An Incremental SAT-Based Approach for Solving the Real-Time Taxi-Sharing Service Problem

Aolong Zha, Qiong Chang and Itsuki Noda

November 11, 2020

An Incremental SAT-Based Approach for Solving the Real-Time Taxi-Sharing Service Problem

Aolong Zha, Qiong Chang, and Itsuki Noda

National Institute of Advanced Industrial Science and Technology, Tsukuba, Japan
{aolong.zha,jou.kyuu,i.noda}@aist.go.jp

Abstract. This paper deals with a combinatorial optimization problem that models real-time taxi-sharing services; this problem has gained much attention in the areas of both artificial intelligence and computational social science. When a passenger α sends a demand to a taxi control center, the center tries to find an optimal solution to assign an appropriate taxi and re-plan its route for the demand so that the service can minimize the sum of the planned travel time for all served passengers (including α) who are allocated to the same taxi. Because finding a feasible solution to the real-time taxi-sharing service problem is $\mathcal{NP}$-hard, most previous studies have focused on developing a semi-optimization algorithm based on well-known metaheuristics. We propose a novel algorithm based on incremental Boolean satisfiability solving with an extensible framework, to optimize the taxi allocation for demands occurring in real time. Our formulation is also suitable for other objective functions with small changes to the weighted constraints. The experiments, which are based on real-map data using a modified simulation of urban mobility, show that our new approach achieves higher performance with a reasonable computational cost compared to an existing insertion method that has been functioned in a real service application.

Keywords: Taxi-sharing · Route planning · Combinatorial optimization.

1 Introduction

In modern cities, the increasing demand for personal mobility requires good transport services that provide a friendlier alternative to private cars while maintaining usability. Recently, several ride-sharing systems have emerged, that use online-enabled platforms to connect drivers who have empty seats in their cars to people who need a ride. In this paper, the problem of planning and scheduling a set of taxis to transport a number of passengers from their pick-up locations to their drop-off locations at specific times is called the *real-time taxi-sharing service problem* (RTSS). The motivation for this work is that, if RTSS can be solved efficiently, providing a convenient and flexible mode of transportation to passengers would be possible.

RTSS is a synthetic combinatorial optimization problem, which is not only strongly related to the *multiple traveling salesmen problem* [5] with time windows but also further considers the constraints of the *vehicle routing problem* [29] with pick-up and delivery [6]. RTSS is also very similar to the *integrated dial-a-ride problem* (IDARP) [23], which integrates the ride-sharing services provided by a set of designated vehicles with their already existing fixed routes. Both RTSS and IDARP are based on real-time on-demand allocation but differ in that, in the latter, each vehicle is limited to travel from its source to destination with possible detours.

To the best of our knowledge, there are no algorithms that can efficiently solve RTSS in the worst-case scenario (e.g., peak time in a largely populated area) when fully considering all potential ride-sharing. For example, for all taxis, re-planning of all of their unfinished and shiftable (i.e., not on-board yet) demands, or further dealing with transfer situations [8]. Because the high-frequency demand occurrence and the limited taxi speed lead to a sustained and rapid growth of the unfinished job list for each taxi, which reflects execution slow down on an increasing search space. It is significant that a real-time application has a reasonable execution time. Therefore, in this paper, we focus on allocating a taxi and re-planning its route in real time only for fresh demands without changing other taxi routes.

1.1 Related Work

RTSS is usually formulated as an *integer programming* (IP) problem. Because finding a feasible solution to RTSS is $\mathcal{NP}$-hard, as shown in Santos and Xavier [26], most previous studies have focused on developing semi-optimization algorithms based on well-known heuristics, such as the *genetic algorithm* or *local search*. Baldacci et al. [3] proposed both an exact and a heuristic method to solve the car-pooling problem based on two IP formulations. In Herbawi and Weber [13], a generational genetic insertion heuristic to solve the IP formulation of a dynamic ride-matching problem with a multi-objective function was presented. In Agatz et al. [1], a simplified real-time ride-sharing problem with a simulation environment based on real-world data, in which each driver can pass through only one pick-up and delivery location, was modeled as a maximum-weight bipartite matching problem and was solved using the optimization software CPLEX. Alonso-Mora et al. [2] also simplified the pick-up and delivery problem to a segmented request-vehicle graph, and reduce it into an optimal assignment problem by IP modeling. Huang et al. [15], presented a method for large-scale real-time ride-sharing and used a real dataset to compare it to some general methods, such as the branch-and-bound algorithm and the IP approach. In Xu et al. [31], a heuristic based on user historical data for consideration of both immediate rewards and future gains was utilized to optimize driver-order dispatch.

In addition, Gørtz [11] analyzed the computational complexity of preemptive finite capacity dial-a-ride, and Simonin and O'Sullivan [27] provided a basis for the development of efficient methods and heuristics. These two studies are based on Boolean satisfiability constraints. According to the results of MaxSAT Evaluation 2017, overall, the performances of maximum satisfiability (MaxSAT) solvers are much better than those of Cplex, MaxSAT techniques may be comparable to IP method for solving the combinatorial optimization problems.[1]

1.2 Contributions

Instead of the traditional IP modeling, we present an approach that is based on Boolean satisfiability testing for solving RTSS. Our approach can be divided into the following two parts: (1) a MaxSAT encoding, and (2) an incremental algorithm. We separate the constraints only related to 0-1 variables in RTSS, and formulate them into the first part. The rest of the constraints are handled by the second part. According to the particularity of RTSS, we also propose a new encoding method for *Hamiltonian path* constraint and give a proof of its correctness, which are the theoretical contribution of this paper. In our implementation, we modify a well-know traffic simulator and the proposed method is compared with an existing method that has been applied in a real RTSS system. Source code and external libraries for our experiments are available at `https://github.com/ReprodSuplem/RTSS/`.

2 The RTSS Problem

This is a system in which a fresh demand is sent to a taxi control center. The center tries to find an optimal solution of RTSS, i.e., how to assign an appropriate taxi and re-plan its route to take into account current demands and to minimize the sum of the planned trip times of all passengers who are allocated to the taxi corresponding to the new demand.[2] Then, the demand will be accepted if a solution is found; otherwise, the demand will be rejected. Each demand contains a pick-up point L, a drop-off point R, an earliest departure time D, a latest arrival time T, and the number of passengers N. Note that, in this paper, each occurring demand will be pushed into a chronological queue. Once a demand is rejected or accepted, it will be popped out of this queue. Therefore, all demands will be orderly processed one by one.

Mathematically, in RTSS, we are given a *complete directed graph* $G = (f, V, E)$ for each on-duty taxi, where $f : E \to V \times V$. The set of vertices V represents all the locations, which consist of the current point O, the unfinished pick-up points P_i, and the drop-off points Q_i of the accepted demands, as well as the pick-up point L and drop-off point R of the new demand. We denote the set of the unfinished pick-up points as Φ, where $\Phi = \{P_1, P_2, \ldots\}$, and denote the set of unfinished drop-off points as Ψ, where $\Psi = \{Q_1, Q_2, \ldots\}$. Note that the size of Φ, i.e., $|\Phi|$, must be less than or equal to $|\Psi|$ and that the number of on-board passengers is $|\Psi| - |\Phi|$. Each edge in the set of E indicates a possible link, denoted as $\langle v_i, v_j \rangle$, where $v_i, v_j \in V$, that directly connects the corresponding

[1] `https://maxsat-evaluations.github.io/2017/results/complete/weighted/table-extended.html`

[2] The planned trip time of a passenger is the total cost time from the pick-up point to the drop-off point.

combinatorial pair of location points.[3] Assume that the cost time of such a link, denoted as $w_{\langle v_i, v_j \rangle}$, can be easily estimated. For each taxi, all of the cost times of the links comprise an *asymmetric time matrix* shown in Fig. 1.

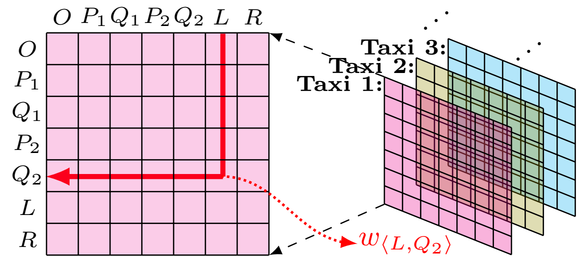


Fig. 1. An illustration of the asymmetric time matrix corresponding to each taxi; and each cell of matrix indicating a cost time of corresponding link.

We define a taxi route as a vector of head-to-tail connected links that represents a sequence of location points consisting of the vertices in V. RTSS involves finding a route for each taxi that is subject to the following constraints.

Order constraint: The first point of a route must be the taxi's current location and a served passenger must be dropped off after being picked up.

Deadline constraint: Each served passenger whose demand has been accepted but is unfinished must be picked up at the set location point P after the corresponding earliest departure time D and must be delivered to the set location point Q before the corresponding latest arrival time T.

Capacity constraint: Each taxi cannot carry more passengers than its capacity.

Monopoly constraint: The pick-up point L and the drop-off point R of the new demand must be assigned to the same taxi.

HP constraint: For the taxi that is assigned the new demand, its route must be a Hamiltonian path (HP), denoted $H = (\delta, V, \mathcal{E})$, while for each of the taxis that are not assigned to the new demand, each of their routes must also be an HP, denoted $H' = (\delta, \mathcal{V}, \mathcal{E})$, where $H, H' \subseteq G$, $\mathcal{V} = V \setminus \{L \cup R\}$, $\mathcal{E} \subseteq E$, and δ are the constraints that guarantee that the route is an HP.

The typical optimization objective function is based on the model of the *shortest path problem* (SPP) [10] to ensure energy conservation and environmental protection. However, the typical objective maybe leads to highly imbalanced solutions in some cases (e.g., when all the requested pick-up points and drop-off points occur on a straight path). If no restriction is imposed on the number of points to be visited by each taxi, RTSS is an ill-posed problem: the optimal solution is obtained when one taxi takes all the demands. Therefore, in this paper,

[3] Link $\langle v_i, v_j \rangle$ is the shortest direct path from v_i to v_j.

we focus on minimizing the sum of the travel time of all served passengers who are allocated to the taxi that corresponds to the fresh demand; such optimization can effectively offer better service. Formally, the objective of most existing research is to minimize the total travel time of all served passengers (i.e., the *min-sum* objective). Considering that continuous demand allocation occurrs in a time series, our objective can be regarded as the min-max objective, which is equivalent to minimize the time that the taxi spent most of the time; such a *min-max* objective function is more valuable to industry research of taxi-sharing service. We will show that these two different criteria can be easily adapted via small modifications to the weighted constraints.

3 MaxSAT Encoding

A well-known *Boolean satisfiability problem* (SAT) was the first problem shown to be $\mathcal{NP}$-complete [9]; in this problem, determining whether there exists a truth assignment that satisfies a given Boolean formula is necessary.[4] Typically, a Boolean formula is expressed in *conjunctive normal form* (CNF), which consists of a conjunction (i.e., logic *and*) of one or more clauses. A *clause* is a disjunction (i.e., logic *or*) of one or more literals, and a *literal* is an occurrence of a Boolean variable or its negation (i.e., logic *not*).

MaxSAT is an optimal version of SAT [17]. In the *weighted partial* MaxSAT, the problem instance is typically expressed as a set of hard and soft clauses, where each soft clause has a bounded positive numerical *weight*. The problem is to find a model that satisfies all the hard clauses and maximizes the sum of the weights of the satisfied soft clauses. Formally, we denote a MaxSAT formula as $\mathcal{F} = C_1 \land \ldots \land C_m \land (C_{m+1}, w_1) \land \ldots \land (C_{m+n}, w_n)$, where the first m clauses are hard and the rest are soft. Solving a MaxSAT instance $\mathcal{F}$ amounts to finding an assignment that satisfies all $\bigwedge_{i=1}^{m} C_i$ and maximizes $\sum_{i=1}^{n}(w_i C_{m+i})$. Technically, $\mathcal{F}$ can be solved via the resolution of a sequence of SAT instances associated with *pseudo-Boolean* (PB) constraints encoding [25] as follows: $\mathcal{F}_k = C_1 \land \ldots \land C_m \land \mathsf{CNF}_{\mathrm{PB}}(\sum_{i=1}^{n}(w_i \neg C_{m+i}) < k)$. $\mathcal{F}_k$ is a CNF that is satisfiable if and only if $\mathcal{F}$ has an assignment $\mathcal{A}$ whose optimum answer (i.e., $\sum_{i=1}^{n}(w_i \mathcal{A}(C_{m+i}))$) is greater than $\sum_{i=1}^{n} w_i - k$. If the optimal assignment of $\mathcal{F}$ is $\mathcal{A}$ and its optimum answer is $\sum_{i=1}^{n} w_i - k_{opt}$, then the SAT problem $\mathcal{F}_k$ for $k \geq k_{opt}$ is satisfiable, while the problem for $k < k_{opt}$ is unsatisfiable. Therefore, searching for the optimum answer for $\mathcal{F}$ means finding the precise location of this transition from satisfiable to unsatisfiable CNF formulas. MaxSAT solvers based on the above-mentioned approach are usually called *satisfiability-based* solvers, which is the term used in the remainder of this paper.

Definition 1. *Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n Boolean variables. We have the following naive CNF encodings corresponding to three special cases of cardinality constraints for X as follows.*

[4] A truth assignment is a function $\mathcal{A} : X \to \{0, 1\}$, where X is a set of Boolean variables and $\mathcal{A}$ is regarded as a conjunction of all elements in X.

At most one constraint: $AMO(X) = \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} (\neg x_i \vee \neg x_j)$.
At least one constraint: $ALO(X) = \bigvee_{i=1}^{n} x_i$.
Exactly one constraint: $EO(X) = AMO(X) \wedge ALO(X)$.

3.1 Two Types of Boolean Variables

In our MaxSAT encoding, there are two different types of Boolean variables that will be used. One type indicates whether a link $\langle a, b\rangle$ is a sub-path of its taxi route. For the sake of convenience, we also denote this type of Boolean variable as $\langle a, b\rangle$. For example, if $\langle a, b\rangle = 1$, then we know that the taxi has arranged a route in which it will go directly to location b when it departs from location a; otherwise, location b cannot be the next arrival point from location a. The other type of Boolean variable is denoted as $\overrightarrow{a, b}$, which indicates a directed reachability. For example, $\overrightarrow{a, b} = 1$ means that the taxi has arranged a route where it can access location b via location a. In these two types of Boolean variables, the pair of parameters a and b cannot be identical vertices in V for a specific taxi; this is restricted by blocking the corresponding variables. For each pair of these two types of Boolean variables, we have a related implication rule as follow:

$$\langle a, b\rangle \rightarrow \overrightarrow{a, b}. \tag{1}$$

3.2 HP Constraint for RTSS

There are several previous works related to encoding HP constraint to CNF, which includes the *log encoding* proposed in Iwama and Miyazaki [16], the *absolute encoding* presented in Hoos [14] and the *relative encoding* described in Prestwich [24]. In contrast with encoding a naive HP constraint, solving RTSS need to further consider encoding objective function that concerns the relative positions of taxi's via points with respect to each other in its permutation. Therefore, our encoding is based on the relative encoding, while holding a different idea in ensuring edge connections.

To encode the HP constraint, we constructed a connective network of the two types of Boolean variables with the following four laws.

Chain transition law: For each taxi, for any three different locations a, b, and c, where $a, b, c \in V$, we have

$$\begin{aligned}
&\overrightarrow{a, b} \wedge \overrightarrow{b, c} \rightarrow (\overrightarrow{a, c} \wedge \neg\langle a, c\rangle), \quad &&\overrightarrow{a, c} \wedge \overrightarrow{c, b} \rightarrow (\overrightarrow{a, b} \wedge \neg\langle a, b\rangle),\\
&\overrightarrow{b, a} \wedge \overrightarrow{a, c} \rightarrow (\overrightarrow{b, c} \wedge \neg\langle b, c\rangle), \quad &&\overrightarrow{b, c} \wedge \overrightarrow{c, a} \rightarrow (\overrightarrow{b, a} \wedge \neg\langle b, a\rangle),\\
&\overrightarrow{c, a} \wedge \overrightarrow{a, b} \rightarrow (\overrightarrow{c, b} \wedge \neg\langle c, b\rangle), \quad &&\overrightarrow{c, b} \wedge \overrightarrow{b, a} \rightarrow (\overrightarrow{c, a} \wedge \neg\langle c, a\rangle).
\end{aligned} \tag{2}$$

Confluence law: For each taxi, for any three different locations a, b, and c, where $a, b, c \in V$, we have

$$\overrightarrow{b, a} \wedge \overrightarrow{c, a} \rightarrow (\overrightarrow{b, c} \vee \overrightarrow{c, b}), \quad \overrightarrow{a, b} \wedge \overrightarrow{c, b} \rightarrow (\overrightarrow{a, c} \vee \overrightarrow{c, a}), \quad \overrightarrow{a, c} \wedge \overrightarrow{b, c} \rightarrow (\overrightarrow{a, b} \vee \overrightarrow{b, a}). \tag{3}$$

Ramification law: For each taxi, for any three different locations a, b, and c, where $a, b, c \in V$, we have

$$\overrightarrow{a,b} \wedge \overrightarrow{a,c} \rightarrow (\overrightarrow{b,c} \vee \overrightarrow{c,b}), \quad \overrightarrow{b,a} \wedge \overrightarrow{b,c} \rightarrow (\overrightarrow{a,c} \vee \overrightarrow{c,a}), \quad \overrightarrow{c,a} \wedge \overrightarrow{c,b} \rightarrow (\overrightarrow{a,b} \vee \overrightarrow{b,a}). \tag{4}$$

Acyclic law: For each taxi, for any two different locations a and b, where $a, b \in V$, we have

$$\neg\overrightarrow{a,b} \vee \neg\overrightarrow{b,a}. \tag{5}$$

Theorem 1. *The simultaneous Eqs. (1)–(4) ensure that, for each taxi, each vertex of its V can be visited at most once and be departed from at most once.*

Proof. Assume that a point a can be visited more than once from other points which include points b and c, where $a, b, c \in V$; therefore, $\langle b, a\rangle = \langle c, a\rangle = 1$. According to Eq. (1), we have $\overrightarrow{b,a} = \overrightarrow{c,a} = 1$. Then, due to Eq. (3), we know that $\overrightarrow{b,c} \vee \overrightarrow{c,b} = 1$. If we let $\overrightarrow{b,c} = 1$ with $\overrightarrow{c,a} = 1$, according to Eq. (2), we have $\overrightarrow{b,a} \wedge \neg\langle b, a\rangle = 1$, which conflicts with the first assumption $\langle b, a\rangle = 1$; otherwise, if we let $\overrightarrow{c,b} = 1$ with $\overrightarrow{b,a} = 1$, according to Eq. (2), we have $\overrightarrow{c,a} \wedge \neg\langle c, a\rangle = 1$, which also conflicts with the first assumption $\langle c, a\rangle = 1$. Therefore, the simultaneous Eqs. (1)–(3) ensure that, for each taxi, each vertex of its V can be visited at most once.

Assume that a point a can be departed from more than once to other points which include points b and c, where $a, b, c \in V$; therefore, $\langle a, b\rangle = \langle a, c\rangle = 1$. According to Eq. (1), we have $\overrightarrow{a,b} = \overrightarrow{a,c} = 1$. Then, due to Eq. (4), we know that $\overrightarrow{b,c} \vee \overrightarrow{c,b} = 1$. If we let $\overrightarrow{b,c} = 1$ with $\overrightarrow{a,b} = 1$, according to Eq. (2), we have $\overrightarrow{a,c} \wedge \neg\langle a, c\rangle = 1$, which conflicts with the first assumption $\langle a, c\rangle = 1$; otherwise, if we let $\overrightarrow{c,b} = 1$ with $\overrightarrow{a,c} = 1$, according to Eq. (2), we have $\overrightarrow{a,b} \wedge \neg\langle a, b\rangle = 1$, which also conflicts with the first assumption $\langle a, b\rangle = 1$. Therefore, the simultaneous Eqs. (1), (2) and (4) ensure that, for each taxi, each vertex of its V can be departed from at most once. $\square$

Let a set of taxis Λ consist of on-duty taxis. We denote a set of vertices V (resp. a vertex v) of a specific taxi λ as V_λ (resp. v_λ). For each taxi, we need to further guarantee that there exists at least one visit to each unfinished pick-up point P and drop-off point Q (Eq. (6)), and that there exists at least one departure from each P (Eq. (7)).

$$\bigwedge_{\lambda \in \Lambda} \bigwedge_{a \in \Phi_\lambda \cup \Psi_\lambda} ALO(\{\langle x, a\rangle \mid x \in V_\lambda\}) \tag{6}$$

$$\bigwedge_{\lambda \in \Lambda} \bigwedge_{P \in \Phi_\lambda} ALO(\{\langle P, y\rangle \mid y \in V_\lambda \setminus O_\lambda\}) \tag{7}$$

Lemma 1. *The simultaneous Eqs. (1)–(7) guarantee that, for each taxi, each vertex of its $\Phi \cup \Psi$, can be visited exactly once, and each vertex of its Φ can be departed from exactly once.*

For all taxis, we need to guarantee that there exists exactly one taxi to visit the pick-up point L and the drop-off point R of a new demand (Eqs. (8) and (9)), that there exists exactly one taxi to depart from L (Eq. (10)), and that there exists at most one taxt to depart from R (Eq. (11)).

$$EO(\bigcup_{\lambda\in\varLambda}\{\langle x, L_\lambda\rangle \mid x \in V_\lambda\}), \quad EO(\bigcup_{\lambda\in\varLambda}\{\langle x, R_\lambda\rangle \mid x \in V_\lambda\}) \tag{8,9}$$

$$EO(\bigcup_{\lambda\in\varLambda}\{\langle L_\lambda, y\rangle \mid y \in V_\lambda \setminus O_\lambda\}), \quad AMO(\bigcup_{\lambda\in\varLambda}\{\langle R_\lambda, y\rangle \mid y \in V_\lambda \setminus O_\lambda\}) \tag{10,11}$$

For each taxi, if there are some unfinished jobs, we need to further ensure that there exists at least one departure from its current point O; otherwise, we do nothing (Eq. (12)).

$$\bigwedge_{\lambda\in\varLambda}\begin{cases} ALO(\{\langle O_\lambda, y\rangle \mid y \in V_\lambda \setminus O_\lambda\}), & \text{if } \varPsi_\lambda \neq \emptyset, \\ \top, & \text{otherwise.}\end{cases} \tag{12}$$

Irreflexivity law: Because self-cyclic paths and the self reachabilities need to be banned, we encode some unit clauses to block the relevant corresponding variables as follow:

$$\bigwedge_{\lambda\in\varLambda}\bigwedge_{x\in V_\lambda} \neg\langle x, x\rangle \wedge \neg\overrightarrow{x, x}. \tag{13}$$

Corollary 1. *The simultaneous Eqs. (1)–(13) ensure HP constraints for RTSS.*

3.3 Order and Monopoly Constraints

Encoding the order and monopoly constraints on the basis of the HP constraint are straightforward. To encode the former, we only need to block all the directed reachabilities from somewhere to the current point for each taxi (Eq. (14)). Besides, for each existing pair of the unfinished pick-up and the drop-off points (i.e., P_i and Q_i) which correspond to the same served passenger, we add the unit clause $\overrightarrow{P_i, Q_i}$ for restricting the order of via-points (Eq. (15)).

$$\bigwedge_{\lambda\in\varLambda}\bigwedge_{x\in V_\lambda} \neg\overrightarrow{x, O_\lambda}, \qquad \bigwedge_{\lambda\in\varLambda}\bigwedge_{i} \overrightarrow{P_{\lambda_i}, Q_{\lambda_i}} \tag{14,15}$$

To encode the latter, we need to ensure that, for each taxi, the reachability of point L to point R must be evaluated to true if this taxi occupies either point of them (Eqs. (16) and (17)), and for all taxis, there exists at most one such a reachability that is evaluated to true (Eq. (18)).

$$\bigwedge_{\lambda\in\varLambda}\left(\neg\overrightarrow{O_\lambda, L_\lambda} \vee \overrightarrow{L_\lambda, R_\lambda}\right), \quad \bigwedge_{\lambda\in\varLambda}\left(\neg\overrightarrow{O_\lambda, R_\lambda} \vee \overrightarrow{L_\lambda, R_\lambda}\right) \tag{16,17}$$

$$AMO(\{\overrightarrow{L_\lambda, R_\lambda} \mid \lambda \in \varLambda\}) \tag{18}$$

3.4 Soft Clauses

To minimize the sum of the travel times of all the served passengers who are allocated to the taxi corresponding to the currently occurring demand, we have the following optimization objective function: $\min\{\sum_{\lambda \in \Lambda} \sum_{x \in V_\lambda} \sum_{y \in V_\lambda} (\overrightarrow{O_\lambda, L_\lambda} \cdot \langle x, y \rangle \cdot w_{\langle x,y \rangle})\}$. Therefore, we have the soft clauses:

$$\bigwedge_{\lambda \in \Lambda} \bigwedge_{x \in V_\lambda} \bigwedge_{y \in V_\lambda} \left(\neg\overrightarrow{O_\lambda, L_\lambda} \vee \neg\langle x, y \rangle, w_{\langle x,y \rangle} \right). \tag{19}$$

Because RTSS is required to continuously solve the path planning problem in a time series scenario, this optimization is equivalent to a min-max objective function. Note that we cannot directly IP-formulate the above objective because it is a nonlinear polynomial which is different from the normal min-max objective function. For an min-sum objective that minimizes the total cost time required for all the taxis to finish their accepted job lists corresponds to solving SPP, Eq. (19) can be converted by simply removing $\neg\overrightarrow{O_\lambda, L_\lambda}$ from each soft clause.

3.5 Space Complexity

Here, we give the space complexity of our encoding, which includes the number of required Boolean variables and clauses. Consider that the number of taxis is m (i.e., $|\Lambda| = m$) and that the size of V that has the largest number of unfinished points is n (i.e., $\max_{\lambda \in \Lambda}\{|V_\lambda|\} = n$), where $n \geq 3$.

Theorem 2. *When encoding RTSS to MaxSAT without deadline and capacity constraints, the simultaneous Eqs. (1)–(19) always produce a pseudo-polynomial-sized CNF that includes the number of required Boolean variables in $O(mn^2)$ and the number of generated clauses in $O(mn^3 + m^2n^2)$.*

Proof. The total number of variables required in our encoding is bounded by $2mn^2$, which is composed of two types of variables that have the same size, mn^2. In Eqs. (1) and (5), the number of clauses involved in our encoding is bounded from above by $3m\binom{n}{2}$. In Eqs. (2)–(4), the number of clauses is bounded by $18m\binom{n}{3}$. In Eqs. (6) and (7), the number of clauses is bounded by $3mn/2$. In Eqs. (8)–(11), the number of clauses is bounded by $2m^2n^2$. In Eq. (12), (resp. Eq. (13)), the number of clauses is bounded by m (resp. $2mn$). In Eqs. (14) and (15), the number of clauses is bounded by $3mn/2$, while in Eqs. (16)–(18), the number of clauses is bounded by $2m + \binom{m}{2}$. When encoding soft clauses in Eq. (19), the number of clauses is bounded by mn^2. Therefore, the simultaneous Eqs. (1)–(19) always produce a pseudo-polynomial-sized CNF that includes the number of required Boolean variables in $O(mn^2)$ and the number of generated clauses in $O(mn^3 + m^2n^2)$. $\square$

Technically, we can further reduce the CNF size via integrating Boolean variables by $\overrightarrow{a, b} \equiv \neg\overrightarrow{b, a}$, where $\forall\lambda \in \Lambda$, $a, b \in \mathcal{V}_\lambda$ and $a \neq b$, which is based on the method in Velev and Gao [30]. Due to such integration, one-third of Eq. (2) and most of Eqs. (3)–(5) can be eliminated. Nonetheless, it does not affect the above asymptotic upper bounds.

Algorithm 1 MaxSAT solver with an incremental approach

```
**Input**: $C_1 \wedge \ldots \wedge C_m \wedge (C_{m+1}, w_1) \wedge \ldots \wedge (C_{m+n}, w_n)$, $\mathit{\Gamma}$
**Initialization**: $sat? \leftarrow \mathsf{true}$, $first \leftarrow \mathsf{true}$, $k \leftarrow 1 + \sum_{i=1}^{n} w_i$, $assumption \leftarrow \emptyset$
**Output**: Unsatisfiability or a pair of the optimal assignment to $\mathcal{F}$ and its optimum answer
1: $\forall_i$ $SATsolver.\textsc{AddClause}(C_i)$, s.t. $1 \leq i \leq m$
2: while $sat?$ do
3:   if $first$ then
4:     $first \leftarrow \mathsf{false}$
5:     $(sat?, \mathcal{A}) \leftarrow SATsolver.\textsc{Solve}(assumption)$
6:   else
7:     $(violate?, reason) \leftarrow \textsc{CheckCondition}(\mathcal{A}, \mathit{\Gamma})$, s.t. $reason \subseteq \mathcal{A}$
8:     if $violate?$ then
9:       $SATsolver.\textsc{AddClause}(\neg reason)$
10:    else
11:      $k \leftarrow \sum_{\mathcal{A}(C_{m+i})=0} w_i$
12:      $\forall PBclause$ $SATsolver.\textsc{AddClause}(PBclause)$, s.t.
         $PBclause \in \mathsf{CNF}_{\mathrm{PB}}(\sum_{i=1}^{n}(w_i \neg C_{m+i}) < k)$
13:    end if
14:    $(sat?, \mathcal{A}) \leftarrow SATsolver.\textsc{Solve}(assumption)$
15:  end if
16: end while
17: if $k = 1 + \sum_{i=1}^{n} w_i$ then
18:   return UNSAT
19: else
20:   return $(\mathcal{A}, \sum_{i=1}^{n} w_i - k)$
21: end if
```

4 Incremental Approach

A general direct encoding for the rest constraints is anticipated that a huge number of auxiliary Boolean variables are required, which are used to represent every possible sum of the combinations of the cost time and the number of on-board passengers. Therefore, in this paper, we prune the search space using a naive incremental SAT-based approach [28], instead of the constraints encoding method. The idea of our approach, shown in Algorithm 1, is to construct a sequence of MaxSAT instances by adding a hard clause whenever the current assignment violates the externality conditions (i.e., the deadline and capacity constraints), denoted by $\mathit{\Gamma}$, during the SAT solving iterations (lines 7–12). Each added clause is the negation of a partial assignment, which is the reason for the violation (lines 7–9). Learning these clauses can effectively avoid the same inconsistency in the remaining procedures. The main difference from the general MaxSAT solver is that k is updated to a smaller value (line 11), and then the PB constraint is associated with a fresh k (line 12) only when $\mathit{\Gamma}$ is satisfied with the current assignment $\mathcal{A}$.

An effective heuristic of variable assignment can greatly reduce the time for solving large-scale RTSS [18,2]. Therefore, it is essential to provide an extensible framework for heuristics in a real-time system. Our incremental SAT-based algorithm can use an *assumption* SAT solving for further modifications (lines 5 and 14). A set of assumptions is defined as a set of literals that are assumed to be true and which are picked for decisions first, always in the top of the search tree. Then, if during the search, it is needed to flip the assignment of one of these assumptions to false, the problem is unsatisfiable under the initial assumptions. Therefore, we can introduce some heuristics methods via this assumption solving interface for preferential search strategies in the early solving stage.

5 Implementation and Evaluation

5.1 The Existing Insertion Method

To better evaluate our proposed approach, we implemented another method called *successive best insertion* (SBI), proposed in Noda et al. [22,21], which is used to provide a comparison to our approach. SBI is an approximation method that finds a semi-optimal result for RTSS via a specific local searching method described as follows:

1. For each taxi λ ($\lambda \in \Lambda$), there exists a via-point list that chronologically stores each vertex of $\mathcal{V}_\lambda$.
2. Each taxi's via-point list cannot be modified except via operations inserting L and R.
3. For each taxi, SBI linearly searches for the best pair of positions to insert L and R, according to the minimum of its self-time consumption plus the sum of the delays to all the served passengers' trip times. During the above search, a temporary best answer is initialized with *no solution* and is updated whenever a better answer is found that does not violate the deadline and capacity conditions.
4. SBI assigns the demand to a taxi whose cost is the minimum for all the taxis. If all taxis report no solution, then the demand is refused.

SBI has used in a real RTSS application system operated in Japan for more than three years [20,19], because SBI requires small computational resources (e.g., computing time) and provides reasonable solutions to vehicle allocation in real services. This work motivated to improve the quality of the solutions under reasonably extended computational resources. Therefore, we compare the performance of the proposed method to SBI. Note that our new method and SBI have different search space. In brief, for each taxi, SBI tries to insert new demand's pick-up and drop-off points into a fixed permutation (i.e., the order of all unfinished points cannot be shifted). However, the proposed MaxSAT approach exhaustively considers all possible permutations.

5.2 Experimental Setting

Our empirical experiments were conducted using *simulation of urban mobility* (SUMO) [4], version 0.32, which is a widely recognized open-source traffic simulation package including a traffic simulator as well as supporting tools. SUMO is microscopic, space continuous, and time discrete, providing a fair approximation of real-world traffic scenarios. We imported the road network for the city of Tsukuba from *OpenStreetMap* [12] into the simulator. To simulate real-world scenarios, there are three discrete areas set to occur demands in the city that were far apart and had high population mobility (e.g., commuting locations, shopping centers, residential areas).

The experiment was performed with the following parameter settings: the number of taxis (#Taxi) was chosen from $\{20, 30, 40, 50\}$; the demand occurrence frequency (Dof) for every set area was chosen from $\{18, 24, 36, 72\}$, where Dof indicates the number of demands occurring during a simulation hour; each demand involved only one passenger (i.e., $N = 1$) with the current time as its D and a calculated deadline as its T by dividing a measured distance between P and Q by an average walking velocity. the capacity of each taxi was 4; and the average taxi speed was 8.33 m/s. Note that each parameter set contains a pair with #Taxi and Dof. In addition, each experiment ran within 43,200 simulation seconds, and a new-demand checker worked during every simulation second to return the current demands to an allocation algorithm (i.e., SBI or SAT).[5] Even though both the SBI and SAT approaches were given a set 10 CPU second time limit for each demand, they also returned an approximate solution if they exceeded that time limit.

All experiments were performed on an Intel Xeon(R) Silver 4108, 1.8 GHz, with an 8-core processor and 93 GB of memory using the Ubuntu 18.04 operation system. Only one CPU core was used for each experiment. We implemented both SBI method and our proposed MaxSAT encoding by using Ruby 2.5.1, and modified a satisfiability-based solver, QMaxSAT, for the presented incremental approach in GCC version 7.3.0 with an n-level modulo-based CNF encoding of the PB constraints to solve all the generated MaxSAT instances [32].

5.3 Evaluation Metrics

There are three different evaluation metrics considered as follows: (1) the passengers' average speed (avg. speed), (2) the cumulative shared ratio (cumul. share) and (3) the average runtime for solving a demand (avg. runtime). Avg. speed was obtained by calculating the average of the passengers' actual moving distances divided by the cost time, which was counted from when the demand occurred to when the passenger arrived at their destination. In transportation research, the avg. speed can be regarded as a metric to evaluate the usability of service, in which a higher avg. speed corresponds to a shorter and less time-consuming trip for each demand. The cumul. share is a ratio of the total number of occurred demands to the cumulated number of shared people in each passenger trip.[6] The cumul. share can be regarded as another metric to evaluate the usability of service,

[5] The proposed approach is called "SAT" in our experiments.

[6] If passengers were in the same taxi at the same time, we say that they "shared" their trip with each other. Due to this mutuality, the cumulation of each demand includes duplicate calculations.

Table 1. Results of the SBI and SAT approaches compared using three different evaluation metrics.

Param.	Avg. speed (m/s)		Cumul. share (person/demand)		Avg. runtime (CPU s/demand)	
setting	SBI	SAT	SBI	SAT	SBI	SAT
n20-f18	3.94 [0.02]	**4.34** [0.03]	0.88 [0.07]	**1.37** [0.19]	**0.18** [0.01]	0.18 + 0.04 [0.01 + 0.00]
n30-f18	4.00 [0.13]	**4.34** [0.04]	0.74 [0.14]	**1.13** [0.06]	**0.16** [0.00]	0.21 + 0.05 [0.01 + 0.00]
n40-f18	3.95 [0.06]	**4.33** [0.06]	0.68 [0.10]	**1.08** [0.09]	**0.16** [0.01]	0.27 + 0.07 [0.02 + 0.00]
n50-f18	3.91 [0.07]	**4.20** [0.11]	0.62 [0.11]	**1.05** [0.05]	**0.16** [0.01]	0.34 + 0.09 [0.03 + 0.01]
n20-f24	3.70 [0.08]	**4.31** [0.03]	1.20 [0.06]	**1.50** [0.10]	**0.18** [0.01]	0.19 + 0.05 [0.01 + 0.00]
n30-f24	4.00 [0.06]	**4.32** [0.07]	0.83 [0.09]	**1.33** [0.06]	**0.17** [0.02]	0.23 + 0.06 [0.01 + 0.00]
n40-f24	3.98 [0.12]	**4.33** [0.02]	0.79 [0.06]	**1.30** [0.07]	**0.16** [0.01]	0.29 + 0.08 [0.02 + 0.00]
n50-f24	3.90 [0.10]	**4.23** [0.14]	0.80 [0.07]	**1.28** [0.12]	**0.16** [0.01]	0.35 + 0.09 [0.03 + 0.00]
n20-f36	2.47 [0.20]	**4.00** [0.07]	2.37 [0.15]	**2.53** [0.16]	**0.21** [0.02]	0.21 + 0.10 [0.02 + 0.01]
n30-f36	3.76 [0.06]	**4.24** [0.04]	1.20 [0.06]	**1.93** [0.14]	**0.18** [0.01]	0.25 + 0.09 [0.01 + 0.01]
n40-f36	3.88 [0.06]	**4.26** [0.05]	0.99 [0.07]	**1.67** [0.15]	**0.17** [0.01]	0.29 + 0.09 [0.02 + 0.00]
n50-f36	3.85 [0.04]	**4.25** [0.02]	0.96 [0.15]	**1.67** [0.16]	**0.17** [0.01]	0.36 + 0.12 [0.02 + 0.00]
n20-f72	0.94 [0.04]	**1.67** [0.17]	3.63 [0.07]	**4.84** [0.29]	**3.91** [0.67]	1.01 + 5.10 [0.26 + 0.55]
n30-f72	1.51 [0.12]	**3.50** [0.04]	2.87 [0.10]	**3.85** [0.19]	1.46 [0.26]	**0.33** + **0.51** [0.01 + 0.02]
n40-f72	2.67 [0.19]	**3.94** [0.03]	2.33 [0.07]	**2.74** [0.13]	**0.23** [0.03]	0.37 + 0.26 [0.03 + 0.03]
n50-f72	3.35 [0.04]	**4.01** [0.05]	1.75 [0.08]	**2.27** [0.12]	**0.20** [0.02]	0.44 + 0.25 [0.02 + 0.03]

in which a higher cumul. share ratio corresponds to a smaller payment amount for each served passenger. Lastly, the avg. runtime represents the computational cost of the allocation method.

5.4 Results and Analyses

Table 1 shows the experimental results for the mentioned three evaluation items. The notation "nx-fy" in the first column of Table 1 lists all pairs of the parameter setting, where x indicates #Taxi and y indicates Dof. For each "nx-fy", we repeated five experiments and then calculated their average and standard deviation rounded to two decimal places, which corresponds to two numbers in each cell, where the left number indicates the average and the right number surrounded by square brackets indicates the standard deviation. Comparing SBI and SAT, SAT outperforms SBI for both avg. speed and cumul. share for each parameter setting. However, SAT is overall inferior to SBI for the avg. runtime. Note that the avg. runtime of SAT is added in two parts, where the first part indicates the avg. runtime of the MaxSAT encoding, and the second part indicates the avg. runtime of the incremental MaxSAT solving. Interestingly, for each parameter setting, the avg. runtime of the encoding is generally longer than that of the solving, except in the cases of "n30-f72" and "n20-f72". This highlights a future avenue of research, that is, to improve our MaxSAT encoding with a more compact space complexity. Here we explain the worst-case scenario in our experiment, "n20-f72", which leads to the service being in short supply. Due to the sustained and rapid growth of the unfinished job list for each taxi, the MaxSAT encoding generates a series of huge size instances (see Theorem 2). In this case, the accepted demand rate of SBI was 64.20%, while that of SAT was 70.24%. Conversely, in all other cases, both of their accepted demand rates were 100%. Therefore, another future avenue of research is to find a method that can dynamically adjust #Taxi depending on specific features (e.g., area information or different time zones). Nevertheless, overall, when evaluating the efficiency of a real-time planning system, SBI and SAT can solve RTSS within a reasonable runtime.

To further compare the performances of SBI and SAT, in Fig. 2, we plot their change trend for avg. speed with different parameter settings, which are depicted as the corresponding contour plots of avg. speed separated by color bars. The horizontal axis indicates increasing values of #Taxi, and the vertical axis indicates increasing values of Dof. For both SBI and SAT, we see that, as #Taxi becomes smaller and Dof becomes greater, the corresponding color mapping is of a lower avg. speed. In addition, the change trend (i.e., the gradient of the surface) of SAT is greater than that of SBI. If we consider whether the avg. speed exceeds 3.5 m/s as a condition for distinguishing service quality, then only about 60% of results in SBI can be satisfied with this condition; while more than 90% of them in SAT can be evaluated to good. Consequently, both SBI and SAT suffer a service quality decline, here reflected in the avg. speed; however, SAT can provide a wider range of treatment than SBI.

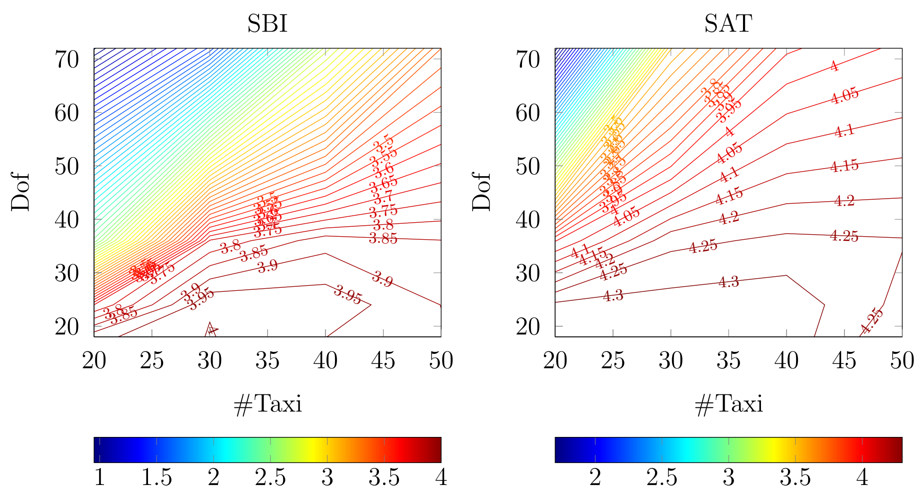


Fig. 2. The change trend for the avg. speed with different parameter settings using the SBI and SAT approaches.

5.5 Real-World Experiment

We simulated the real-world data of the city of Yokohama that were obtained from a transport service company to further demonstrate our proposed approach. In this experiment, we chose the collected one-week dataset (from December 1 to 7, 2018), fixed the number of taxis to 10 with an 8-capacity for each taxi, and kept the rest of the other parameter settings and our experimental environment. Note

Table 2. Results of the SBI and SAT approaches compared on real-world data.

Real-world data	#Rejected (demand)		Avg. speed		Cumul. share		Avg. runtime	
(#Demand [date])	SBI	SAT	SBI	SAT	SBI	SAT	SBI	SAT
702 [Dec. 1]	55	**0**	1.58	**1.85**	1.75	**2.12**	**0.15**	$0.21 + 1.75$
714 [Dec. 2]	2	**0**	1.67	**1.84**	**1.96**	1.80	**0.14**	$0.19 + 1.46$
508 [Dec. 3]	0	0	2.08	**2.50**	0.64	0.64	**0.13**	$0.15 + 0.05$
593 [Dec. 4]	0	0	1.99	**2.28**	**1.12**	1.08	**0.13**	$0.16 + 0.47$
518 [Dec. 5]	1	**0**	2.26	**2.46**	0.62	**0.66**	**0.12**	$0.14 + 0.05$
537 [Dec. 6]	0	0	2.03	**2.39**	1.06	**1.10**	**0.12**	$0.16 + 0.28$
931 [Dec. 7]	99	**0**	1.34	**1.48**	2.36	**3.31**	**0.17**	$0.32 + 3.21$

that the elements of each demand (i.e., its L, R, D, T and N) were based on the real-world data. The results are shown in Table 2, where the first column indicates the information of real-world data corresponding to the total number of occurred demands during that day surrounded by square brackets. In addition to the three mentioned evaluation metrics, we also listed the number of rejected demands in both SBI and SAT. Obviously, SAT outperforms SBI in this item. Overall, the other results have the same trend and conclusions with the random experiment, except there are some dates that SAT is slightly inferior to SBI in terms of cumul. share. The reason for this shortage may be that in our proposed algorithm there is a single objective function which can be regarded as the maximization of avg. speed but not that of cumul. share.

6 Conclusions

In this paper, we studied RTSS and proposed a novel approach to solve RTSS that is based on the incremental SAT technique. Our method is suitable for other objective functions (e.g., the shortest path optimization) and is extendible for other heuristic search strategies. We also proved the correctness of our encoding and analyzed its space complexity. Finally, we conducted a comparative experiment based on real-map data using a modified SUMO simulator to compare our method to another existing insertion method–SBI, which has been practiced in a real RTSS system. The experimental result shows that, even though the two methods can solve RTSS within a reasonable computational time in general and both suffer a service quality decline in the worst-case scenarios, our new approach can provide a wider range of treatments and a more rapid transport service with a cheaper price on average than SBI.

References

1. Agatz, N.A., Erera, A.L., Savelsbergh, M.W., Wang, X.: Dynamic ride-sharing: A simulation study in metro atlanta. Transportation Research Part B: Methodological **45**(9), 1450 – 1464 (2011). https://doi.org/https://doi.org/10.1016/j.trb.2011.05.017

2. Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., Rus, D.: On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. Proceedings of the National Academy of Sciences **114**(3), 462–467 (2017). https://doi.org/10.1073/pnas.1611675114
3. Baldacci, R., Maniezzo, V., Mingozzi, A.: An exact method for the car pooling problem based on lagrangean column generation. Operations Research **52**(3), 422–439 (2004). https://doi.org/10.1287/opre.1030.0106
4. Behrisch, M., Krajzewicz, D., Weber, M. (eds.): Simulation of Urban Mobility - First International Conference, SUMO 2013, Berlin, Germany, May 15-17, 2013. Revised Selected Papers, Lecture Notes in Computer Science, vol. 8594. Springer (2014). https://doi.org/10.1007/978-3-662-45079-6
5. Bektas, T.: The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega **34**(3), 209 – 219 (2006). https://doi.org/https://doi.org/10.1016/j.omega.2004.10.004
6. Berbeglia, G., Cordeau, J., Laporte, G.: Dynamic pickup and delivery problems. European Journal of Operational Research **202**(1), 8–15 (2010). https://doi.org/10.1016/j.ejor.2009.04.024
7. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
8. Coltin, B., Veloso, M.M.: Ridesharing with passenger transfers. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014. pp. 3278–3283. IEEE (2014). https://doi.org/10.1109/IROS.2014.6943018
9. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151–158. STOC ’71, ACM (1971). https://doi.org/10.1145/800157.805047
10. Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.): The Shortest Path Problem, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, November 13-14, 2006, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 74. DIMACS/AMS (2009), `http://dimacs.rutgers.edu/Volumes/Vol74.html`
11. Gørtz, I.L.: Hardness of preemptive finite capacity dial-a-ride. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2006 and 10th International Workshop on Randomization and Computation, RANDOM 2006, Barcelona, Spain, August 28-30 2006, Proceedings. Lecture Notes in Computer Science, vol. 4110, pp. 200–211. Springer (2006). https://doi.org/10.1007/11830924_20
12. Haklay, M.M., Weber, P.: Openstreetmap: User-generated street maps. IEEE Pervasive Computing **7**(4), 12–18 (2008). https://doi.org/10.1109/MPRV.2008.80
13. Herbawi, W., Weber, M.: A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows. In: Soule, T., Moore, J.H. (eds.) Genetic and Evolutionary Computation Conference, GECCO ’12, Philadelphia, PA, USA, July 7-11, 2012. pp. 385–392. ACM (2012). https://doi.org/10.1145/2330163.2330219
14. Hoos, H.H.: Sat-encodings, search space structure, and local search performance. In: Dean, T. (ed.) Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages. pp. 296–303. Morgan Kaufmann (1999), `http://ijcai.org/Proceedings/99-1/Papers/044.pdf`

15. Huang, Y., Bastani, F., Jin, R., Wang, X.S.: Large scale real-time ridesharing with service guarantee on road networks. PVLDB **7**(14), 2017–2028 (2014). https://doi.org/10.14778/2733085.2733106
16. Iwama, K., Miyazaki, S.: Sat-variable complexity of hard combinatorial problems. In: In: Proceedings of the world computer congress of the IFIP. pp. 253–258. Elsevier science B.V (1994)
17. Li, C.M., Manyà, F.: Maxsat, hard and soft constraints. In: Biere et al. [7], pp. 613–631. https://doi.org/10.3233/978-1-58603-929-5-613
18. Ma, S., Zheng, Y., Wolfson, O.: T-share: A large-scale dynamic taxi ridesharing service. In: Jensen, C.S., Jermaine, C.M., Zhou, X. (eds.) 29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013. pp. 410–421. IEEE Computer Society (2013). https://doi.org/10.1109/ICDE.2013.6544843
19. Nakashima, H., Matsubara, H., Tayanagi, E. (eds.): Smart Mobility Revolution — Advanced AI Public Transportation Service, SAVS —. FUN Press (March 2019), `https://books.google.co.jp/books?id=QTEAwwEACAAJ`, (in Japanese)
20. Nakashima, H., Sano, S., Hirata, K., Shiraishi, Y., Matsubara, H., Kanamori, R., Koshiba, H., Noda, I.: One Cycle of Smart Access Vehicle Service Development, pp. 247–262. Springer Japan, Tokyo (2016). https://doi.org/10.1007/978-4-431-55861-3_17
21. Noda, I., Ohta, M., Kumada, Y., Shinoda, K., Nakashima, H.: Usability of dial-a-ride systems. In: Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M.P., Wooldridge, M.J. (eds.) 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands. pp. 1281–1282. ACM (2005). https://doi.org/10.1145/1082473.1082733
22. Noda, I., Ohta, M., Shinoda, K., Kumada, Y., Nakashima, H.: Evaluation of usability of dial-a-ride systems by social simulation. In: Hales, D., Edmonds, B., Norling, E., Rouchier, J. (eds.) Multi-Agent-Based Simulation III, 4th International Workshop, MABS 2003, Melbourne, Australia, July 14th, 2003, Revised Papers. Lecture Notes in Computer Science, vol. 2927, pp. 167–181. Springer (2003). https://doi.org/10.1007/978-3-540-24613-8_12
23. Posada, M., Andersson, H., Häll, C.H.: The integrated dial-a-ride problem with timetabled fixed route service. Public Transport **9**(1-2), 217–241 (2017). https://doi.org/10.1007/s12469-016-0128-9
24. Prestwich, S.D.: SAT problems with chains of dependent variables. Discrete Applied Mathematics **130**(2), 329–350 (2003). https://doi.org/10.1016/S0166-218X(02)00410-9
25. Roussel, O., Manquinho, V.M.: Pseudo-boolean and cardinality constraints. In: Biere et al. [7], pp. 695–733. https://doi.org/10.3233/978-1-58603-929-5-695
26. Santos, D.O., Xavier, E.C.: Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem. In: Rossi, F. (ed.) IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013. pp. 2885–2891. IJCAI/AAAI (2013), `http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6779`
27. Simonin, G., O'Sullivan, B.: Optimisation for the ride-sharing problem: a complexity-based approach. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014). Frontiers in Artificial Intelligence and Applications, vol. 263, pp. 831–836. IOS Press (2014). https://doi.org/10.3233/978-1-61499-419-0-831

28. Soh, T., Berre, D.L., Roussel, S., Banbara, M., Tamura, N.: Incremental sat-based method with native boolean cardinality handling for the hamiltonian cycle problem. In: Fermé, E., Leite, J. (eds.) Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8761, pp. 684–693. Springer (2014). https://doi.org/10.1007/978-3-319-11558-0_52
29. Toth, P., Vigo, D.: Vehicle Routing: Problems, Methods, and Applications, Second Edition. No. 18 in MOS-SIAM Series on Optimization, SIAM (2014)
30. Velev, M.N., Gao, P.: Efficient SAT techniques for relative encoding of permutations with constraints. In: Nicholson, A.E., Li, X. (eds.) AI 2009: Advances in Artificial Intelligence, 22nd Australasian Joint Conference, Melbourne, Australia, December 1-4, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5866, pp. 517–527. Springer (2009). https://doi.org/10.1007/978-3-642-10439-8_52
31. Xu, Z., Li, Z., Guan, Q., Zhang, D., Li, Q., Nan, J., Liu, C., Bian, W., Ye, J.: Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In: Guo, Y., Farooq, F. (eds.) Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018. pp. 905–913. ACM (2018). https://doi.org/10.1145/3219819.3219824
32. Zha, A., Koshimura, M., Fujita, H.: N-level modulo-based CNF encodings of pseudo-boolean constraints for maxsat. Constraints **24**(2), 133–161 (2019). https://doi.org/10.1007/s10601-018-9299-0

Appendix

A Function CheckCondition($\mathcal{A}, \varGamma$) in Algorithm 1

The pseudo-code of function CheckCondition is shown in Algorithm 2, which is called in line 7 in Algorithm 1. The externality conditions in $\varGamma$ contain *deadlineArr*, *currenTime*, *capacityArr*, *numPassengArr* and *pickDropArr*, where *deadlineArr* and *pickDropArr* are two-dimensional arrays respectively storing the deadline and the number of pick-up/drop-off passengers of each taxi's job list. We also give an illustration of such data structures in Fig. 3 to help understand this algorithm. In Algorithm 2, the solution $\mathcal{A}$ need to be decoded and be found out which taxi's size of route has grown two, i.e., points L and R are allocated in its route (line 1), and we extract the partial assignment corresponding to this taxi from $\mathcal{A}$ (line 2).

Algorithm 2 A function to check whether the current assignment violates the deadline or capacity constraints

```
Input: $\mathcal{A}$, $\varGamma$, where $deadlineArr$, $currenTime$, $capacityArr$, $numPassengArr$,
    $pickDropArr \in \varGamma$ (note that, all indices of arrays range from 1)
Initialization: $sumDelay \leftarrow 0$, $sumCarried \leftarrow 0$, $violate? \leftarrow$ false, $reason \leftarrow \emptyset$
Output: $violate?$, $reason$, s.t. $reason \subseteq \mathcal{A}$
1: Decode $\mathcal{A}$ and find out the index $\lambda$ of taxi who occupies new demand
2: Extract $\mathcal{A}_\lambda$ which indicates the partial assignment involved with taxi $\lambda$
3: for each $\langle i,j\rangle$ ($\langle i,j\rangle \in \mathcal{A}_\lambda$) in route_order do
4:     $reason \leftarrow reason \wedge \langle i,j\rangle$
5:     $sumDelay \leftarrow sumDelay + w_{\langle i,j\rangle}$
6:     if $sumDelay > (deadlineArr[\lambda][j] - currenTime)$ then
7:         $violate? \leftarrow$ true
8:         break
9:     end if
10:    $sumCarried \leftarrow numPassengArr[\lambda]$
11:    $sumCarried \leftarrow sumCarried + pickDropArr[\lambda][j]$
12:    if $sumCarried > capacityArr[\lambda]$ then
13:        $violate? \leftarrow$ true
14:        break
15:    end if
16: end for
17: return $(violate?, reason)$
```

B Simulation

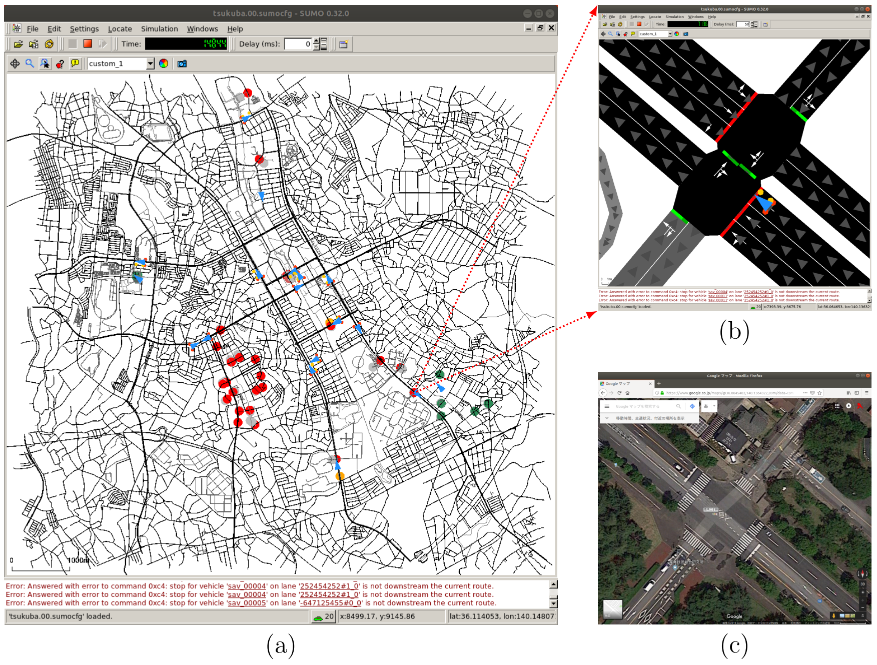
In order to confirm the result in the real city, Tsukuba, we investigated the features of mobility and transportation in this city based on actual road map, costs, and realistic conditions (e.g., acceleration, deceleration, boarding and alighting time, etc). Our simulation environment consists of four parts: (1) traffic

Taxi λ:	O	P_1	L	Q_1	P_2	P_3	Q_3	R	Q_2
deadlineArr[λ]:	t	7:10	7:05	7:30	7:20	7:15	7:35	7:25	7:40
pickDropArr[λ]:	c	$+2$	$+1$	-2	$+3$	$+1$	-1	-1	-3

t: *currenTime* c: *numPassengArr*[λ]

Fig. 3. An example of deadline and capacity conditions checking for the current assignment in Algorithm 2.

physical simulator, (2) demand generator, (3) vehicle routing controller, and (4) simulation GUI. All geometric distances in our experiment were estimated according to the corresponding *Manhattan distance*. We imported the road network of this city into our modified SUMO simulator, which is schematically shown in Fig. 4 (a). A magnified scale zooming in a specific location point picked from (a) with SUMO GUI is also given in Fig. 4 (b), which can highly simulate the road and traffic information of the real-map (see Fig. 4 (c)).



```
Error: Answered with error to command 0xc4: stop for vehicle 'sav_00004' on lane '252454252#1_0' is not downstream the current route.
Error: Answered with error to command 0xc4: stop for vehicle 'sav_00004' on lane '252454252#1_0' is not downstream the current route.
Error: Answered with error to command 0xc4: stop for vehicle 'sav_00005' on lane '-647125455#0_0' is not downstream the current route.
'tsukuba.00.sumocfg' loaded.    20    x:8499.17, y:9145.86    lat:36.114053, lon:140.14807
```

Fig. 4. A modified SUMO simulator: **(a)** The road network for the city of Tsukuba from OpenStreetMap. **(b)** A magnified scale zooming in a specific location point picked from (a) with SUMO GUI. **(c)** The aerial photograph of real-map corresponding to (b) obtained by Google Map.