1, b_3
$\cdots$	$\cdots$	$\cdots$
$p_{\|\mathcal{P}\|}$	$false$	b_1

Fig. 2. Inverted list

Time Complexity. In each iteration, Algorithm 1 needs to scan all the buses in $\mathcal{B} \backslash \mathcal{F}$ and computes their marginal gain to the chosen set. Each marginal gain computation needs to traverse $\mathcal{P}$ once in the worst case. Thus, adding one bus into $\mathcal{F}$ takes $O(|\mathcal{P}| \cdot |\mathcal{B}|)$ time, and the total complexity is $O(n \cdot |\mathcal{P}| \cdot |\mathcal{B}|)$.

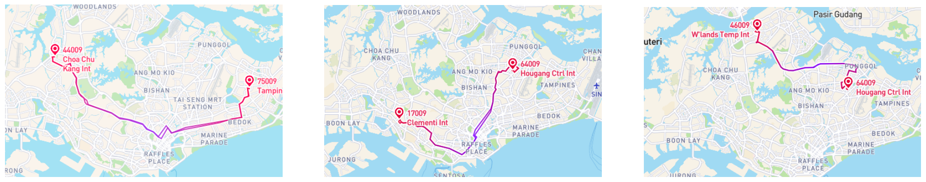
4.2 Index for Efficient Marginal Gain Computation

To accelerate the marginal gain computation, which is the main bottleneck of Algorithm 1, we propose two mapping indexes, *forward list* and *inverted list* as shown in Fig. 1 and Fig. 2 respectively. The former is for buses $b_i \in \mathcal{B}$, maintaining a list of passengers L_P that could be served by bus b_i. Note that a passenger could be served by multiple buses. To avoid counting the same passenger multiple times when we calculate the marginal gain, we maintain another parameter $N_{ToBeServed}$ to capture the number of passengers in L_P that are still waiting for services. The initial value of $N_{ToBeServed}$ is set to be the cardinality of L_P, and its value will be reduced every time when a passenger in L_P is served by another bus. The latter is for passengers $p \in \mathcal{P}$, maintaining a list of buses that could serve the passenger p. The boolean $IsServed$ is to indicate whether any of the optional buses has been scheduled with an initial value being $false$. For example, if bus b_1 is selected, it could serve three passengers based on $N_{ToBeServed}$'s value associated with b_1 in forward list. Meanwhile, $IsServed$'s value of passengers in L_P of b_1 (i.e., $p_1, p_3, p_{|\mathcal{P}|}$) will be changed to $true$, all the buses that could serve p_1 or p_3 or $p_{|\mathcal{P}|}$ have to update $N_{ToBeServed}$'s value to reflect the fact that some of their potential passengers have already been served.

5 Partition-based Greedy Method

In practice, a bus network is designed to cover different parts of a city to meet residents' various travel demands. By design, it tries to avoid unnecessary overlapping among routes [4,16]. For example, Figure 3 plots three popular bus routes in Singapore. A passenger whose travel demand could be served by route 67 will not consider route 161 or route 147 as these routes have *zero* overlap. This observation suggests that it might be unnecessary to scan the entire bus network when calculating the marginal gains of certain buses. This motivates us to design a partition-based greedy method. In the following, we first introduce a novel concept namely *service overlap ratio* to guide the partitioning process, and then present the algorithm.

Our main idea is to partition the bus routes (and buses) into disjoint clusters, and then use a divide-and-conquer strategy to find local optimal frequencies for



(a) Bus Route 67 (b) Bus Route 147 (c) Bus Route 161

Fig. 3. Visualization of three popular bus routes in Singapore

routes in each partition. This approach is expected to reduce the time complexity of the basic greedy by a factor of m^2 with m being the number of partitions. The speedup is contributed by the fact that it invokes the greedy algorithm for each cluster and hence it only needs to scan the buses and passengers corresponding to the routes in a cluster during the greedy search. Meanwhile, in term of accuracy, we introduce a novel concept called *service overlap ratio* to achieve an approximation ratio with non-trivial theoretical guarantee, as shown later.

Definition 4 (Partition). *A partition of a set S is denoted as a cluster set $\mathcal{C}=\{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_m\}$, where m denotes the total number of clusters, such that $S = \cup_{i=1}^{m} \mathcal{C}_i$, $\forall \mathcal{C}_i \in \mathcal{C}$, $\mathcal{C}_i \neq \phi$, and $\forall \mathcal{C}_i, \mathcal{C}_j \in \mathcal{C}$ with $i \neq j$, $\mathcal{C}_i \cap \mathcal{C}_j = \phi$.*

To better illustrate the service overlap ratio, we define a function $\mathsf{Serve}(P,R)$ that takes a passenger set P and a route set R as inputs and returns the passengers in P that could be served by any route in R without considering the temporal factor. To be more specific, a passenger p will be returned by $\mathsf{Serve}(P,R)$ if there is a route $r_i \in R$ such that r_i contains $p.s_b$ and $p.s_e$ in order, which is different from the "bus serves passengers" defined in Definition 1. We name the set of passengers returned by $\mathsf{Serve}(P,R)$ as the passenger pool w.r.t. bus routes R.

As stated in Definition 5, the service overlap ratio ρ_i of a bus route cluster $\mathcal{C}_i^R$ tries to measure the number of passengers in the passenger pool w.r.t. $\mathcal{C}_i^R$ that actually also belong to the passenger pools w.r.t. other clusters. Let $|A|$ denote the cardinality of the set A, and $\overline{\mathcal{F}}_i$ denote a bus service frequency returned by $\mathrm{Greedy}(\mathcal{C}_\mathrm{i}^\mathrm{B}, \mathcal{C}_\mathrm{i}^\mathrm{R}, \mathcal{C}_\mathrm{i}^\mathrm{P}, \mathrm{N_{min}})$. $\mathcal{C}_i^B$, $\mathcal{C}_i^R$, and $\mathcal{C}_i^P$ refer to a cluster of buses, a cluster of routes and a cluster of passengers respectively, and N_{min} refers to a $|\mathcal{C}_i^R|$-dimensional vector in the form of $\langle n_{min}, n_{min}, \cdots, n_{min}\rangle$. The parameter n_{min} is set to the minimum number of buses required by any route. Although there are different ways to quantify the overlaps between bus routes, we define ρ_i in such a way that a partition-based greedy guided by ρ_i can achieve a theoretical bound, as to be detailed next.

Definition 5 (Service overlap ratio). *Given a partition $\mathcal{C}^R$ of the original bus route database $\mathcal{R}$, for a cluster $\mathcal{C}_i^R$, the ratio ρ_i of the service overlap between $\mathcal{C}_i^R$ and the rest clusters is $\frac{\left|\bigcup_{\mathcal{C}_j^R \in \mathcal{C}^R \setminus \mathcal{C}_i^R} \mathsf{Serve}(\mathcal{P},\mathcal{C}_i^R) \cap \mathsf{Serve}(\mathcal{P},\mathcal{C}_j^R)\right|}{\mathcal{G}(\overline{\mathcal{F}}_i)}$.*

Partitioning of bus routes and buses. Algorithm 3 lists the pseudo-code of a bus route partitioning method guided by service overlap ratio. It first partitions

Algorithm 2: PartGreedy $(\mathcal{B}, \mathcal{R}, \mathcal{P}, \mathcal{N}, \rho)$

```
2.1 Input: a bus database $\mathcal{B}$, a bus route database $\mathcal{R}$, a passenger database $\mathcal{P}$, and a
    vector $\mathcal{N}$ $\langle n_1, n_2, \cdots, n_{|\mathcal{R}|}\rangle$, a controlling threshold $\rho$
2.2 Output: a bus service frequency $\mathcal{F}$
2.3 initialize $\mathcal{C}^R \leftarrow \phi$, $\mathcal{C}^B \leftarrow \phi$, $S_P \leftarrow \phi$, $n_{min} \leftarrow Min_{1\leq i\leq|\mathcal{R}|} n_i$, $\mathcal{F} \leftarrow \phi$
2.4 $(\mathcal{C}^B, \mathcal{C}^R) \leftarrow$ BusRoutePartitioning$(\mathcal{B}, \mathcal{R}, \mathrm{n_{min}}, \rho)$
2.5 for each cluster $\mathcal{C}_i^R \in \mathcal{C}^R$ do
2.6     $S_P \leftarrow$ Serve$(\mathcal{P}, Cluster_i^R)$, $\mathcal{F} \leftarrow \mathcal{F} \cup$ Greedy$(\mathcal{C}_\mathrm{i}^\mathrm{B}, \mathcal{C}_\mathrm{i}^\mathrm{R}, \mathrm{S_P}, \mathcal{N})$
2.7 return $\mathcal{F}$
```

the routes using the finest granularity by forming a cluster for each bus route. Thereafter, it checks the service overlap ratio ρ_i for each cluster $\mathcal{C}_i^R$ and picks the one with the largest ρ_i, denoted as $\mathcal{C}_k^R$, for expansion (Line 3.9). It selects the cluster $\mathcal{C}_j^R$ that shares the largest common passenger pool with $\mathcal{C}_k^R$(Line 3.11) and merges $\mathcal{C}_j^R$ with $\mathcal{C}_k^R$ (Lines 3.12 - 3.14). Note that when cluster $\mathcal{C}_k^R$ is expanded, let $\overline{\mathcal{F}}_k$ denote the new frequency returned by Greedy$(\mathcal{C}_k^B, \mathcal{C}_k^R, \mathcal{P}, N_{min})$. $\mathcal{G}(\overline{\mathcal{F}}_k)$ is actually required when calculating ρ_k for this expanded cluster, by Definition 5. However, to reduce the computation cost and the complexity, we use $\mathcal{L} = max\{\mathcal{G}(\overline{\mathcal{F}}_k) + \mathcal{G}(\overline{\mathcal{F}}_j) - |S_k \cap S_j|, \mathcal{G}(\overline{\mathcal{F}}_k), \mathcal{G}(\overline{\mathcal{F}}_j)\}$ as an approximation of $\mathcal{G}(\overline{\mathcal{F}}_k)$. According to our merger rules, $\mathcal{L}$ is a lower bound of $\mathcal{G}(\overline{\mathcal{F}}_k)$ and it does not affect the accuracy of our partition algorithm. This merge-and-expansion process continues until the ρ_is associated with all the clusters $\mathcal{C}_i^R$ fall below the input threshold ρ.

When the bus routes and buses are partitioned, it invokes the basic greedy method (Section 4) to find the frequency for each cluster, and merges the local frequencies for $|\mathcal{C}^R|$ clusters as the final answer. We name this approach as PartGreedy. Its pseudo-code is shown in Algorithm 2 and its approximation ratio is analyzed in Lemma 1.

Lemma 1. *Given a partition $\mathcal{C}^R$=$\{\mathcal{C}_1^R, \mathcal{C}_2^R, \cdots, \mathcal{C}_i^R, \cdots, \mathcal{C}_m^R\}$ of the bus route database $\mathcal{R}$ and the maximum service overlap ratio ρ, PartGreedy achieves a $(1-\rho)(1-1/e)$ approximation ratio to solve the FAST problem.*

Proof. Let $\mathcal{F}_i$ denote the solution obtained by Greedy for cluster $\mathcal{C}_i^R$, $\mathcal{F}^*$ denote the solution obtained by PartGreedy, $\mathcal{O}_i$ denote the optimal solution for cluster $\mathcal{C}_i^R$, and $\mathcal{O}$ denote the global optimal solution. In Algorithm 3, it uses the lower bound of the $\mathcal{G}(\overline{\mathcal{F}}_k)$ to compute the upper bound of ρ_k and terminates when the upper bound of ρ_i for every cluster $\mathcal{C}_i^R \in \mathcal{C}^R$ is no greater than the given threshold ρ. Then we have $\rho \geq \rho_i$ for any $\mathcal{C}_i^R \in \mathcal{C}^R$. Recall Section 3, the basic greedy method is proved to achieve $(1-1/e)$-approximation. Therefore, we have $\mathcal{G}(\mathcal{F}_i) \geq (1-1/e)\mathcal{G}(\mathcal{O}_i)$. Because of the submodularity and monotonicity of $\mathcal{G}$, we have $\sum_{i=1}^m \mathcal{G}(\mathcal{O}_i) \geq \mathcal{G}(\mathcal{O})$ and $\mathcal{G}(\mathcal{F}_i) \geq \mathcal{G}(\overline{\mathcal{F}}_i)$. Then, by Definition 5 we have:

$$\left|\bigcup_{\mathcal{C}_j^R \in \mathcal{C}^R \setminus \mathcal{C}_i^R} \mathsf{Serve}(\mathcal{P}, \mathcal{C}_i^R) \cap \mathsf{Serve}(\mathcal{P}, \mathcal{C}_j^R)\right| = \rho_i \mathcal{G}(\overline{\mathcal{F}}_i) \leq \rho \mathcal{G}(\mathcal{F}_i). \qquad (5)$$

Algorithm 3: BusRoutePartitioning $(\mathcal{B}, \mathcal{R}, n_{min}, \rho)$

```
3.1  Input: a bus database $\mathcal{B}$, a bus route database $\mathcal{R}$, an integer $n_{min}$, and a
     controlling threshold $\rho$
3.2  Output: a partition $\mathcal{C}^B$ of $\mathcal{B}$ and a partition $\mathcal{C}^R$ of $\mathcal{R}$
3.3  for each bus route $r_i \in Route$ do
3.4      initialize $\mathcal{C}_i^R \leftarrow \{r_i\}$, $\mathcal{C}_i^B \leftarrow \{b_{ab} \in \mathcal{B} | a = i\}$, $S_i \leftarrow \mathsf{Serve}(\mathcal{P}, Cluster_i^R)$
3.5      $\overline{\mathcal{F}}_i \leftarrow \mathrm{Greedy}(\mathcal{C}_\mathrm{i}^\mathrm{B}, \mathcal{C}_\mathrm{i}^\mathrm{R}, \mathcal{P}, \mathrm{N_{min}})$
3.6  initialize $\mathcal{C}^R \leftarrow \cup_{r_i \in \mathcal{R}} \mathcal{C}_i^R$
3.7  for $\mathcal{C}_i^R \in \mathcal{C}^R$ do
3.8      $\rho_i \leftarrow \left| \bigcup_{\mathcal{C}_j^R \in \mathcal{C}^R \setminus \mathcal{C}_i^R} S_i \cap S_j \right| / \mathcal{G}(\overline{\mathcal{F}}_i)$
3.9  $k \leftarrow \mathrm{argmax}_{\mathcal{C}_k^R \in \mathcal{C}^R} \rho_k$, $Max \leftarrow \rho_k$
3.10 while $Max > \rho$ do
3.11     $j \leftarrow \mathrm{argmax}_{\mathcal{C}_j^R \in \mathcal{C}^R \setminus \mathcal{C}_k^R} |(S_j \cap S_k)|$
3.12     $\mathcal{C}_k^R \leftarrow \mathcal{C}_k^R \cup \mathcal{C}_j^R$, $\mathcal{C}^R \leftarrow \mathcal{C}^R - \mathcal{C}_j^R$, $\mathcal{C}_k^B \leftarrow \mathcal{C}_k^B \cup \mathcal{C}_j^B$, $\mathcal{C}^B \leftarrow \mathcal{C}^B - \mathcal{C}_j^B$
3.13     $\mathcal{G}(\overline{\mathcal{F}}_k) \leftarrow max\{\mathcal{G}(\overline{\mathcal{F}}_k) + \mathcal{G}(\overline{\mathcal{F}}_j) - |S_k \cap S_j|, \mathcal{G}(\overline{\mathcal{F}}_k), \mathcal{G}(\overline{\mathcal{F}}_j)\}$, $S_k \leftarrow S_k \cup S_j$
3.14     $\rho_k \leftarrow \frac{\left| \bigcup_{\mathcal{C}_l^R \in \mathcal{C}^R \setminus \mathcal{C}_k^R} S_l \cap S_k \right|}{\mathcal{G}(\overline{\mathcal{F}}_k)}$
3.15     $k \leftarrow \mathrm{argmax}_{\mathcal{C}_k^R \in \mathcal{C}^R} \rho_k$, $Max \leftarrow \rho_k$
3.16 return $\mathcal{C}^B$, $\mathcal{C}^R$
```

In addition, Inequality (6) holds according to Definition 3.

$$\left| \bigcup_{\mathcal{C}_j^R \in \mathcal{C}^R \setminus \mathcal{C}_i^R} \mathsf{Serve}(\mathcal{P}, \mathcal{C}_i^R) \cap \mathsf{Serve}(\mathcal{P}, \mathcal{C}_j^R) \right| \geq \mathcal{G}(\mathcal{F}_i) - (\mathcal{G}(\mathcal{F}^*) - \mathcal{G}(\mathcal{F}^* \setminus \mathcal{F}_i)) \quad (6)$$

Based on Inequality (5) and Inequality (6), we have $\mathcal{G}(\mathcal{F}^*) - \mathcal{G}(\mathcal{F}^* \setminus \mathcal{F}_i) \geq (1 - \rho)\mathcal{G}(\mathcal{F}_i)$. Using the principle of inclusion-exclusion, we have $\mathcal{G}(\mathcal{F}^*) = \mathcal{G}(\mathcal{F}_1 \cup \mathcal{F}_2 \cup ... \cup \mathcal{F}_m) \geq \sum_{i=1}^{m}(\mathcal{G}(\mathcal{F}^*) - \mathcal{G}(\mathcal{F}^* \setminus \mathcal{F}_i)) \geq (1 - \rho) \sum_{i=1}^{m} \mathcal{G}(\mathcal{F}_i) \geq (1 - \rho)(1 - 1/e) \sum_{i=1}^{m} \mathcal{G}(\mathcal{O}_i) \geq (1 - \rho)(1 - 1/e)\mathcal{G}(\mathcal{O})$. Thus, this lemma is proved. ■

6 Progressive Partition-based Greedy Method

Although PartGreedy improves the efficiency of basic greedy by conducting the search within each partition (though not the original route/bus database), it still suffers from a high computational cost. To be more specific, in each iteration of the greedy search (either a global search or a local search by Greedy), in order to find the one with the maximum gain, it has to recalculate the marginal gain $\mathcal{G}(\mathcal{F} \cup b) - \mathcal{G}(\mathcal{F})$ for all the buses not yet scheduled.

Motivated by this observation, we propose a progressive partition-based greedy method (ProPartGreedy). It selects multiple, but not only one, buses in each local greedy search iteration to cut down the total number of iterations required and hence the computation cost. The pseudo-code of ProPartGreedy is the same as Algorithm 2 except that the call of Greedy is replaced with Function 1 (ProGreedy) in line 2.6 of Algorithm 2. Meanwhile, we will prove that it can achieve

Function 1: ProGreedy $(\mathcal{B}, \mathcal{R}, \mathcal{N}, \varepsilon)$

```
1.1  Input: a bus database $\mathcal{B}$, a bus route database $\mathcal{R}$, a vector $\mathcal{N}$, and a parameter $\varepsilon$
1.2  Output: a bus service frequency $\mathcal{F}$
1.3  Initialize $\mathcal{F} \leftarrow \phi$, $n \leftarrow \sum_{i=1}^{|\mathcal{N}|} n_i$
1.4  Initialize a $|\mathcal{N}|$-dimension vector $\langle k_1, k_2, \cdots, k_{|\mathcal{N}|} \rangle$ with zero
1.5  Sort $b \in \mathcal{B}$ based on descending order of $\mathcal{G}(b)$
1.6  Initialize $h \leftarrow \max_{b \in \mathcal{B}}(\mathcal{G}(b))$
1.7  while $|\mathcal{F}| \leq n$ do
1.8      for each $b_{jl} \in \mathcal{B}$ do
1.9          if $|\mathcal{F}| \leq n$ then
1.10             $\mathcal{G}_{b_{jl}}(\mathcal{F}) \leftarrow \mathcal{G}(\mathcal{F} \cup b_{jl}) - \mathcal{G}(\mathcal{F})$
1.11             if $\mathcal{G}_{b_{jl}}(\mathcal{F}) \geq h$ then
1.12                 $\mathcal{F} \leftarrow \mathcal{F} \cup b_{jl}$, $\mathcal{B} \leftarrow \mathcal{B} \backslash b_{jl}$
1.13                 $k_j + +$
1.14                 if $k_j \geq n_j$ then
1.15                     remove all bus serve the route $r_j$ from $\mathcal{B}$
1.16             if $\mathcal{G}(b_{jl}) < h$ then
1.17                 break
1.18         else
1.19             break
1.20     $h \leftarrow \frac{h}{1+\epsilon}$
1.21 return $\mathcal{F}$
```

an approximation ratio of $(1 - \rho)(1 - 1/e - \varepsilon)$, where ρ and ε are tunable parameters that provide a trade-off between efficiency and accuracy.

As presented in Function 1, ProGreedy first sorts $b \in \mathcal{B}$ by $\mathcal{G}(b)$ and initializes the threshold h to the value of $\max_{b \in \mathcal{B}}(\mathcal{G}(b))$. Then, it iteratively fetches all the buses with their marginal gains not smaller than h into $\mathcal{F}$ and meanwhile lowers the threshold h by a factor of $(1 + \varepsilon)$ for next iteration (Lines 1.8-1.20). The iteration continues until there are n buses in $\mathcal{F}$. Unlike the basic greedy method that has to check all the potential buses in $\mathcal{B}$ or a cluster of $\mathcal{B}$ in each iteration, it is not necessary for ProGreedy as it implements an early termination (Lines 1.16-1.17). Since buses are sorted by $\mathcal{G}(b)$ values, if $\mathcal{G}(b_{jl})$ of the current bus is smaller than h, all the buses b pending for evaluation will have their $\mathcal{G}(b)$ values smaller than h and hence could be skipped from evaluation. In the following, we first analyze the approximation ratio of Function 1 by Lemma 2. Based on Lemma 2, we show the approximation ratio of ProPartGreedy by Lemma 3.

Lemma 2. *ProGreedy achieves a $(1 - 1/e - \varepsilon)$ approximation ratio.*

Proof. Let b_i be the bus selected at a given threshold h and $\mathcal{O}$ denote the optimal local solution to the problem of selecting n *buses* that can maximize $\mathcal{G}$. Because of the submodularity of $\mathcal{G}$, we have:

$$\mathcal{G}_b(\mathcal{F}) = \begin{cases} \geq h & \text{if } b = b_i \\ \leq h \cdot (1 + \varepsilon) & \text{if } b \in \mathcal{O} \backslash (\mathcal{F} \cup b_i), \end{cases} \tag{7}$$

where $\mathcal{F}$ is the current partial solution. Equation (7) implies that $\mathcal{G}_{b_i}(\mathcal{F}) \geq \mathcal{G}_b(\mathcal{F})/(1+\varepsilon)$ for any $b \in \mathcal{O}\backslash\mathcal{F}$. Thus, we have $\mathcal{G}_{b_i}(\mathcal{F}) \geq \frac{1}{(1+\varepsilon)|\mathcal{O}\backslash\mathcal{F}|}\sum_{b\in\mathcal{O}\backslash\mathcal{F}}\mathcal{G}_b(\mathcal{F}) \geq \frac{1}{(1+\varepsilon)n}\sum_{b\in\mathcal{O}\backslash\mathcal{F}}\mathcal{G}_b(\mathcal{F})$. Let $\mathcal{F}_i$ denote the partial solution that b_i has been included and b_{i+1} be the bus selected at the $(i+1)$th step. Then we have $\mathcal{G}(\mathcal{F}_{i+1}) - \mathcal{G}(\mathcal{F}_i) = \mathcal{G}_{b_{i+1}}(\mathcal{F}_i) \geq \frac{1}{(1+\varepsilon)n}\sum_{b\in\mathcal{O}\backslash\mathcal{F}_i}\mathcal{G}_b(\mathcal{F}_i) \geq \frac{1}{(1+\varepsilon)n}(\mathcal{G}(\mathcal{O}\cup\mathcal{F}_i) - \mathcal{G}(\mathcal{F}_i)) \geq \frac{1}{(1+\varepsilon)n}(\mathcal{G}(\mathcal{O}) - \mathcal{G}(\mathcal{F}_i))$.

The solution $\mathcal{F}^*$ obtained by Function 1 with $|\mathcal{F}^*| = n$. Using the geometric series formula, we have $\mathcal{G}(\mathcal{F}^*) \geq \left(1 - \left(1 - \frac{1}{(1+\varepsilon)n}\right)^n\right)\mathcal{G}(\mathcal{O}) \geq \left(1 - e^{\frac{-n}{(1+\varepsilon)n}}\right)\mathcal{G}(\mathcal{O}) = \left(1 - e^{\frac{-1}{(1+\varepsilon)}}\right)\mathcal{G}(\mathcal{O}) \geq ((1 - 1/e - \varepsilon))\,\mathcal{G}(\mathcal{O})$. Hence, the lemma is proved. ■

Lemma 3. *Given a partition $\mathcal{C}^R$=$\{\mathcal{C}_1^R, \mathcal{C}_2^R, \cdots, \mathcal{C}_i^R, \cdots, \mathcal{C}_m^R\}$ of the bus route database $\mathcal{R}$ and the maximum service overlap ratio ρ, ProPartGreedy achieves a $(1-\rho)(1-1/e-\varepsilon)$ approximation ratio to solve the FAST problem.*

Proof. Based on Lemma 2, this proof is similar to the proof of Lemma 1, so we omit it due to space limit. ■

Table 1. Statistics of datasets

Database	Amount	AvgDistance	AvgTravelTime
$\mathcal{B}$	451k	N.A.	N.A.
$\mathcal{R}$	396	19.91km	5159s
$\mathcal{P}$	28m	4.2km	1342s

7 Experiment

In this section, we first explain the experimental setup; we then conduct sensitivity tests to tune the parameters to their reasonable settings, as our algorithms have several tunable parameters; we finally report the performance, in terms of effectiveness, efficiency, and scalability, of all the algorithms.

Datasets. We crawl the real bus routes ($\mathcal{R}$) from transitlink[5] in Singapore. Each route is represented by the sequence of bus stop IDs it passes sequentially, together with the distance between two consecutive bus stops. The travel time from a stop to another stop via a route r_i is estimated by the ratio of the distance between those two stops along the route to the average bus speed of the route. We use bus touch-on record data (shown later) to find the average travel speed of a particular bus line. For the passenger database ($\mathcal{P}$), due to the exhibit regular travel patterns of passengers [15], we use the real bus touch-on record data in a week of April 2016 in Singapore, which is obtained from the authors of [15] and contains 28 million trip records. Each trip record includes the IDs/timestamps of the boarding and alighting bus stops, the bus route, and the trip distance. We assume passengers spend x minutes waiting for their buses, with x following a random distribution between 1 and 5 minutes. Then, we generate the bus candidate set ($\mathcal{B}$) based on the route and service time range. For each route, we

[5] `https://www.transitlink.com.sg/eservice/eguide/service_idx.php`

Table 2. Parameter settings

Parameter	Values
number of bus departures $\mathcal{N} = \langle n_1, n_2, \cdots \rangle$	$\langle 10 \rangle$, $\langle 20 \rangle$, $\boldsymbol{\langle 30 \rangle}$, $\langle 40 \rangle$, $\langle 50 \rangle$
total passenger number $\lvert \mathcal{P} \rvert$	100k, 200k, **300k**, 400k, 500k
waiting time threshold θ	1min, 2min, **3min**, 4min, 5min
tunable parameter used byProPartGreedy ε	10^{-4}, 10^{-3}, $\mathbf{10^{-2}}$, 10^{-1}
controlling threshold used by PartGreedy ρ	0.1, **0.2**, 0.3, 0.4

use buses that depart every minute between 5am and 12am as the superset of candidate buses. The statistics of those datasets are shown in Table 1.

Parameters. Table 2 lists the parameter settings, with values in bold being default. In all experiments, we vary one parameter and set the rest to their defaults. We assume all bus routes require the same number of bus departures in our study. Notation $\langle 20 \rangle$ represents the vector $\langle 20, \cdots, 20 \rangle$ for brevity.

Algorithm. To the best of our knowledge, this is the first work to study the FAST problem, and thus no previous work is available for direct comparison. In particular, we compare the following five methods. FixInterval that fixes the time interval between two bus departures as $\lfloor$(service time range) / (bus number)$\rfloor$ for each line and chooses the bus that departures at 5am as the first bus; Top-k that picks top-k buses, which could serve the most number of passengers $(k = n_i)$; Greedy, PartGreedy, and ProPartGreedy, i.e., Algorithm 1, Algorithm 2, and the progressive partition-based method proposed in this paper.

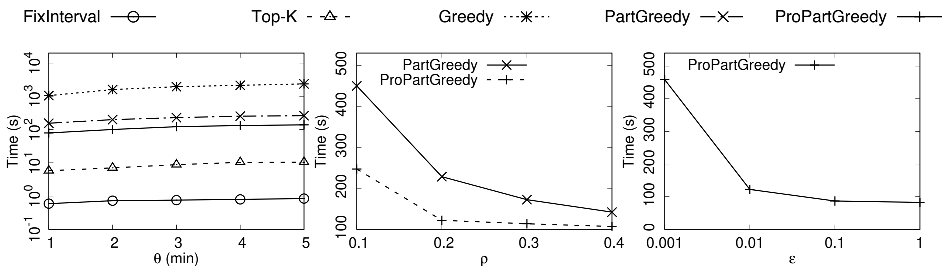
Performance measurement. We adopt the *total running time* of each algorithm and the *total served passenger number (SPN)* of the scheduled buses as the main performance metrics. We randomly choose 5 million passengers from a week of data and pre-process the passenger dataset to build the index, which takes $5,690$ seconds and occupies 585MB disk space. Each experiment is repeated ten times, and the average result is reported.

Setup. All codes are implemented in C++. Experiments are conducted on a server with 24 Intel X5690 CPU and 140GB memory running CentOS release 6.10. We will release the code publicly once the paper is published.

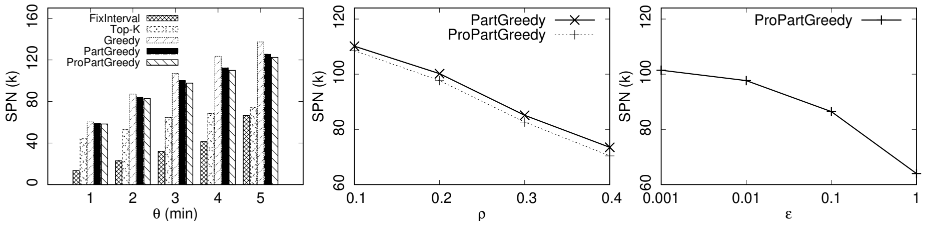
Parameter Sensitivity Test - θ. The impact of waiting time threshold θ on the running time and SPN are reported in Figure 4(a) and Figure 4(d), respectively. Parameter θ has an almost-zero impact on the running time. On the other hand, it affects SPN. As θ increases, all the algorithms are able to serve more passengers, which is consistent with our expectations. We set $\theta = 3$, the mean value.

Parameter Sensitivity Test - ρ. The impact of parameter ρ on the running time and SPN are reported in Figure 4(b) and Figure 4(e), respectively. It has a positive impact on the running time performance but a negative impact on SPN. As ρ increases its value, PartGreedy and ProPartGreedy both incur shorter running time but serve less number of passengers. We choose $\rho = 0.2$ as the default setting.

Parameter Sensitivity Test - ε. Parameter ε only affects ProPartGreedy. It controls the trade-off between efficiency and accuracy. As ε increases its value, ProPartGreedy incurs shorter running time and serves less number of passengers,

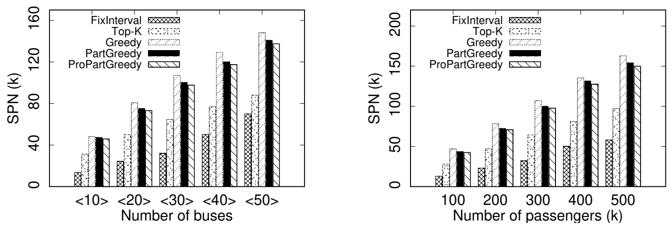


(a) Running time vs. θ (b) Running time vs. ρ (c) Running time vs.ε



(d) SPN vs. θ (e) SPN vs. ρ (f) SPN vs. ε

Fig. 4. Effect of parameters



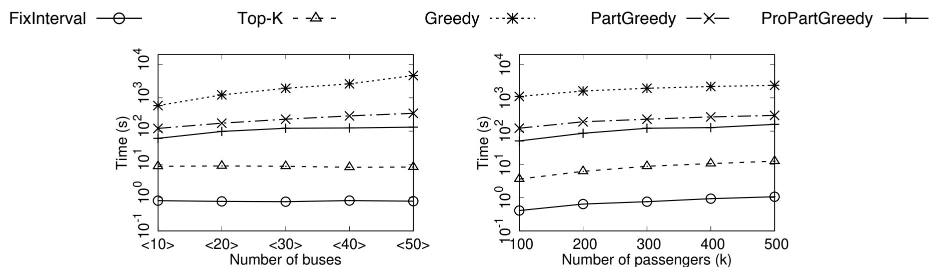
(a) SPN vs. $\mathcal{N}$ (b) SPN vs. $|\mathcal{P}|$

Fig. 5. Effectiveness Study: SPN vs. $\mathcal{N}$ or $|\mathcal{P}|$

as reported in Figure 4(b) and Figure 4(f), respectively. We choose $\varepsilon = 0.01$ as the default setting.

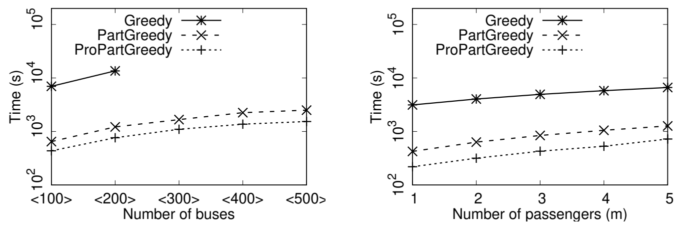
Effectiveness Study. We report the effectiveness of different algorithms in Figure 5. We observe that (1) FixInterval is most *ineffective*; (2) the three algorithms proposed in this work perform much better than the other two, e.g., ProPartGreedy doubles (or even triples in some cases) the SPN of FixInterval; and (3) Greedy performs the best while PartGreedy and ProPartGreedy achieve comparable performance (only up to 9.4% below that of Greedy).

Efficiency Study. Figure 6 shows the running time of each method w.r.t. varying $\mathcal{N}$ and $|\mathcal{P}|$. We have two main observations. (1) The time gap among Greedy, PartGreedy and ProPartGreedy becomes more significant with the increase of $\mathcal{N}$. This could be the increase of $\mathcal{N}$ causes an increase in the number of clusters and n_{min}. On the other hand, PartGreedy and ProPartGreedy only need to scan one cluster when selecting buses. (2) The improvement of PartGreedy and ProPartGreedy over Greedy decreases with the increase of $|\mathcal{P}|$. This is because



(a) Running time vs. $\mathcal{N}$ (b) Running time vs. $|\mathcal{P}|$

Fig. 6. Efficiency Study: Total Running Time vs. $\mathcal{N}$ or $|\mathcal{P}|$



(a) Running time vs. $\mathcal{N}$ (b) Running time vs. $|\mathcal{P}|$

Fig. 7. Scalability Study

the overlap between clusters increases with the increase of $|\mathcal{P}|$, which leads to a reduction in the number of clusters and an increase in partition time.

Scalability Study. To evaluate the scalability of our methods, we vary $\mathcal{N}$ from $\langle 100\rangle$ to $\langle 500\rangle$, and $|\mathcal{P}|$ from 1 million to 5 million. From Figure 7(a), we find that the efficiency of Greedy is more sensitive to $\mathcal{N}$, as compared to PartGreedy and ProPartGreedy. It's worth noting that the results are omitted for Greedy when it cannot terminate within 10^4 seconds. As shown in Figure 7(b), PartGreedy and ProPartGreedy are about ten times faster than Greedy when $|\mathcal{P}|$ is varying.

8 Conclusion

In this paper we studied the bus frequency optimization problem considering user satisfaction for the first time. Our target is to schedule the buses in such a way that the total number of passengers who could receive their bus services within the waiting time threshold is maximized. We showed that this problem is NP-hard, and proposed three approximation algorithms with non-trivial theoretical guarantees. Lastly, we conducted experiments on real-world datasets to verify the efficiency, effectiveness, and scalability of our methods.

Acknowledgements. Zhiyong Peng is supported in part by the National Key Research and Development Program of China (Project Number: 2018YFB1003400), Key Project of the National Natural Science Foundation of China (Project Number: U1811263) and the Research Fund from Alibaba Group. Zhifeng Bao is supported in part by ARC DP200102611, DP180102050, NSFC 91646204, and a Google Faculty Award. Baihua Zheng is supported in part by Prime Ministers Office, Singapore under its International Research Centres in Singapore Funding Initiative.

References

1. Antonides, G., Verhoef, P.C., Van Aalst, M.: Consumer perception and evaluation of waiting time: A field experiment. Journal of consumer psychology **12**(3), 193–202 (2002)
2. Ceder, A., Golany, B., Tal, O.: Creating bus timetables with maximal synchronization. Transportation Research Part A: Policy and Practice **35**(10), 913–928 (2001)
3. Constantin, I., Florian, M.: Optimizing frequencies in a transit network: a nonlinear bi-level programming approach. International Transactions in Operational Research **2**(2), 149–164 (1995)
4. Fletterman, M., et al.: Designing multimodal public transport networks using metaheuristics. Ph.D. thesis, University of Pretoria (2009)
5. Gao, Z., Sun, H., Shan, L.L.: A continuous equilibrium network design model and algorithm for transit systems. Transportation Research Part B: Methodological **38**(3), 235–250 (2004)
6. Ibarra-Rojas, O.J., Delgado, F., Giesen, R., Muñoz, J.C.: Planning, operation, and control of bus transport systems: A literature review. Transportation Research Part B: Methodological **77**, 38–75 (2015)
7. Ibarra-Rojas, O.J., Rios-Solis, Y.A.: Synchronization of bus timetabling. Transportation Research Part B: Methodological **46**(5), 599–614 (2012)
8. Kong, M.C., Camacho, F.T., Feldman, S.R., Anderson, R.T., Balkrishnan, R.: Correlates of patient satisfaction with physician visit: differences between elderly and non-elderly survey respondents. Health and Quality of Life Outcomes **5**(1), 62 (2007)
9. Lin, N., Ma, W., Chen, X.: Bus frequency optimisation considering user behaviour based on mobile bus applications. IET Intelligent Transport Systems **13**(4), 596–604 (2019)
10. Martínez, H., Mauttone, A., Urquhart, M.E.: Frequency optimization in public transportation systems: Formulation and metaheuristic approach. European Journal of Operational Research **236**(1), 27–36 (2014)
11. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions - I. Math. Program. **14**(1), 265–294 (1978)
12. Parbo, J., Nielsen, O.A., Prato, C.G.: User perspectives in public transport timetable optimisation. Transportation Research Part C: Emerging Technologies **48**, 269–284 (2014)
13. Schéele, S.: A supply model for public transit services. Transportation Research Part B: Methodological **14**(1-2), 133–146 (1980)
14. Shafahi, Y., Khani, A.: A practical model for transfer optimization in a transit network: Model formulations and solutions. Transportation Research Part A: Policy and Practice **44**(6), 377–389 (2010)
15. Tian, X., Zheng, B.: Using smart card data to model commuters responses upon unexpected train delays. In: Big Data. pp. 831–840. IEEE (2018)
16. Wang, S., Bao, Z., Culpepper, J.S., Sellis, T., Cong, G.: Reverse k nearest neighbor search over trajectories. IEEE Trans. Knowl. Data Eng. **30**(4), 757–771 (2018)
17. Wu, Y.: Combining local search into genetic algorithm for bus schedule coordination through small timetable modifications. International Journal of Intelligent Transportation Systems Research **17**(2), 102–113 (2019)
18. Zhang, P., Bao, Z., Li, Y., Li, G., Zhang, Y., Peng, Z.: Trajectory-driven influential billboard placement. In: SIGKDD. pp. 2748–2757. ACM (2018)