Towards an Optimal Bus Frequency Scheduling: When the Waiting Time Matters

Songsong Mo, Zhifeng Bao, Baihua Zheng, and Zhiyong Peng

Abstract—Reorganizing bus frequencies to cater for the actual travel demands can significantly save the cost of the public transport system. Many, if not all, previous studies formulate this as a bus frequency optimization problem that tries to minimize passengers' average waiting time. On the other hand, 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 can be maximized. We propose two variants of the problem, FAST and FASTCO, to cater for different application needs and prove that both are NP-hard. To solve FAST effectively and efficiently, we first present an index-based $(1-1/e)$-approximation algorithm. By exploiting the locality property of routes in a bus network, we further propose a partition-based greedy method that achieves a $(1-\rho)(1-1/e)$ approximation ratio. Then we propose a progressive partition-based greedy method to further boost the efficiency while achieving a $(1-\rho)(1-1/e-\varepsilon)$ approximation ratio. For the FASTCO problem, two greedy-based heuristic methods are proposed. Experiments on a real city-wide bus dataset in Singapore have been conducted to verify the efficiency, effectiveness, and scalability of our methods in addressing FAST and FASTCO respectively.

Index Terms—bus frequency scheduling optimization, user waiting time minimization, approximation algorithm.

1 Introduction

Public transport is not only an important contributing factor to urban sustainability, but also an essential part of tackling both pollution and congestion. To encourage more commuters to take public transport (e.g., buses), it is critical to make travelling by public transport a convenient choice for commuters. In this paper, we study bus frequency optimization with a focus on rescheduling bus frequencies to meet the actual travel demands, which 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[1]. If we re-organize the 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 past, there have been several studies on bus frequency optimization. Most of them share a common objective of minimizing the *average travel cost* (in term of waiting time) of passengers [1], [2], [3], [4], [5]. In addition, their solutions are usually heuristic rather than approximate algorithms with theoretical guarantees. However, most, if not all, existing studies ignore *user satisfaction*, a very important aspect. Meanwhile, the commuter satisfaction report has confirmed that the user satisfaction drops faster as the waiting time increases [6], [7], [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, which is an input that could be changed based on user or application needs.

To this end, we focus on studying the satisfaction-boosted bus scheduling. To be more specific, we propose the *SatisFAction-BooSTed Bus Scheduling (FAST)* problem and *SatisFAction-BooSTed Bus Scheduling under the COnstraint of Limited Vehicles (FASTCO)* to cater for two different scenarios.

Given a bus service database $\mathcal{B}$, a bus route database $\mathcal{R}$, a passenger database $\mathcal{P}$, and a waiting time threshold θ, FAST takes 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 as an input and it schedules n_i bus departures for each route $r_i \in \mathcal{R}$ such that the whole bus system is able to serve the most passengers within the threshold θ. FASTCO takes a terminal database $\mathcal{T}$ and a vector $\mathcal{N}\langle n_1, n_2, \cdots, n_i, \cdots, n_{|\mathcal{T}|}\rangle$ with n_i (≥ 1) denoting the initial number of buses parking at terminal t_i as inputs, and it arranges buses to serve the bus services under the condition that a bus parking at a terminal t_i could serve any route that starts from the terminal t_i such that the whole bus system is able to serve the most passengers within the threshold θ. In short, FAST assumes the number of bus departures for each bus route is fixed; FASTCO assumes the number of buses parking at each bus terminal is fixed but buses could be scheduled to serve different routes started from the same terminal. Our analysis shows that the objective functions of both FAST and FASTCO are submodular, and both FAST and FASTCO are 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

This work was done when Songsong Mo was a visiting student at RMIT.

- *S. Mo and Z. Peng (corresponding author) are with the school of computer science, Wuhan University, Wuhan 430000, Hubei, China. E-mail: {songsong945, peng}@whu.edu.cn*
- *Z. Bao is with the Computer Science and Information Technology, RMIT University, Melbourne, VIC 3000, Australia. E-mail: zhifeng.bao@rmit.edu.au*
- *B. Zheng is with Singapore Management University, Singapore, Singapore. E-mail: bhzheng@smu.edu.sg*

Manuscript received xx, 2020.

1. https://www.liveabout.com/bus-cost-to-purchase-and-operate\-2798845

(Greedy) as the baseline, which can provide $(1 - 1/e)$ approximation factor. Then, we propose two enhanced algorithms, namely PartGreedy and ProPartGreedy. PartGreedy is inspired by the fact that a bus network is designed to cover different parts of the city and avoid unnecessary overlapping among routes [9], [10], [11], [12]. It adopts a partitioning algorithm to divide the bus network into several disjoint partitions. Accordingly, it invokes a local greedy search within each partition to effectively reduce the computation cost of the original Greedy. 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 provides a trade-off between efficiency and effectiveness.

Next, to resolve the FASTCO problem, we design two greedy-based heuristic methods. First, we extend basic greedy method (GreedySel) to solve FASTCO as a baseline. GreedySel, in each greedy iteration, checks all the bus routes that can be served by currently available physical buses and chooses the route with the maximal marginal gain. GreedySel simply selects the buses based on the contribution to the objective function. However, it might schedule a bus to serve a bus route in a much later time slot, which significantly affects the utility of buses. In order to improve the utilization of physical buses, we present a composite score based greedy method (ScoreSel) to strike a balance between the idle time and the marginal gains of each bus.

In summary, we make the following contributions.

- We propose and study the FAST and FASTCO problems. To the best of our knowledge, this is the first study on bus frequency optimization that considers commuters' tolerance on waiting time. We prove that both FAST and FASTCO are NP-hard (Section 3).
- 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 two user-defined parameters (Sections 4-5).
- We design two greedy-based heuristic methods to solve the FASTCO problem effectively and efficiently (Section 6).
- We conduct extensive experiments on real-world bus routes 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 (Section 7).

This paper extends our previous preliminary study [13] by introducing a new problem called FASTCO, and presenting two greedy-based heuristic methods as the solutions, along with the corresponding experimental studies. In addition, we have also developed a system that implements the main ideas [14].

2 Related Work

In this section, we will review existing studies related to this paper and present the main difference between existing works and our study presented in this paper.

We cluster the literature into two categories based on the overall optimization goal, i.e., *travel time driven bus frequency optimization problem (Travel-BFO)* and *transfer time driven bus frequency optimization problem (Transfer-BFO)*. The former treats each ride as a new trip and aims to minimize the average/total travel time of passengers for either one bus route or a bus route network, based on passenger demands. Here, a commuting journey from the origin to the destination is different from a bus ride, as a commuting journey could involve multiple rides if transfers are required. The latter tries to minimize the total transfer time of the transfer passengers, who have to take more than one bus in order to travel from their origins to the destinations.

Travel-BFO. Here, the passenger demand is usually abstracted as an origin-destination (OD) pair. The model proposed in [1] 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 formulated as a nonconvex objective function with linear or convex constraints, and a heuristic method is proposed by refining a set of frequencies according to a descent strategy. In [2], it is modelled as a nonlinear bi-level problem: the upper level represents the planner who wants to ensure the minimum total travel time under fleet size constraints; the lower level represents the users who act by minimizing the travel time. In [3], 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). The salient characteristic of this work is the internalization of the congestion in the behavior of the users. The proposed heuristic solution starts with an initial set of frequencies, which is successively improved by a sensitivity analysis procedure. Martínez et al. [4] study the transit frequency optimization problem to determine the time interval between subsequent buses for a set of bus routes. They propose a mixed-integer linear programming (MILP) formulation for an existing bilevel model [2], and present a meta-heuristic method. A new model considering user behavior is proposed in [5]. 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 target bus route. In [15], [16], [17], they aim to plan some routes to minimize some objectives, such as the total travel time and the maximum flow time.

Differences. Although different bus frequency optimization models have been proposed, they share a very similar optimization goal, i.e., minimizing the average/total travel cost of passengers. Different from the above studies, we aim to improve the *overall passenger satisfaction* by scheduling the buses such that they can serve more passengers within a 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 consumer satisfaction surveys [6], [7], [8]. *Second*, the waiting time threshold is tunable, hence the bus company

TABLE 1: Notations for problem formulation and solutions

Symbol	Description
$\mathcal{R}$	A bus route database
$\mathcal{P}$	A passenger database
$\mathcal{B}$	A bus service database
$\mathcal{F}$	A bus service frequency
$\mathcal{T}$	A bus terminal database
$\mathcal{G}(\mathcal{F})$	The total number of passengers served by $\mathcal{F}$
θ	The waiting time threshold

can adjust thresholds to cater to various concerns on budget, government needs, passengers' tolerance of waiting, etc.

Transfer-BFO. The transfer time driven bus frequency optimization problem is an extension of single bus route scheduling. It determines the departure time of each trip of all the bus routes in the bus network with the consideration of passenger transfer activities at transfer stations [11]. This problem is modelled by mixed integer programming models to maximize the number of synchronized bus arrivals at transfer nodes [18]. Ibarra-Rojas et al. [19] extend [18] to address a flexible Transfer-BFO problem with almost evenly-spaced departures to prevent bus bunching. The model proposed in [20] tries to minimize the total transfer time experienced by passengers. Parbo et al. [21] study a bi-level bus timetabling problem to minimize the weighted transfer waiting time of passengers, and applies a Tabu Search algorithm to solve this bi-level problem. Recently, a non-linear mixed integer-programming model has been proposed to maximize the total number of transfer passengers with small excess transfer time [22].

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 waiting time threshold.

Remark. (1) Methodology-wise, all the above studies in both categories only propose heuristic methods without theoretical guarantees, although they have different problem settings. On the other hand, most of our approaches are approximate algorithms with non-trivial theoretical guarantees. (2) Scalability-wise, our approaches by design have superior scalability as they are proposed to handle city-scale datasets with hundreds of bus routes and millions of travel records; in contrast, most of the existing studies are not able to handle large datasets.

3 Problem Formulation

In this section, we first formally define FAST and FASTCO. Then we analyze the monotonicity and submodularity of the respective objective function and prove the NP-hardness for both problems. To facilitate our presentation, frequently used notations are listed in Table 1.

3.1 Problem Definition

In a bus route database $\mathcal{R}$, a route r is represented as a sequence of bus stations $(s_1, s_2, \cdots, s_i, \cdots, s_m)$, where each bus station s_i is represented by (latitude, longitude). In a passenger database $\mathcal{P}$, a passenger $p \in \mathcal{P}$ is in the form of a tuple $\{s_b, s_e, rt\}$, where s_b denotes the boarding station, s_e denotes the alighting station, and rt denotes the time when p reaches s_b. A bus service bs_{ij} is in the 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 3.1. We define that a bus service bs_{ij} can serve a passenger p, if r_i contains $p.s_b$ and $p.s_e$ in order, and $0 \leq dt_j + T(r_i.s_1, p.s_b) - rt \leq \theta$, where $T(r_i.s_1, p.s_b)$ denotes the travel time required by the bus service bs_{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 average historical 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 3.1, we formally introduce $\mathcal{S}(bs_{ij}, p_k)$ to denote the service of bs_{ij} to passenger p_k, as presented in Equation (1).

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

Next, we introduce the concept of bus service frequency in Definition 3.2.

Definition 3.2. A bus service frequency (f_i) for bus route r_i refers to a set of bus services (bs_{i1}, bs_{i2}, $\cdots$, bs_{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.

Given a bus route database $\mathcal{R}$, f_i represents the bus service frequency corresponding to a bus route $r_i \in \mathcal{R}$. Consequently, $\cup_{\forall r_i \in \mathcal{R}} f_i$ forms a set, denoted as the bus service frequency $\mathcal{F}$. Then, the bus services scheduled based on $\mathcal{F}$ to a passenger p_k can be computed by Equation (2). Note that $\mathcal{S}(\mathcal{F}, p_k) = 1$ as long as any $bs_{ij} \in \mathcal{F}$ can serve p_k; otherwise, $\mathcal{S}(\mathcal{F}, p_k) = 0$.

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

To this end, we proceed to formalize FAST and FASTCO in Definition 3.3 and 3.4, respectively, and show their NP-hardness in Section 3.2.

Definition 3.3 (SatisFAction-BooSTed Bus Scheduling (FAST)). Given a bus service database $\mathcal{B}$, 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 denotes the total number of bus departures of bus route $r_i \in \mathcal{R}$, FAST aims to find a bus service frequency $\mathcal{F}$ that can maximize

$$\mathcal{G}(\mathcal{F}) = \sum\nolimits_{p_k \in \mathcal{P}} \mathcal{S}(\mathcal{F}, p_k),$$

where $\mathcal{G}(\mathcal{F})$ denotes the total number of passengers served by $\mathcal{F}$.

Example 1. Assume we have a bus route database $\mathcal{R} = \{r_1, r_2\}$, a bus service database $\mathcal{B} = \{bs_{11}, bs_{12}, bs_{21}, bs_{22}, bs_{23}\}$, and a passenger database $\mathcal{P} = \{p_1, p_2, p_3, p_4\}$, as shown in Figure 1. Given a vector $\mathcal{N}\langle 1, 2\rangle$ that indicates the expected number of departures for bus routes, we need to select one bus service from $\{bs_{11}, bs_{12}\}$ and two bus services from $\{bs_{21}, bs_{22}, bs_{23}\}$ to $\mathcal{F}$ such that $\mathcal{G}(\mathcal{F})$ could be maximized according to Definition 3.3. In this example, we should output

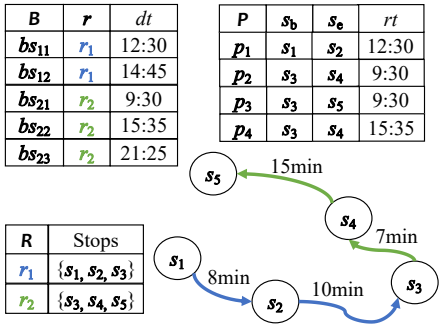


Fig. 1: An example of FAST

$\mathcal{F} = \{bs_{11}, bs_{21}, bs_{22}\}$, which can serve the maximum number of passengers ($\mathcal{G}(\mathcal{F}) = 4$), as the optimal result.

FAST focuses on the case where the number of departures for each bus route is fixed and known in advance. It tries to find the proper departure times for each route in order to serve the maximum number of passengers within the given waiting time threshold.

Apart from this case, another different, yet equally important, case is that, a certain scheduling might allow a bus vehicle to serve multiple bus routes. Taking Singapore as an example, many bus routes share common starting and/or ending locations (e.g., bus terminus or bus interchanges). For example, there are 24 routes starting from Bedok Bus Interchange and 31 routes starting from Jurong East Bus Interchange. Motivated by the above observation, we introduce the concept of *bus terminal*, which refers to the bus stations that normally serve as the starting station and/or terminating station of bus routes. With the help of bus terminals, we further assume that buses parking at a given bus terminal could be scheduled to serve any bus routes that starts from that bus terminal. Consequently, we consider the case where the number of bus vehicles is limited and study how to maximize the user satisfaction by fully utilizing the limited buses. To cater to the bus scheduling problem with limited number of buses constraint, we formulate FASTCO in Definition 3.4.

To be more specific, let $\mathcal{T} = \{t_1, t_2, \cdots, t_i, \cdots, t_{|\mathcal{T}|}\}$ denote the set of bus terminals (i.e., $\mathcal{T} = \cup_{\forall r_i \in \mathcal{R}}(r_i.s_1 \cup r_i.s_m)$). We further introduce $t_i.routes$ to represent the set of routes starting from terminal t_i, i.e., $\forall r_j \in t_i.routes$, $r_j.s_1 = t_i$, and $t_i.buses$ to denote the set of physical buses parking at terminal t_i. A physical bus b is represented by (id, at), where at denotes the time that b arrives at a terminal. Note that $t_i.buses$ changes its buses as buses are scheduled to serve different routes. For example, when a bus b is scheduled to serve a route r_i that is from one terminal t_a to another terminal t_b at time dt_j, bus b will be removed from $t_a.buses$ at dt_j to reflect the fact that b departs from the terminal t_a, and it will be included into $t_b.buses$ at time $dt_j + T(r_i.s_1, r_i.s_m)$ to reflect the fact that bus b arrives at terminal t_b at $dt_j + T(r_i.s_1, r_i.s_m)$. Then, we define the following problem to output a bus service frequency $\mathcal{F}$ that can maximize the total number of passengers served by $\mathcal{F}$ subject to the number of vehicle constraint.

Definition 3.4 (SatisFAction-BooSTed Bus Scheduling under the COnstraint of Limited Vehicles (FASTCO)). Given a bus service database $\mathcal{B}$, a bus route database $\mathcal{R}$, a passenger database $\mathcal{P}$, a terminal database $\mathcal{T}$, a waiting time threshold θ, and a vector $\mathcal{N}\langle n_1, n_2, \cdots, n_i, \cdots, n_{|\mathcal{T}|}\rangle$ where n_i (≥ 1) denotes the initial number of buses parking at terminal t_i, FASTCO aims to find a bus service frequency $\mathcal{F}$ that can maximize

$$\mathcal{G}(\mathcal{F}) = \sum\nolimits_{p_k \in \mathcal{P}} \mathcal{S}(\mathcal{F}, p_k),$$

under the condition that a bus parking at a terminal t_i could serve any route in $t_i.routes$. Here, $\mathcal{G}(\mathcal{F})$ denotes the total number of passengers served by $\mathcal{F}$.

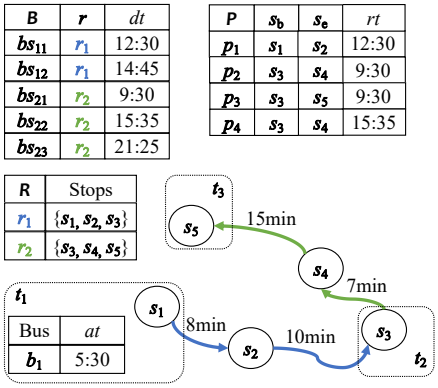


Fig. 2: An example of FASTCO

Example 2. Assume we have a bus route database $\mathcal{R} = \{r_1, r_2\}$, a bus service database $\mathcal{B} = \{bs_{11}, bs_{12}, bs_{21}, bs_{22}, bs_{23}\}$, a passenger database $\mathcal{P} = \{p_1, p_2, p_3, p_4\}$ (same as Example 1), and a terminal database $\mathcal{T} = \{t_1, t_2, t_3\}$, as shown in Figure 2. There is only one bus b_1, parking at terminal t_1, i.e., $t_1.buses = \{b_1\}$, $t_2.buses = t_3.buses = \emptyset$. According to Definition 3.4, we can first arrange b_1 to serve either bus service bs_{11} or bs_{12}, as $t_1.routes = \{r_1\}$, $b_1.at \leq bs_{11}.dt$ and $b_1.at \leq bs_{12}.dt$. Here $b_1.at$ denotes the arrival time of bus b_1. Once the selection is made (say bs_{11} is selected and added to $\mathcal{F}$), bus b_1 will reach t_2 at 12:48. As $t_2.routes = \{r_2\}$ and $b_1.at$ =12:48, bus b_1 could serve either bs_{22} or bs_{23}. Suppose bs_{22} is selected, $\mathcal{F} = \{bs_{11}, bs_{22}\}$ achieves the best performance by serving two passengers in total.

3.2 Problem Hardness

In this section, we proceed to conduct theoretical analysis on the hardness of FAST and FASTCO.

Theorem 3.1. 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 bus services, and b refers to a bus service in $\mathcal{B} \backslash T$. According to [23], $\mathcal{G}(V)$ 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:

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

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 bus services in V or bus services in $T \backslash V$ or bus service$b$, there are in total four cases corresponding to Inequality (4).

Case 1: p_k can be served by a bus service $b_0 \in V$. 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, $\mathcal{S}(V_b, p_k) - \mathcal{S}(V, p_k) - \mathcal{S}(T_b, p_k) + \mathcal{S}(T, p_k) = 0$.

Case 2: p_k cannot be served by any bus service $b_0 \in V$ but it can be served by a bus service $b_1 \in T \backslash V$. 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, $\mathcal{S}(V_b, p_k) - \mathcal{S}(V, p_k) - \mathcal{S}(T_b, p_k) + \mathcal{S}(T, p_k) \geq 0$.

Case 3: p_k cannot be served by any bus service $b_0 \in T$ and can be served by the bus service 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, $\mathcal{S}(V_b, p_k) - \mathcal{S}(V, p_k) - \mathcal{S}(T_b, p_k) + \mathcal{S}(T, p_k) = 0$.

Case 4: p_k cannot be served by any bus service $b_0 \in T$ or the bus service 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 3.2. The FAST problem is NP-hard.

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 served by a bus service $b \in \mathcal{B}$. 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 bus services). The Set Cover problem is equivalent to deciding if there is a k-bus service 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. ■

Theorem 3.3. The objective function $\mathcal{G}$ of FASTCO is monotone and submodular.

The proof of Theorem 3.3 is similar to the proof of Theorem 3.1. We omit it here for space saving.

Theorem 3.4. The FASTCO problem is NP-hard.

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

1.1 **Input:** a bus service 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 service $bs_{jl} \leftarrow \arg\max_{bs \in \mathcal{B} \backslash \mathcal{F}} (\mathcal{G}(\mathcal{F} \cup bs) - \mathcal{G}(\mathcal{F}))$
1.7 $k_j ++$
1.8 **if** $k_j \leq n_j$ **then**
1.9 $\mathcal{F} \leftarrow \mathcal{F} \cup bs_{jl}$
1.10 **if** $k_j \geq n_j$ **then**
1.11 remove all the bus services serving route r_j from $\mathcal{B}$
1.12 **return** $\mathcal{F}$

Proof. We prove the NP-hardness of FASTCO by reducing the Set Cover problem to FASTCO problem. The proof of Theorem 3.4 is similar to that of Theorem 3.2. The only difference is the way to set the number of subsets. In the case where all routes start from different terminals and no route starts from other routes' end terminals, the total number of physical buses in all terminals from which routes start is set to k (selecting k bus services). ■

4 Basic Greedy Method for FAST

To address FAST, we first present a basic greedy method. To accelerate the marginal gain computation, we propose a mapping structure to index the bus service and the passenger databases. The basic greedy method is guaranteed to achieve (1 - $1/e$)-approximation, as proved by Nemhauser et al. [23].

The pseudo-code of the greedy method is listed in Algorithm 1. In each iteration, it selects a bus service $bs_{jl} \in \mathcal{B} \backslash \mathcal{F}$ with the largest marginal gain, such that $bs_{jl} = \arg\max_{bs \in \mathcal{B} \backslash \mathcal{F}} (\mathcal{G}(\mathcal{F} \cup bs) - \mathcal{G}(\mathcal{F}))$, 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 r_j, which bs_{jl} serves, has reached the total number of bus departures required by this route. If so, it removes all bus services corresponding to the route r_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.

Time Complexity. In each iteration, Algorithm 1 needs to scan all the bus services 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 service into $\mathcal{F}$ takes $O(|\mathcal{P}| \cdot |\mathcal{B}|)$ time, and the total complexity is $O(n \cdot |\mathcal{P}| \cdot |\mathcal{B}|)$.

Index for Efficient Marginal Gain Computation. It is noticed that the computation of the marginal gain for scheduling one bus service to serve a bus route is the main bottleneck of Algorithm 1. To address this issue, we propose two mapping indexes, *forward list* and *inverted list*, as shown in Fig. 3 and Fig. 4 respectively. The former is for bus services $bs_i \in \mathcal{B}$, maintaining a list of passengers L_P that could be served by bus service bs_i. Note that a passenger

Bus Service List	$N_{ToBeServed}$	L_P
bs_1	3	$p_1, p_3, p_{\|\mathcal{P}\|}$
bs_2	2	p_1, p_2
bs_3	1	p_3
$\cdots$	$\cdots$	$\cdots$
$bs_{\|\mathcal{B}\|}$	1	p_2

Fig. 3: Forward list

Passenger List	$IsServed$	Optional Bus Services
p_1	$false$	bs_1, bs_2
p_2	$false$	$bs_2, bs_{\|\mathcal{B}\|}$
p_3	$false$	bs_1, bs_3
$\cdots$	$\cdots$	$\cdots$
$p_{\|\mathcal{P}\|}$	$false$	bs_1

Fig. 4: Inverted list

could be served by multiple bus services. To avoid counting the same passenger multiple times when calculating the marginal gain, we maintain another parameter $N_{ToBeServed}$ to capture the number of passengers in L_P that are still waiting for services. Given one bus service bs_i, the initial value of $N_{ToBeServed}$ is set to be the cardinality of the corresponding L_P, and its value will be reduced every time when a passenger in L_P is served by another bus service bs_j.

The latter is for passengers $p \in \mathcal{P}$, maintaining a list of bus services that could serve the passenger p. The boolean $IsServed$ associated with each passenger is to indicate whether any of the desired bus services has been scheduled and its initial is $false$ to reflect the fact that the passenger is waiting for buses. For example, if bus service bs_1 is selected, it could serve three passengers based on $N_{ToBeServed}$'s value associated with bs_1 in forward list. Meanwhile, $IsServed$'s value of passengers in L_P of bs_1 (i.e., $p_1, p_3, p_{|\mathcal{P}|}$) will be changed to $true$, and 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 Methods for FAST

As reported in the aforementioned time complexity analysis, the basic greedy suffers from low efficiency and poor scalability for FAST. For example, as to be reported in Figure 13(a), it cannot complete the scheduling within 3 hours when the number of departures required by each bus route reaches 300. Therefore, we further propose two efficient partition-based methods, inspired by [24], [25], both of which hold a theoretical guarantee on the approximation ratio. Moreover, they offer tunable parameters to cater to different degrees of tradeoff between efficiency and effectiveness of the solution.

5.1 Partition-based Greedy Method

In practice, a bus network is designed to cover different parts of a city in order to meet residents' various travel demands and to avoid unnecessary overlapping among routes [9], [10], [11], [26]. For example, Figure 5 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, because these routes have *zero* overlap. This observation suggests that it might not be necessary to scan the entire bus network when calculating the marginal gains of certain bus services. 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.

The main idea is to partition the bus routes (and bus services) into disjoint clusters, and use a divide-and-conquer strategy to find local optimal frequencies for 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. Note when $m = 1$, the time complexity of ProPartGreedy is the worst, which is the same as that of Greedy. The speedup is contributed by the fact that it invokes the greedy algorithm for each partition and hence it only needs to scan the bus services and passengers corresponding to the routes in a partition during the greedy search. Meanwhile, in terms of accuracy, we introduce a novel concept called *service overlap ratio* and use it to derive an approximation ratio with a non-trivial theoretical guarantee.

Definition 5.1 (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 $i \neq j$, $\mathcal{C}_i \cap \mathcal{C}_j = \phi$.

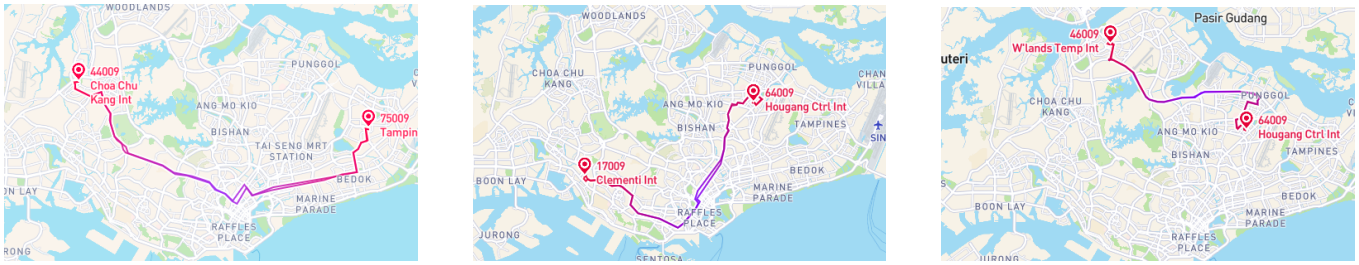
To better illustrate the service overlap ratio, we define a function Serve(P, R) that takes a passenger set P and a route set R as inputs and returns the passengers in P that could be served by any route in R without considering the temporal factor. To be more specific, a passenger p will be returned by 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 service serves passengers" defined in Definition 3.1. We name the set of passengers returned by Serve(P, R) as the passenger pool w.r.t. bus routes R.

Intuitively, 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 Greedy($\mathcal{C}_k^B, \mathcal{C}_k^R, \mathcal{P}, N_{min}$). $\mathcal{G}(\overline{\mathcal{F}}_k)$, and $\mathcal{C}_i^P$ refer to a cluster of bus services, 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 bus services required by any route. To this end, we define the service overlap ratio to quantify the overlaps between clusters.

Definition 5.2 (Service overlap ratio). Given a partition $\mathcal{C}^R = \{\mathcal{C}_1^R, \cdots, \mathcal{C}_m^R\}$ of the original bus route database $\mathcal{R}$, for a cluster $\mathcal{C}_i^R \in \mathcal{C}^R$, the ratio of the service overlap between $\mathcal{C}_i^R$ and the rest clusters is

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

Partitioning of bus routes and bus services. Algorithm 2 lists the pseudo-code of a bus route partitioning method, guided by service overlap ratio. It first partitions 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 2.9). It selects the



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

Fig. 5: Visualization of three popular bus routes in Singapore

Algorithm 2: BusRoutePartitioning $(\mathcal{B}, \mathcal{R}, n_{min}, \rho)$

2.1 **Input:** a bus database $\mathcal{B}$, a bus route database $\mathcal{R}$, an integer n_{min}, and a controlling threshold ρ
2.2 **Output:** a partition $\mathcal{C}^B$ of $\mathcal{B}$ and a partition $\mathcal{C}^R$ of $\mathcal{R}$
2.3 **for** *each bus route* $r_i \in Route$ **do**
2.4 initialize $\mathcal{C}_i^R \leftarrow \{r_i\}$, $\mathcal{C}_i^B \leftarrow \{b_{ab} \in \mathcal{B} | a = i\}$, $S_i \leftarrow \mathsf{Serve}(\mathcal{P}, Cluster_i^R)$
2.5 $\overline{\mathcal{F}}_i \leftarrow \mathrm{Greedy}(\mathcal{C}_\mathrm{i}^\mathrm{B}, \mathcal{C}_\mathrm{i}^\mathrm{R}, \mathcal{P}, \mathrm{N_{min}})$
2.6 initialize $\mathcal{C}^R \leftarrow \cup_{r_i \in \mathcal{R}} \mathcal{C}_i^R$
2.7 **for** $\mathcal{C}_i^R \in \mathcal{C}^R$ **do**
2.8 $\rho_i \leftarrow \left|\bigcup_{\mathcal{C}_j^R \in \mathcal{C}^R \setminus \mathcal{C}_i^R} S_i \cap S_j\right| / \mathcal{G}(\overline{\mathcal{F}}_i)$
2.9 $k \leftarrow \mathrm{argmax}_{\mathcal{C}_k^R \in \mathcal{C}^R} \rho_k$, $Max \leftarrow \rho_k$
2.10 **while** $Max > \rho$ **do**
2.11 $j \leftarrow \mathrm{argmax}_{\mathcal{C}_j^R \in \mathcal{C}^R \setminus \mathcal{C}_k^R} |(S_j \cap S_k)|$
2.12 $\mathcal{C}_k^R \leftarrow \mathcal{C}_k^R \cup \mathcal{C}_j^R$, $\mathcal{C}^R \leftarrow \mathcal{C}^R - \mathcal{C}_j^R$, $\mathcal{C}_k^B \leftarrow \mathcal{C}_k^B \cup \mathcal{C}_j^B$, $\mathcal{C}^B \leftarrow \mathcal{C}^B - \mathcal{C}_j^B$
2.13 $\mathcal{G}(\overline{\mathcal{F}}_k) \leftarrow max\{\mathcal{G}(\overline{\mathcal{F}}_k) + \mathcal{G}(\overline{\mathcal{F}}_j) - |S_k \cap S_j|, \mathcal{G}(\overline{\mathcal{F}}_k), \mathcal{G}(\overline{\mathcal{F}}_j)\}$
2.14 $S_k \leftarrow S_k \cup S_j$, $\rho_k \leftarrow \frac{\left|\bigcup_{\mathcal{C}_l^R \in \mathcal{C}^R \setminus \mathcal{C}_k^R} S_l \cap S_k\right|}{\mathcal{G}(\overline{\mathcal{F}}_k)}$
2.15 $k \leftarrow \mathrm{argmax}_{\mathcal{C}_k^R \in \mathcal{C}^R} \rho_k$, $Max \leftarrow \rho_k$
2.16 **return** $\mathcal{C}^B$, $\mathcal{C}^R$

Algorithm 3: PartGreedy $(\mathcal{B}, \mathcal{R}, \mathcal{P}, \mathcal{N}, \rho)$

3.1 **Input:** a bus service 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 ρ
3.2 **Output:** a bus service frequency $\mathcal{F}$
3.3 initialize $\mathcal{C}^R \leftarrow \phi$, $\mathcal{C}^B \leftarrow \phi$, $S_P \leftarrow \phi$, $n_{min} \leftarrow Min_{1 \le i \le |\mathcal{R}|} n_i$, $\mathcal{F} \leftarrow \phi$
3.4 $(\mathcal{C}^B, \mathcal{C}^R) \leftarrow \mathrm{BusRoutePartitioning}(\mathcal{B}, \mathcal{R}, \mathrm{n_{min}}, \rho)$
3.5 **for** *each cluster* $\mathcal{C}_i^R \in \mathcal{C}^R$ **do**
3.6 $S_P \leftarrow \mathsf{Serve}(\mathcal{P}, Cluster_i^R)$, $\mathcal{F} \leftarrow \mathcal{F} \cup \mathrm{Greedy}(\mathcal{C}_\mathrm{i}^\mathrm{B}, \mathcal{C}_\mathrm{i}^\mathrm{R}, \mathrm{S_P}, \mathcal{N})$
3.7 **return** $\mathcal{F}$

cluster $\mathcal{C}_j^R$ that shares the largest common passenger pool with $\mathcal{C}_k^R$(Line 2.11) and merges $\mathcal{C}_j^R$ with $\mathcal{C}_k^R$ (Lines 2.12 - 2.14). 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 5.2. 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 merge 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. We use this partition method instead of the existing ones to enable PartGreedy to get an approximation ratio, which is introduced in the following.

After the bus routes and bus services are partitioned, it invokes the basic greedy method (Section 4) to find the frequency for each cluster, and merges the local frequencies corresponding to $|\mathcal{C}^R|$ clusters as the final answer. We name this approach as PartGreedy. Its pseudo-code is shown in Algorithm 3 and its approximation ratio is analyzed in Lemma 5.1.

Lemma 5.1. 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 2, 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 $\rho \ge \rho_i$ for any $\mathcal{C}_i^R \in \mathcal{C}^R$. Recall in Section 3, the basic greedy is proved to achieve $(1-1/e)$-approximation. Therefore, we have $\mathcal{G}(\mathcal{F}_i) \ge (1-1/e)\mathcal{G}(\mathcal{O}_i)$. Because of the submodularity and monotonicity of $\mathcal{G}$, we have $\sum_{i=1}^m \mathcal{G}(\mathcal{O}_i) \ge \mathcal{G}(\mathcal{O})$ and $\mathcal{G}(\mathcal{F}_i) \ge \mathcal{G}(\overline{\mathcal{F}}_i)$. Then, by Definition 5.2 we have:

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

In addition, Inequality (6) holds according to Definition 3.3.

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

Based on Inequality (5) and Inequality (6), we have

$$\mathcal{G}(\mathcal{F}^*) - \mathcal{G}(\mathcal{F}^* \setminus \mathcal{F}_i) \ge (1-\rho)\mathcal{G}(\mathcal{F}_i).$$

Using the principle of inclusion-exclusion, we have

$$\begin{aligned}\mathcal{G}(\mathcal{F}^*) &= \mathcal{G}(\mathcal{F}_1 \cup \mathcal{F}_2 \cup \ldots \cup \mathcal{F}_m) \\ &\ge \sum_{i=1}^m (\mathcal{G}(\mathcal{F}^*) - \mathcal{G}(\mathcal{F} \setminus \mathcal{F}_i)) \\ &\ge (1-\rho) \sum_{i=1}^m \mathcal{G}(\mathcal{F}_i) \\ &\ge (1-\rho)(1-1/e) \sum_{i=1}^m \mathcal{G}(\mathcal{O}_i) \\ &\ge (1-\rho)(1-1/e)\mathcal{G}(\mathcal{O}).\end{aligned}$$

Thus, this lemma is proved. ■

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

1.1 **Input:** a bus service database $\mathcal{B}$, a bus route database $\mathcal{R}$, a vector $\mathcal{N}$, and a parameter ε
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 $bs \in \mathcal{B}$ based on descending order of $\mathcal{G}(bs)$
1.6 Initialize $h \leftarrow \max_{bs \in \mathcal{B}}(\mathcal{G}(bs))$
1.7 **while** $|\mathcal{F}| \leq n$ **do**
1.8 **for** *each* $bs_{jl} \in \mathcal{B}$ **do**
1.9 **if** $|\mathcal{F}| \leq n$ **then**
1.10 $\mathcal{G}_{b_{jl}}(\mathcal{F}) \leftarrow \mathcal{G}(\mathcal{F} \cup bs_{jl}) - \mathcal{G}(\mathcal{F})$
1.11 **if** $\mathcal{G}_{b_{jl}}(\mathcal{F}) \geq h$ **then**
1.12 $\mathcal{F} \leftarrow \mathcal{F} \cup bs_{jl}$, $\mathcal{B} \leftarrow \mathcal{B} \backslash bs_{jl}$
1.13 k_j++
1.14 **if** $k_j \geq n_j$ **then**
1.15 remove all bus services serving 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** $\mathcal{F}$

5.2 Progressive Partition-based Greedy Method

Although PartGreedy improves the efficiency of basic greedy by conducting the search within each partition, 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 $\mathcal{G}(\mathcal{F} \cup bs) - \mathcal{G}(\mathcal{F})$ for all the bus services not yet scheduled.

Motivated by this observation, we propose a progressive partition-based greedy method (ProPartGreedy). It selects multiple, but not only one, bus services 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 3 except that the call of Greedy is replaced with Function 1 (ProGreedy) in line 3.6 of Algorithm 3. Meanwhile, we will prove that it can achieve 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 bus services $bs \in \mathcal{B}$ based on descending order of $\mathcal{G}(bs)$ and initializes the threshold h to the value of $\max_{bs \in \mathcal{B}}(\mathcal{G}(bs))$. Then, it iteratively fetches all the bus services 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 bus services in $\mathcal{F}$.

Unlike the basic greedy that has to check all potential bus services in $\mathcal{B}$ or a cluster of $\mathcal{B}$ in each iteration, brute-force checking is not necessary for ProGreedy as it implements an early termination (Lines 1.16-1.17). Since bus services are sorted by their $\mathcal{G}(bs)$ values, if $\mathcal{G}(bs_{jl})$ of the current bus service is smaller than h, all the bus services bs pending for evaluation will have their $\mathcal{G}(bs)$ values smaller than h and hence could be skipped from evaluation. It is worth noting that the efficiency of ProGreedy may be very low, or even lower than that of Greedy, when ε is close to 0. In the following, we first analyze the approximation ratio of Function 1 in Lemma 5.2. Based on Lemma 5.2, we show the approximation ratio of ProPartGreedy in Lemma 5.3.

Lemma 5.2. ProGreedy achieves a $(1-1/e-\varepsilon)$ approximation ratio.

Proof. Let b_i be the bus service selected at a given threshold h and $\mathcal{O}$ denote the optimal local solution to the problem of selecting n bus services 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 $\mathcal{G}_{bs_i}(\mathcal{F}) \geq \mathcal{G}_{bs}(\mathcal{F})/(1+\varepsilon)$ for any $bs \in \mathcal{O} \backslash \mathcal{F}$. Thus, we have

$$\begin{aligned} \mathcal{G}_{bs_i}(\mathcal{F}) &\geq \frac{1}{(1+\varepsilon)|\mathcal{O} \backslash \mathcal{F}|} \sum\nolimits_{bs \in \mathcal{O} \backslash \mathcal{F}} \mathcal{G}_{bs}(\mathcal{F}) \\ &\geq \frac{1}{(1+\varepsilon)n} \sum\nolimits_{bs \in \mathcal{O} \backslash \mathcal{F}} \mathcal{G}_{bs}(\mathcal{F}). \end{aligned}$$

Let $\mathcal{F}_i$ denote the partial solution that bs_i has been included and bs_{i+1} be the bus service selected at the $(i+1)$th step. Then, we have

$$\begin{aligned} \mathcal{G}(\mathcal{F}_{i+1}) - \mathcal{G}(\mathcal{F}_i) &= \mathcal{G}_{b_{i+1}}(\mathcal{F}_i) \\ &\geq \frac{1}{(1+\varepsilon)n} \sum\nolimits_{bs \in \mathcal{O} \backslash \mathcal{F}_i} \mathcal{G}_{bs}(\mathcal{F}_i) \\ &\geq \frac{1}{(1+\varepsilon)n} (\mathcal{G}(\mathcal{O} \cup \mathcal{F}_i) - \mathcal{G}(\mathcal{F}_i)) \\ &\geq \frac{1}{(1+\varepsilon)n} (\mathcal{G}(\mathcal{O}) - \mathcal{G}(\mathcal{F}_i)). \end{aligned}$$

The solution $\mathcal{F}^*$ is obtained by Function 1 with $|\mathcal{F}^*| = n$. Using the geometric series formula, we have

$$\begin{aligned} \mathcal{G}(\mathcal{F}^*) &\geq \left(1 - \left(1 - \frac{1}{(1+\varepsilon)n}\right)^n\right) \mathcal{G}(\mathcal{O}) \\ &\geq \left(1 - e^{\frac{-n}{(1+\varepsilon)n}}\right) \mathcal{G}(\mathcal{O}) \\ &= \left(1 - e^{\frac{-1}{(1+\varepsilon)}}\right) \mathcal{G}(\mathcal{O}) \\ &\geq ((1-1/e-\varepsilon)) \mathcal{G}(\mathcal{O}). \end{aligned}$$

Hence, the lemma is proved. ■

Lemma 5.3. 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 5.2, this proof is similar to the proof of Lemma 5.1, so we omit it to save space. ■

6 Greedy-based Heuristic Methods for FASTCO

Although the greedy method is able to support FASTCO, it is not able to guarantee $(1-1/e)$-approximation, because of the constraint of limited number of vehicles and the reusability of vehicles to serve different bus routes, as shown in the following example.

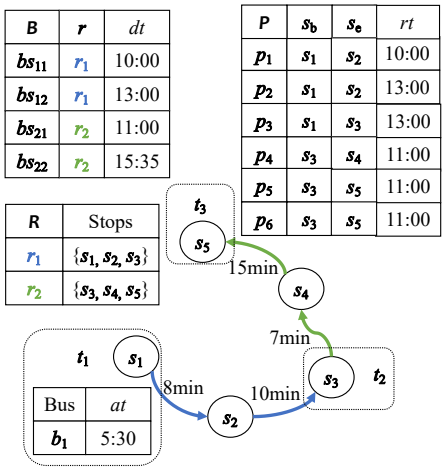


Fig. 6: A counterexample

Example 3. As shown in Figure 6, the greedy strategy will schedule b_1 that is currently parking at terminal t_1 to serve bus route r_1 at 13:00 (select bs_{12} to $\mathcal{F}$), as it is able to serve the most passengers ($\{p_2, p_3\}$). Thereafter, b_1 will reach t_2 at 13:18 and it could not serve any other passengers. In other words, the total number of passengers served by the bus service frequency output by greedy method is 2. On the other hand, we could schedule b_1 to serve route r_1 at 10:00 (select bs_{11} to $\mathcal{F}$), which allows it to serve one passenger ($\{p_1\}$). Then, b_1 reaches t_2 at 10:18 and is available to serve bus route r_2 at 11:00 (select bs_{21} to $\mathcal{F}$). This schedule allows three more passengers ($\{p_4, p_5, p_6\}$) to be served. In total, this frequency serves 4 passengers. Obviously, $2/4 = 0.5 < (1-1/e) \approx 0.63$. We could conclude that the greedy method can not achieve $(1-1/e)$-approximation when supporting FASTCO.

Therefore, we decide to devise heuristic methods for FASTCO. In this section, we first the extend basic greedy method (namely GreedySel) to solve FASTCO as a baseline; we then present a composite score based greedy method (namely ScoreSel) to trade off the bus's idle time and marginal gains of each bus service.

6.1 Basic Greedy Method

Algorithm 4 shows the pseudo-code of the basic greedy-based heuristic method. It first performs initializations (Lines 4.3-4.6). For example, it initializes the arrival time of all the buses in $t.buses$ for all the terminals $t \in \mathcal{T}$ to τ_s, the starting timestamp of all the bus services in a given city (e.g., 5am in our experimental study); it initializes a temporary bus service database $\mathcal{B}_0$ by including all the bus services in $\mathcal{B}$ with **Available**(bs)$\neq \emptyset$. Function **Available**(bs) takes a bus service bs as an input and returns whether a physical bus is available to provide the service. To be more specific, given a bus service $bs_{ij}(r_i, dt_j)$ to be served in the near future, if there is at least one bus parking at terminal $r_i.s_1$, it returns the bus that reaches terminal $r_i.s_1$ the earliest; otherwise, it returns an empty set. In other words, given a bus service bs_{ij} (r_i, dt_j), if $\exists b \in t_x.buses$ such that $b.at \leq dt_j \wedge t_x = r_i.s_1 \wedge \forall b' \in t_x.buses$, $b'.at \geq b.at$, **Available**(bs) returns b.

It then starts iterations until the temporary bus service database $\mathcal{B}_0$ becomes empty. In each iteration, it selects the bus service (say bs_{ij}) in $\mathcal{B}_0$ with the largest marginal gain and inserts it into the current service frequency $\mathcal{F}$ (Lines 4.8 - 4.9). The physical bus bus_k that is returned by **Available**(bs_{ij}) will be scheduled to serve bs_{ij}. We then remove the bus bus_k from the departure terminal t_x and add it to the arrival terminal t_y (Lines 4.10 - 4.14). Finally, we invoke function **Update**($\mathcal{B}_0$, bus_k) to update the temporary bus service database $\mathcal{B}_0$ because of the dispatch of bus_k (Line 4.15). Note that there is no need to re-form $\mathcal{B}_0$ from scratch, since the only change is that bus_k is going to depart from its original terminal t_x and head towards the terminal t_y. We only need to invoke **Available**(bs) for i) all the services bs' with the initial return of **Available**(bs') being bus_k and ii) all the services $bs(r_a, dt_b) \in \mathcal{B} - \mathcal{B}_0$ with $r_a.s_1$ being t_y and $dt_b \geq bus_k.at$. Finally, it returns $\mathcal{F}$ as the solution.

Algorithm 4: GreedySel $(\mathcal{B}, \mathcal{R}, \mathcal{P}, \mathcal{T}, \mathcal{N})$

4.1 **Input:** a bus service database $\mathcal{B}$, a bus route database $\mathcal{R}$, a passenger database $\mathcal{P}$, a terminal database $\mathcal{T}$, a vector $\mathcal{N}$ $\langle n_1, n_2, \cdots, n_i \rangle$, and two constants τ_s, τ_e that indicate the starting time and end time of all the bus services in a given city respectively
4.2 **Output:** a bus service frequency $\mathcal{F}$
4.3 Initialize $\mathcal{F} \leftarrow \phi$, $\mathcal{B}_0 \leftarrow \emptyset$
4.4 $\forall t \in \mathcal{T}, \forall b \in t.buses$, initialize $b.at \leftarrow \tau_s$
4.5 **for** *each* $bs \in \mathcal{B}$ **do**
4.6 $\quad \mathcal{B}_0 \leftarrow \mathcal{B}_0 \cup \{bs\}$ if (**Available**(bs)$\neq \emptyset$)
4.7 **while** $\mathcal{B}_0 \neq \emptyset$ **do**
4.8 $\quad$ Select a bus service $bs_{ij} \leftarrow \arg\max_{bs \in \mathcal{B}_0 \setminus \mathcal{F}} (\mathcal{G}(\mathcal{F} \cup bs) - \mathcal{G}(\mathcal{F}))$
4.9 $\quad \mathcal{F} \leftarrow \mathcal{F} \cup \{bs_{ij}\}$
4.10 $\quad b_k \leftarrow$ **Available**(bs)
4.11 $\quad$ Remove b_k from $t_x.buses$ with $t_x \leftarrow r_i.r_1$
4.12 $\quad$ **if** $dt_j + T(r_i.s_1, r_i.s_m) \leq \tau_e$ **then**
4.13 $\quad\quad b_k.at \leftarrow dt_j + T(r_i.s_1, r_i.s_m)$
4.14 $\quad\quad$ Add b_k into $t_y.buses$ with $t_y \leftarrow r_i.r_m$
4.15 $\quad \mathcal{B}_0 \leftarrow$ **Update**($\mathcal{B}_0$, b_k)
4.16 **return** $\mathcal{F}$

Time Complexity. In each iteration, Algorithm 4 needs to scan all the bus services in $\mathcal{B}_0 \setminus \mathcal{F}$ and compute their marginal gain to the chosen set. In the worst case, it needs to scan $|\mathcal{B}|$ bus services. Each marginal gain computation needs to traverse $\mathcal{P}$ once in the worst case. Thus, adding one bus service into $\mathcal{F}$ takes $O(|\mathcal{P}| \cdot |\mathcal{B}|)$ time. Let $k = \sum_{t_i \in \mathcal{T}} |t_i.buses|$ denote the total number of bus vehicles, and let T_r and T_t denote the time length of service time range and the shortest route transfer time, respectively. Then, the maximum value of $|\mathcal{F}|$ is $(T_r/T_t) \cdot k$ and the total complexity is $O((T_r/T_t) \cdot k \cdot |\mathcal{P}| \cdot |\mathcal{B}|)$.

6.2 Composite Score Based Greedy Method

GreedySel (i.e., Algorithm 4) simply selects the bus service with maximum marginal gain in each iteration, without considering the idle time of a bus. Here, the idle time refers to the duration between the timestamp a bus reaches its terminal and the timestamp it departs from its terminal to serve a bus route. Since it only considers the marginal gain, it is possible that a bus reaching a terminal in the early morning is scheduled to serve a bus service in the afternoon during the rush hour as it is expected to achieve the highest marginal gain. However, it does not consider the

fact that this schedule forces a bus to stay in the terminal for hours and hence causes unnecessary waste of bus resources. Consequently, the bus frequency returned by Algorithm 4 is not able to guarantee a high utility rate of buses. In order to avoid the case that certain buses are forced to stay in terminals for long hours because they are scheduled to serve bus services in much later timestamps, we introduce a novel concept called *composite score*, which is designed to trade off between the bus idle time and the marginal gain achieved by a bus service.

Accordingly, we propose a new algorithm, namely *Greedy Scheduling by Composite Score* (in short ScoreSel), based on composite score. It shares the same procedure as Algorithm 4, but adopts a different strategy when scheduling bus services. To be more specific, in each iteration, instead of selecting the bus service with the maximum marginal gain, it selects the service with the maximum composite score. In other words, the selection criterion in Line 4.8 of Algorithm 4 will be replaced by $bs_{ij} \leftarrow \arg\max_{bs \in \mathcal{B}_0 \setminus \mathcal{F}}(CoScore(bs_{ij}, \mathcal{F}, \lambda)$. Here, $CoScore$ stands for the composite score and notations $\mathcal{F}$ and λ refer to the current bus schedule and a controlling parameter (that adjusts the relative importance of bus idle time and the marginal gain of a bus service), respectively. Due to the similarity between GreedySel and ScoreSel, we skip the pseudo code of ScoreSel.

In the following, we first explain how to compute composite scores for each bus service bs; we then analyze the time complexity of the composite score based greedy method.

Composite Score. We introduce $I_t(bs_{ij})$ to indicate the bus idle time, which refers to the duration between the time when a bus reaches a terminal (i.e., $b.at$) and the time when the bus is scheduled to leave the terminal to serve a bus service bs_{ij}. In particular, let b_k be the output of **Service**(bs_{ij}). Then, $I_t(bs_{ij}) = dt_j - b_k.at$. Recall that in Algorithm 4, we only scan the bus services that are in the temporary bus service database $\mathcal{B}_0$. The fact that a bus service $bs_{ij} \in \mathcal{B}_0$ guarantees that **Service**(bs_{ij})$\neq \emptyset$, i.e., the services bs_{ab} with **Service**(bs_{ab})$= \emptyset$ are not considered here. Given the fact that the departure time dt_j of a bus service is fixed, $I_t(bs_{ij})$ reflects the idle duration of a bus. A smaller $I_t(bs_{ij})$ is preferred in order to achieve a higher utility rate. On the other hand, we use $g(bs_{ij}, \mathcal{F})$ $(= \mathcal{G}(\mathcal{F} \cup bs_{ij}) - \mathcal{G}(\mathcal{F}))$ to represent the marginal gain achieved if the bus service bs_{ij} is scheduled. Based on these two notations, we formally define *composite score* of a bus service in Definition 6.1. Thresholds θ_i and θ_g are for normalization purpose such that both $\frac{I_t(bs_{ij})}{\theta_i}$ and $\frac{g(bs_{ij},\mathcal{F})}{\theta_g}$ are in the range of $[0, 1]$.

Definition 6.1. Given a bus service bs_{ij}, a bus service frequency $\mathcal{F}$, an idle time threshold θ_i and a marginal gain threshold θ_g, the composite score corresponding to the bus service bs_{ij} is defined as follows:

$$CoScore(bs_{ij}, \mathcal{F}, \lambda) = \lambda(1 - \frac{I_t(bs_{ij})}{\theta_i}) + (1-\lambda)\frac{g(bs_{ij}, \mathcal{F})}{\theta_g},$$

where parameter $\lambda \in [0, 1]$ controls the relative importance of the idle time and the marginal gain.

Time Complexity. Let $k = \sum_{t_i \in \mathcal{T}} |t_i.buses|$ denote the total number of bus vehicles. In each iteration, ScoreSel needs to scan all the bus services in $\mathcal{B} \setminus \mathcal{F}$ and compute their composite scores. Each composite score computation needs to traverse $\mathcal{P}$ and k physical buses once in the worst case. Therefore, adding one bus service into $\mathcal{F}$ takes $O((k + |\mathcal{P}|) \cdot |\mathcal{B}|)$ time. In practice, $\mathcal{P}$ is much larger than k. Thus, we update the time complexity of adding one bus service into $\mathcal{F}$ to $O(|\mathcal{P}| \cdot |\mathcal{B}|)$. We use T_r and T_t to denote the time length of service time range and the shortest route transfer time, respectively. Then, the maximum value of $|\mathcal{F}|$ is $(T_r/T_t) \cdot k$ and the worst time complexity is $O((T_r/T_t) \cdot k \cdot |\mathcal{P}| \cdot |\mathcal{B}|)$.

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.

7.1 Experimental setup

Datasets. We collect the terminal and interchange information ($\mathcal{T}$) from Land Transport GURU[2] in Singapore. We crawl the real bus routes ($\mathcal{R}$) from Transitlink[3] 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 route.

For the passenger database ($\mathcal{P}$), due to the exhibit regular travel patterns of passengers [27], we use the real bus touch-on record data in a week of April 2016 in Singapore that 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 service candidate set ($\mathcal{B}$) based on the route and service time range. For each route, we use buses that depart every minute between 5am and 12am as the superset of candidate bus services. The statistics of those datasets are shown in Table 3.

Parameters. Table 2 lists the parameter settings, with values in bold being default. In all the 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.

Algorithms. To the best of our knowledge, this is the first work to study the FAST and FASTCO problems, and thus no previous work is available for direct comparison. In particular, we compare the following five methods for FAST.

2. https://landtransportguru.net/bus-infrastructure/bus-interchanges-and-terminals/
3. https://www.transitlink.com.sg/eservice/eguide/service_idx.php

TABLE 2: Parameter settings

Parameter	Values
number of bus departures $\mathcal{N} = \langle n_1, n_2, \cdots \rangle$	$\langle 10 \rangle$, $\langle 20 \rangle$, $\langle \mathbf{30} \rangle$, $\langle 40 \rangle$, $\langle 50 \rangle$
number of vehicles $\mathcal{N} = \langle n_1, n_2, \cdots \rangle$	$\langle 20 \rangle$, $\langle 40 \rangle$, $\langle \mathbf{60} \rangle$, $\langle 80 \rangle$, $\langle 100 \rangle$
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
relative importance used by ScoreSel λ	0, 0.1, **0.2**, 0.3, 0.4, 0.5

TABLE 3: Statistics of datasets

Database	Amount	AvgDistance	AvgTravelTime
$\mathcal{T}$	12	N.A.	N.A.
$\mathcal{B}$	451k	N.A.	N.A.
$\mathcal{R}$	396	19.91km	5159s
$\mathcal{P}$	28m	4.2km	1342s

- FixInterval that fixes the time interval between two bus departures as $\lfloor$(service time range) / (bus number)$\rfloor$ for each service 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, the three algorithms proposed in this paper.

For the FASTCO problem, we compare two heuristic methods, GreedySel and ScoreSel, 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 bus services (i.e., $\mathcal{G}(\mathcal{F})$) as the main performance metrics. We randomly choose the trips (in total 28 million) made by 5 million passengers within a week 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 algorithms 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.

7.2 Parameter Sensitivity Test

The first set of experiments is to evaluate the sensitivity of different parameters on different algorithms, including waiting time threshold θ, controlling threshold ρ that decides the partitioning of bus routes/bus services, controlling threshold ε that determines how fast the threshold h required by ProPartGreedy reduces its value, and parameter λ that controls the relative importance of the bus idle time and the marginal gain achieved by a bus service when calculating the composite score. Note parameter θ affects both FAST and FASTCO, parameters ρ and ε affect FAST only, and parameter λ affects FASTCO only. Their impacts are detailed below.

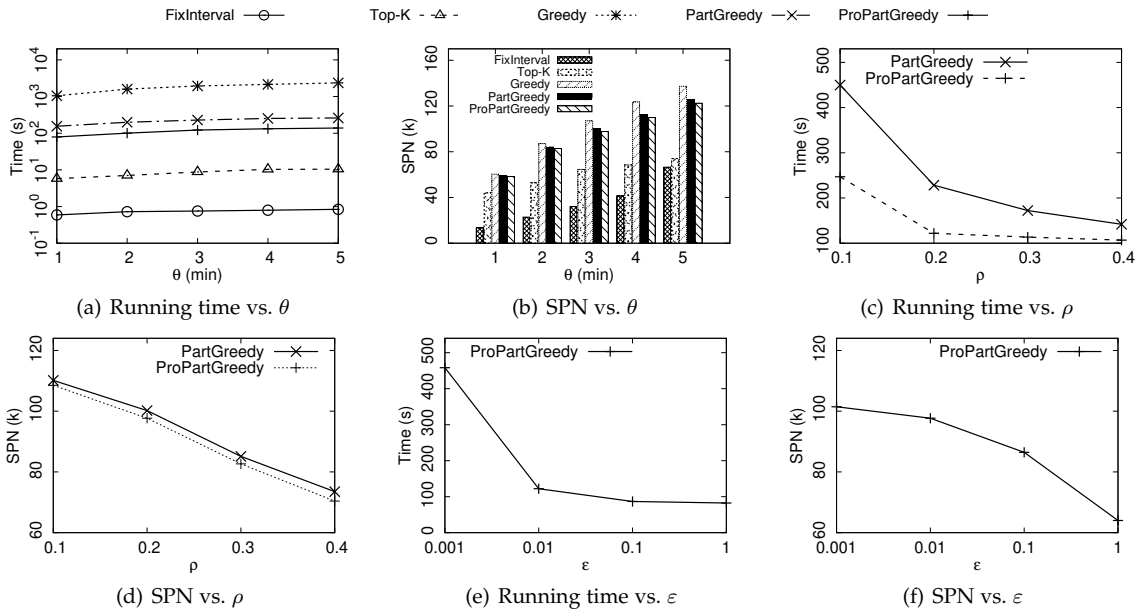
The impact of θ on FAST. The impact of waiting time threshold θ on the running time and SPN of the algorithms for FAST is reported in Figure 7(a) and Figure 7(b), respectively. Parameter θ has an almost-zero impact on the running time. This is because the value of θ only affects the value of marginal gain $\mathcal{G}(\mathcal{F})$ but not the computation cost of $\mathcal{G}(\mathcal{F})$. This also explains why θ affects SPN. As θ increases, all the algorithms are able to serve more passengers, which is consistent with our expectations. In the following experiments, we set $\theta = 3$.

The impact of ρ on FAST. The impact of parameter ρ on the running time and SPN is reported in Figure 7(c) and Figure 7(d), 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 passengers. We choose $\rho = 0.2$ as the default setting.

The impact of ε on FAST. 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 passengers, as reported in Figure 7(e) and Figure 7(f), respectively. We choose $\varepsilon = 0.01$ as the default setting.

The impact of θ on FASTCO. The impact of waiting time threshold θ on the running time and SPN of the algorithms for FASTCO is reported in Figure 8(a) and Figure 8(b), respectively. Parameter θ has a negative impact on the running time. This is because the value of θ directly affects the size of the temporary bus service database $\mathcal{B}_0$ and the size of $\mathcal{B}_0$ determines the total number of iterations algorithms GreedySel and ScoreSel (for FASTCO) have to perform. As θ becomes larger, $\mathcal{B}_0$ contains more bus services and hence both GreedySel and ScoreSel incur longer running time. On the other hand, as θ increases its value, both algorithms are able to serve more passengers, which is consistent with our expectations. We set $\theta = 3$, the mean value.

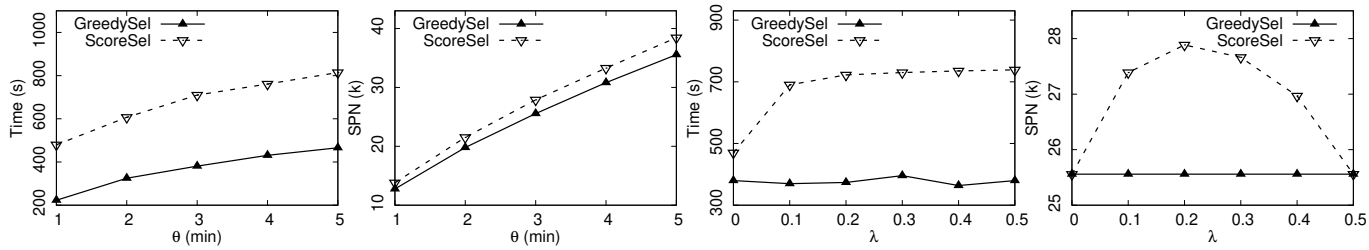
The impact of λ on FASTCO. Parameter λ only affects ScoreSel. It controls the relative importance of the bus idle time and marginal gain achieved by a bus service. It has a negative impact on the running time performance, as shown in Figure 8(c). As λ increases, bus idle time plays a more important role in composite scores. More buses will be scheduled to serve bus services in the near future, which increases the scheduling overhead. On the other hand, as λ increases its value, the effectiveness of ScoreSel increases first and then decreases, as reported in Figure 8(d). This is because as λ increases its value from 0 to 0.2, the composite score starts considering the impact of bus idle time to avoid scheduling buses to serve bus services that are much later. The improved bus utility actually has a positive impact on the number of served passengers. However, as λ further increases its value, the impact of bus idle time is overly emphasized which affects the marginal gain. We choose $\varepsilon = 0.2$ as the default setting, because it achieves high SPN with a reasonable running time.



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

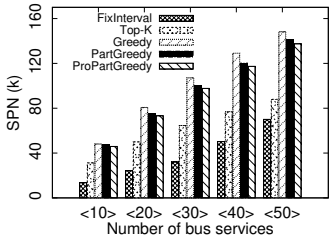
(d) SPN vs. ρ (e) Running time vs. ε (f) SPN vs. ε

Fig. 7: Effect of parameters for FAST

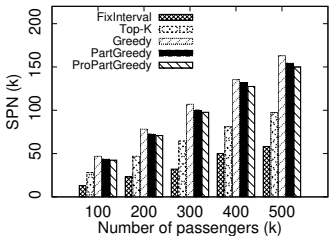


(a) Running time vs. θ (b) SPN vs. θ (c) Running time vs. λ (d) SPN vs. λ

Fig. 8: Effect of parameters for FASTCO

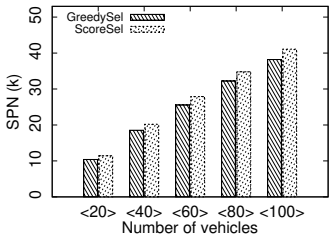


(a) SPN vs. $\mathcal{N}$

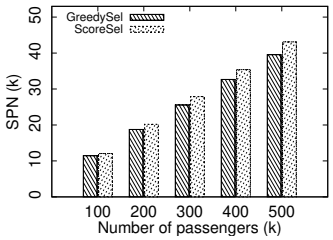


(b) SPN vs. $|\mathcal{P}|$

Fig. 9: Effectiveness study for FAST: SPN vs. $\mathcal{N}$ or $|\mathcal{P}|$



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



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

Fig. 10: Effectiveness study for FASTCO: SPN vs. $\mathcal{N}$ or $|\mathcal{P}|$

7.3 Effectiveness Study

The second set of experiments is to evaluate how effective the proposed algorithms are. As mentioned previously, both FAST and FASTCO are NP-hard, and all the algorithms proposed in this paper provide approximate solutions. SPN demonstrates their effectiveness.

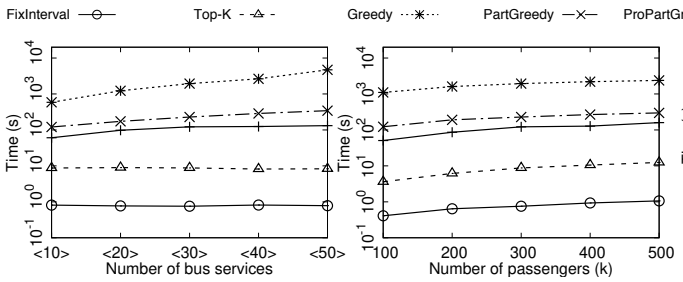
Effectiveness Study for FAST. We report the effectiveness of different algorithms in Figure 9. We observe that (1) FixInterval is significantly less effective than others; (2) the three algorithms proposed in this work perform much better than the other two, e.g., ProPartGreedy doubles (or even triples 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% less than that of Greedy).

Effectiveness Study for FASTCO. Figure 10 reports the effectiveness of ScoreSel and GreedySel for FASTCO, under different $\mathcal{N}$ or $|\mathcal{P}|$ settings. We observe that ScoreSel outperforms GreedySel by at least 5.2%.

7.4 Efficiency Study

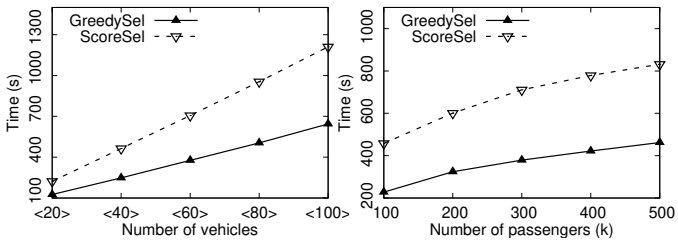
The third set of experiments is to evaluate the efficiency of different algorithms.

Efficiency Study for FAST. Figure 11 reports the running time of five algorithms under varying $\mathcal{N}$ or $|\mathcal{P}|$. We have two main observations. First, the time gap among Greedy, PartGreedy and ProPartGreedy becomes more significant with the increase of $\mathcal{N}$. A possible reason is that 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 bus services.



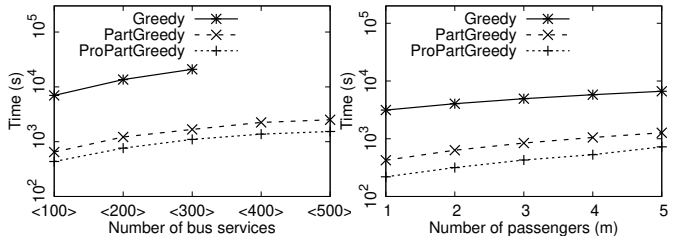
(a) Running time vs. $\mathcal{N}$ (b) Running time vs. $|\mathcal{P}|$

Fig. 11: Efficiency study for FAST: Total running time vs. $\mathcal{N}$ or $|\mathcal{P}|$



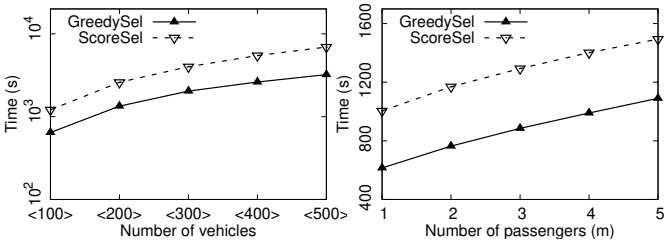
(a) Running time vs. $\mathcal{N}$ (b) Running time vs. $|\mathcal{P}|$

Fig. 12: Efficiency study for FASTCO: Total running time vs. $\mathcal{N}$ or $|\mathcal{P}|$



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

Fig. 13: Scalability study for FASTCO



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

Fig. 14: Scalability study for FASTCO

Second, the improvement of PartGreedy and ProPartGreedy over Greedy decreases with the increase of $|\mathcal{P}|$. This is because 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.

Efficiency Study for FASTCO. We report the efficiency of GreedySel and ScoreSel in Figure 12. Again, we have made two main observations. First, the running time of both algorithms increases almost linearly w.r.t. $\mathcal{N}$ and $|\mathcal{P}|$, which is consistent with the time complexity analysis conducted in Section 6. Second, the time gap between GreedySel and ScoreSel becomes more significant with the increase of $\mathcal{N}$. This is because, compared to GreedySel, ScoreSel takes the utility of physical buses into consideration and enables each physical bus to serve more bus services.

7.5 Scalability Study

Our last set of experiments is to evaluate the scalability of the algorithms proposed in this paper. Ideally, we would like all the algorithms to have good scalability and are able to provide approximate solutions even when the number of passengers and the number of bus services increase significantly. In the following study, we change $\mathcal{N}$ from $\langle 100 \rangle$ to $\langle 500 \rangle$ to demonstrate the increase of bus service frequency, and change $|\mathcal{P}|$ from 1 million to 5 million to simulate the increase of passengers to be served.

Scalability Study for FAST. As reported in Figure 13(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 we skip the results of Greedy when it cannot terminate within 10^4 seconds. This is because both PartGreedy and ProPartGreedy adopt partitioning strategy, which helps to limit the side effect of $\mathcal{N}$ on the running time incurred. Similarly, as the increases of passengers, all the algorithms require longer running time because the computation cost of marginal gains increases. As reported in Figure 13(b), PartGreedy and ProPartGreedy are about ten times faster than Greedy under different $|\mathcal{P}|$ settings.

Scalability Study for FASTCO. As shown in Figure 14, both GreedySel and ScoreSel methods can terminate within 10^4 seconds even when the number of bus departures for each bus route is increased to 500 and the number of passengers to be served is increased to 5 million. In addition, GreedySel is about two times faster than ScoreSel as it does not need to derive the bus idle time for each physical bus.

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. The FAST and FASTCO problems are proposed to cater for different application needs in this paper. We showed that these problems are NP-hard, and proposed three approximation algorithms with non-trivial theoretical guarantees and two greedy-based heuristic methods for FAST and FASTCO, respectively. Lastly, we conducted experiments on real-world datasets to verify the efficiency, effectiveness, and scalability of our methods.

Acknowledgment

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, and a Google Faculty Award. Baihua Zheng is supported in part by Prime Minister's Office, Singapore under its International Research Centres in Singapore Funding Initiative.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

REFERENCES

[1] S. Schéele, "A supply model for public transit services," *Transportation Research Part B: Methodological*, vol. 14, no. 1-2, pp. 133–146, 1980.

[2] I. Constantin and M. Florian, "Optimizing frequencies in a transit network: a nonlinear bi-level programming approach," *International Transactions in Operational Research*, vol. 2, no. 2, pp. 149–164, 1995.

[3] Z. Gao, H. Sun, and L. L. Shan, "A continuous equilibrium network design model and algorithm for transit systems," *Transportation Research Part B: Methodological*, vol. 38, no. 3, pp. 235–250, 2004.

[4] H. Martínez, A. Mauttone, and M. E. Urquhart, "Frequency optimization in public transportation systems: Formulation and metaheuristic approach," *European Journal of Operational Research*, vol. 236, no. 1, pp. 27–36, 2014.

[5] N. Lin, W. Ma, and X. Chen, "Bus frequency optimisation considering user behaviour based on mobile bus applications," *IET Intelligent Transport Systems*, vol. 13, no. 4, pp. 596–604, 2019.

[6] G. Antonides, P. C. Verhoef, and M. Van Aalst, "Consumer perception and evaluation of waiting time: A field experiment," *Journal of consumer psychology*, vol. 12, no. 3, pp. 193–202, 2002.

[7] M. C. Kong, F. T. Camacho, S. R. Feldman, R. T. Anderson, and R. Balkrishnan, "Correlates of patient satisfaction with physician visit: differences between elderly and non-elderly survey respondents," *Health and Quality of Life Outcomes*, vol. 5, no. 1, p. 62, 2007.

[8] M. Cantwell, B. Caulfield, and M. O'Mahony, "Examining the factors that impact public transport commuting satisfaction," *Journal of Public Transportation*, vol. 12, 06 2009.

[9] M. Fletterman *et al.*, "Designing multimodal public transport networks using metaheuristics," Ph.D. dissertation, University of Pretoria, 2009.

[10] S. Wang, Z. Bao, J. S. Culpepper, T. Sellis, and G. Cong, "Reverse k nearest neighbor search over trajectories," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 4, pp. 757–771, 2018.

[11] O. J. Ibarra-Rojas, F. Delgado, R. Giesen, and J. C. Muñoz, "Planning, operation, and control of bus transport systems: A literature review," *Transportation Research Part B: Methodological*, vol. 77, pp. 38–75, 2015.

[12] S. Wang, Z. Bao, J. S. Culpepper, and G. Cong, "A survey on trajectory data management, analytics, and learning," *arXiv preprint arXiv:2003.11547*, 2020.

[13] S. Mo, Z. Bao, B. Zheng, and Z. Peng, "Bus frequency optimization: When waiting time matters in user satisfaction," in *Database Systems for Advanced Applications - 25th International Conference*. Springer, 2020, pp. 192–208.

[14] ——, "FASTS: A satisfaction-boosting bus scheduling assistant," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 2873–2876, 2020.

[15] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu, "A unified approach to route planning for shared mobility," *Proc. VLDB Endow.*, vol. 11, no. 11, pp. 1633–1646, 2018.

[16] Y. Xu, Y. Tong, Y. Shi, Q. Tao, K. Xu, and W. Li, "An efficient insertion operator in dynamic ridesharing services," in *35th IEEE International Conference on Data Engineering, ICDE*. IEEE, 2019, pp. 1022–1033.

[17] Y. Zeng, Y. Tong, and L. Chen, "Last-mile delivery made practical: An efficient route planning framework with theoretical guarantees," *Proc. VLDB Endow.*, vol. 13, no. 3, pp. 320–333, 2019.

[18] A. Ceder, B. Golany, and O. Tal, "Creating bus timetables with maximal synchronization," *Transportation Research Part A: Policy and Practice*, vol. 35, no. 10, pp. 913–928, 2001.

[19] O. J. Ibarra-Rojas and Y. A. Rios-Solis, "Synchronization of bus timetabling," *Transportation Research Part B: Methodological*, vol. 46, no. 5, pp. 599–614, 2012.

[20] Y. Shafahi and A. Khani, "A practical model for transfer optimization in a transit network: Model formulations and solutions," *Transportation Research Part A: Policy and Practice*, vol. 44, no. 6, pp. 377–389, 2010.

[21] J. Parbo, O. A. Nielsen, and C. G. Prato, "User perspectives in public transport timetable optimisation," *Transportation Research Part C: Emerging Technologies*, vol. 48, pp. 269–284, 2014.

[22] Y. Wu, "Combining local search into genetic algorithm for bus schedule coordination through small timetable modifications," *International Journal of Intelligent Transportation Systems Research*, vol. 17, no. 2, pp. 102–113, 2019.

[23] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions - I," *Math. Program.*, vol. 14, no. 1, pp. 265–294, 1978.

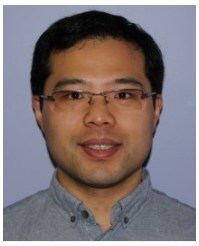
[24] P. Zhang, Z. Bao, Y. Li, G. Li, Y. Zhang, and Z. Peng, "Trajectory-driven influential billboard placement," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2748–2757.

[25] Y. Zhang, Y. Li, Z. Bao, S. Mo, and P. Zhang, "Optimizing impression counts for outdoor advertising," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019, pp. 1205–1215.

[26] L. Duan, T. Pang, J. Nummenmaa, J. Zuo, P. Zhang, and C. Tang, "Bus-olap: A data management model for non-on-time events query over bus journey data," *Data Sci. Eng.*, vol. 3, no. 1, pp. 52–67, 2018.

[27] X. Tian and B. Zheng, "Using smart card data to model commuters' responses upon unexpected train delays," in *International Conference on Big Data*. IEEE, 2018, pp. 831–840.

Songsong Mo is working towards his Master degree in Computer Science at Wuhan University. He received his Bachelor of Computer Science degree from Wuhan University in 2018. His research interests include database and big data analytics.



Zhifeng Bao received the Ph.D. degree in computer science from the National University of Singapore in 2011 as the winner of the Best PhD Thesis in school of computing. He is currently an Associate Professor at RMIT University. He is also an Honorary Senior Fellow with University of Melbourne in Australia. His current research interests include data usability, spatial database, graph data analytics and data integration.

Baihua Zheng received the PhD degree in computer science from Hong Kong University of Science & Technology, China, in 2003. She is currently a professor in the School of Information Systems, Singapore Management University, Singapore. Her research interests include mobile/pervasive computing, spatial databases, and big data analytics.

Zhiyong Peng is a professor of computer school, Wuhan University of China. He received B.Sc from Wuhan University, M.Eng. from Changsha Institute of Technology of China in 1985 and 1988, respectively, and Ph.D degree from Kyoto University of Japan in 1995. He worked as a researcher in Advanced Software Technology & Mechatronics Research Institute of Kyoto from 1995 to 1997 and a member of technical staff in Hewlett-Packard Laboratories Japan from 1997 to 2000. His research interests include complex data management, web data management, trusted data management. He is a member of IEEE Computer Society, ACM SIGMOD and vice director of Database Society of Chinese Computer Federation. He was general co-chair of WAIM2011, DASFAA2013 and PC Co-chair of DASFAA2012, WISE2006 and CIT2004.