Published in IEEE Transactions on Knowledge & Data Engineering, 35 (7), 6534-6547, 2023.
DOI: 10.1109/TKDE.2022.3188243

A Data-Driven Approach for Scheduling Bus Services Subject to Demand Constraints

Janaka Chathuranga Brahmanage, Thivya Kandappu, and Baihua Zheng

Abstract—Passenger satisfaction is extremely important for the success of a public transportation system. Many studies have shown that passenger satisfaction strongly depends on the time they have to wait at the bus stop (waiting time) to get on a bus. To be specific, user satisfaction drops faster as the waiting time increases. Therefore, service providers want to provide a bus to the waiting passengers within a threshold to keep them satisfied. It is a two-pronged problem: (a) to satisfy more passengers the transport planner may increase the frequency of the buses, and (b) in turn, the increased frequency may impact the service operational costs. To address it, we propose PASS and COST as the two variants that satisfy different optimization criteria mentioned above. The optimization goal of PASS is the number of satisfied passengers while the optimization goal of COST is the number of passengers served per unit of driving time. Consequently, PASS utilizes resources to the maximum to satisfy the highest number of passengers, while COST optimizes for both passenger satisfaction and operational costs. Accordingly, we propose two algorithms to solve PASS and COST respectively and evaluate their performance based on real passenger demand data-set.

Index Terms—Bus schedule optimization, dynamic bus scheduling, greedy search, operational cost, waiting time threshold

1 Introduction

The optimal bus scheduling problem has been extensively studied in recent history under the vision of realising smart and sustainable urban transportation. More specifically, in an effort to optimally schedule bus services, objectives, that are both (a) *endogenous*, such as operational costs [1], [2] and fleet size [3], [4], and (b) *exogenous*, such as spatio-temporal passenger demand variations [4], [5], [6], [7], [8] and passenger satisfaction (i.e., passenger wait time) [4], [9], [10], are optimised to achieve near-best bus schedules. The success of these approaches ultimately leads to improved system performance (e.g., reduced fleet size and/or operational cost) and enhanced passenger experience (e.g., shorter waiting time). However, the problem of optimal bus scheduling remains challenging, because it concerns multiple dimensions (e.g., spatio-temporal demand variants, passenger waiting time, operational costs etc.).

Recent years have seen a considerable amount of work on bus schedule optimisation. The prior focus in this area was on optimizing (a) passenger waiting time or passenger satisfaction [4], [9], [10], and (b) operational cost [1], [2]. In general, these approaches solve the scheduling problem, independently, while optimisation is considered along any *single* dimension. Further, several models were under unrealistic assumptions that the arrival rate of passengers at a bus stop can be modelled as a stochastic process or that physical bus vehicles are always available to serve a bus service when required (in other words, unlimited fleet size).

In this paper, we visualise the bus scheduling as a two-pronged problem: (a) to satisfy more passengers (measured by whether the waiting time of a passenger is within a certain threshold), the transport planner may increase the frequency of bus services, and (b) in turn, the increased frequency may impact the service operational costs. More specifically, we focus on the bus services scheduling problem subjected to a limited number of physical vehicles by maximising the number of passengers served by bus services while minimising the operational cost. To be more specific, we propose PASS and COST as the two variants that satisfy different optimisation criteria mentioned above. PASS aims at optimising the number of satisfied passengers while COST is to optimize the number of passengers served per unit driving time, both under the constraint of limited fleet size (i.e., the number of physical vehicles is limited). Therefore, PASS greedily utilizes resources to satisfy the highest number of passengers, while COST optimizes for both passenger satisfaction and operational costs.

To aid our analyses, in this paper, we specifically focus on (a) *Passenger dataset* $\mathcal{P}$, a smart-card generated city-scale trip data for public buses in Singapore, where the vast majority of passengers tap-in and tap-out when boarding and disembarking from a bus, respectively, thereby providing $\langle time_{in}, origin, destination \rangle$ records for individual trips (a month's worth of anonymity-preserving smart-card generated bus trip data accounts to 108 million trips, taken by $\approx$ 5 million passengers), and (b) *Route network* $\mathcal{R}$, obtained from Singapore's publicly available government facilitated portal called *DataMall* (https://datamall.lta.gov.sg/), encompasses 5036 bus stops and 715 unique routes with the fleet size of $\approx$4000.

- *The authors are with the School of Computing and Information Systems, Singapore Management University, Singapore 178902. E-mail: {janakab, thivyak, bhzheng}@smu.edu.sg.*

Key Contributions. Our key contributions are:

- *Considering passengers' tolerance on waiting time.* To the best of our knowledge, previous work [4] is the only study on bus schedule optimization that considers passengers' tolerance on waiting time. It modelled the problem as a submodular optimization and proposed a constant factor approximation algorithm if there is no fleet size limitation. In our work, we propose a constant factor approximation algorithm (Sequence Greedy) for the same problem while considering the fleet size limitation.
- *Proposing two algorithms* namely *Bidirectional Greedy* and *Sequence Greedy* to solve the bus service scheduling problem while using passengers' tolerance on waiting time as the optimization goal.
- *Designing two optimization techniques* to improve the running time of algorithms; one of them can be directly applied to previous work [4] as well.
- *Introducing three models* (M_{one}, M_{ss}, and M_f), to share the limited fleet across multiple routes during the day, and we evaluate the performance of our algorithms on each of them.
- *Conducting the experiments via real city-scale trip data and bus route network.*

2 Related Work

Schedule optimization of transportation networks has been studied for decades. In this section, we will review existing approaches that are related to our study and discuss the differences between our work and existing approaches based on: (a) the optimization goals, (b) the way the travel demand is modelled, and (c) the architecture of the solution.

Base on optimization goals.The main goals of the schedule optimization problem include reducing (i) total travelling time, (ii) passenger waiting time, (iii) passenger transfer time, and (iv) operational cost.

Passengers' total travel time/travel cost (that includes both the waiting time to board the bus and the travel time to reach the destination of each passenger) is a popular optimization goal, considered in many existing works [5], [11], [12], [13]. For example, [5] considers, for each origin-destination pair, the weighted sum of travel time w.r.t. different route options, weighted by the route probability of selecting them by passengers.

The passenger waiting time is another commonly used optimization option. Most of the existing works (e.g., [9], [10], [14]) only consider the total waiting time or the average waiting time as the optimization goal. However, it is important to note that neither the total waiting time nor the average waiting time is a precise indicator of passenger satisfaction, as passenger dissatisfaction does not have a linear relationship with the waiting time. In an actual scenario, passenger satisfaction drops faster as the waiting time increases [15], [16], [17]. A novel optimization goal is proposed in [4] by considering the number of passengers served within a waiting time threshold.

Transfer time required by transfer passengers, though less popular, is considered as an optimization goal too by some works [10], [12], [18]. However, these works only serve the passengers who need to make transfers when travelling from their origin to their destination, which refers to a small subset of the entire passenger population.

The other most important aspect of the optimization goal is the operational cost, which consists of time cost, personnel cost, space cost, and vehicle operational cost (e.g., fuel cost and maintenance cost) [19]. Multiple works have taken it into account [1], [11], [20] and the works presented [1], [11] have further abstracted all operational costs into a variable cost per unit distance.

Based on the Model of Demand. Most of the early works consider the travel demand as a statistical distribution [7], [8]. They have used datasets collected using surveys and electronic detection devices such as infrared beams to derive the probabilistic distribution of the travel demand over time and routes. However, the recent development of Automated Fare Collection (AFC) systems allows us to collect more precise data about the travel demand as origin-destination pairs. Data collected by most AFC systems include origin-stop, destination-stop, boarding-time, and alighting-time. Recent studies such as [4], [21] have utilized them to model more precise travel demand.

Based on Solution Architecture. Studies presented in [9], [13], [22], [23], [24] use Mixed Integer Linear Programming (MILP) to model the problem and then solve it using various standard methods. Though MILP is powerful enough to support complex constraints and optimization goals, it suffers from a major disadvantage, i.e., poor scalability. As the number of variables grows, the running time increases exponentially. To address this problem, most of these works propose approximation methods such as Heuristic Search, Harmony Search, and Genetic Algorithm (GA) [9], [13], [18]. The other frequently used approach is bi-level programming [2], [12], [25], [26], which divides the problem into two levels. They have used different optimization techniques such as sub-gradient optimization, MILP, and GA at each level.

Discussion. Similar to the existing work [4], we aim at improving the overall passenger satisfaction by scheduling the physical vehicles such that they can serve more passengers within a given waiting time threshold. In a public transport network, the bus routes are fixed and the travelling time from one stop to another via a fixed bus route is not affected by the scheduling much. On the other hand, the scheduling does affect passengers' waiting time. Based on our observations, passengers become unhappy and frustrated once the waiting time exceeds a certain threshold. Though different passengers might have different tolerance levels to the waiting time, we introduce a waiting time threshold and aim at serving as many passengers within their waiting time thresholds as possible by scheduling the bus services properly. In our work, we consider physical vehicles as limited resources and propose two problems, namely PASS and COST. PASS tries to optimize the total number of satisfied passengers, while COST considers not only the number of satisfied passengers but also the operational cost.

Existing work [4] also considers the constraint of limited physical vehicles and formulates a problem of FASTCO. However, we want to highlight that though FASTCO and PASS share a similar optimization goal, their inputs are

TABLE 1
Notations for Problem Formulation and Solution

Symbol	Description
r	A conventional bus route which consists of a sequence of $\|r\|$ bus stops $\langle s_1, s_2, \cdots, s_{\|r\|}\rangle$
$\mathcal{R}$	A bus route database with $r \in \mathcal{R}$ referring to a bus route
dt	Departure time of a bus service candidate
$D(r)$	Total travel time of a bus route $r \in \mathcal{R}$
$D(r, s_a, s_b)$	Travel time from stop s_a to stop s_b on bus route $r \in R$ where $s_a, s_b \in r$
bs	A bus service candidate in the form of $\{r, dt\}$, representing a service w.r.t. route r and having dt as the departure time from the starting stop
$\mathcal{B}$	A bus service candidates database with $bs \in \mathcal{B}$ referring to a bus service
seq	A sequence of bus services $\langle bs_1, bs_2, \cdots, bs_n\rangle$ to be served by a physical vehicle
Seq_{valid}	Set of all valid sequences seq, in the bus service candidate database $\mathcal{B}$
$\mathcal{S}$	A bus schedule, a vector of valid bus service sequences $\langle seq_1, seq_2, \cdots, seq_m\rangle$ which can be served by m physical vehicles
$\mathcal{V}$	Set of all physical bus vehicles with $v_i \in \mathcal{V}$ referring to one vehicle
$Set(\mathcal{S})$	Set of all the bus services in the bus schedule $\mathcal{S}$
p	a passenger represented by $\langle s_b, s_e, rt\rangle$, where s_b and s_e denote the boarding stop and the alighting stop respectively, and rt denotes the time when p reaches s_b
$\mathcal{P}$	A passenger database, with $p \in \mathcal{P}$ referring to a passenger
θ	The passenger waiting time threshold
w_{min}/w_{max}	Minimum/maximum sojourn time for a physical vehicle between two consecutive services

different and FASTCO considers only a special case of PASS. First, FASTCO assumes the physical vehicles are distributed across different terminals and takes it as an input, where a terminal refers to a bus stop that serves as the first stop of at least one bus route. However, the initial distribution of physical vehicles directly affects the overall performance of the underlying scheduling algorithm and FASTCO does not discuss that. Second, FASTCO assumes after a physical vehicle reaches the last stop of a route and completes the current service, it can serve any route that is started from the same stop. This is one of the three models considered by PASS, which will be detailed in Section 3. Third, we introduce parameters to control the sojourn time of a vehicle (the duration from the end of the previous service to the start of the next service) in a bus terminal, which effectively avoids the undesirable schedules encountered in FASTCO, where a physical vehicle is assigned to serve services that could meet the demand of many passengers without considering the time gap between services. Last but not least, the algorithms proposed in this paper can improve the performance of FASTCO, e.g., SG is able to improve the performance of FASTCO by 18.03% and 6.96% when the number of vehicles is set to 1000 and 3000 respectively. Please refer to Section 5 for more detailed comparisons between FASTCO and our work.

3 Formulating Bus Service Scheduling Problem

In this paper, our main goal is to identify optimal bus schedules, so as to maximise the total number of satisfied passengers or other objectives under the constraint of limited fleet size. Table 1 lists the major symbols used in the rest of this paper.

3.1 Preliminary

In a bus route database $\mathcal{R}$, a route $r \in \mathcal{R}$ has the following attributes: (i) a route number/identifier, denoted by id, and (ii) a sequence of bus stops $\langle s_1, s_2, \cdots, s_{|r|}\rangle$ visited by r in chronological order, where $r.s_1$ and $r.s_{|r|}$ denote the first and the last bus stops of r respectively. Please note that the same route identifier can be used to denote two routes $r_1, r_2 \in \mathcal{R}$ that operate in opposite directions. Notation $D(r)$ refers to the total travel time of the bus route r. How to derive $D(r)$ values for different routes is orthogonal to the work presented in this paper. There are multiple ways to estimate the travel time – for example, we can adopt a simple historical average approach to approximate $D(r)$ by computing the average travel time of r over the past x days.

A service (or bus service) bs represents a single trip instance over a bus route and is denoted by a tuple $\langle r, dt\rangle$, where $bs.r$ is the route served by the service bs, and $bs.dt$ is the departure time of bs from the first bus stop $r.s_1$. We assume bus service database $\mathcal{B}$ consists of all the possible bus services, e.g., {⟨r, 5:00⟩, ⟨r, 5:01⟩, ⟨r, 5:02⟩, $\cdots$, ⟨r, 23:00⟩} represent all the possible bus services w.r.t. bus route r under the assumptions that i) bus services start at 5:00 and end at 23:00 in a city, and ii) the bus scheduling is performed at the granularity of minute.

In typical bus scheduling, each physical vehicle v is bound to serve multiple services and/or routes in a day. Intuitively, various policies and models can be devised to assign the next service bs_j after v completes its current service bs_i. For example, while one model would allow the bs_j to be the service that runs in the opposite direction of bs_i, another model might suggest bs_j to be the service that caters for a route that is high in demand. As the scheduling of the services depends on various models and applications in need, we define $\mathcal{A}(bs_i, bs_j, M)$ in Eq. (1) to estimate the validity of assigning v to bs_j after it finishes bs_i under a specific model M.

$$\mathcal{A}(bs_i, bs_j, M) = \begin{cases} 1 & bs_j \text{ after } bs_i \text{ is valid under } M \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

As mentioned above, M represents various models adopted to schedule the next service to a physical vehicle. In this paper, we consider three different models to cater for different applications, namely, M_{one}, M_{ss}, and M_f, referring to *one route model*, *same stop model*, and *flexible model*, respectively. The sojourn time of a vehicle in a bus terminal (i.e., the time between any two consecutive services assigned to a vehicle) is parameterized by w_{min} and w_{max} to denote the minimum and maximum time gaps. On the one hand, w_{min} enforces a break between two services assigned to a vehicle, hence, the driver could have some rest and the vehicle can be cleaned/disinfected properly; on the other hand, w_{max} controls the duration of the break such that each vehicle could achieve a certain utilization threshold.

To simplify our presentation, let t $(= bs_i.dt + D(bs_i.r))$ denote the time when the vehicle reaches the last stop to complete the current service bs_i, where $bs_i.dt$ denotes the departure time of bs_i from its first stop, and $D(bs_i.r)$ denotes the total travel time. We now describe our three models in detail, below.

- M_{one}: Each physical vehicle is only assigned to bus services corresponding to the same route but different directions, i.e., $\mathcal{A}(bs_i, bs_j, M_{one}) = 1 \Leftrightarrow bs_i.r.id = bs_j.r.id \wedge bs_i.r.s_{|bs_i.r|} = bs_j.r.s_1 \wedge bs_j.dt \in [t + w_{min}, t + w_{max}]$.
- M_{ss}: Each vehicle, once finishing a bus service bs_i, can serve another bus service bs_j that originates from the last stop of $bs_i.r$, i.e., $\mathcal{A}(bs_i, bs_j, M_{ss}) = 1 \Leftrightarrow bs_i.r.s_{|bs_i.r|} = bs_j.r.s_1 \wedge bs_j.dt \in [t + w_{min}, t + w_{max}]$.
- M_f: Each physical vehicle, once finishing a bus service bs_i, has the flexibility to determine whether it stays at the same stop or moves to a nearby stop to serve the next bus service, i.e., $\mathcal{A}(bs_i, bs_j, M_f) = 1 \Leftrightarrow D(bs_i.r.s_{|bs_i.r|}, bs_j.r.s_1) \leq d_{max} \wedge bs_j.dt \in [t + w_{min}, t + w_{max}]$. Here, $D(s_a, s_b)$ returns the time required by a vehicle to move from a bus stop s_a to another stop s_b, and parameter d_{max} is introduced to define the maximum distance (in the form of travelling time) a vehicle could travel under this model.

Definition 3.1. *We define a* valid *sequence of bus services a physical vehicle v_i can serve under a particular model M as seq_i, in the form of $\langle bs_1, bs_2, \ldots, bs_{n_i}\rangle$, i.e., $\forall j \in [1, n_i)$, $\mathcal{A}(bs_j, bs_{j+1}, M) = 1$.*

We introduce the concept of sequence to define the list of bus services that is served by a vehicle. In other words, a sequence seq_i associated with a vehicle v_i describes the scheduled bus services to be served by vehicle v_i. To facilitate the presentation, operator $\oplus$ is introduced to append a new bus service to a sequence to extend the sequence. To be more specific, let seq_i be a sequence in the form of $\langle bs_1, \ldots, bs_{n_i}\rangle$. Then, $seq_i \oplus bs_j$ updates sequence seq_i to $\langle bs_1, \ldots, bs_{n_i}, bs_j\rangle$ or $\langle bs_j, bs_1, \ldots, bs_{n_i}\rangle$, dependent on the scheduled departure time of bs_j. After the extension, n_i is increased by one to reflect the updated number of bus services in the sequence seq_i. In addition, we assume only bus service bs_j satisfying the condition $\mathcal{A}(bs_{n_i}, bs_j, M) = 1$ or $\mathcal{A}(bs_j, bs_1, M) = 1$ could be appended to seq_i.

Similarly, we can extend $\oplus$ to a schedule $\mathcal{S}$ that is defined in Definition 3.2. To be more specific, notation $\mathcal{S} \oplus^i bs_j$ indicates that a new bus service bs_j is appended to sequence $seq_i \in \mathcal{S}$, i.e., $\mathcal{S} \oplus^i bs_j = \mathcal{S} - \{seq_i\} + \{seq_i \oplus bs_j\}$.

Definition 3.2. *We define a bus schedule $\mathcal{S}$ w.r.t. a vehicle database $\mathcal{V}$ and a model M as a set of sequences corresponding to vehicles in $\mathcal{V}$ under the constraint that all the sequences in $\mathcal{S}$ have zero overlap, i.e., $\mathcal{S} = \cup_{v_i \in \mathcal{V}} seq_i \wedge \forall v_i, v_j \in \mathcal{V}$, $seq_i \cap seq_j = \emptyset$. Here, seq_i refers to a valid sequence corresponding to a vehicle $v_i \in \mathcal{V}$ under the model M.*

In a passenger database $\mathcal{P}$, a passenger $p \in \mathcal{P}$ is in the form of a tuple $\langle s_b, s_e, rt\rangle$, where $p.s_b$ denotes the boarding stop, $p.s_e$ denotes the alighting stop, and $p.rt$ denotes the time when p reaches s_b [4].

Definition 3.3. *We define that a bus service bs_j $\langle r, dt\rangle$ can serve a passenger p, w.r.t. a given waiting time threshold θ_p, if r passes stops $p.s_b$ and $p.s_e$ in order, and $0 \leq dt + D(r, r.s_1, p.s_b) - p.rt \leq \theta_p$, where $D(r, r.s_1, p.s_b)$ denotes the travel time required by the bus service bs_j from $r.s_1$ to $p.s_b$ via the bus route r.*

Based on Definition 3.3 and work presented in [4], we formally introduce $Ser(bs_j, p_k, \theta_p)$ to denote the service of bs_j to passenger p_k under the waiting time threshold θ_p, as presented in Eq. (2).

$$Ser(bs_j, p_k, \theta_p) = \begin{cases} 1 & \text{if } bs_j \text{ can serve } p_k \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Then, for bus service schedule $\mathcal{S}$, its services to a passenger p_k can be derived by Eq. (3). Note that $Ser(\mathcal{S}, p_k, \theta_p) = 1$ as long as any bus service $bs_j \in \mathcal{S}$ can serve p_k; otherwise, $Ser(\mathcal{S}, p_k, \theta_p) = 0$. Accordingly, the total number of passengers that can be served by a given schedule $\mathcal{S}$ can be calculated by Eq. (4).

$$Ser(\mathcal{S}, p_k, \theta_p) = 1 - \prod\nolimits_{bs_j \in \mathcal{S}} (1 - Ser(bs_j, p_k, \theta_p)) \tag{3}$$

$$\mathcal{G}(\mathcal{S}, \theta_p) = \sum\nolimits_{p_k \in \mathcal{P}} Ser(\mathcal{S}, p_k, \theta_p) \tag{4}$$

3.2 Problem Definition

We study the bus scheduling problem under the constraint of a limited number of physical vehicles. To ease our presentation, let Seq_{valid} define the complete set of all possible valid bus service sequences.

We first introduce PASS(short for *Passenger Oriented Bus Scheduling*). PASS maximizes a set function from Seq_{valid} to the total number of satisfied passengers, where the cardinality constraint is that the total number of sequences shall be equivalent to the number of physical vehicles $|\mathcal{V}|$ (i.e., fleet size). To be more specific, for a given fleet size $\mathcal{V}$, PASS aims at finding $\mathcal{S}$ such that $|\mathcal{S}| = |\mathcal{V}|$ and $\mathcal{S} \subset Seq_{valid}$ that can maximize $\mathcal{G}(\mathcal{S}, \theta_p)$, as defined in Eq. (5).

$$\mathcal{S}^p_{optimal} = argmax_{\mathcal{S} \subset Seq_{valid} \wedge |\mathcal{S}| = |\mathcal{V}|} \mathcal{G}(\mathcal{S}, \theta_p) \tag{5}$$

We then introduce COST, which stands for *Cost Effective Bus Scheduling*. Different from PASS, COST considers the total number of passengers served per unit cost. Without

loss of generality, to estimate the operational cost, we consider the total mileage, as it directly affects the fuel consumption (e.g., the fuel consumption of a full-sized city bus is around 2.5–3 km per litre), the number of drivers required, and maintenance cost. As vehicles in a city, travel at a relatively stable speed, we measure the mileage by adopting the time duration $D(r)$ of a route r instead of the travel distance, as defined in Eq. (6). With that assumption, we consider the number of passengers served per unit vehicle-operation-time as the optimization goal for COST. Accordingly, COST aims at finding $\mathcal{S}$ such that $|\mathcal{S}| = |\mathcal{V}|$ and $\mathcal{S} \subset Seq_{valid}$ that can maximize $\frac{\mathcal{G}(\mathcal{S},\theta_p)}{\mathcal{M}(\mathcal{S})}$, as defined in Eq. (7).

$$\mathcal{M}(\mathcal{S}) = \sum\nolimits_{seq_i \in \mathcal{S}} \sum\nolimits_{bs_j \in seq_i} D(bs_j.r) \tag{6}$$

$$\mathcal{S}^c_{optimal} = argmax_{\mathcal{S} \subset Seq_{valid} \wedge |\mathcal{S}| = |\mathcal{V}|} \frac{\mathcal{G}(\mathcal{S},\theta_p)}{\mathcal{M}(\mathcal{S})} \tag{7}$$

3.3 Problem Analysis

In the following, we perform a theoretical analysis on PASS and COST. First, we prove PASS is submodular and monotonically increasing in Theorem 3.1, while COST is not submodular nor increasing in Theorem 3.2. A submodular function $f(E)$ is a set function with diminishing returns. In other words, the difference of the value increment (i.e., $f(E \cup \{e\}) - f(E)$) when a new element e is added to the input set E decreases as size $|E|$ of the input set E grows. We will report how this property is used to obtain a constant-factor approximation algorithm for PASS in Section 4. Next, we discuss the hardness of both PASS and COST in Theorem 3.3.

Theorem 3.1. *PASS is submodular and monotonically increasing.*

Let $X \subseteq Y \subseteq Seq_{valid}$, where Seq_{valid} denotes all valid bus service sequences, and X, Y are two potential bus schedules. In addition, seq refers to a bus service sequence in $Seq_{valid} \backslash Y$. According to [27], $\mathcal{G}_s(V)$ is submodular if it satisfies: $\mathcal{G}_s(X \cup seq) - \mathcal{G}_s(X) \geq \mathcal{G}_s(Y \cup seq) - \mathcal{G}_s(Y)$.

To facilitate the proof, we define $X_s = X \cup seq$, $Y_s = Y \cup seq$, $\mathcal{G}_s(X) = \mathcal{G}(X_s,\theta_p) - \mathcal{G}(X,\theta_p)$, and $\mathcal{G}_s(Y) = \mathcal{G}(Y_s,\theta_p) - \mathcal{G}(Y,\theta_p)$. Further, for the simplicity, we write $Ser(X,p_k)$ for $Ser(X,p_k,\theta_p)$. Then, we have:

$$\begin{aligned}&\mathcal{G}_s(X) - \mathcal{G}_s(Y)\\ &= \sum\nolimits_{p_k \in \mathcal{P}} (Ser(X_s,p_k) - Ser(X,p_k))\\ &\quad - \sum\nolimits_{p_k \in \mathcal{P}} (Ser(Y_s,p_k) - Ser(Y,p_k))\end{aligned} \tag{8}$$

If we prove Inequality (9) for any $p_k \in \mathcal{P}$, then we can show the submodularity of $\mathcal{G}$,

$$Ser(X_s,p_k) - Ser(X,p_k) - Ser(Y_s,p_k) + Ser(Y,p_k) \geq 0 \tag{9}$$

According to whether p_k can be served by bus services in X or bus services in $Y \backslash X$ or bus service sequence seq, there are in total four cases corresponding to Inequality (9).

	$X \subseteq Y$	Y	$seq \in Seq_{valid}$
case 1	✓	-	-
case 2	✗	✓	-
case 3	✗	✗	✓
case 4	✗	✗	✗

Case 1: p_k can be served by a bus service in sequence $seq_0 \in X$. Then, we have $Ser(X,p_k) = Ser(X_s,p_k) = Ser(Y,\ p_k) = Ser(Y_s,p_k) = 1$, because $X \subset X_s$ and $X \subseteq Y \subset Y_s$. Thus, $Ser(X_s,p_k) - Ser(X,p_k) - Ser(Y_s,p_k) + Ser(Y,p_k) = 0$.

Case 2: p_k cannot be served by a bus service in any sequence $seq_0 \in X$ but it can be served by a bus service in a sequence $seq_1 \in Y \backslash X$. Then, we have $Ser(X,p_k) = 0$, $Ser(X_s,p_k) \geq 0$ and $Ser(Y,p_k) = Ser(Y_s,p_k) = 1$. Thus, $Ser(X_s,p_k) - Ser(X,p_k) - Ser(Y_s,p_k) + Ser(Y,p_k) \geq 0$.

Case 3: p_k cannot be served by a bus service in any sequence $seq_0 \in Y$ and can be served by a bus service in the sequence seq. Then we have $Ser(X,p_k) = Ser(Y,p_k) = 0$ and $Ser(X_s,p_k) = Ser(Y_s,p_k) = 1$. Thus, $Ser(X_s,p_k) - Ser(X,p_k) - Ser(Y_s,p_k) + Ser(Y,p_k) = 0$.

Case 4: p_k cannot be served by a bus service in any sequence $seq_0 \in Y$ or a bus service in the service sequence seq. Then, we have $Ser(X,p_k)$=$Ser(X_s,p_k)$=$Ser(Y,p_k) = Ser(Y_s,p_k) = 0$. Thus, $Ser(X_s,p_k) - Ser(X,p_k) - Ser(Y_s,p_k) + Ser(Y,p_k) = 0$. The above shows the correctness of Inequality (9).

Based on Eq. (8) and Inequality (9), we conclude that $\mathcal{G}$ is a submodular function.

To show that $\mathcal{G}$ is increasing, we have to show that if $X \subseteq Y \Rightarrow \mathcal{G}(Y,\theta_p) - \mathcal{G}(X,\theta_p) \geq 0$. Again, we assume $X \subseteq Y \subseteq Seq_{valid}$. Then, we have

$$\begin{aligned}\mathcal{G}(Y,\theta_p) - \mathcal{G}(X,\theta_p) &= \sum_{p_k \in \mathcal{P}} Ser(Y,p_k) - \sum_{p_k \in \mathcal{P}} Ser(X,p_k)\\ &= \sum\nolimits_{p_k \in \mathcal{P}} (Ser(Y,p_k) - Ser(X,p_k))\end{aligned} \tag{10}$$

Similarly, we try to prove Inequality (11) for any $p_k \in \mathcal{P}$ in order to prove that $\mathcal{G}$ is increasing.

$$Ser(Y,p_k) - Ser(X,p_k) \geq 0 \tag{11}$$

There are three cases in total depending on whether p_k can be served by bus services in X, Y.

Case 1: p_k can be served by a bus service in sequence $seq_0 \in X$. Then, we have $Ser(X,p_k) = Ser(Y,p_k) = 1$ because $X \subseteq Y$, i.e., $Ser(Y,p_k) - Ser(X,p_k) = 0$.

Case 2: p_k cannot be served by any bus service in X but a bus service in sequence $seq_0 \in Y \backslash X$. Then, we have $Ser(X,p_k) = 0$ and $Ser(Y,p_k) = 1$ because $X \subseteq Y$, i.e., $Ser(Y,p_k) - Ser(X,p_k) = 1$.

Case 3: p_k cannot be served by any bus service in X or Y. Then, we have $Ser(X,p_k) = Ser(Y,p_k) = 0$ because $X \subseteq Y$, i.e., $Ser(Y,p_k) - Ser(X,p_k) = 0$.

The above cases show the correctness of Inequality (11). It proves that $\mathcal{G}$ is monotonically increasing and our proof completes. □

Theorem 3.2. *COST is neither submodular nor monotonically increasing.*

The optimization function of COST is $Opt_{cost}(\mathcal{S}) = \frac{\mathcal{G}(\mathcal{S},\theta_p)}{\mathcal{M}(\mathcal{S})}$. First, we provide a counterexample to show that the function is not submodular. Let $X \subset Y \subset Seq_{valid}$. Then, according to the definition of submodular, Opt_{cost} should satisfy Eq. (12), if it is submodular.

$$Opt_{cost}(X \cup seq) - Opt_{cost}(X) \geq Opt_{cost}(Y \cup seq) - Opt_{cost}(Y) \quad (12)$$

To eaze the presentation, we assume $lhs = Opt_{cost}(X \cup seq) - Opt_{cost}(X)$ and $rhs = Opt_{cost}(Y \cup seq) - Opt_{cost}(Y)$. Let consider a situation where X, Y both serve only one passenger with $\mathcal{M}(X) = 1$ and $\mathcal{M}(Y) = 2$. Then, we have $\mathcal{G}(X, \theta_p) = 1$ and $\mathcal{G}(Y, \theta_p) = 1$. Let $seq \in Seq_{valid} \backslash Y$, $\mathcal{G}(seq) = 0$ and $\mathcal{M}(seq) = 1$. We have $lhs = \frac{1}{2} - \frac{1}{1} = -\frac{1}{2}$ and $rhs = \frac{1}{3} - \frac{1}{2} = -\frac{1}{6}$. As $lhs < rhs$, it can be concluded that Opt_{cost} is not submodular.

Further note that in above example $X \subseteq Y$ but $Opt_{cost}(X) > Opt_{cost}(Y)$. It contradicts with Eq. (11). Therefore, COST is not monotonically increasing. Our proof completes. □

Theorem 3.3. *Both PASS and COST are NP-complete.*

In the following, we try to prove that PASS is NP-Complete. First, it is important to note that PASS is NP because we can use a simple procedure to determine whether a given schedule $\mathcal{S}$ can serve the given number of passengers. We skip the details because it is obvious.

Then, we prove the problem is NP-hard by reducing the Vertex Cover Problem into PASS. Given a graph $G(V, E)$ and a positive integer k, the Vertex Cover Problem is to find whether there is a subset V' of vertices of size at most k, such that every edge in the graph is connected to some vertex in V'. We map each vertex in V into a service-sequence in Seq_{valid} such that $|V| = |Seq_{valid}|$, and map each edge in E to a passenger in P. Note that passengers are placed in such a way that a passenger (represented by an edge $e(v_1, v_2)$) can only be served by any of the two sequences corresponding to v_1 and v_2. Then, we map the positive integer k into the number of available vehicles $|\mathcal{V}|$. Based on the above arrangements, the Vertex Cover Problem is equivalent to deciding whether there is a schedule $\mathcal{S}$ that can serve all the passengers. Since the Vertex Cover Problem is a well-known NP-Complete problem, we can conclude that PASS is NP-Complete.

Next, we can show that the Vertex Cover Problem can be reduced to a special configuration of COST. For this, we are going to consider a bus network with $\forall r \in \mathcal{R},\ \mathcal{D}(r) = t$, where all the bus services share the same duration t. The duration within which the bus services are provided is set to T with T approaching $2t$ but smaller than $2t$. In other words, each sequence contains one and only one bus service. Further note that for any final solution $\mathcal{S}$, the number of sequences included in $\mathcal{S}$ (i.e., $|\mathcal{S}|$) equals the number of available vehicles $|\mathcal{V}|$. Under these special arrangements, we can say $Opt_{cost} = \frac{\mathcal{G}(\mathcal{S},\theta_p)}{t \times |\mathcal{V}|}$, where $(t \times |\mathcal{V}|)$ is a constant. We can ignore $\frac{1}{t \times |\mathcal{V}|}$ as optimization process does not depend on a constant multiplier. In other words, we can map the Vertex Cover problem into COST as same as PASS and we can conclude that COST is also NP-Complete.

4 Solution

We introduce two algorithms to solve PASS and COST, namely *Bidirectional Greedy (BG)* and *Sequence Greedy (SG)*. In addition, we also propose a few techniques to further improve search efficiency. To ease the presentation, we focus on PASS. However, we can easily replace the objective function from Eq. (5) to Eq. (7) so that the algorithms can support COST.

4.1 Bidirectional Greedy (BG) Search

Bidirectional greedy search tries to allocate available vehicles to bus services with the highest marginal gains. To be more specific, among all the possible bus services, it initializes a sequence with the bus service having the highest gain and slowly expands and completes the sequence by repeatedly appending valid bus services with the highest marginal gains to the sequence.

There are different ways to implement the above mentioned principle. For example, we can follow the service-first strategy to locate a bus service with the highest gain and include the identified service in a suitable sequence. In this way, we actually expand all $|\mathcal{V}|$ sequences in the final solution concurrently. Alternatively, we can follow the sequence-first strategy by performing the search to complete one sequence first before starting allocating bus services to another sequence.

Algorithm 1. Bidirectional Greedy $(\mathcal{B}, M, \mathcal{V}, \theta_p)$

Input: a candidate bus database $\mathcal{B}$, Sequencing Model M, available bus vehicles $\mathcal{V}$, passenger waiting time threshold θ_p
Output: a bus service schedule $\mathcal{S}$

$\mathcal{S} \leftarrow \emptyset$
for *each* $v_i \in \mathcal{V}$ **do**
  $seq \leftarrow argmax_{bs_i \in \mathcal{B}}(\mathcal{G}(\mathcal{S} \cup \{bs_i\}, \theta_p) - \mathcal{G}(\mathcal{S}, \theta_p))$
  $\mathcal{B} \leftarrow \mathcal{B} / \{bs_i\}$
  while *True* **do**
    $bs_{first} \leftarrow seq[1], bs_{last} \leftarrow seq[|seq|]$
    $\mathcal{B}_{possible} \leftarrow \{bs_i \in \mathcal{B} | \mathcal{A}(bs_i,\ bs_{first}, M) = 1 \vee \mathcal{A}(bs_{last}, bs_i, M) = 1\}$
    if $\mathcal{B}_{possible} = \emptyset$ **then**
      break
    $bs_{best} \leftarrow argmax_{bs_i \in \mathcal{B}_{possible}}(\mathcal{G}(\mathcal{S} \cup \{seq \oplus bs_i\}, \theta_p) - \mathcal{G}(\mathcal{S} \cup \{seq\}, \theta_p))$
    $seq \leftarrow seq \oplus bs_i, \mathcal{B} \leftarrow \mathcal{B} / \{bs_i\}$
  $\mathcal{S} \leftarrow \mathcal{S} \cup seq$
return $\mathcal{S}$

After some preliminary evaluation, we find that the sequence-first strategy outperforms the service-first strategy. This is because the service-first strategy associates a vehicle with a bus service bs_i (corresponding to route r) that could achieve a very high gain. Though vehicles assigned to route r can be assigned to other routes, the initial assignment is very critical (e.g., model M_{one} does not allow a vehicle assigned to route r to serve another route, and M_{ss} only allows a vehicle to serve a route that starts from the bus stop that the vehicle is located). The fact that route r is in high demand during the service of bs_i does not guarantee that route r always has a relatively high demand. Consequently, it might allocate vehicles to serve the routes with

very high demand in a very short duration which affects the overall performance. Consequently, we only include the bidirectional greedy based on the sequence-first strategy in this paper.

Algorithm 1 lists the pseudo-code for BG. It uses a FOR loop to complete $|\mathcal{V}|$ sequences. Within each iteration, it focuses on one single sequence that will be assigned to a physical vehicle. It identifies the bus service bs_i with the highest gain and initiates a new sequence seq with bus service bs_i (Lines 3-4). Thereafter, it slowly expands and completes this sequence seq by repeatedly including valid bus services with the highest marginal gain to the sequence. Following the idea of greedy search, we can just locate a bus service with the highest gain and then check whether the current sequence seq could accommodate it. However, this process could be very time-consuming, as many services might not be valid, dependent on the first/last bus services in the sequence. To improve the search performance and avoid evaluating bus services that cannot be added to the current sequence, we only consider valid bus services from $\mathcal{B}$ that can be added to the current sequence seq under M to form a candidate bus service set $\mathcal{B}_{possible}$. It then locates the one bs_{best} with the highest gain from this candidate set and inserts it to seq (Lines 6-11). This process continues until the candidate set $\mathcal{B}_{possible}$ becomes empty.

4.2 Sequence Greedy (SG)

Bidirectional Greedy examines bus services one by one and tries to include bus services with higher gains into the solution earlier. However, an insertion of a bus service to a sequence has a direct impact on the services that could be inserted into the same sequence later on and hence the overall gain achieved by the sequence. A sequence containing a bus service with the highest gain might not be able to achieve an overall high gain. Consequently, we propose our second algorithm that employs sequence as the unit to evaluate the gain for the entire sequence when we construct the solution, namely *Sequence Greedy* (in short SG).

SG search follows the greedy approach proposed [27] over the set of all valid service sequences Seq_{valid} and it is guaranteed to achieve (1 - $1/e$)-approximation, according to the proof provided by [27]. We iterate over all vehicles in $\mathcal{V}$. In each iteration, we find the sequence with the highest marginal gain and assign it to the vehicle. The size of Seq_{valid}, the set of all valid service sequences, is exponential and at a very large scale, because it includes all possible combinations of bus services that can be served by a physical vehicle under the given model M. For example, Seq_{valid} in our experiments has over 10^{20+} combinations, when we consider 700+ bus routes over 5,000+ bus stops, use minute as the smallest scheduling unit, and have w_{min} and w_{max} set to 10 minutes and 40 minutes respectively. Therefore, it is extremely expensive, if not possible, to evaluate all possible sequences to find the one with the highest gain. However, the way we introduce the model M allows us to utilize dynamic programming to find the sequence with the highest gain in a polynomial time. Details of the implementation of dynamic programming are presented in Section 4.3.

Algorithm 2. Sequence Greedy($\mathcal{B}, M, \mathcal{V}, \theta_p$)

Input: a candidate bus database $\mathcal{B}$, Sequencing Model M, vehicles $\mathcal{V}$, passenger waiting time threashold θ_p
Output: a bus service schedule $\mathcal{S}$

1: $\mathcal{S} \leftarrow \emptyset$
2: **for** $v_i \in \mathcal{V}$ **do**
3: $seq_{best} \leftarrow \emptyset$, $g_{best} \leftarrow 0$
4: **for** *each bus service* $bs_j \in \mathcal{B}$ **do**
5: $\langle g_j, seq_j \rangle \leftarrow best\_seq\_startfrom(bs_j, \mathcal{S}, \mathcal{B}, M, \theta_p)$
6: **if** $g_j > g_{best}$ **then**
7: $g_{best} \leftarrow g_j$, $seq_{best} \leftarrow seq_j$
8: $\mathcal{S} \leftarrow \mathcal{S} \cup seq_{best}$, $\mathcal{B} \leftarrow \mathcal{B}/\{bs' | bs' \in seq_{best}\}$
9: **return** $\mathcal{S}$;
10: **Function 1** best_seq_startfrom($v_i, bs_s, \mathcal{S}, \mathcal{B}, M, \theta_p$):
11: $seq_{best} \leftarrow Oslash;$, $g_{best} \leftarrow 0$
12: **for** *each* $bs_j \in \mathcal{B}$ **do**
13: $g_j, seq_j \leftarrow best\_seq\_startfrom(v_i, bs_s, \mathcal{S}, \mathcal{B}, M, \theta_p)$
14: **if** $g_j > g_{best}$ **then**
15: $g_{best} \leftarrow g_j$, $seq_{best} \leftarrow seq_j$
16: **return** seq_{best};

The pseudo-code for SG search is listed in Algorithm 2. In each iteration, it picks the bus service sequence $seq \in Seq_{valid}$ with the highest marginal gain, such that $seq = \arg\max_{seq_i \in Seq_{valid}}(\mathcal{G}(\mathcal{S} \cup seq_i, \theta_p) - \mathcal{G}(\mathcal{S}, \theta_p))$ and assigns it to a physical vehicle. This process is continued until it assigns a sequence to each available vehicle. It uses a recursive search (i.e., $best\_seq\_startfrom(\mathcal{S}, \mathcal{B}, M, \theta_p)$) to find the sequence with the highest marginal gain. The search space of the recursive search is exponential but it is reduced to $O(|\mathcal{B}|)$ via a dynamic programming approach. More details will be presented next.

4.3 Optimization Techniques

Indexes. The computation of the marginal gain for scheduling one bus service is the first bottleneck faced by both algorithms proposed above. To address this issue, we use several indexes that are similar to the indexes proposed by [4], including *forward list*, *current marginal gains*, *backward list* and *served passengers*. The *forward list* is a mapping from $\mathcal{B}$ to a set of passengers from $\mathcal{P}$ who can be served by particular bus service in $\mathcal{B}$. The *current marginal gains* is a mapping from $\mathcal{B}$ to its current marginal gain, i.e., the number of passengers served by particular bus services. The *backward list* is a mapping from $\mathcal{P}$ to a set of bus services that can serve them. In addition, we keep the set of already served passengers as *served passengers* in a binary search tree. Whenever we include a new bus service in the schedule, we can find the list of passengers who can be served by the bus service using *forward list* and meanwhile ignore already served passengers and use *backward list* to find all the bus services which could have served those passengers. Then, we subtract them from *current marginal gains*. Further, we maintain two indexes, namely *next bus services* and *previous bus services*. For a given bus service bs, we often want to access the list of valid next bus services ($\{bs_i \in \mathcal{B} | \mathcal{A}(bs, bs_i, M) = 1\}$) and the list of valid previous bus services ($\{bs_i \in \mathcal{B} | \mathcal{A}(bs_i, bs, M) = 1\}$). Therefore, instead of iterating over all the bus services in $\mathcal{B}$, we pre-process the data and keep them in *next bus services* and *prev bus services*.

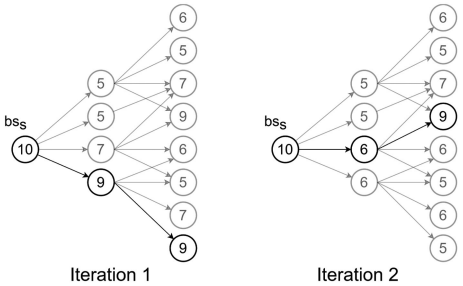


Fig. 1. Sequence gain for any $\mathcal{B}_s$ does not increase in each iteration.

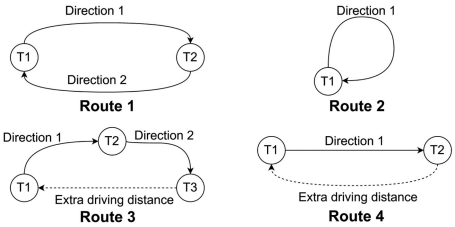


Fig. 2. Example forms of bus routes.

Dynamic Programming. Function $best\_seq\_start\_from(\mathcal{S}, \mathcal{B}, M, \theta_p)$ (invoked in Line 5 of Algorithm 2 and defined as Function 1 in Lines 10-16 of Algorithm 2) is a recursive function. It utilizes a *divide and concur* approach to find the sequence with the highest gain. Since $|Seq_{valid}|$ is exponential, it is *impossible* to solve this recursive function within a practical time duration. To address this issue, we use the well-known Dynamic Programming approach [28]. To be more specific, we use a hash map to cache the results. The key of the hash map is the starting bus service bs_s and the value of the hash map is the value returned by Function 1.

Priority Queues. In both algorithms presented above, we aim at finding the service with the highest marginal gain. Since the marginal gain of any service could change in each iteration, we have to scan all the services again to find the service with the highest gain. To avoid repeatedly re-calculating the gain for each service in each iteration, we use a priority queue data structure that has an insert and update time complexity of $O(\log(|\mathcal{B}|))$. Whenever we include a bus service bs to the schedule, we update the marginal gain of *only* affected services, i.e., all the services that can serve at least one passenger served by bs $\{bs' \in \mathcal{B} | \exists p \in \mathcal{P}, Ser(bs,' p, \theta_p) = 1 \wedge Ser(bs, p, \theta_p) = 1\}$.

Sequence Gain Upper Bound. Whenever we include a new service to the existing schedule, it will serve some passengers, which will reduce the marginal gains of all the bus services that could serve those passengers. Therefore, given a sequence seq formed by m ($m \geq 1$) bus services, its marginal gain will not increase but might be decreased in each iteration, as a newly scheduled bus service might serve some passengers who are initially served by bus services included into seq and hence reduce, but definitely not increase, the gain of the sequence seq.

We depict an example in Fig. 1 to facilitate the understanding. Each circle in the example represents a bus service and the number within the circle stands for the corresponding gain it can achieve at the current iteration. A direct link from a circle w.r.t. bus service bs_i to another circle w.r.t. bus service bs_j indicates that a vehicle is able to serve the bus service bs_j after it finishes the service bs_i, i.e., $\mathcal{A}(bs_i, bs_j, M) = 1$. It is noticed that, as compared with Iteration 1, some gains in Iteration 2 decrease their values. This is caused by the fact that a newly scheduled bus service in Iteration 1 serves certain passengers, which affects the gain of other bus services.

We use this fact to reduce the branching factor of Function 1. To be more specific, given a starting bus service bs_s, the first value g_{best} returned by Function 1 corresponds to the highest gain of any sequence starting from bs_s. Similarly, in Line 13 of Algorithm 2, the value of g_j does not increase but might decrease in subsequent iterations. Therefore, the value corresponding to g_j from the previous iteration provides an upper bound of g_j returned by subsequent iterations. We use g_j^{prev} and g_j to denote the value of g_j returned in the previous iteration and that returned in the current iteration respectively. Accordingly, we have $g_j \leq g_j^{prev}$.

In an iteration, given the starting bus service bs_s, we evaluate a potential sequence seq in the form of $\langle bs_s, bs_j, bs_{j+1}, \cdots \rangle$. The marginal gain of seq is $\mathcal{G}(\mathcal{S}_j \oplus^i bs_s, \theta_p) - \mathcal{G}(\mathcal{S}, \theta_p)$, with $\mathcal{S}_j = \mathcal{S} \cup \langle bs_j, bs_{j+1}, \cdots \rangle$. We want to compare it with g_{best} at Function 1. In addition, we have g_j^{prev} and marginal gain of bs_s: $\mathcal{G}(\mathcal{S} \oplus^i bs_s, \theta_p) - \mathcal{G}(\mathcal{S}, \theta_p)$. From the second function call afterwards, we have a value for g_j^{prev}. If $g_j^{prev} + \mathcal{G}(\mathcal{S} \oplus^i bs_s, \theta_p) - \mathcal{G}(\mathcal{S}, \theta_p) \leq g_{best}$, then it is guaranteed that $\mathcal{G}(\mathcal{S}_j \oplus^i bs_s, \theta_p) - \mathcal{G}(\mathcal{S}, \theta_p) \leq g_{best}$, i.e., the if condition listed in Function 1 will NOT be met. Accordingly, we can safely ignore bs_j (and hence all the sub-sequences that start from bs_j).

5 Experiment

In this section, we first present the experimental setup; we next conduct experiments to verify the improvements introduced by different optimization techniques presented in Section 4.3; we then compare the performance of the two newly proposed algorithms against three baselines under different parameter settings; we finally present a case study to demonstrate the potential savings (in terms of the number of bus vehicles saved and the total vehicle travelling time shortened) that could be brought by newly proposed algorithms in a real setup.

5.1 Experimental Setup

Dataset. We collect the bus route network of Singapore from Land Transport Datamall (https://datamall.lta.gov.sg/content/datamall/en.html). In total, there are 715 routes. The length of bus routes ranges from 2 to 105 bus stops, with an average length of 36.34 bus stops. 475 out of 715 routes (66.4%) are in the form of a loop, corresponding to example Route 1 and Route 2 in Fig. 2. In our experiments, we only consider those 475 routes that form closed loops. Other routes (e.g., Route 3 and Route 4 in Fig. 2) are not considered in our bus network as we are not very sure how vehicles serving those routes are mobilized. However, we want to highlight that most of the removed routes are not regular routes (e.g., only in operation during certain duration on certain days) and the 475 routes considered in our experiments cover more than 97% of the daily travel demand from passengers. In addition, we crawl Gothere API (https://gothere.sg/maps) to get the travel time (D) between every two consecutive stops of every route.

TABLE 2
Parameters and Their Settings

Parameter	Values
Model	M_{one}, M_{ss}, M_f
No Vehicles $\|\mathcal{V}\|$	1000, 2000, **3000**, 4000, 5000
w_{min} (in minutes)	6, 8, **10**, 12, 14
w_{max} (in minutes)	20, 30, **40**, 50, 60
θ_p (in minutes)	1, 3, **5**, 7, 9

For the passenger database ($\mathcal{P}$), we use the real bus touch-on record data on a single day (15 February 2016) in Singapore, which contains 3.4 million trip records. Each trip record captures the details of a real bus ride, including i) the corresponding bus route; ii) the IDs of the boarding and alighting bus stops; and iii) the timestamps of the boarding and alighting. Since the real trip records do not capture when the passengers reached the boarding bus stops, we assume passengers spend x minutes waiting for their buses, with x following a random distribution between 1 and 5 minutes.

For the bus service database ($\mathcal{B}$), we adopt minute as the minimum unit when we schedule bus services (e.g., if the daily bus service starts at 5:00 am in the morning, we can schedule bus services at 5:00 am, 5:01 am, 5:02 am and so on). We assume daily bus services are operated within a duration T (e.g., it is from 5:00 in the morning until 11:00 at night in our study), within which we generate all possible services for each bus route.

Key Parameters. Table 2 lists the parameters and their values, with their default values underlined.

Baselines.Our bus scheduling algorithms consider the number of passengers served within a waiting time threshold as the main optimization goal. To the best of our knowledge, previous work FASTCO [4] is the only work sharing a similar optimization goal. Hence, we have adapted FASTCO to support PASS and COST as a baseline, and compare its performance with the two methods proposed in this work, including bidirectional greedy (in short BG) and sequence greedy (in short SG). In addition, we also implement two general bus scheduling baselines namely *Top-K*, and *Fixed-Interval*. The three baselines are briefly explained below. Top-K picks the k bus sequences with the highest gains and allocates them to $|\mathcal{V}|$ vehicles with $k = |\mathcal{V}|$. Note, that when Top-K derives the gain of a sequence, it does not consider the impact of other sequences on its gain. Fixed-Interval schedules vehicles on all routes using a fixed time interval. In this paper, we set the interval to the maximum possible frequency achievable based on $\mathcal{V}$, i.e., $\frac{\sum_{r \in R}(D(r)+w_{min})}{|\mathcal{V}|}$. FASTCO [4] adopts a greedy algorithm to support an objective function that is similar to PASS under the model M_{ss}. We want to highlight that we have improved FASTCO (as compared with the original FASTCO proposed in [4]) from multiple aspects, including i) an initial allocation of vehicles to starting bus stops based on real demands, i.e., the number of vehicles allocated to a starting bus stop is proportional to the demand; ii) priority queue technique which manages to save the average running time of FASTCO by up to 96% (to be detailed in Section 5.2); and iii) the consideration of w_{min} and w_{max} to avoid the cases of assigning vehicles to services with high gains without considering the time gap and hence forcing vehicles to have long idle time that is not preferable.

Experiment Environment. All codes are implemented in C ++. Experiments are conducted on a machine with an 8-Core 1.8 GHz CPU and 16 GB memory running Ubuntu Operating System.

5.2 Optimization Techniques

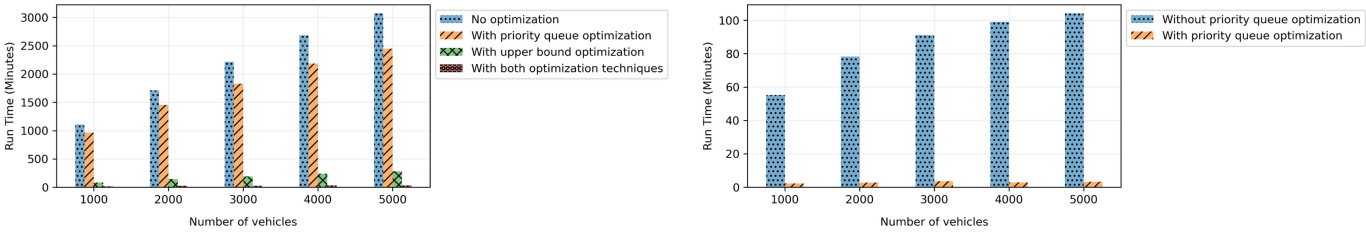
In our first set of experiments, we evaluate the contribution of different optimization techniques proposed in Section 4.3 to the search efficiency. We mainly adopt running time as the performance metric. There are in total four optimization techniques proposed. However, indexes and dynamic programming are essential, without which the search algorithms (especially SG) are not able to be performed. Consequently, we are only able to demonstrate the improvement achieved by *Priority Queues* and *Sequence Gain Upper Bound* by switching off each of these two optimization techniques when running the algorithms.

As these two techniques affect SG the most, we report the performance of SG with neither technique, the performance of SG with both techniques implemented and that of SG with either priority queue or sequence gain upper bound in Fig. 3a. As observed, both optimization techniques are able to improve the search efficiency significantly. For example, when there are 3000 vehicles, priority queue optimization is able to reduce the overall running time of SG from 2213 minutes to 1830 minutes; upper bound optimization could reduce SG's overall running time from 2213 minutes to 193 minutes. Though the upper bound optimization technique is observed to be more effective, we want to highlight that these two optimization techniques address different aspects of the problem. The priority queue reduces the number of route services we need to check in order to find the highest gain route service by keeping them in a heap. Upper bound optimization reduces the search space of sequences by avoiding low gain branches in the search tree. The effect of the Priority Queue optimization technique is negligible without the upper bound optimization because the search space of sequences is huge and the sequence search dominates the overall running time of SG. However, the saving achieved by Priority Queue optimization becomes more significant when the upper bound optimization is also adopted, e.g., using both techniques together can reduce the running time of SG from 2213 minutes to just 26 minutes when $|\mathcal{V}| = 3000$.

In addition, we want to highlight that the priority queue is applicable to FASTCO. As reported in Fig. 3b, it can help reduce the average running time of FASTCO by up to 96%.

5.3 Algorithm Performance Under PASS

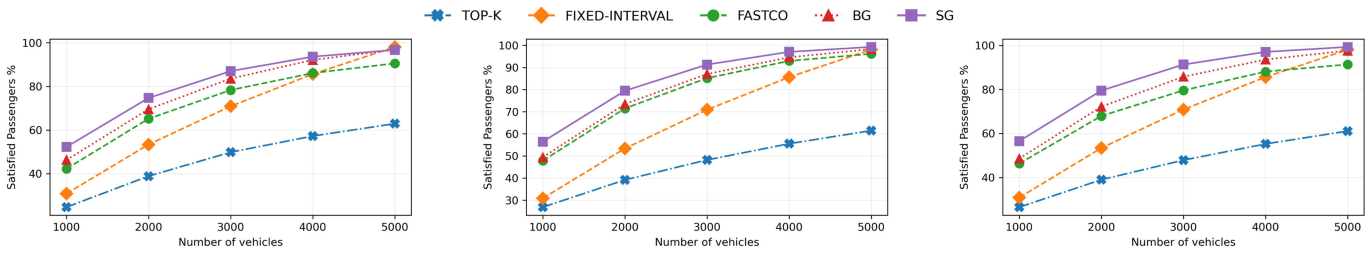
In our second set of experiments, we report the performance of the two newly proposed algorithms and that achieved by baselines when supporting PASS. Without loss of generality, we follow the definition of PASS and use the total number of satisfied passengers as the main performance metric, which is reported in the form of *the percentage of* $|\mathcal{P}|$ (as $|\mathcal{P}|$ is fixed). In the following, we first report the overall performances of different algorithms; we then evaluate the impact of major parameters, including w_{min}, w_{max}, and θ, on different algorithms. In each experiment, we adjust the value of



(a) Impact of optimization techniques on SG

(b) Impact of priority queue on FASTCO

Fig. 3. Impacts of different optimization techniques on the search performance of SG and FASTCO.

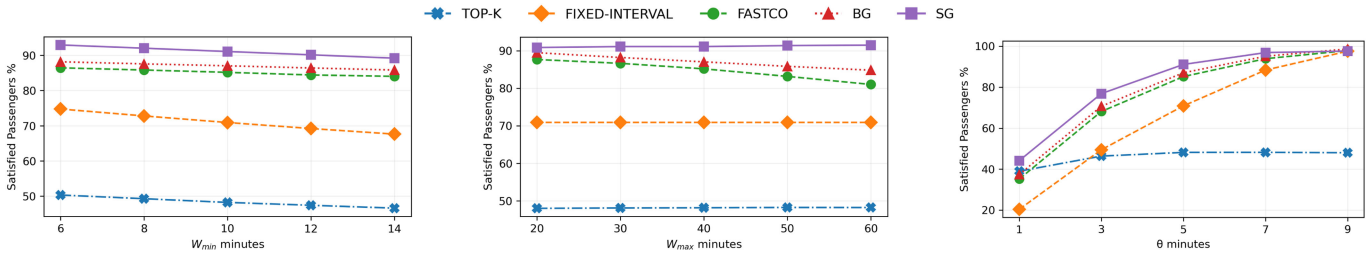


(a) PASS with M_{one}

(b) PASS with M_{ss}

(c) PASS with M_f

Fig. 4. Performances of PASS versus different $|\mathcal{V}|$.



(a) vs. w_{min}

(b) vs. w_{max}

(c) vs. θ

Fig. 5. The impacts of different parameters on the performances of PASS.

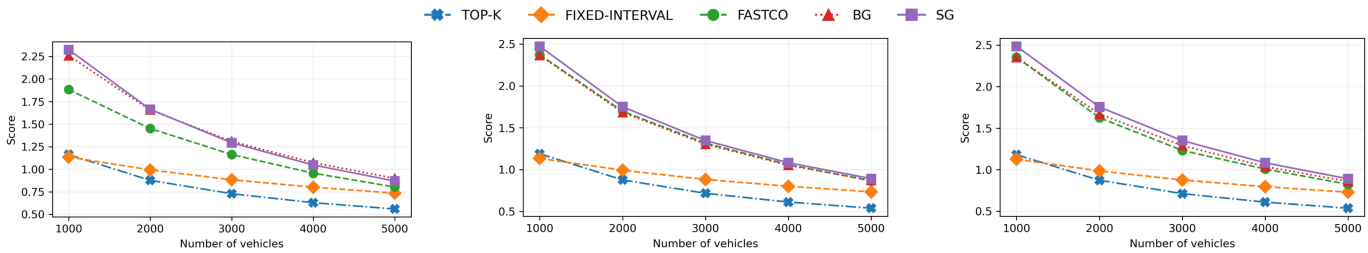
one parameter within the value range listed in Table 2, while keeping the other parameters at their default values.

Overall Comparison. The results with respect to the available number of vehicles are depicted in Fig. 4, under three different models. SG performs consistently the best for each model. This is likely contributed by the fact that SG evaluates the performance achieved by the entire sequence when scheduling the bus services, while all other algorithms focus on one bus service at a time. BG achieves the second-best performance for all models. Recall that BG schedules the bus services by completing one sequence first before constructing another sequence, which again demonstrates that focusing on one sequence is a better strategy. Fixed Interval improves its performance as $|\mathcal{V}|$ increases because we set the interval to the maximum frequency that can be achieved based on available $\mathcal{V}$. When $|\mathcal{V}| = 5000$, the interval is set to a very small number and all the passengers will be satisfied.

As the overall performance trends of different algorithms remain relatively stable under different models M, we only report the experimental results under the M_{ss} model when we evaluate the impacts of major parameters to save space and minimize redundancy.

Impact of w_{min}. Parameter w_{min} sets the minimum gap between two consecutive bus services assigned to a vehicle. Intuitively, its value determines the total number of services each vehicle can serve. As reported in Fig. 5a, when w_{min} increases its values, all algorithms serve fewer passengers. This is consistent with our expectation, as a larger w_{min} enforces all the vehicles to have a longer break after they finish the current bus services and hence the total number of services assigned to each vehicle becomes smaller.

Impact of w_{max}. The impact of w_{max} is slightly different. Parameter w_{max} determines the maximum gap between two consecutive bus services assigned to the same physical vehicle. A larger w_{max} allows each vehicle to stay longer before being assigned to another service, which provides more opportunities for each vehicle to be assigned to a service with higher marginal gain and meanwhile reduces the total number of services assigned to each vehicle. In other words, the increase of w_{max} brings both positive impact and negative impact on the performance of different algorithms. For FASTCO and BG, the negative impact is dominant, as shown in Fig. 5b. Those algorithms follow a greedy approach when selecting the next bus service to be served. Consequently, when vehicles are allowed to rest longer before being assigned to new services, they might be able to find services with higher marginal gains to be served next. However, the assigned services to be served next have a



(a) COST with M_{one} (b) COST with M_{ss} (c) COST with M_f

Fig. 6. Performance of COST versus different $|\mathcal{V}|$.

direct impact on the subsequent services to be assigned, and both algorithms only consider the next services but not those to be assigned in the future. Eventually, the physical vehicles serve less number of services in total, resulting in a reduction of the total gain they can achieve. However, SG considers the whole sequence, which provides room for the positive impact to demonstrate its power. When w_{max} becomes larger, there are more valid sequences to be considered. Since SG selects the highest gain sequence by checking all possible sequences, it gives more freedom to choose the service sequence that does not decrease the gain.

Impact of θ. Parameter θ is the threshold to simulate the tolerance level of passengers when waiting for bus services. As θ increases, all the passengers are willing to wait longer for buses before they become unsatisfied with the services. Naturally, it becomes easier for all the algorithms to serve more passengers within their waiting time threshold, as shown in Fig. 5c. However, its impact on Top-K is minor, as Top-K picks the highest gain sequences while the gain of a sequence is derived without considering the impact of other sequences.

5.4 Algorithm Performance Under COST

Next, we study the performance of different algorithms under COST. In addition to the number of passengers that can be served within a waiting time threshold θ_p by a given schedule $\mathcal{S}$, COST also considers the total travelling distances $\mathcal{M}(\mathcal{S})$ of physical vehicles as an indicator of the cost corresponding to the given schedule. Consequently, we adopt $\frac{\mathcal{G}(\mathcal{S},\theta_p)}{\mathcal{M}(\mathcal{S})}$, namely *score*, in the experimental results to be presented next, as the performance metric.

First, Fig. 6 reports the scores achieved by different algorithms under the three models, when the available number of physical vehicles changes. The primary observation is that unlike in PASS, the score does not increase but decreases as more vehicles become available. This is because COST is neither monotone nor submodular, as stated in Theorem 3.2. The count of satisfied passengers (i.e., $\mathcal{G}(\mathcal{S},\theta_p)$) is submodular but not the total vehicle driving time $\mathcal{M}(\mathcal{S})$. Consequently, when more vehicles become available, the number of satisfied passengers increases at a slower pace, while driving time increases without a constraint, resulting in a decreased score.

Second, SG is still the best performer while its advantage over others becomes less significant, especially with more vehicles. According to our observations, the reason for this is that SG tends to cover more passengers by driving a long distance. We also report both $\mathcal{G}(\mathcal{S},\theta_p)$ and $\mathcal{M}(\mathcal{S})$ in Table 3 for $|\mathcal{V}| = 3000$. Note that the satisfied passenger percentage (i.e., $\mathcal{G}(\mathcal{S},\theta_p)$) of SG and BG is significantly higher than that of other baselines (including FASTCO).

In the following, we investigate the impacts of different parameters on the performance of various algorithms when supporting COST. Again, we focus on model M_{ss} only for space saving.

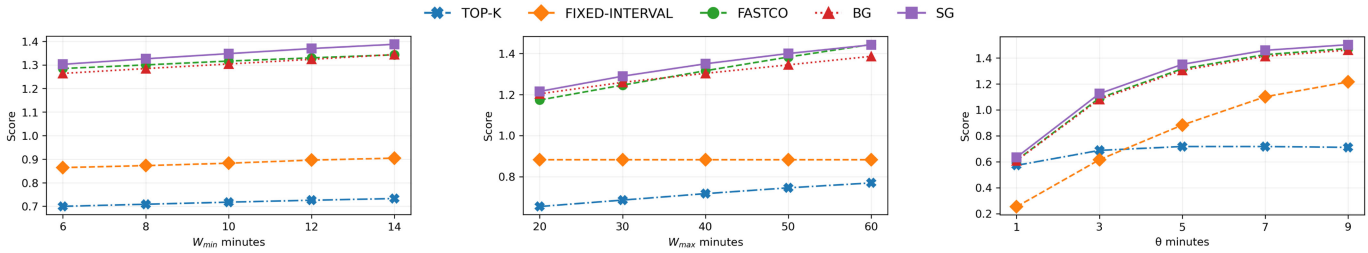
Impact of w_{min}. The impact of w_{min} is significantly different from that in PASS. It is observed that w_{min} has a positive impact on the score, as shown in Fig. 7a. This is consistent with our expectations. As w_{min} increase its value, vehicles are forced to rest longer between two consecutive services. Accordingly, it helps reduce the total driving time of each vehicle. Even though it could also decrease the satisfied passenger count as happened in PASS, the impact of driving time seems to be significant enough to increase the score.

*Impact of w_{max}.*The impact of w_{max} is also different from that in PASS but again is still within our expectations. w_{max} has a positive impact on the score, as reported in Fig. 7b. w_{max} is the longest allowed duration between two consecutive services. The increase of w_{max} will allow all algorithms to pick services with a higher score while resting longer between two consecutive services.

Impact of θ. The impact of θ under COST is almost the same as it under PASS, as shown in Fig. 7c. As θ increases, all passengers are willing to wait longer for a service; therefore, it becomes easier for all algorithms to serve more passengers with a shorter total driving time, resulting in an increased score.

TABLE 3
Duration and Satisfied Passengers % of COST With 3000 Vehicles

Model	Algorithm	Duration hours ($\times 10^3$)	Satisfied Passengers %
M_{one}	TOP-K	34.96	44.96
	FIXED-IN	45.52	70.90
	FASTCO	38.29	78.56
	BG	35.80	82.71
	SG	36.87	84.14
M_{ss}	TOP-K	34.85	44.11
	FIXED-IN	45.52	70.90
	FASTCO	36.97	85.87
	BG	37.90	87.18
	SG	36.69	87.25
M_f	TOP-K	34.88	43.81
	FIXED-IN	45.88	70.90
	FASTCO	37.62	81.78
	BG	38.43	87.46
	SG	36.74	87.49



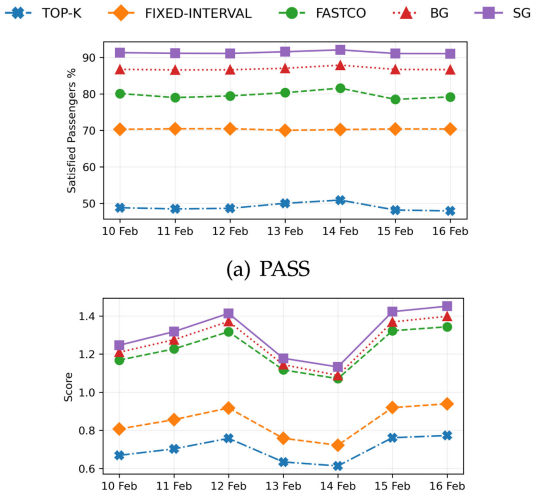
(a) vs. w_{min} (b) vs. w_{max} (c) vs. θ

Fig. 7. The impacts of different parameters on the performances of COST.

5.5 Sensitivity

All the previous experiments are based on a passenger dataset that is derived from the real commuting records on a single day (15 February 2016). In order to verify that the performance comparisons among different algorithms and our observations are reliable, we report the performance of different algorithms using different passenger datasets. To be more specific, we construct 7 passenger datasets, again based on real commuting records on a single day in one week from Wednesday 10 February 2016 to Tuesday 16 February 2016. The results w.r.t. $|\mathcal{V}| = 3000$ under M_{ss} are reported in Fig. 8.

It is observed that the results corresponding to different days do not differ much. The only noticeable deviation is that COST has a slightly lower score on 13 February and 14 February. This is because 13 February and 14 February are weekends, and the corresponding passenger databases are relatively smaller.



(a) PASS

(b) COST

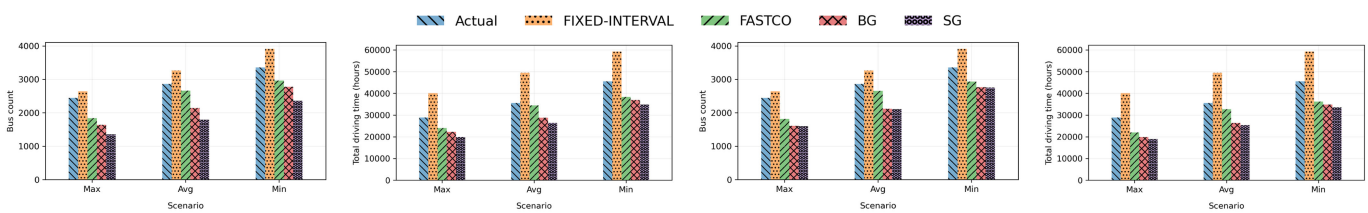
Fig. 8. Sensitivity test of different algorithms of commuting demands (10-16 February 2016, $|\mathcal{V}| = 3000$) under M_{ss}.

5.6 Case Study

In our following set of experiments, we compare our algorithms with the actual vehicle allocation of the Singapore bus network. We first collect the frequency of the actual schedule from Transitlink eGuide (https://www.transit- link.com.sg/eservice/eguide/service_idx.php); we then derive the minimum vehicle count required, say v_{req}, to operate the buses under this frequency. Thereafter, we calculate the performance (e.g., the number of satisfied passengers under PASS) of the actual bus services operated based on the fixed schedules served by v_{req} vehicles. Considering the performance w.r.t. actual schedule as the target, we execute our proposed algorithms to find out the required number of vehicles in order to achieve the same performance. This is to demonstrate the potential savings that could be achieved if bus services are scheduled in a dynamic manner guided by real travel demands (as supported by the two algorithms proposed in this paper) instead of fixed schedules.

Since we do not know the exact number of vehicles used by the Transport Authority, we calculate it according to the following approach. We first assume that they do not share vehicles between multiple routes, so vehicles have to operate as a loop within their route repeatedly (model M_{one}). The transit link website provides time intervals, in terms of ranges, between consecutive services for peak and off-peak hours, as shown in Table 4. Accordingly, we consider three actual scenarios namely *Min*, *Avg*, and *Max*, which correspond to minimum, average, and maximum intervals respectively. For example, given a frequency of 7-13 minutes, *Min*, *Avg*, and *Max* will schedule a service every 7, 10 and 13 minutes. We calculate the required number v_{req} of vehicles to operate in given frequencies for each route, and add them up based on $v_{req} = \sum_{r \in R}(D(r) + w_{min})/T_r$, whose values are reported in Table 5.

T_r is the time interval between two services of the route r collected from Transitlink, i.e., Frequency listed in Table 4. v_{req} is the required number of vehicles, with its values corresponding to different scenarios/time periods reported in



(a) Number of vehicles PASS (b) Total driving time PASS (c) Number of vehicles COST (d) Total driving time COST

Fig. 9. Comparison with actual scenario.

TABLE 4
Transitlink Frequency Information on Exam Route 7

Route 7 Period	Frequency (Minutes) Direction 1	Direction 2
06:30 - 08:30	9-11	8-11
08:31 - 16:59	6-13	8-12
17:00 - 19:00	9-12	7-13
After 19:00	8-13	11-13

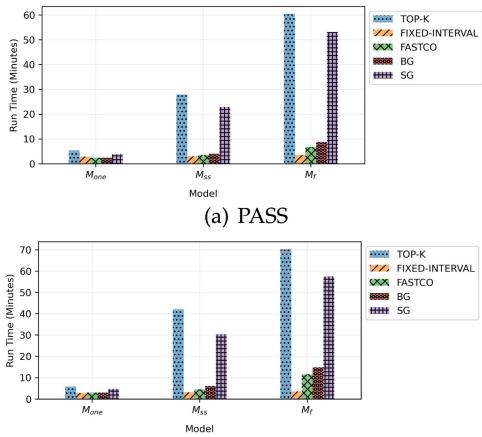
TABLE 5
Minimum Number of Vehicles Required to Operate

Period	Actual Scenarioes Min	Avg	Max
0630 - 0830	3356	2860	2444
0831 - 1659	3265	2373	1842
1700 - 1900	3031	2549	2153
after 1900	2460	1993	1668
v_a	3356	2860	2444

Table 5. As v_{req} is related to T_r that has different values at different periods, we consider the largest v_{req}, indicated by v_a in Table 5. Given v_a vehicles serving the bus routes based on the fixed frequencies, we are able to derive the performance (e.g., the total number of passengers served under PASS).

We then derive the numbers of vehicles required by other algorithms, in order to achieve the same performance, with the numbers and the corresponding total driving duration (of all the vehicles) reported in Fig. 9. It is observed that our algorithms achieve significant savings. For example, as compared with the actual scenario Avg, SG reduces the number of required vehicles by 37% and shortens the total driving time by 26% for PASS; it reduces vehicles by 26% and shortens the total driving time by 29% for COST. One important thing to notice is that FASTCO performs significantly poorer than the actual scenario in all cases. According to our observation, this is highly related to the initial vehicle distribution. For example, FASTCO initially allocates vehicles to the highest gain route services, leading to uneven distribution of vehicles throughout the network. Top-K tends to stagnate around 70-60% of the passenger satisfaction even under 10000 vehicles, and still, it does not reach the satisfaction level of the actual scenario. Therefore we have excluded it from the case-study section.

In the previous comparison, we allow algorithms to utilize all vehicles anytime. However, in the actual scenario, not all the vehicles are operated during the off-peak period.



(a) PASS

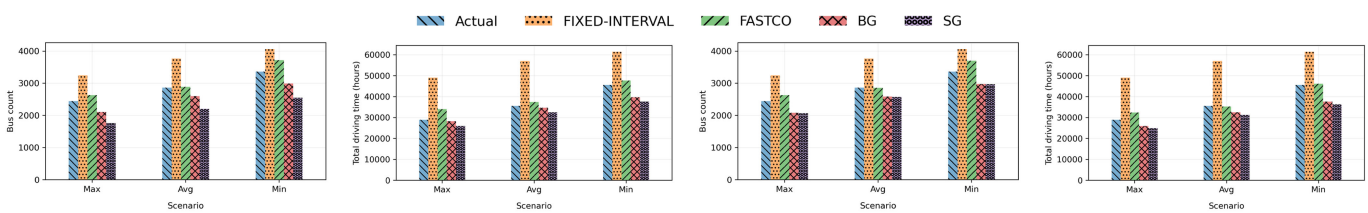
(b) COST

Fig. 11. Running time of different algorithms.

Thus, it gives a competitive advantage to our algorithms. To provide a fairer comparison, we perform a second experiment by assuming that the actual schedule operates at the highest frequency throughout the whole day, i.e., all periods listed in Table 4 share the same frequency. The results are reported in Fig. 10. In this setup, SG also achieves a 23% reduction in the required vehicle count and a 9% reduction in the total travel time under PASS. In terms of COST, SG is able to reduce the vehicle counts by 10% and shorten the total driving time by 12%.

5.7 Running Time

In the last set of experiments, we evaluate the running time of all the algorithms. Fig. 11 reports the experimental results with 5000 vehicles. Out of all five algorithms, Top-K incurs the highest running time, followed by SG, while Fixed Interval is the simplest and it incurs the shortest running time. In addition, the model M_{one} has a smaller running time, because the vehicles are only shared within the same route. Accordingly, its search space for the best sequence is much smaller than the other two models. Consistent with our expectation, the model M_f requires the longest running time as its search space is the largest among the three models considered in this work. However, we want to highlight that SG, though requiring a longer running time, is able to achieve the best performance with a running time very affordable (e.g., it can be completed within an hour for a passenger database that contains over 3 million of trip data and a bus network that has 5000+ bus stops and 400+ bus routes).



(a) Number of vehicles PASS (b) Total driving time PASS (c) Number of vehicles COST (d) Total driving time COST

Fig. 10. Comparison with the modified actual scenario.

6 Conclusion

In this paper, we study the problem of bus scheduling under the constraint of limited physical vehicles. We consider not only the number of satisfied passengers within a given waiting time threshold but also the bus service operational cost as our optimization objectives and formulate PASS and COST. We propose two different algorithms to perform the scheduling and several optimization techniques to further reduce the time complexity of our algorithms. A comprehensive evaluation based on a real passenger demand dataset has been performed to demonstrate the advantages of our algorithms. In the near future, we would like to propose a prediction model that is able to predict the travel demand from the passengers based on historical demand data to provide input to our algorithms in real-time with high accuracy to further enhance the work presented in this paper.

Acknowledgments

Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not reflect the views of the Ministry of Education, Singapore.

References

[1] D. J. Sun, Y. Xu, and Z.-R. Peng, "Timetable optimization for single bus line based on hybrid vehicle size model," *J. Traffic Transp. Eng.*, vol. 2, no. 3, pp. 179–186, 2015.
[2] L. dell'Olio, A. Ibeas, and F. Ruisánchez, "Optimizing bus-size and headway in transit networks," *Transportation*, vol. 39, no. 2, pp. 449–464, 2012.
[3] P. Delle Site and F. Filippi, "Service optimization for bus corridors with short-turn strategies and variable vehicle size," *Transp. Res. Part A Policy Pract.*, vol. 32, no. 1, pp. 19–38, 1998.
[4] S. Mo, Z. Bao, B. Zheng, and Z. Peng, "Towards an optimal bus frequency scheduling: When the waiting time matters," *IEEE Trans. Knowl. Data Eng.*, early access, Nov. 6, 2020, doi: 10.1109/TKDE.2020.3036573.
[5] N. Lin, W. Ma, and X. Chen, "Bus frequency optimisation considering user behaviour based on mobile bus applications," *IET Intell. Transport Syst.*, vol. 13, no. 4, pp. 596–604, 2019.
[6] Z. Wang, J. Yu, W. Hao, and J. Xiang, "Joint optimization of running route and scheduling for the mixed demand responsive feeder transit with time-dependent travel times," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 4, pp. 2498–2509, Apr. 2021.
[7] A. Ceder, "Bus frequency determination using passenger count data," *Transp. Res. Part A Gen.*, vol. 18, no. 5–6, pp. 439–453, 1984.
[8] A. Ceder, "Methods for creating bus timetables," *Transp. Res. Part A Gen.*, vol. 21, no. 1, pp. 59–83, 1987.
[9] Y. Zhang, R. Su, Y. Zhang, and N. S. G. Guruge, "A multi-bus dispatching strategy based on boarding control," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 6, pp. 5029–5043, Jun. 2022.
[10] G. F. Newell, "Dispatching policies for a transportation route," *Transp. Sci.*, vol. 5, no. 1, pp. 91–105, 1971.
[11] V. Tom and S. Mohan, "Transit route network design using frequency coded genetic algorithm," *J. Transp. Eng.*, vol. 129, no. 2, pp. 186–195, 2003.
[12] B. Yu, Z. Yang, and J. Yao, "Genetic algorithm for bus frequency optimization," *J. Transp. Eng.*, vol. 136, no. 6, pp. 576–583, 2010.
[13] H. Martínez, A. Mauttone, and M. E. Urquhart, "Frequency optimization in public transportation systems: Formulation and metaheuristic approach," *Eur. J. Oper. Res.*, vol. 236, no. 1, pp. 27–36, 2014.
[14] S. Hassold and A. Ceder, "Multiobjective approach to creating bus timetables with multiple vehicle types," *Transp. Res. Rec.*, vol. 2276, no. 1, pp. 56–62, 2012.
[15] G. Antonides, P. C. Verhoef, and M. Van Aalst, "Consumer perception and evaluation of waiting time: A field experiment," *J. Consum. Psychol.*, vol. 12, no. 3, pp. 193–202, 2002.
[16] 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 Qual. Life Outcomes*, vol. 5, no. 1, pp. 1–6, 2007.
[17] M. Cantwell, B. Caulfield, and M. O'Mahony, "Examining the factors that impact public transport commuting satisfaction," *J. Public Transp.*, vol. 12, no. 2, 2009, Art. no. 1.
[18] F. Cevallos and F. Zhao, "Minimizing transfer times in public transit network with genetic algorithm," *Transp. Res. Rec.*, vol. 1971, no. 1, pp. 74–79, 2006.
[19] R. Oldfield and P. Bly, "An analytic investigation of optimal bus size," *Transp. Res. Part B Methodological*, vol. 22, no. 5, pp. 319–337, 1988.
[20] W. Jin and W. Wu, "Single-line transit mixed scheduling model with financial constraints," *J. Jilin Univ. Eng. Technol. Ed.*, vol. 44, no. 1, pp. 54–61, 2014.
[21] X. Tian, B. Zheng, Y. Wang, H.-T. Huang, and C.-C. Hung, "Tripdecoder: Study travel time attributes and route preferences of metro systems from smart card data," *ACM/IMS Trans. Data Sci.*, vol. 2, no. 3, pp. 1–21, 2021.
[22] L. Li, H. K. Lo, W. Huang, and F. Xiao, "Mixed bus fleet location-routing-scheduling under range uncertainty," *Transp. Res. Part B: Methodological*, vol. 146, pp. 155–179, 2021.
[23] K. An and H. K. Lo, "Two-phase stochastic program for transit network design under demand uncertainty," *Transp. Res. Part B Methodological*, vol. 84, pp. 157–181, 2016.
[24] L. Li, H. K. Lo, and F. Xiao, "Mixed bus fleet scheduling under range and refueling constraints," *Transp. Res. Part C Emerg. Technol.*, vol. 104, pp. 443–462, 2019.
[25] I. Constantin and M. Florian, "Optimizing frequencies in a transit network: A nonlinear bi-level programming approach," *Int. Trans. Oper. Res.*, vol. 2, no. 2, pp. 149–164, 1995.
[26] Z. Gao, H. Sun, and L. L. Shan, "A continuous equilibrium network design model and algorithm for transit systems," *Transp. Res. Part B Methodological*, vol. 38, no. 3, pp. 235–250, 2004.
[27] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions - I," *Math. Prog.*, vol. 14, no. 1, pp. 265–294, 1978.
[28] R. Bellman, "The theory of dynamic programming" *Bulletin Amer. Math. Soc.*, vol. 60, no. 6, pp. 503–515, 1954.

Janaka Chathuranga Brahmanage received the BSc degree in engineering (Hons) computer science from the University of Moratuwa, Sri Lanka. He is a research engineer with the School of Computing and Information Systems, Singapore Management University, Singapore. Prior to that, he co-founded Stack Technologies in Sri Lanka. His main research interests include artificial intelligence & data science and data management & analytics

Thivya Kandappu received the PhD degree in electrical engineering and telecommunications from the University of New South Wales, Australia, in 2014. She is currently an assistant professor with the School of Computing and Information Systems, Singapore Management University, Singapore. The specific themes of her research interests are urban transportation modelling & planning, human behaviour analytics for smart-urban systems and mobile/pervasive computing.

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