Hindawi
Mathematical Problems in Engineering
Volume 2018, Article ID 4868906, 13 pages
https://doi.org/10.1155/2018/4868906

Research Article

Solving a Large-Scale Multi-Depot Vehicle Scheduling Problem in Urban Bus Systems

Xiaomei Xu,[1,2,3] **Zhirui Ye**,[1,2,3] **Jin Li**,[4] **and Chao Wang**[1,2,3]

[1] *Jiangsu Key Laboratory of Urban ITS, Southeast University, Nanjing 210096, China*
[2] *Jiangsu Province Collaborative Innovation Center of Modern Urban Traffic Technologies, Southeast University, Nanjing 210096, China*
[3] *School of Transportation, Southeast University, Nanjing 210096, China*
[4] *CCDI (Suzhou) Exploration & Design Consultant CO., Ltd., Suzhou 215123, China*

Correspondence should be addressed to Zhirui Ye; yezhirui@seu.edu.cn

Received 12 February 2018; Revised 29 June 2018; Accepted 26 July 2018; Published 23 August 2018

Academic Editor: Francisco Chicano

This study proposes an improved model and algorithm for the large-scale multi-depot vehicle scheduling problem (MDVSP) with departure-duration restrictions. In this study, the time-space network is applied to model the large-scale MDVSP. Considering that crews usually change shifts in the depot, departure-duration restrictions are added to the classic set-partitioning model to ensure that buses return to the depot when crews reach their working time limits. By embedding a preliminary exploring tactic to the shortest path faster algorithm (SPFA), researchers developed an improved large neighborhood search (LNS) algorithm to solve large-scale instances of MDVSP with departure-duration restrictions. The proposed methodology is applied to a real-life case in China and several test instances. The results show that the improved LNS algorithm can achieve very good performance in computational efficiency without deteriorating solution quality, which is important for large-scale systems. More specifically, the total cost of the improved LNS algorithm is approximately equal to branch-and-price, but the computational time is much shorter in the case study. For test instances with different number of timetabled trips (500, 1000, 1500, and 2000), the Quality Gap (QG) is very small, approximately 0.35%, 0.38%, 0.63%, and 0.93%, while the Efficiency Ratio (ER) reaches up to 2.89, 2.98, 3.65, and 3.79, respectively.

1. Introduction

Bus operation planning commonly includes four components: (1) bus network design; (2) timetable design; (3) bus scheduling; and (4) crew scheduling [1]. Among them, bus scheduling is extremely complex. An efficient scheduling plan is of great benefit to both bus companies and passengers. Compared to the single-line scheduling mode universally used in China, the multi-depot vehicle scheduling mode which belongs to the regional scheduling mode is more efficient. In the multi-depot vehicle scheduling problem (MDVSP), buses are allocated globally, helping bus companies reduce operating costs and promote operational efficiency. Although several models and algorithms have been developed for the vehicle routing problem (VRP) or vehicle scheduling problem (VSP), the extremely large-scale MDVSP involving thousands of timetabled trips and several depots remains relatively unexplored. Most algorithms are not efficient enough to provide a relatively optimal solution within an acceptable computational time. Moreover, bus scheduling systems in China are particularly large and busy due to the large population base and city size. In 2013, there were 21,293 vehicles running on 754 bus lines in Beijing every day, and, by 2014, the number of bus lines had increased to 652 in a third-tier city like Ningbo. For these large-scale systems, it is considerably difficult to apply the multi-depot vehicle scheduling mode into practice. The key issue of application lies in the availability of methods to downsize the scheduling system and improve the algorithm. As background, previous studies are analyzed from three aspects (i.e., network description, model formulation, and solution algorithm) as follows.

According to previous studies, both connection-based networks and time-space networks could be applied to model the VSP. In early studies, vehicle scheduling problems were modeled as connection-based networks [2]. The time-space network was first proposed for air scheduling due to its simplicity in modeling possible connections between flights [3]. Kliewer et al. were the first to apply the time-space network to VSP; they then proposed a commodity network flow model on the basis of the time-space network structure [4, 5]. For a small-scale VSP, the connection-based network was superior, whereas, for a middle or large-scale VSP, the time-space network could achieve better performance in downsizing the system by reducing the number of deadhead arcs [6]. Gintner et al. and Naumann et al. efficiently solved a practical large-scale and middle-scale MDVSP, respectively, using the commodity network flow model based on the time-space network [7, 8]. However, the reduction of deadhead arcs in the time-space network has not been sufficiently considered in previous studies. Instead of checking and reducing deadhead arcs after the network takes its shape, we avoid generating unnecessary deadhead arcs in the network building process to achieve the goal of reducing the number of deadhead arcs. In this way, we can reduce not only the number of deadhead arcs but also the time used in constructing the time-space network so as to improve efficiency.

For the VRP, there are three basic models from previous studies (i.e., vehicle flow model, commodity network flow model, and set-partitioning model) and several variations involving different constraints (e.g., VRP with time windows [9], VRP with backhauls [10], and VRP with pickup and delivery [11]). The goals of these models normally fall into two categories, minimizing fleet size or minimizing operating costs. The set-partitioning model was originally proposed by Balinski and Quandt [12]. As the route feasibility is implicitly considered in the definition of the circuit set, the set-partitioning model can easily take extra constraints into account [13]. In recent years, there have been too many safety incidents related to bus drivers working overtime in China. Drivers working overtime not only jeopardize their own health but also threaten the safety of passengers. Therefore, compared to other constraints, crews' working time limits are the primary consideration taken into account in this study. Drivers should strictly adhere to the change-of-shift time in order to get sufficient rest, which will ensure the safety of passengers on board. As drivers usually change shifts in the depot, buses should return to the depot when the crews reach their working time limits to ensure safe driving. Based on these conditions, the MDVSP in this study is formulated as a set-partitioning model with departure-duration restrictions.

As VRP is an extension of the Traveling Salesman Problem (TSP) and the VSP explored in this study is a VRP that arises in the public transport area, the algorithms for VSP can build on the extensive and successful work done for TSP and VRP. The algorithms in previous studies for VRP are categorized into two groups: exact algorithms and heuristic algorithms. Exact algorithms can achieve high quality solutions when dealing with small-scale systems; however, they suffer great limitations in practice. In contrast, the heuristic algorithms that usually perform a relatively limited exploration of the search space are capable of solving larger VRP problems within modest computational time. However, for extremely large-scale real-life scheduling instances involving several thousand trips, most of the available algorithms are computationally intensive, requiring thousands of CPU seconds. The branch-and-bound algorithm belongs to the exact algorithms category and has been extensively applied to solve the VRP and its variants in recent decades [14–16]. As noted by Toth and Michalewicz, branch-and-bound could find a significantly high quality solution but was not capable of solving large instances encountered in practice [13, 17]. The column generation was an important technique that could solve larger Linear Programing (LP) problems [18]. Many scholars have applied column generation to solve VRP [19, 20]. By integrating column generation techniques with the branch-and-bound scheme (which is also known as branch-and-price), many systems can be solved with significantly good quality solutions, as demonstrated by many successful studies in recent years [21–25]. However, the computational efficiency of the branch-and-price is not so satisfying, in particular for large-scale systems. As the MDVSP explored in this study is extremely large, a heuristic algorithm is established on the large neighborhood search (LNS) framework. By integrating branch-and-price into LNS, an algorithm characterized by both high quality solutions and computational efficiency is established. As the MDVSP in this study has departure-duration restrictions, the shortest path faster algorithm [26] is modified by embedding a preliminary exploring tactic to solve the subproblem.

In contrast to previous studies on MDVSP, the contributions of this study are as follows: (1) presenting a novel way to deal with deadhead arcs so as to improve the efficiency of downsizing the time-space network based MDVSP, (2) modeling the MDVSP by adding departure-duration restrictions to ensure safe driving which is imperative in China, (3) proposing an improved LNS algorithm to ensure good performance both in solution quality and computational efficiency, and (4) modifying the shortest path faster algorithm (SPFA) by embedding a preliminary exploring tactic to address departure-duration restrictions.

The remaining sections are organized as follows. Section 2 introduces basic descriptions of the MDVSP and network modeling. Sections 3 and 4 present the model formulation and solution algorithm to solve a large-scale MDVSP with departure-duration restrictions. Sections 5 and 6 evaluate the performance of the proposed methodology using a real-life case in China and several test instances. The conclusions and perspectives are summarized in the last section.

2. Problem Statement and Network Modeling

Before constructing the mathematical formulation, the time-space network is first introduced. In this section, basic descriptions and assumptions are presented. The methods of network modeling are explored at length. Some necessary definitions and terms are defined in this section.

2.1. Basic Descriptions and Assumptions. The VSP is a VRP encountered by bus companies when addressing the task of assigning buses to cover a given set of timetabled trips with the goal of minimizing fleet size or operating costs. Each

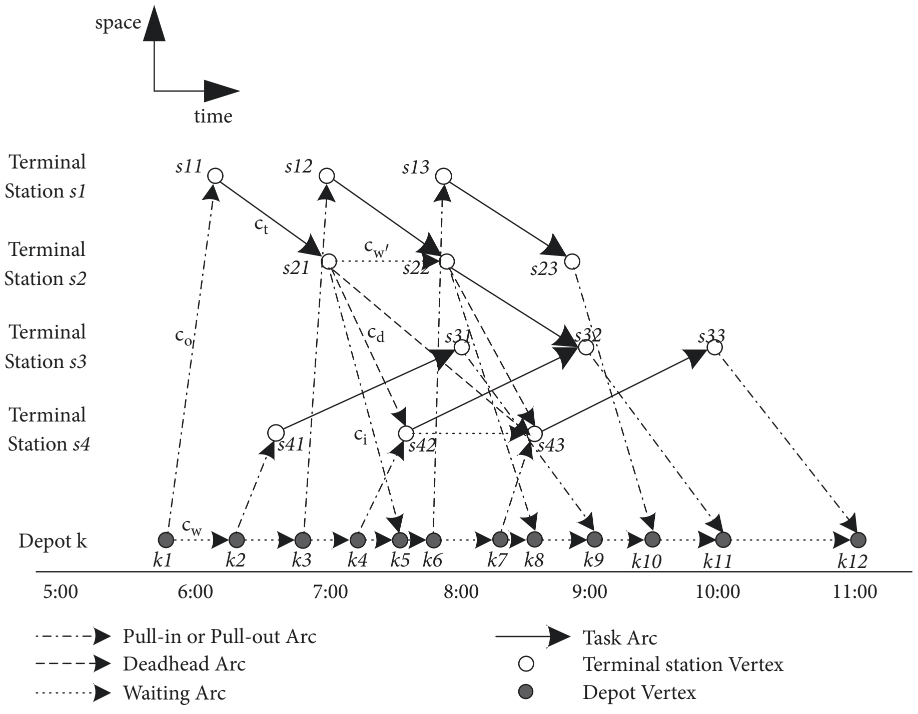

FIGURE 1: Time-space network of depot k in MDVSP.

timetabled trip is accomplished exactly once by a bus, and each bus performs a feasible sequence of trips. The MDVSP in this study involves several depots, and a timetabled trip can be run by any bus from any depot. The large-scale MDVSP with departure-duration restrictions as addressed in this study is built upon the following hypotheses:

(1) The scheduling scheme is organized in terms of individual days, and minutes are the smallest unit of time.

(2) All buses are homogeneous and the number of buses is limited.

(3) Every bus belongs to a home depot. At the end of the day, a bus has to return to the same home depot where it started in the morning.

(4) The cost for buses waiting outside the depot is very high. It is more favorable to wait at a depot than at other stations. If there is enough time, the bus should return to the depot and wait.

2.2. Network Description. The VSP can be modeled by either a connection-based network or a time-space network. Using the connection-based network, the problem can only be slightly downsized as the reducible long arcs are in a relatively small quantity. In contrast, there are significant numbers of deadhead arcs that can be removed in the time-space network. If the problem contains m terminal stations and n trips, the number of deadhead arcs in a time-space network can be reduced to $o(nm)$, in comparison to $o(n^2)$ in the connection-based network. In general, for a large-scale problem, the number of timetabled trips n is much larger than the number of terminal stations m. Consequently, the problem scale is much smaller when using a time-space network compared to using a connection-based network for the same scheduling system. Moreover, as each timetabled trip is characterized by a departure time and arrival time as well as an origin and destination station (time and space attributes), the time axis and space axis in the time-space network constitute a two-dimensional space which can more clearly identify the two attributes of each timetabled trip and the time-space relationships of different bus tours. On that account, it is more favorable to use a time-space network to handle the large-scale MDVSP as done in this study. Figure 1 illustrates the time-space network of depot k in MDVSP.

The time-space network is a directed graph composed of many vertices and arcs. All vertices have two attributes (time and space) and each of them connects a group of possible arrival arcs to the possible departure arcs of the following group. Each arc corresponds to a transition in time or space. Every depot and terminal station vertex indicates a possible arrival or departure event (an arc) at the depot or a given station at a certain time. For instance, *s11* denotes a bus that is reaching and then leaving terminal station s1 to run a timetabled trip (*s11, s21*) at a time around 6:00.

For a depot k ($k \in K$, K is the set of all depots), let G^k=(V^k, A^k) be a complete graph, where $V^k = \{o(k), d(k)\} \cup T \cup E$ is the vertex set. $o(k)$ corresponds to the set of depot vertices where buses pull out from depot k at a certain time(e.g., *k1*, *k2*, and *k3* in Figure 1), whereas $d(k)$ corresponds to the set of depot vertices where buses pull into depot k at a certain time (e.g., *k5*, *k8*, and *k9* in Figure 1). T is the set of terminal station vertices where tasks start at a certain station and time (e.g., *s11*, *s12*, and *s13* in Figure 1), whereas E is the set of terminal station vertices where tasks end at a certain station and time (e.g., *s21*, *s22*, and *s23* in Figure 1).

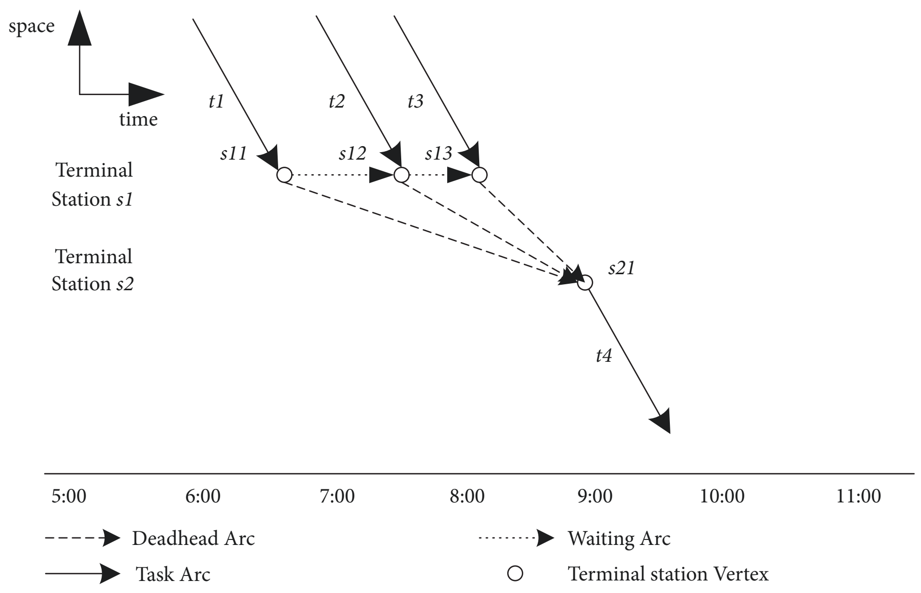

FIGURE 2: Nearest match description.

There are five types of arcs in arc set A^k, including pull-in arcs, pull-out arcs, deadhead arcs, waiting arcs, and task arcs (as shown in Figure 1). A pull-out arc (o,i'), $o \in o(k)$, $i' \in T$, denotes a bus pulling out from depot k in order to start a sequence of timetabled trips. A pull-in arc (i'', d), $i'' \in E$, $d \in d(k)$, denotes a bus pulling into depot k after completing a sequence of timetabled trips. A task arc (i', i''), $i' \in T$, $i'' \in E$, represents a timetabled trip. After serving a timetabled trip, each bus can wait to serve one of the trips starting later from the same station where it is holding, or it can change its location by moving unloaded to another station in order to serve the next loaded trip starting there. The former is a waiting arc, which indicates a bus that waits at a place for some time. There are two kinds of waiting arcs depending on whether the bus waits at a terminal station or at the depot. The latter is an unloaded trip corresponding to the deadhead arc. For a deadhead arc (i'', j'), $i'' \in E$, $j' \in T$, i'' is an end vertex in set E and j' is a start vertex in set T. For a pair of task arcs $(i1', i1'')$ and $(i2', i2'')$ that end at station vertex $i1''$ at time t_a and start at station vertex $i2'$ at time t_b, respectively, if $t_a + t_{ab} < t_b$, where t_{ab} is the travel time of a bus from station $i1''$ to $i2'$, task arc $(i1', i1'')$ and $(i2', i2'')$ are termed as a compatible pair of tasks. The compatibility relationship represents whether trip $(i2', i2'')$ can be served after trip $(i1', i1'')$ by the same bus. Any pair of compatible task arcs can be connected by a deadhead arc showing a possible tour of a bus. For example, a pair of compatible task arcs (*s11,s21*) and (*s42,s32*) are connected by a deadhead arc (*s21,s42*) in Figure 1, and the deadhead arc (*s21,s42*) indicates that a bus moves unloaded from station *s2* to station *s4* at time t_{s21} and then waits at station *s4* until t_{s42}. Figure 1 is the time-space network of depot k. For multiple depots (K), there are K layers of Figure 1 that stack up together with shared timetabled trips (task arcs).

Most deadhead arcs can be removed from the time-space network. Among all the tasks compatible with task arc $(i1', i1'')$, the nearest one is designated as the first match arc. Let D be the set of deadhead arcs that connect task arc $(i1', i1'')$ with all its compatible task arcs. Deadhead arc f, $f \in D$, is the connection between task arc $(i1', i1'')$ and its first match arc. Other arcs apart from f in set D can be removed as each of them can be represented by the deadhead arc f and a waiting arc. The total number of arcs will be reduced significantly compared to the original situation. Nevertheless, all possible connections remain feasible. For example, task arcs (*s42,s32*) and (*s43,s33*) are both compatible with task arc (*s11,s21*) in Figure 1, whereas task arc (*s42,s32*) is the first match arc with arc (*s11,s21*). Deadhead arc (*s21, s43*) can be replaced by deadhead arc (*s21, s42*) and waiting arc (*s42, s43*). Let T be the set of task arcs that end at terminal station *s1* with the same first match task arc t that starts at terminal station *s2*. If task arc $t1 \in T$ is the nearest task arc to t among all elements in T, *t1* is designated as the nearest match arc of t. For instance, in Figure 2, arc *t4* is the first match of arc *t1*, *t2*, and *t3*. As *t3* is nearest to *t4*; *t3* is the nearest match arc of *t4*. Deadhead arc (*s11,s21*) in Figure 2 can be removed as it can be replaced by deadhead arc (*s13,s21*) and waiting arc (*s11,s13*). Similarly, deadhead arc (*s12,s21*) can be replaced by deadhead arc (*s13,s21*) and waiting arc (*s12,s13*). Only deadhead arc (*s13,s21*) is retained.

Each arc is associated with a nonnegative cost.

$$c_o = u \cdot l_{ds} \tag{1}$$

$$c_i = u \cdot l_{sd} \tag{2}$$

$$c_{w'} = w' \cdot t_s \tag{3}$$

$$c_w = w \cdot t_d \tag{4}$$

$$c_t = u' \cdot t_t \tag{5}$$

$$c_d = u \cdot l_{ss} + c_{w'} \tag{6}$$

where c_o, c_i, $c_{w'}$, c_w, c_t, and c_d in Figure 1 denote the cost of a pull-out arc, pull-in arc, waiting arc outside the depot, waiting arc inside the depot, task arc, and deadhead arc, respectively; u, u', w', and w represent the unit cost of a bus running unloaded and loaded and waiting outside the depot and inside the depot, respectively; l_{ds}, l_{sd}, and l_{ss} denote the distance from the depot to a station, from a station to the depot, and from a station to another station, respectively; t_s, t_d, and t_t represent the waiting time at a station and at the depot and the duration time of a timetabled trip. A deadhead arc cost may include the waiting cost as a bus may arrive early before the next timetabled trip starts.

In the time-space network, the MDVSP can be described as many circuits (bus tours) starting from their respective depot vertex, passing by different kinds of arcs, and eventually ending at their depot vertex. Every task arc is covered exactly once. Every circuit, consisting of a feasible sequence of different types of arcs chained with each other, starts with a pull-out arc and ends up with a pull-in arc, which means a bus leaves the depot to start performing timetabled trips and goes back to the depot after finishing all timetabled trips. Take a circuit "*k1-s11-s21-k5-k6-k7-s43-s33-k12*" in Figure 1, for example: the bus first drives out from depot *k* (pull-out arc (*k1, s11*) to station *s1* to perform a timetabled trip (task arc (*s11, s21*)) and then drives back to the depot *k* (pull-in arc (*s21, k5*)) to rest (waiting arc (*k5, k6*) and(*k6, k7*)) and then drives out from depot *k* (pull-out arc (*k7, s43*)) to station *s4* to perform another timetabled trip (task arc (*s43, s33*)) and finally drives back to depot *k* (pull-in arc (*s33, k12*)). There are two pull-out arcs and two pull-in arcs in the circuit, but the starting pull-out arc and the ending pull-in arc of the circuit are unique. The goal is to find a series of desirable circuits with minimum total cost. The number of circuits that start from the depot vertex is equal to the number of buses used in the scheduling system.

2.3. Network Modeling. The key issue of the time-space network is to reduce the number of deadhead arcs. Most deadhead arcs can be removed as each of them can be represented by another deadhead arc and a waiting arc without reducing the number of compatible task pairs. For a large-scale system, the number of compatible task pairs is tremendous. Therefore, it is not desirable to connect all compatible task pairs by deadhead arcs beforehand and then reduce them. Instead, the two steps are carried out simultaneously.

For a depot $k \in K$, the network is constructed with four steps.

Step 1 (generation of terminal station vertices and task arcs). Generate task arcs and corresponding terminal station vertices according to the known bus timetable. Every task arc has a start terminal station vertex and an end terminal station vertex. Every terminal station has several terminal station vertices that are arranged in a chronological order.

Step 2 (generation of depot vertices, pull-in arcs, and pull-out arcs). A time line connected by depot vertices that represents all possible arrival and departure events at the depot is built. The pull-out arcs are constructed by connecting the departure depot vertices to the corresponding start terminal station vertices (generated in Step 1). Similarly, the pull-in arcs are constructed by connecting the relative end terminal station vertices (generated in Step 1) to the arrival depot vertices.

Step 3 (generation of waiting arcs). For a given terminal station, the waiting arcs outside the depot are constructed by connecting all of its terminal station vertices. For the depot, the waiting arcs inside the depot are constructed by connecting all of its depot vertices.

Step 4 (generation and reduction of deadhead arcs). Only those deadhead arcs that correlate to the first matches and the nearest matches are introduced into the model. Moreover, a very long deadhead arc (*i,j*), starting from terminal station vertex *i* at time t_i and ending at terminal station vertex *j* at time t_j, should also be removed if t_j-t_i $>t$, where t is the duration time of pull-in arc (from i to the depot) plus the duration time of pull-out arc (from the depot to j). These techniques result in a dramatic reduction in the number of deadhead arcs.

3. Model Formulation

The MDVSP in this study is formulated as a set-partitioning model calling for the determination of a collection of circuits with minimum cost, which serves each task once and satisfies strict departure-duration restrictions.

Let K be the set of all depots. v_k denotes the number of buses in depot k, $k \in K$. Λ^k is the set of all task arcs in the time-space network of depot k. Ω^k represents the set of all possible circuits, each corresponding to a possible bus tour, which starts with a pull-out arc and ends up with a pull-in arc. Every circuit $p \in \Omega^k$ has an associated cost c_p. Apart from the variable operating cost (cost of different arcs covered by p), c_p also includes fixed cost for the required bus. For every task arc $i\in\Lambda^k$, let a_{ip} be a binary coefficient that takes value 1 if task arc i is covered by circuit p and takes value 0 otherwise. For every circuit $p\in\Omega^k$, the binary decision variable x_p takes value 1 if and only if the circuit p is selected in the optimal solution. Otherwise, x_p takes value 0. The circuit p consists of a feasible sequence of different kinds of arcs chained with each other. The pull-out and pull-in arcs appear in pairs. Γ^p is the set of pairs of pull-out and pull-in arcs. For a pair of pull-out and pull-in arcs j, $j\in \Gamma^p$, t_j^a denotes the start time of the pull-out arc and t_j^b denotes the end time of the pull-in arc. t_0 represents the working time limits of bus crews. The model is as follows.

$$\text{Minimize} \quad \sum_{k\in K} \sum_{p\in\Omega^k} c_p x_p \tag{7}$$

$$\text{subject to:} \quad \sum_{k\in K} \sum_{p\in\Omega^k} a_{ip} x_p = 1, \quad \forall i \in \Lambda^k, \tag{8}$$

$$\sum_{p\in\Omega^k} x_p \le v_k, \quad \forall k \in K, \tag{9}$$

$$t_j^b - t_j^a \le t_0, \quad \forall j \in \Gamma^p,\ p \in \Omega^k,\ k \in K, \quad (10)$$

$$x_p \in \{0,1\}, \quad \forall p \in \Omega^k,\ k \in K. \quad (11)$$

Objective functions (7) serve to minimize the total scheduling cost of a bus company. Constraint (8) ensures that every timetabled trip is covered by exactly one bus. Constraint (9) imposes that the number of buses selected to run the timetabled trips should be no more than the total number in depot k. Constraint (10) imposes that the departure-duration of every selected bus should be no longer than the working time limits of bus crews. Constraint (11) defines the binary decision variables to depict whether a bus tour p is selected or not.

4. Solution Algorithm

In this section, an improved LNS algorithm based on the branch-and-price algorithm is constructed to solve the large-scale MDVSP. We first present the framework of the algorithm before introducing the details of solving the restricted master problem and subproblem.

4.1. The Improved Large Neighborhood Search Algorithm. The MDVSP explored in this study is an extremely large-scale Integer Programming (IP) problem. For one depot k, the number of binary decision variables is equal to the number of all possible circuits in Ω^k. As the size of the circuit set (Ω^k, $k \in K$) is very large, we do not compute the set of all possible circuits when solving the IP problem. Traditional exact algorithms are not applicable. On the framework of LNS, an improved heuristic algorithm that is both accurate and efficient is put forward. As there are departure-duration restrictions in the explored large-scale MDVSP, the SPFA [26] is modified by adding an embedded preliminary exploring tactic to solve the subproblem.

The improved LNS algorithm adheres to the following steps.

Step 1 (obtain initial feasible solution). Generate an initial feasible solution S_0 by insertion heuristic. The reader is referred to [27] for detailed analysis of the insertion heuristic.

Step 2 (initialize parameters). Let parameters *iter*=0 and S_{best}=S_0, where *iter* denotes the number of iterations and S_{best} denotes the best solution set at present. r represents the number of bus circuits selected from the best solution set S_{best} at present. *maxIter* is the maximum number of iterations.

Step 3 (generate neighborhood). Generate a neighborhood by selecting r bus circuits from the best solution set at present S_{best}. Select r bus circuits from the set S_{best}. The set of r bus circuits is termed as S_r. Generate a new problem using all tasks (timetabled trips) contained in the r bus circuits. This new problem has a smaller scale which will take much less time to be solved. The solution space of this new problem is the neighborhood. As to the selecting strategy of the r bus circuits, randomly select r bus circuits from those circuits that were selected less often in set S_{best}.

Step 4 (search neighborhood). Search the neighborhood generated in Step 3 by branch-and-price and we get a better feasible solution of the new problem, termed as S_r'. Reinsert S_r' into set S_{best} instead of S_r. We obtain a feasible solution S' of the original problem.

Step 5 (parameters update). Compare S' with $S_{best'}$; if S' <S_{best}, let S_{best}= S' and *iter*=*iter*+1.

Step 6 (termination criterion). If *iter*≤*maxIter*, return to Step 3; otherwise proceed to Step 7.

Step 7 (the end). S_{best} is the best solution that can be found.

Figure 3 shows the flowchart of the improved LNS algorithm. The main idea of this proposed algorithm is to find an initial feasible solution S_0 at first and then improve this feasible solution through iteration until certain iteration stopping conditions are satisfied and we obtain the optimal solution. The initial feasible solution and the optimal solution are both a subset of (Ω^k, $k \in K$). As to how to improve this feasible solution S_0, the main idea is to optimize a part of S_0 (r bus circuits in S_0) to get a better solution and repeat this operation again and again until certain iteration stopping conditions are satisfied. This operation is the process of generating a neighborhood and then searching the neighborhood.

The branch-and-price algorithm is based on the framework of the branch-and-bound algorithm. As the linear programming relaxation of the set-partitioning model is typically very tight [13], the original integer programming problem is relaxed into a linear programming (LP) problem (master problem) at each node in the branch-and-bound tree.

For a general LP problem (master problem):

$$\text{Minimize} \quad \sum_{q\in Q} c_q x_q \quad (12)$$

$$\text{subject to:} \quad \sum_{q\in Q} a_q x_q \ge \text{b}, \quad (13)$$

$$\sum_{q\in Q} x_q \ge 0, \quad \forall q \in Q \quad (14)$$

Q is the set of all columns and q is a column in Q. The simplex method can be used to solve this problem if the number of columns in Q is not large. Let λ be the corresponding dual variable. In each iteration of the simplex method, if there's a column q whose $c_q - \lambda a_q < 0$, we should select $a_s : c_s = {}^{min}_{q\in Q}(c_q - \lambda a_q)$. However, generally the number of columns in Q is very large, so we can solve this LP relaxation (master problem) using the column generation approach by first solving the LP relaxation over a subset of the possible columns. We refer to a master problem with only a subset of the possible columns as a restricted master problem. Additional columns can be generated as needed for the restricted master problem by solving the subproblem.

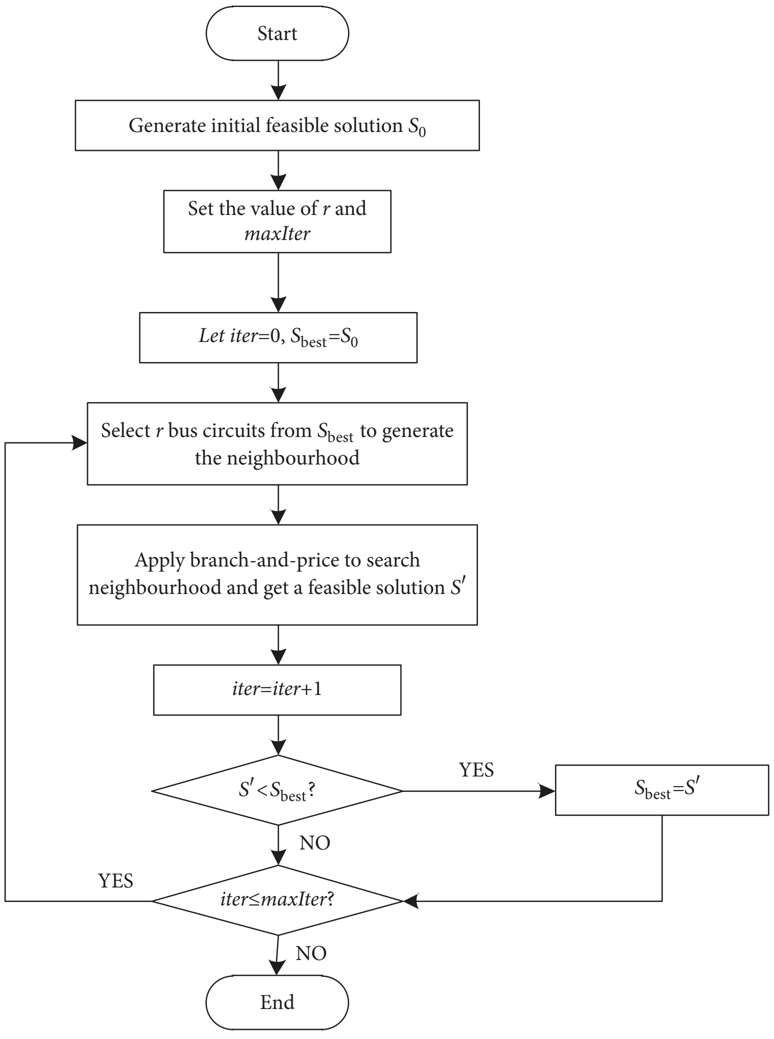
Start
Generate initial feasible solution S_0
Set the value of r and $maxIter$
Let $iter$=0, S_{best}=S_0
Select r bus circuits from S_{best} to generate the neighbourhood
Apply branch-and-price to search neighbourhood and get a feasible solution S'
$iter$=$iter$+1
$S' < S_{\text{best}}$?
YES
S_{best}=S'
NO
YES
$iter \le maxIter$?
NO
End

FIGURE 3: Improved large neighborhood searching algorithm.

The Restricted Master Problem (RMP) is

$$\text{Minimize} \quad \sum_{q\in Q'} c_q x_q \tag{15}$$

$$\text{subject to:} \quad \sum_{q\in Q'} a_q x_q \ge \text{b}, \tag{16}$$

$$\sum_{q\in Q'} x_q \ge 0, \quad \forall q \in Q', \; Q' \subseteq Q \tag{17}$$

Let x^* be the solution of this restricted master problem and λ^* be the corresponding dual variable. The subproblem can be expressed as $c^* = \underset{q\in Q}{\overset{min}{}}(c_q - \lambda^* a_q)$.

For the MDVSP in this paper, the subproblem is the shortest path problem with departure-duration restrictions. Sections 4.2 and 4.3 elaborate the details of branch-and-price techniques for neighborhood searching.

4.2. Restricted Master Problem. At each node in the branch-and-bound tree, the original integer programming problem is relaxed into a linear programming (LP) problem (master problem). Relax the integer variable x_p into a continuous variable between 0 and 1. Let Ω_0^k be the subset of Ω^k. When applying column generation to solve the relaxed version of the original model in this paper, the master problem is divided into two parts, a restricted master problem and a subproblem. The Restricted Master Problem (RMP) is

$$\text{Minimize} \quad \sum_{k\in K} \sum_{p\in\Omega_0^k} c_p x_p \tag{18}$$

$$\text{subject to:} \quad \sum_{k\in K} \sum_{p\in\Omega_0^k} a_{ip} x_p = 1, \quad \forall i \in \Lambda^k, \tag{19}$$

$$\sum_{p\in\Omega_0^k} x_p \le v_k, \quad \forall k \in K, \tag{20}$$

$$t_j^b - t_j^a \le t_0, \quad \forall j \in \Gamma^p, \; p \in \Omega_0^k, \; k \in K, \tag{21}$$

$$0 \le x_p \le 1, \quad \forall p \in \Omega_0^k, \; k \in K. \tag{22}$$

The dual problem of the Restricted Master Problem (DRMP) is

$$\text{Maximum} \quad \sum_{i=1}^{n} \pi_i - \sum_{k=1}^{m} v_k \lambda_k \tag{23}$$

$$\text{subject to:} \quad \sum_{i=1}^{n} a_{ip}\pi_i - \lambda_k \leq c_p, \quad \forall p \in \Omega_0^k, \; k \in K \tag{24}$$

$$\lambda_k \geq 0, \quad k = 1, 2, \ldots, m. \tag{25}$$

ILOG CPLEX or Lingo can be applied to solve the DRMP above and obtain the value of decision variables (π_i and λ_k). The solution of the dual problem (π_i and λ_k) in this paper is obtained by applying ILOG CPLEX to the DRMP. Update the cost of arcs in the network according to the value of π_i and λ_k to solve the subproblem.

4.3. Subproblem. For the MDVSP in this paper, the subproblem is the shortest path problem with departure-duration restrictions. As there is a possibility of negative weighted arcs, the SPFA [26] is applied to solve the subproblem. In essence, SPFA is a Bellman-Ford algorithm with queue optimization.

As there are departure-duration restrictions in the MDVSP, the SPFA is modified by embedding a preliminary exploring tactic. Specifically, check if the vertex meets the departure-duration restriction before adding it into the queue. If the restriction is satisfied, add the vertex into the queue. Otherwise, do not add it. The depot vertices do not need to be checked by the preliminary exploring tactic. Only terminal station vertices have to be checked. Let u be the terminal station vertex that has to be checked by the preliminary exploring tactic. $T[u]$ denotes the departure-duration time of a bus tour that starts from a depot vertex and ends at vertex u. $D[u]$ represents the corresponding tour cost. $F[u]$ is the vertex just before u on the bus tour with minimal tour cost among all bus tours that start from a depot vertex and end at vertex u. For a terminal station vertex u, traverse all arcs (including task arcs and waiting arcs) starting from u. If it is a task arc and $T[u]+t+td\leq t_0$, where t is the trip time of the task arc, td is the time from the end of the task arc to a depot, and t_0 is the known working time limits, continue the SPFA. Otherwise, vertex u cannot be added to the queue. If it is a waiting arc, continue searching until a task arc appears.

The flowchart of the modified SPFA is shown in Figure 4.

5. Case Study

In this section, a real-life case in China with 3 depots, 8 bus lines, and 930 timetabled trips is selected to evaluate the performance of the proposed methodology. Some basic information about the real-life case is shown in the "Supplementary Materials" (available here). There are 96 buses in total for the real-life case with 52 buses in Depot 1, 23 buses in Depot 2, and 21 buses in Depot 3.

The proposed algorithm is implemented by c# language in Visual Studio 2005. ILOG CPLEX is used to solve the DRMP. In this paper, let r=20. The program runs on a computer with CPU Intel T7500 2.20GHz 2G. The optimal schedule scheme for the real-life case is shown as Table S5 in the "Supplementary Materials". 87 buses are used to perform the 930 timetabled trips with 48 buses in Depot 1 performing 511 timetabled trips, 21 buses in Depot 2 performing 207 timetabled trips, and 18 buses in Depot 3 performing 212 timetabled trips, respectively.

To highlight the efficiency of the improved LNS algorithm, the computational result is compared with the result using the branch-and-price algorithm. The branch-and-price algorithm is also implemented by c# language in Visual Studio 2005 and runs on a computer with CPU Intel T7500 2.20GHz 2G. The comparison of the two results is shown in Figure 5.

As shown in Figure 5, both of the algorithms have a fast convergence speed at first and then slow down. However, compared to the improved LNS algorithm, the number of feasible solutions from the branch-and-price algorithm is much smaller. Moreover, it takes much longer to obtain the first feasible solution for branch-and-price. The total cost of the improved LNS algorithm is 299,672 which is slightly more than that of branch-and-price (299,017). This result does make sense as the LNS algorithm optimizes the problem locally while the branch-and-price optimizes the problem globally. The difference in solution quality is due to the difference between local optimization and global optimization. Therefore, the branch-and-price obtains higher quality solutions in the real-life case study. But, as we can see from the results, the difference in the total cost is very small, which can be ignored. However, the improved LNS algorithm begins to converge at 34 minutes, which is much shorter than 104 minutes using branch-and-price. In summary, the improved LNS algorithm can ensure not only the quality of the solution but above all the computational efficiency. On that account, the proposed LNS algorithm is of great significance for large-scale MDVSP, which attaches great importance to the algorithmic efficiency.

In Figure 5 the total cost of the improved LNS algorithm has two hops at 12 minutes and 31 minutes during the iteration process. The reason probably lies in the fixed cost of buses. Compared to other costs like c_o, c_w in Section 2, the fixed cost of a bus is relatively larger. The objective function's value may change significantly as the number of buses needed decreases during the optimization process.

6. Sensitivity Analysis

Although the proposed methodology is tested using a real-life case, extra computational experiments should be designed to further validate whether the algorithm is appropriate for large-scale systems. The most effective method for this purpose is to compare the performance of the improved LNS algorithm and branch-and-price in terms of quality and efficiency, using test instances with different number of timetabled trips. Test instances in this section are generated in a 6km$*$6km square using the same methods as Carpaneto et al. [28]. Test instances with n=500, 1000, 1500, and 2000 timetabled trips are generated. For each n, 10 test instances have been generated as done by Carpaneto et al. [28]. To be comparable, let the number of depots K=3 (which is the same

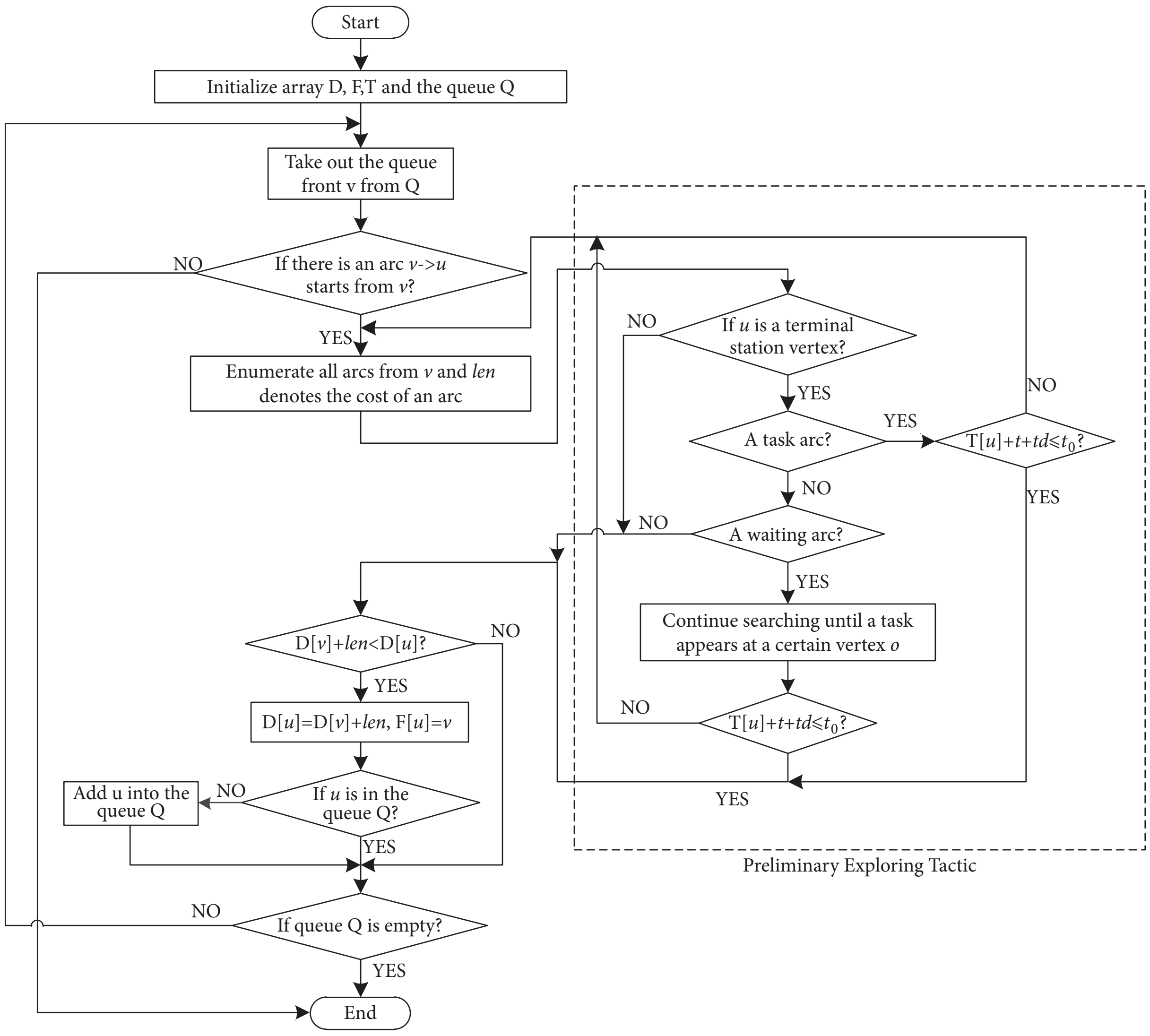
Start
Initialize array D, F,T and the queue Q
Take out the queue front v from Q
If there is an arc v->u starts from v?
NO
YES
Enumerate all arcs from v and len denotes the cost of an arc
If u is a terminal station vertex?
YES
A task arc?
YES
T[u]+t+td≤t0?
NO
YES
NO
A waiting arc?
YES
Continue searching until a task appears at a certain vertex o
T[u]+t+td≤t0?
NO
YES
Preliminary Exploring Tactic
D[v]+len<D[u]?
NO
YES
D[u]=D[v]+len, F[u]=v
If u is in the queue Q?
NO
Add u into the queue Q
YES
If queue Q is empty?
NO
YES
End

FIGURE 4: Modified SPFA with a preliminary exploring tactic.

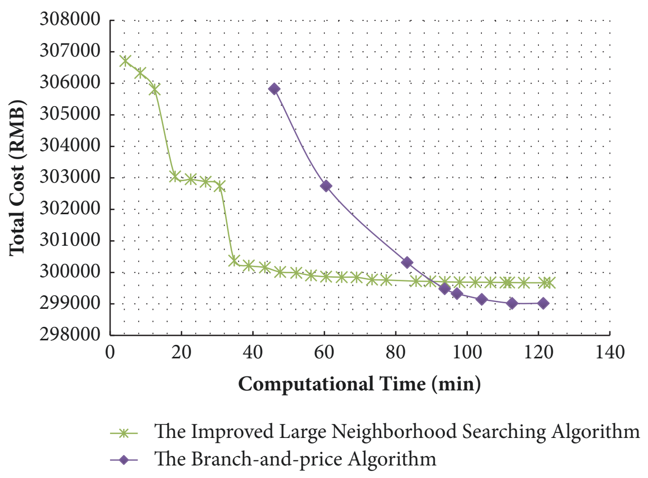

FIGURE 5: Iterative outcomes of two algorithms solving a real-life case.

as the real-life case in Section 5). Quality Gap (QG) is used to compare the solution quality of the two algorithms and Efficiency Ratio (ER) is used to compare the computational efficiency of the two algorithms. QG and ER are defined as follows:

$$QG = \frac{C_l - C_b}{C_b} * 100\% \quad (26)$$

$$ER = \frac{T_b}{T_l} \quad (27)$$

where C_b and T_b are the total cost and computational time of the best solution obtained by branch and-price. C_l and T_l are the total cost and computational time of the best solution obtained by the improved LNS algorithm.

All the test instances are run on a computer with CPU Intel T7500 2.20GHz 2G. The computational results of the test instances with different number of timetabled trips are presented in Table 1.

Table 1: Computational results for test instances with different number of timetabled trips.

Test Number	Number of timetabled trips	Total Cost (RMB)		Quality Gap (%)	Computational Time(min)		Efficiency Ratio
		LNS(C_l)	BP(C_b)		LNS(T_l)	BP(T_b)	
NO. 1	500	151327	150844	0.32	6.0	17.7	2.95
	1000	306801	305670	0.37	36.8	111.1	3.02
	1500	463116	460629	0.54	104.4	380.0	3.64
	2000	605876	600516	0.89	153.1	597.1	3.90
NO. 2	500	167877	167408	0.28	6.7	19.3	2.88
	1000	325414	324021	0.43	36.7	110.8	3.02
	1500	468142	465026	0.67	101.9	379.1	3.72
	2000	588228	582392	1.00	151.8	567.7	3.74
NO. 3	500	159752	159274	0.30	7.4	20.8	2.81
	1000	335644	334573	0.32	35.6	104.0	2.92
	1500	448323	445457	0.64	104.9	377.6	3.60
	2000	631646	625950	0.91	150.4	562.5	3.74
NO. 4	500	161611	160999	0.38	7.8	23.4	3.00
	1000	306715	305493	0.40	34.5	102.1	2.96
	1500	426785	423881	0.69	108.4	396.7	3.66
	2000	619895	614609	0.86	162.8	613.8	3.77
NO. 5	500	165670	164944	0.44	6.5	18.9	2.91
	1000	309025	307641	0.45	38.2	114.2	2.99
	1500	466090	463218	0.62	114.7	410.6	3.58
	2000	664978	658492	0.98	154.2	573.6	3.72
NO. 6	500	149889	149381	0.34	6.9	19.5	2.83
	1000	318111	317160	0.30	35.0	103.6	2.96
	1500	414619	411818	0.68	104.7	383.2	3.66
	2000	629447	623647	0.93	152.6	567.7	3.72
NO. 7	500	167467	166717	0.45	7.6	21.9	2.88
	1000	314987	313701	0.41	36.8	109.3	2.97
	1500	459879	456909	0.65	117.0	437.6	3.74
	2000	673854	668001	0.88	166.3	623.6	3.75
NO. 8	500	156926	156441	0.31	7.0	19.9	2.84
	1000	313965	312652	0.42	40.5	123.1	3.04
	1500	458466	455279	0.70	110.3	401.5	3.64
	2000	630643	624708	0.95	164.2	635.5	3.87
NO. 9	500	157143	156642	0.32	7.0	20.6	2.94
	1000	318107	317092	0.32	39.4	117.8	2.99
	1500	467748	465143	0.56	103.6	371.9	3.59
	2000	623549	617987	0.90	161.2	617.4	3.83
NO. 10	500	162175	161658	0.32	5.9	16.8	2.85
	1000	310107	308995	0.36	36.1	105.4	2.92
	1500	474581	471985	0.55	111.8	413.7	3.7
	2000	629727	623369	1.02	160.6	611.9	3.81

As seen in Table 1, the total cost of the improved LNS algorithm is slightly more than that of branch-and-price algorithm for each test instance of different trip size (500, 1000, 1500, and 2000). But, as we can see from the Quality Gap, the difference in the total cost is very small, which can be ignored. However, the computational time of the improved LNS algorithm is much less than that of branch-and-price algorithm for each test instance of different trip size (500, 1000, 1500, and 2000) which indicates that the improved LNS algorithm is much more efficient than branch-and-price, and

Table 2: Results of Wilcoxon Signed Ranks Test.

(a) Costs

Ranks

		N	Mean Rank	Sum of Ranks
BP Cost-LNS Cost	Negative Ranks	40[a]	20.50	820.00
	Positive Ranks	0[b]	.00	.00
	Ties	0[c]		
	Total	40		

a. BP Cost < LNS Cost
b. BP Cost > LNS Cost
c. BP Cost = LNS Cost

Test Statistics[a]

	BP Cost-LNS Cost
Z	-5.511[b]
Asymp. Sig. (2-tailed)	.000

a. Wilcoxon Signed Ranks Test
b. Based on negative ranks.

(b) Time

Ranks

		N	Mean Rank	Sum of Ranks
BP Time-LNS Time	Negative Ranks	0[a]	.00	.00
	Positive Ranks	40[b]	20.50	820.00
	Ties	0[c]		
	Total	40		

a. BP Time < LNS Time
b. BP Time > LNS Time
c. BP Time = LNS Time

Test Statistics[a]

	BP Time-LNS Time
Z	-5.511[b]
Asymp. Sig. (2-tailed)	.000

a. Wilcoxon Signed Ranks Test
b. Based on negative ranks.

the advantages of algorithmic efficiency continue to expand as the number of timetabled trips increases.

To test whether there are statistically significant differences between the improved LNS algorithm and the branch-and-price algorithm in total cost and computational time, "Wilcoxon Signed Ranks Test" in SPSS is applied to the data in Table 1. The results are shown in Table 2. The variable name in Table 2 consists of two parts. Take "BP Cost", for example, "BP" represents the method name (branch-and-price) and "Cost" represents the total cost.

As we can see from the **Ranks** table in Table 2, LNS algorithm had a higher total cost but a lower computational time than the branch-and-price algorithm for all test instances. By examining the final **Test Statistics** table in Table 2, we can discover whether there are statistically significant differences between the improved LNS algorithm and the branch-and-price algorithm in total cost and computational time. We are looking for the "**Asymp. Sig. (2-tailed)**" value, which in this case is 0.000, illustrating that the observed differences are statistically significant. However, compared with the significant improvement in algorithmic efficiency brought by the LNS algorithm, the increase in total cost is very small, which can be ignored.

In order to better show the results, the minimum value, lower-quartile value, median value, upper-quartile value, maximum value, and average value of Quality Gap and Efficiency Ratio over 10 test instances for each n (n=500, 1000, 1500, 2000) are calculated, as shown in Tables 3 and 4.

As seen in Tables 3 and 4, there is not a big change in Quality Gap or Efficiency Ratio for different test instances of each n (n=500, 1000, 1500, and 2000). The Quality Gap is approximately 0.35%, 0.38%, 0.63%, and 0.93%, respectively,

Table 3: Quality gap (*QG*).

Quality Gap (%)						
	Different Values					
Number of Trips	Minimum Value	Lower quartile Value	Median Value	Upper quartile Value	Maximum Value	Average Value
500	0.28	0.31	0.32	0.37	0.45	0.35
1000	0.30	0.33	0.39	0.42	0.45	0.38
1500	0.54	0.58	0.65	0.68	0.70	0.63
2000	0.86	0.89	0.92	0.97	1.02	0.93

Table 4: Efficiency ratio (*ER*).

Efficiency Ratio						
	Different Values					
Number of Trips	Minimum Value	Lower quartile Value	Median Value	Upper quartile Value	Maximum Value	Average Value
500	2.81	2.84	2.88	2.93	3.00	2.89
1000	2.92	2.96	2.98	3.01	3.04	2.98
1500	3.58	3.61	3.65	3.69	3.74	3.65
2000	3.72	3.74	3.76	3.83	3.90	3.79

for test instances with different number of timetabled trips (500, 1000, 1500, and 2000), while the Efficiency Ratio reaches up to 2.89, 2.98, 3.65, and 3.79, respectively, for test instances with different number of timetabled trips (500, 1000, 1500, and 2000). The small Quality Gap demonstrates the effectiveness of the improved LNS algorithm to produce high quality solutions. Meanwhile, the large Efficiency Ratio clearly highlights the excellent computational efficiency of the proposed heuristic.

Based on Tables 1 and 2 and the computational results of Quality Gap and Efficiency Ratio presented in Tables 3 and 4, the findings suggest that the improved LNS algorithm is effective in obtaining faster solutions without deteriorating the quality, especially for large-scale instances.

7. Conclusions and Future Work

To solve this large-scale MDVSP, the only feasible approach is to downsize the problem and improve the algorithm simultaneously. Crews working overtime increase safety risks for themselves and others, so buses should return to the depot and change shifts when the crews reach their working time limits. Therefore, an improved set-partitioning model with departure-duration restrictions was established based on the time-space network, calling for the determination of a collection of circuits with minimum cost. In order to solve the MDVSP model, an improved LNS algorithm was proposed. To deal with departure-duration restrictions, the SPFA was modified by embedding a preliminary exploring tactic to solve the subproblem. A case study in China was presented to evaluate the performance of the proposed algorithm. The results indicate that the improved LNS algorithm, which encompasses attributes of both fast computing speed and high quality solutions, is especially appropriate for an extremely large-scale MDVSP. The sensitivity analysis in Section 6 further verifies the effectiveness of the proposed methodology to solve a large-scale MDVSP.

In this study, only bus scheduling is considered, which is just one part of the whole bus operating planning process. Future study should be directed at researching all four parts (i.e., bus network design, timetable design, bus scheduling, and crew scheduling) and their integration to continue developing a more stable and sophisticated bus scheduling plan. In addition, in the proposed LNS algorithm, the selecting strategy of the r bus circuits and the value of parameter r affect the computational efficiency of the proposed algorithm. A good selecting strategy and r value will further improve the LNS algorithm. The selecting strategy is a recommended future research topic. How to scientifically select the value of parameter r also needs further research.

Data Availability

The data used to support the findings of this study are available from the first author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Supplementary Materials

Some basic information and results about the real-life case including (1) line routes and the depot positions of the real-life case, (2) line length, average speed, opening time, closing time, and the number of timetabled trips of each bus line, (3) travel time between the depots and the origin (destination) stations of each bus line, (4) travel time between different origin and destination stations of each bus line, (5) timetable information of each bus line, and (6) the optimal schedule scheme for the real-life case are shown as Figure S1, Table S1, Table S2, Table S3, Table S4, and Table S5 in the "Supplementary Materials". In Table S2 and Table S3, " *i*-O" represents the origin station of Line *i* (upward) and " *i*-D" represents the destination station of Line i (upward). The origin station of an upward bus line is the destination station of its corresponding downward bus line and the destination station of an upward bus line is the origin station of its corresponding downward bus line. Some lines have the same origin station or destination station; for example, Line 1 (upward), Line 2 (upward), and Line 6 (upward) have the same destination station. *(Supplementary Materials)*

References

[1] X. Zuo, C. Chen, W. Tan, and M. Zhou, "Vehicle scheduling of an urban bus line via an improved multiobjective genetic algorithm," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 1030–1041, 2015.

[2] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse, "The vehicle routing problem: State of the art classification and review," *Computers & Industrial Engineering*, vol. 99, pp. 300–313, 2016.

[3] C. A. Hane, C. Barnhart, E. Johnson, R. E. Marsten, G. L. Nemhauser, and G. Sigismondi, "The fleet assignment problem: solving a large-scale integer program," *Mathematical Programming*, vol. 70, no. 1, pp. 211–232, 1995.

[4] N. Kliewer, T. Mellouli, and L. Suhl, "A new solution model for multi-depot multi-vehicle-type vehicle scheduling in (sub) urban public transport," in *Proceedings of the 13th Mini-EURO Conference and the 9th Meeting of the EURO Working Group on Transportation*, 2002.

[5] N. Kliewer, T. Mellouli, and L. Suhl, "A time-space network based exact optimization model for multi-depot bus scheduling," *European Journal of Operational Research*, vol. 175, no. 3, pp. 1616–1627, 2006.

[6] Z. Y. Chen, *The Comparison between Connection-Based Network and Time-Space Network on Multiple-Depot Multiple-Vehicle-Type Scheduling Problem [Master thesis]*, National Taiwan University of Science and Technology, 2005.

[7] V. Gintner, N. Kliewer, and L. Suhl, "Solving large multiple-depot multiple-vehicle-type bus scheduling problems in practice," *OR Spectrum*, vol. 27, no. 4, pp. 507–523, 2005.

[8] M. Naumann, L. Suhl, and S. Kramkowski, "A stochastic programming approach for robust vehicle scheduling in public bus transport," *Procedia-Social and Behavioral Sciences*, vol. 20, pp. 826–835, 2011.

[9] M. Saint-Guillain, C. Solnon, and Y. Deville, "The static and stochastic vrp with time windows and both random customers and reveal times," in *Proceedings of the European Conference on the Applications of Evolutionary Computation*, pp. 110–127, 2017.

[10] A. García-Nájera, J. A. Bullinaria, and M. A. Gutiérrez-Andrade, "An evolutionary approach for multi-objective vehicle routing problems with backhauls," *Computers & Industrial Engineering*, vol. 81, pp. 90–108, 2015.

[11] M. Avci and S. Topaloglu, "A hybrid metaheuristic algorithm for heterogeneous vehicle routing problem with simultaneous pickup and delivery," *Expert Systems with Applications*, vol. 53, pp. 160–171, 2016.

[12] M. L. Balinski and R. E. Quandt, "On an integer program for a delivery problem," *Operations Research*, vol. 12, no. 2, pp. 300–304, 1964.

[13] P. Toth and D. Vigo, *Vehicle Routing: Problems, Methods, and Applications*, Society for Industrial and Applied Mathematics, SIAM Publishing, Philadelphia, PA, USA, 2014.

[14] M. Goerigk, B. Grün, and P. Heßler, "Branch and bound algorithms for the bus evacuation problem," *Computers & Operations Research*, vol. 40, no. 12, pp. 3010–3020, 2013.

[15] Y. Kumar and S. Jain, "School bus routing based on branch and bound approach," in *Proceedings of the IEEE International Conference on Computer, Communication and Control, IC4 2015*, pp. 1–4, Indore, India, September 2015.

[16] L. C. Coelho and G. Laporte, "The exact solution of several classes of inventory-routing problems," *Computers & Operations Research*, vol. 40, no. 2, pp. 558–565, 2013.

[17] Z. Michalewicz and D. Fogel, *How to Solve It: Modern Heuristics*, Springer-Verlag, New York, NY, USA, 2000.

[18] X. Li, G. Lin, and C. Shen, "Learning hash functions using column generation," in *Proceedings of the International Conference on Machine Learning*, pp. 142–150, 2013.

[19] A. Ceselli, G. Righini, and M. Salani, "A column generation algorithm for a rich vehicle-routing problem," *Transportation Science*, vol. 43, no. 1, pp. 56–69, 2009.

[20] C. C. Ribeiro and F. Soumis, "Column generation approach to the multiple-depot vehicle scheduling problem," *Operations Research*, vol. 42, no. 1, pp. 41–52, 1994.

[21] S. Dabia, S. Ropke, T. Van Woensel, and T. De Kok, "Branch and price for the time-dependent vehicle routing problem with time windows," *Transportation Science*, vol. 47, no. 3, pp. 380–396, 2013.

[22] D. Taş, M. Gendreau, N. Dellaert, T. van Woensel, and A. G. de Kok, "Vehicle routing with soft time windows and stochastic travel times: a column generation and branch-and-price solution approach," *European Journal of Operational Research*, vol. 236, no. 3, pp. 789–799, 2014.

[23] I. Muter, J.-F. Cordeau, and G. Laporte, "A branch-and-price algorithm for the multidepot vehicle routing problem with interdepot routes," *Transportation Science*, vol. 48, no. 3, pp. 425–441, 2014.

[24] A.-S. Pepin, G. Desaulniers, A. Hertz, and D. Huisman, "A comparison of five heuristics for the multiple depot vehicle scheduling problem," *Journal of Scheduling*, vol. 12, no. 1, pp. 17–30, 2009.

[25] M. Savelsbergh, "A branch-and-price algorithm for the generalized assignment problem," *Operations Research*, vol. 45, no. 6, pp. 831–841, 1997.

[26] D. Fanding, "A faster algorithm for shortest path-SPFA," *Journal of Southwest Jiaotong University*, vol. 29, no. 2, pp. 207–212, 1994 (Chinese).

[27] A. M. Campbell and M. Savelsbergh, "Efficient insertion heuristics for vehicle routing and scheduling problems," *Transportation Science*, vol. 38, no. 3, pp. 369–378, 2004.

[28] G. Carpaneto, M. Dell'Amico, M. Fischetti, and P. Toth, "A branch and bound algorithm for the multiple depot vehicle scheduling problem," *Networks*, vol. 19, no. 19, pp. 531–548, 1989.