Computers & Operations Research 59 (2015) 78–93

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

A tabu search algorithm for the multi-period inspector scheduling problem

Hu Qin [a], Wei Ming [a], Zizhen Zhang [b,*], Yubin Xie [c], Andrew Lim [d]

[a] School of Management, Huazhong University of Science and Technology, No. 1037, Luoyu Road, Wuhan, China
[b] School of Mobile Information Engineering, Sun Yat-Sen University, Tang Jia Bay, Zhuhai, Guangdong, China
[c] School of Information Science and Technology, Sun Yat-Sen University, Guangzhou, China
[d] Department of Management Sciences, City University of Hong Kong, Tat Chee Ave., Kowloon Tong, Kowloon, Hong Kong

ARTICLE INFO

Available online 23 January 2015

Keywords:
Tabu search
Routing
Meta-heuristics
Inspector scheduling problem
Hotel selection

ABSTRACT

This paper introduces a multi-period inspector scheduling problem (MPISP), which is a new variant of the multi-trip vehicle routing problem with time windows (VRPTW). In the MPISP, each inspector is scheduled to perform a route in a given multi-period planning horizon. At the end of each period, each inspector is not required to return to the depot but has to stay at one of the vertices for recuperation. If the remaining time of the current period is insufficient for an inspector to travel from his/her current vertex *A* to a certain vertex *B*, he/she can choose either waiting at vertex *A* until the start of the next period or traveling to a vertex *C* that is closer to vertex *B*. Therefore, the shortest transit time between any vertex pair is affected by the length of the period and the departure time. We first describe an approach of computing the shortest transit time between any pair of vertices with an arbitrary departure time. To solve the MPISP, we then propose several local search operators adapted from classical operators for the VRPTW and integrate them into a tabu search framework. In addition, we present a constrained knapsack model that is able to produce an upper bound for the problem. Finally, we evaluate the effectiveness of our algorithm with extensive experiments based on a set of test instances. Our computational results indicate that our approach generates high-quality solutions.

1. Introduction

This paper studies a new manpower routing and scheduling problem faced by a company that procures products from over one thousand suppliers across Asia. The company places orders with a large number of suppliers and must inspect the goods at the factories of the suppliers before shipment. Therefore, the suppliers are required to make inspection requests with the company when the ordered goods are ready for delivery. An inspection request is characterized by the workload, the inspection site and the time window within which the inspection can be started. In turn, the company dispatches a team of professional quality inspectors to perform all on-site inspections. In order to facilitate coordination between inspectors and suppliers, the inspections could only be carried out during working hours (e.g., 8:00 am–6:00 pm). Usually, a weekly schedule is created to assign inspectors to requests for the upcoming week. The company has a stable of in-house inspectors, each having a specified weekly workload limit, and the unfulfilled inspection requests are outsourced to external agencies with additional costs. After receiving their weekly inspection schedules, the inspectors depart from the regional office and will not report back until they have performed all their assigned inspections for the week. More specifically, they leave the regional office on Monday, visit a set of inspection sites and return to the regional office on Friday or some earlier workday. In each workday, an inspector generally travels to diffident locations, completes several inspections and finds overnight accommodation (i.e., hotel) in the vicinity of his/her last/next inspection site at or before the end of the office hours. The objective of the problem is to assign as many inspection workloads as possible to the stable of in-house inspectors while satisfying all the above-mentioned practical constraints.

We call this problem the *multi-period inspector scheduling problem* (MPISP), which can be viewed as a variant of the multi-trip vehicle routing problem with time windows (VPRTW) [2,30]. There are four main features that distinguish the MPISP from the multi-trip VRPTW. First, the scheduling subjects, e.g., vehicles or inspectors, are not required to return to the regional office every workday. Second, at the end of each workday, each scheduling subject must stay at one of the vertices for recuperation. Third, each vertex can be visited more than once. If the remaining time of

* Corresponding author. Tel.: +86 13826411848.
E-mail addresses: tigerqin@hust.edu.cn, tigerqin1980@gmail.com (H. Qin), mingwayway@gmail.com (W. Ming), zhangzizhen@gmail.com (Z. Zhang), xyb.0606@gmail.com (Y. Xie), lim.andrew@cityu.edu.hk (A. Lim).

http://dx.doi.org/10.1016/j.cor.2015.01.003
0305-0548/

the current period is insufficient for an inspector to travel from his/her current vertex A to a certain vertex B, he/she can choose either waiting at vertex A until the start of the next period or traveling to a vertex C that is closer to vertex B. The vertex C is called a *waypoint*, which only acts as the intermediate point in a route. Fourth, the objective is to maximize the total inspected workload rather than to minimize the number of inspectors used and/or the total distance traveled.

In this study, we propose a tabu search algorithm to solve the MPISP. This algorithm employs a tailored fitness function consisting of three lexicographically ordered components, a local improvement procedure with tabu moves, an ejection pool improvement process and a perturbation phase. The contributions of this study are fourfold. First, we introduce a new and practical multi-period manpower routing and scheduling problem that considers multiple working periods. Second, we provide an effective tabu search algorithm that uses a set of problem-specific neighborhood search operators. Third, we construct a constrained knapsack model that can produce an upper bound for the MPISP. Fourth, the comprehensive experimental results on a large number of test instances show the effectiveness of our approach.

The remainder of this paper is organized as follows. We first provide an overview of related research in Section 2. In Section 3, we then give a formal definition of the MPISP. In Section 4, we describe an approach of computing the shortest transit time for any pair of vertices with any departure time. Our proposed tabu search algorithm is detailed in Section 5 and the constrained knapsack model is presented in Section 6. Section 7 reports the experiments results and Section 8 concludes this study with some closing remarks.

2. Related work

The MPISP is one type of manpower scheduling problems. Scheduling staff members is a traditional research area; example problems include the nurse rostering problem [10], the technician planning problem [15] and the airline crew rostering problem [23]. As for the manpower scheduling problems that involve creating routes for staff members, we refer the reader to Li et al. [27], Tang et al. [36], Zäpfel and Bögl [43], Cai et al. [7], Zhang et al. [44].

Since each inspector has to perform inspections at different locations, the MPISP is essentially a variant of the vehicle routing problem [37]. One of the defining characteristics is its objective of maximizing the total inspected workload. Two previously studied problems with similar objective are the team orienteering problem with time windows (TOPTW) [41,39,25,22] and the vehicle routing problem with time windows and a limited number of vehicles (m-VRPTW) [26,28]. The m-VRPTW is an extension of the TOPTW with the consideration of vehicle capacity and customer demands. These two problems both aim to determine a set of routes that maximizes the total reward of the vertices visited during a single period with a distance or duration limit. The multi-period planning horizon of the MPISP is related to the periodic vehicle routing problem (PVRP) [16,21] and the multiple trip vehicle routing problem (MTVRP) [4]. However, the MPISP is quite different from the PVRP and MTVRP. In the PVRP, each customer requires a certain number of visits within the planning horizon, and two types of decisions are involved in the planning, namely determining the visit days for each customer and the routing plan for each time period. The PVRP and MTVRP both require that each vehicle must return to the depot at the end of each period.

Another defining characteristic of the MPISP is the consideration of multiple working periods. Working hour regulations have recently received increasing attention from some researchers studying vehicle routing problems. Savelsbergh and Sol [34] proposed a dynamic and general pickup and delivery problem in which lunch and night breaks must be taken into account. Xu et al. [42] applied column generation based solution approaches to solve a pickup and delivery vehicle routing problem that involves a set of practical complications, such as heterogeneous vehicles, last-in-first-out loading and unloading operations, pickup and delivery time windows, and working hour restrictions by the United States Department of Transportation. Similarly, Goel [18,19] and Kok et al. [24] investigated combined vehicle routing and driver scheduling problems under the European Union regulations for drivers.

The MPISP problem can be viewed as a natural extension of the orienteering problem with hotel selection (OPHS) [13,14]. In the OPHS, a scheduling subject can visit a set of vertices each with a score and find accommodation at a given set of hotels. The tour is divided into multiple trips, each with a limited duration and starting from and ending at one of the hotels. The objective of the OPHS is to determine a tour that maximizes the total collected score. The OPHS is a variant of the traveling salesperson problem with hotel selection (TSPHS) [40,8], which aims to serve all vertices with the minimum number of connected trips and the minimum total travel distance. The common characteristic of the above three problems is the involvement of hotel selection.

Tang et al. [36] and Zäpfel and Bögl [43] introduced two manpower routing and scheduling problems that involve maximization of profits, multiple periods and working hour restrictions. However, their problems require that the trip in each period must start from and end at the depot. Most recently, Zhang et al. [44] proposed an inspector scheduling problem which is very similar to our problem. Their problem differs from our problem in the following four assumptions: (1) each vertex can only be visited at most once; (2) if a vertex is visited by an inspector, its inspection request must be fulfilled by this inspector; (3) an inspector reaches a vertex and completes the corresponding inspection task in the same period; and (4) each vehicle stays at the last served vertex at the end of each period and begins the trip of the next period from that vertex. By ignoring these four assumptions, the MPISP is more difficult but practical.

3. Problem description

The MPISP is defined on a directed graph $G=(V,E)$, where $V=\{0,1,\dots,n\}$ is the vertex set and $E=\{(i,j): i,j\in V, i\neq j\}$ is the edge set. Vertex 0 represents the depot location and $V_C=\{1,\dots,n\}$ denotes the locations of n suppliers. Each supplier i is characterized by a location $i\in V_C$, a workload d_i, a required service time s_i and a time window $[e_i,l_i]$. For notational convenience, we assign $d_0=0$ and $s_0=0$ for the depot. Each edge $(i,j)\in E$ requires a non-negative traveling time $t_{i,j}$, where the matrix $[t_{i,j}]$ satisfies the triangle inequality.

We are given a set K of m homogeneous inspectors, each of which has a workload limit Q and can only work within a set $P=\{1,\dots,w\}$ of w *working periods* (or called *working time windows*). For any period $p\in P$, a_p and b_p ($a_p<b_p$) are its starting and closing working times, respectively, and b_p-a_p equals a positive constant T that is not less than s_i for any $i\in V$. An inspector can arrive at vertex $i\in V_C$ prior to e_i and wait at no cost until the service of supplier i becomes possible. All inspectors must leave the depot after e_0 ($e_0=a_1=0$) and return to the depot before l_0 ($l_0=b_w$), where $[e_0,l_0]$ is called *depot time window*. At the end of each period, each inspector is not required to return to the depot but has to stop traveling and stay at one of the vertices. Moreover, service cannot be interrupted, i.e., if the service of some supplier cannot be completed before the end of a period, it must be restarted in the later periods. Each vertex can be visited more than once while each supplier can be served by at most one inspector, so some supplier locations can be used as *waypoints*. The objective of the

MPISP is to construct m inspector routes to complete as many workloads as possible while respecting depot time window, workload limit, supplier time windows and inspector working time windows. We provide a mixed integer programming model for the MPISP in Appendix A.

In reality, a_p should be larger than b_{p-1}, and the duration between a_p and b_{p-1} is the downtime for rest and recuperation. Without loss of generality, we can assume that the length of the downtime is extremely small by setting $b_{p-1} = a_p$ and imposing a break at time b_{p-1}. As illustrated in Fig. 1, we can easily transform the non-zero downtime cases to zero downtime ones. In Fig. 1(a), T=20 and the time windows of suppliers 1, 2, and 3 are [5, 90], [10, 50] and [85, 95], respectively. After transformation, their time windows become [5, 50], [10, 30] and [45, 55] (see Fig. 1(b)).

To further describe the MPISP, we convert the graph $G = (V, E)$ into a directed (not complete) graph $G' = (V', E')$ by the following three steps: (1) split each vertex $i \in V_C$ into two vertices $i+$ and $i-$, and create an edge $(i+, i-)$, where vertex $i+$ represents the arrival of vertex i and vertex $i-$ represents the completion of supplier i's service; (2) create edges $(0, i+)$, $(i+, 0)$, $(i-, 0)$, $(i+, j+)$ and $(i-, j+)$, where $i, j \in V_C$ and $i \neq j$; and (3) set $t_{i+,i-} = s_i$, $t_{0,i+} = t_{0,i}$, $t_{i+,0} = t_{i-,0} = t_{i,0}$ and $t_{i+,j+} = t_{i-,j+} = t_{i,j}$. An example to illustrate this conversion is shown in Fig. 2, where Fig. 2(b) is the resultant graph derived from Fig. 2(a).

We can denote a feasible solution of the MPISP by S, consisting of m routes, namely $S = \{r_1, r_2, \ldots, r_m\}$. A route r_k $(1 \le k \le m)$ is divided into w sub-routes by periods and therefore can be expressed as $r_k = (r_k^1, r_k^2, \ldots, r_k^w)$, where r_k^p $(1 \le p \le w)$ denotes the trip in period p. If an inspector returns to the depot before period w, he/she will stay at the depot for the remaining periods. The sub-route r_k^p is a sequence of vertices, where its starting and ending vertices are denoted by $v_s(r_k^p)$ and $v_e(r_k^p)$, respectively. If an inspector k stays at the depot during the whole period p, we set $r_k^p = (0)$ and $v_s(r_k^p) = v_e(r_k^p) = 0$. According to the definition of our problem, an inspector must stay at vertex $v_e(r_k^p)$ for rest and will start the next trip from this vertex in period $p+1$, i.e, $v_e(r_k^p) = v_s(r_k^{p+1})$ for all $1 \le p \le w-1$. Obviously, the starting vertex of period 1 and the ending vertex of period w for each route must be vertex 0. Fig. 3 gives a feasible solution to an MPISP instance involving 10 suppliers, four inspectors and three periods.

In Fig. 3, the solid and dash lines denote the working (i.e., traveling or providing service) and idle statuses of the inspectors, respectively. The set $U = \{9, 10\}$ indicates that suppliers 9 and 10 are not served by any inspector. The route $r_1 = (r_1^1, r_1^2, r_1^3)$ comprises three trips, i.e., $r_1^1 = (0, 1+)$, $r_1^2 = (1+, 1-, 2+, 2-)$ and $r_1^3 = (2-, 3+, 3-, 0)$. This route shows that inspector 1 arrives at vertex 1, but does not have sufficient time to complete the service for supplier 1 in the first period. Thus, he/she has to wait until the start of the second period and then provides service to supplier 1. Subsequently, inspector 1 travels to vertex 2, completes the service of supplier 2 and stays at vertex 2 for recuperation. In the third period, inspector 1 travels from vertex 2 to vertex 3, provides service for supplier 3, and finally returns to the depot. Since each supplier i can be served by at most one inspector, edge $(i+, i-)$ can be included in at most one route. In route r_2, after completing the service of supplier 4, inspector 2 travels to vertex 5 via a waypoint, namely vertex 3. Note that any waypoint must be the ending vertex of a certain period (and also be the starting vertex of the following period) due to the rule of triangle inequality. As shown

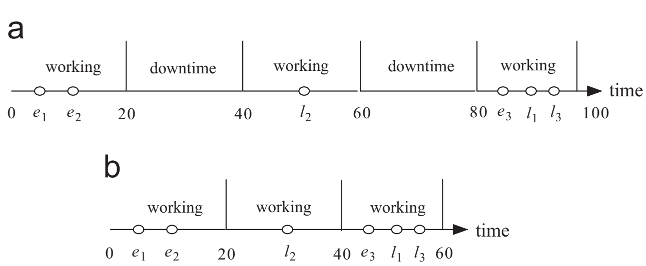

Fig. 1. (a) The original case. (b) The case after transformation.

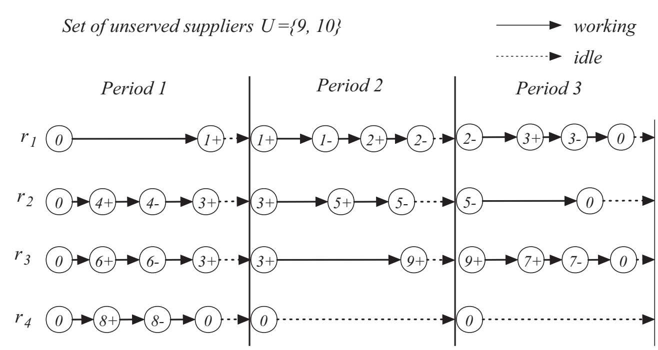

Fig. 3. An example feasible solution to an MPISP instance.

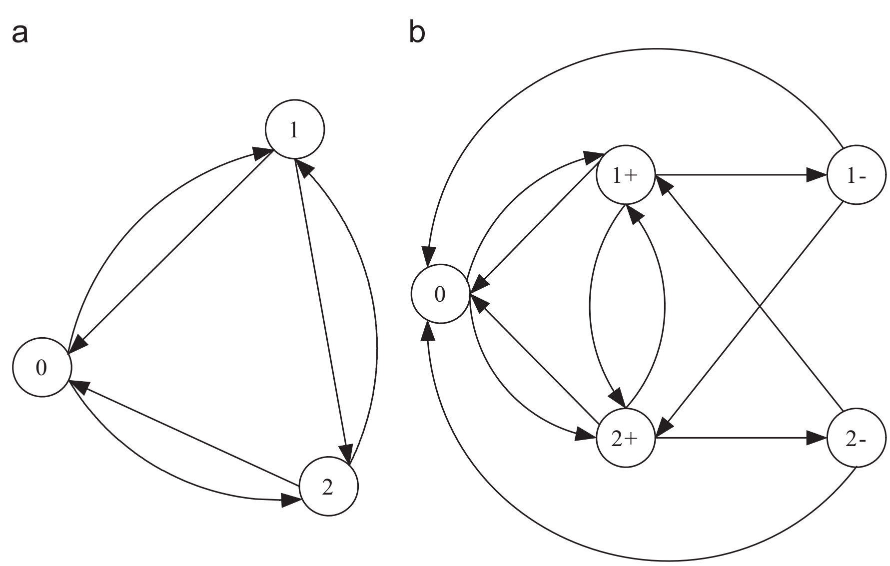

Fig. 2. (a) $G = (V, E)$. (b) $G' = (V', E')$.

in route r_3, an inspector may use two or more waypoints between two consecutively served suppliers. In this route, inspector 3 visits but does not serve supplier 9, i.e., vertex 9 only acts as a waypoint. As no route traverses edge $(9+, 9-)$, supplier 9 is not served by any inspector in this solution. The route r_4 illustrates that an inspector may be idle during some periods; its three trips are represented by $r_4^1 = (0, 8+, 8-, 0)$, $r_4^2 = (0)$ and $r_4^3 = (0)$.

4. Shortest transit time

In a complete graph that satisfies the triangle inequality, the shortest path from vertex i to vertex j must be edge (i,j). When working periods are imposed on the inspectors, edge (i,j) may be unusable in some situations and therefore the shortest transit time may be greater than $t_{i,j}$. The simplest such situation can be encountered when $t_{i,j} > T$. To move from vertex i to vertex j, an inspector has to use some waypoints and the transit time may cross several periods. We illustrate this situation in Fig. 4, where an inspector has completed the service of supplier i and departs from vertex i at the beginning of a certain period.

Unlike the classical VRP models, in the MPISP the shortest transit times from vertex i to other vertices are affected by the departure time (denoted by dt_i) of the inspector. Therefore, we define $\hat{t}_{i,j}(dt_i)$ as the shortest transit time from vertex i to vertex j with departure time dt_i. If dt_i is the opening time of a certain period, i.e., $dt_i = a_p$ for some $p \in P$, $\hat{t}_{i,j}(dt_i)$ can be simplified to $\hat{t}_{i,j}$. Further, we define $\text{ceil}(dt_i)$ as the closing time of the period within which dt_i lies, i.e., if $a_p < dt_i \le b_p$, then $\text{ceil}(dt_i) = b_p$. If $\text{ceil}(dt_i) - dt_i \ge t_{i,j}$, an inspector can travel across edge (i,j) within the current period and thus $\hat{t}_{i,j}(dt_i) = t_{i,j}$. Otherwise, the inspector has to either wait at vertex i until the start of the next period or travel to some waypoint u. We illustrate these situations in Fig. 5, where an inspector may travel from vertex i to vertex j via some waypoint.

As previously mentioned, a waypoint u can only be positioned as the last or the first vertex in the trip of some period. More precisely, if an inspector travels to a waypoint u, he must stay at u for downtime (see Fig. 5(b)). Taking $N(dt_i) = \{u \in V \mid t_{i,u} \le \text{ceil}(dt_i) - dt_i\}$ to be the set of all vertices that can act as waypoints for vertex i, the value of $\hat{t}_{i,j}(dt_i)$ can be calculated by

$$\hat{t}_{i,j}(dt_i) = \begin{cases} t_{i,j} & \text{if } \text{ceil}(dt_i) - dt_i \ge t_{i,j}; \\ \text{ceil}(dt_i) - dt_i + \min\limits_{u \in N(dt_i) \cup \{i\}} \{\hat{t}_{u,j}\} & \text{otherwise} \end{cases} \tag{1}$$

The above expression shows that computing any $\hat{t}_{i,j}(dt_i)$ requires $O(n)$ time given the values of all $\hat{t}_{i,j}$, which can be calculated prior to applying any algorithm to the problem. If the last waypoint between vertex i and vertex j is vertex u, the corresponding shortest transit time, denoted by $\hat{t}^u_{i,j}$, can be obtained by

$$\hat{t}^u_{i,j} = \begin{cases} \hat{t}_{i,u} + t_{u,j} & \text{if } \text{ceil}(\hat{t}_{i,u}) - \hat{t}_{i,u} \ge t_{u,j}; \\ \text{ceil}(\hat{t}_{i,u}) + t_{u,j} & \text{if } \text{ceil}(\hat{t}_{i,u}) - \hat{t}_{i,u} < t_{u,j} \le T \\ +\infty & \text{otherwise}. \end{cases} \tag{2}$$

Obviously, we have

$$\hat{t}_{i,j} = \min_{u \in V} \{\hat{t}^u_{i,j}\}$$

To compute all $\hat{t}_{i,j}$, we can apply an algorithm modified from Dijkstra's algorithm [1], one of the most well-known label-setting algorithms for the classical shortest path problem. This modified Dijkstra's algorithm employs expression (2) as the extension function and has a time complexity of $O(n^2)$. Since we need to compute the shortest transit time between each vertex pair, the total time complexity for all $\hat{t}_{i,j}$ is bounded by $O(n^3)$.

We can accelerate the computation of $\hat{t}_{i,j}(dt_i)$ by the following procedure. We first remove from the graph all edges whose lengths are greater than T and then sort all neighbors u of vertex i in ascending order of $t_{i,u}$, generating a vertex sequence $(i_0, i_1, \ldots, i_h)$. Note that we have $i_0 = i$ since $t_{i,i} = 0$. For $0 \le k \le h$, let $\hat{t}^{(k)}_{i,j} = \min_{0 \le k' \le k}\{\hat{t}_{i_{k'},j}\}$ be the shortest transit time from one of the first $k+1$ vertices in the sequence to vertex j. The values of all $\hat{t}^{(k)}_{i,j}$ can be computed by Algorithm 1 in time complexity of $O(n^3)$. Fig. 6 pictorially shows the process of computing all $\hat{t}^{(k)}_{i,j}$. According to expression (1),

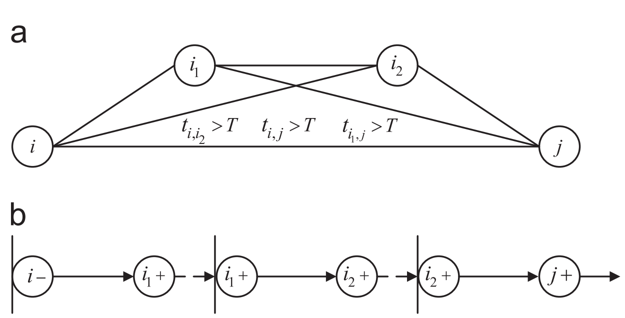

Fig. 4. (a) The vertex locations. (b) The route associated with the shortest transit time from vertex i to vertex j, where vertices i_1 and i_2 are waypoints.

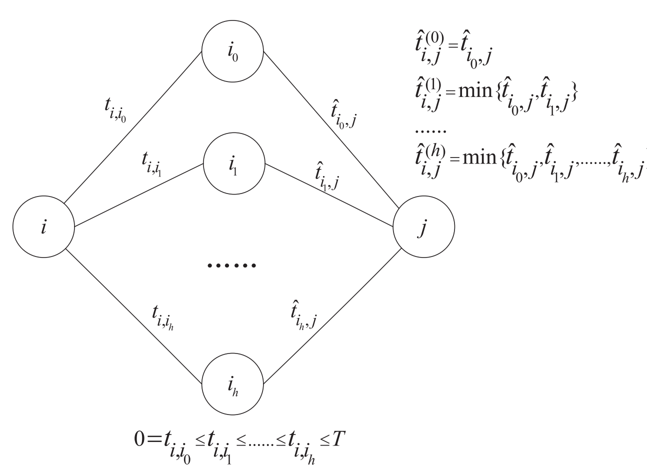

Fig. 6. The process of computing all $\hat{t}^{(k)}_{i,j}$.

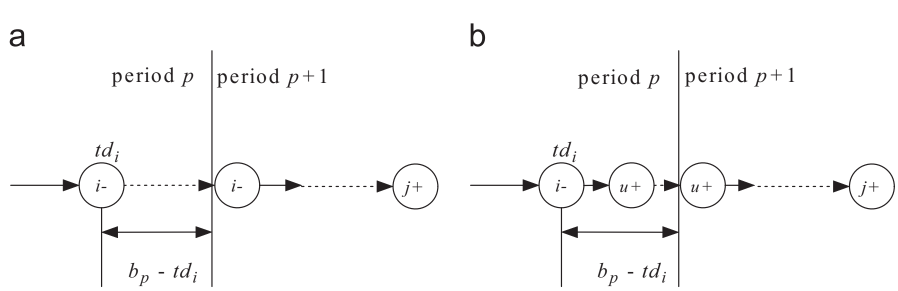

Fig. 5. (a) Wait at vertex i. (b) Travel to a waypoint u.

$\hat{t}_{i,j}(dt_i) = \text{ceil}(dt_i) - dt_i + \min_{u \in N(dt_i) \cup \{i\}} \{\hat{t}_{u,j}\}$ if $\text{ceil}(dt_i) - dt_i < t_{i,j}$. To achieve this $\hat{t}_{i,j}(dt_i)$, we identify the largest k satisfying ceil$(dt_i) - dt_i \geq t_{i,i_k}$ using binary search on $t_{i,i_0}, t_{i,i_1}, \ldots, t_{i,i_h}$, and retrieve the value of $\hat{t}^{(k)}_{i,j}$, which is equal to $\min_{u \in N(dt_i) \cup \{i\}} \{\hat{t}_{u,j}\}$. The above procedure shows that the time complexity of computing $\hat{t}_{i,j}(dt_i)$ can be reduced to $O(\log n)$ given that all $\hat{t}^{(k)}_{i,j}$ are available.

Algorithm 1. The algorithm for preprocessing all $\hat{t}^{(k)}_{i,j}$.

INPUTS: all $\hat{t}_{i,j}$;
for $i = 0$ to n **do**
  Sort all neighbors u of vertex i in ascending order of $t_{i,u}$ to generate a vertex sequence $(i = i_0, i_1, \ldots, i_h)$;
  for $j = 0$ to n **do**
    $\hat{t}^{(0)}_{i,j} = \hat{t}_{i,j}$;
    for $k = 1$ to h **do**
      $\hat{t}^{(k)}_{i,j} = \min\{\hat{t}^{(k-1)}_{i,j}, \hat{t}_{i_k,j}\}$;
    end for
  end for
end for

The computation of all $\hat{t}_{i,j}$ and $\hat{t}^{(k)}_{i,j}$ can be done in a preprocessing stage, which requires a time complexity of $O(n^3)$. The approach described in the following section needs to frequently compute $\hat{t}_{i,j}(dt_i)$. Thus, this preprocessing stage is particularly useful to save the overall computation time.

5. Tabu search algorithm

Tabu search algorithm has been successfully applied to a wide variety of routing and scheduling problems, such as the classical VRP [17,38], the VRPTW [11,20], the three-dimensional loading capacitated VRP [45], the job-shop scheduling problem [3] and the nurse rostering problem [6]. Basically, tabu search algorithm starts from an initial solution and iteratively proceeds from the *incumbent* solution to its best allowable neighbor. The neighborhood of a solution is a set of solutions that can be reached from that solution by a certain operation. Each type of operation corresponds to a neighborhood and the procedure of identifying the best allowable neighbor in the neighborhood is called an *operator*. The transition from the incumbent solution to one of its neighbors is called a *move*.

Our tabu search algorithm employs several operations adapted from classical operations for the VRPTW, namely *2-opt, Or-opt, 2-opt*, Relocate* and *Exchange* [5], and an *ejection pool* [28,32]. The most noteworthy characteristic that distinguishes these adapted operations from their classical counterparts is the procedure of checking the feasibility of the modified solution. For example, after performing an operation on a VRPTW solution, we can check the feasibility of the resultant solution in $O(1)$ time (it is assumed that for each vertex the latest arrival time that does not lead to the violation of the time windows of all successive vertices has been calculated in a preprocessing step). However, for a modified MPISP solution, we may require up to $O(n \log n)$ time to check its feasibility due to the re-computation of the shortest transit times associated with the affected vertices, which will be elaborated in Section 5.4.1.

The pseudocode of our tabu search algorithm is presented in Algorithm 2, which is an iterative approach that follows a four-phase framework: initialization, local search with tabu moves, ejection pool algorithm and perturbation. At the beginning of the algorithm, we generate an initial solution S_0 using the function *best_init* (see Section 5.3) and then initialize both the best solution S_{best} and the current solution S by S_0. In each iteration, we first invoke the local search procedure with tabu moves (function *local_search*, see Section 5.4) and set S' to be the best solution found by this procedure. Subsequently, we try to improve on S' by an ejection pool algorithm (function *EPA*, see Section 5.5) and then update S_{best} if possible. Finally, the search process is diversified by perturbing the best solution found in this iteration. The above process is repeated until the perturbation procedure (function *perturb*, see Section 5.6) is consecutively performed *maxPerturbation* times without improving on S_{best}.

Algorithm 2. Framework of the tabu search algorithm.

$S_0 \leftarrow best\_init()$;
$S_{best} \leftarrow S_0$ and $S \leftarrow S_0$;
$i \leftarrow 0$;
while $i \leq$ *maxPerturbation* **do**
  $S' \leftarrow$ the best solution found by *local_search*(S);
  $S' \leftarrow EPA(S')$
  if S' is better than S_{best} **Then**
    $S_{best} \leftarrow S'$ and $i \leftarrow 0$;
  else
    $i \leftarrow i + 1$;
  end if
  $S \leftarrow perturb\ (S')$;
end while
return S_{best}.

5.1. Solution representation

In Section 3, we have used the sequences of visited vertices to represent the problem solution (see Fig. 3). However, in our tabu search algorithm, we represent the route of each inspector by a sequence of served suppliers. For example, Fig. 7 shows a solution that is exactly the same as the one in Fig. 3. The routes r_1, r_2, r_3 and r_4 include the served suppliers and the ejection pool U contains the leftover suppliers. All waypoints are not displayed in this solution representation and there may exist waypoints and/or breaks between two consecutively served suppliers.

5.2. Fitness function

The tabu search algorithm ranks solutions using a fitness function. It is natural to define the fitness value of a solution S as the total completed workload, denoted by $P(S)$. However, many distinct solutions have the same value of $P(S)$. To further differentiate solutions, we incorporate into the fitness function another two measures denoted by $D(S)$ and $F(S)$, respectively, which is inspired by Lim and Zhang [28]. As a result, the fitness function consists of three lexicographically ordered components, namely $P(S)$, $D(S)$ and $F(S)$.

The second component $D(S)$ employs a function $mv(u, S)$ that estimates the difficulty of inserting supplier $u \in U$ into the routes of solution S. Denoting any route in S by $r = (v_0, v_1, \ldots, v_{|r|}, v_{|r|+1})$, where $|r|$ is the number of served suppliers in route r and $v_0 = v_{|r|+1} = 0$, the definition of $mv(u, S)$ is given by

$$mv(u,S) = \min_{r \in S} mv(u,r)$$

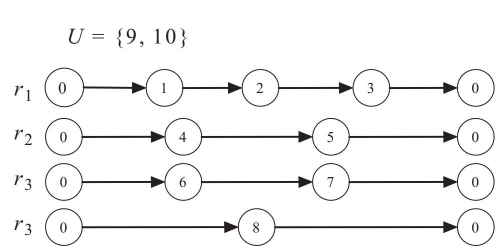

Fig. 7. An example of the solution representation used in our tabu search algorithm.

where

$$mv(u,r) = \max\{\eta \times mv_l(u,r), \quad mv_t(u,r)\} \tag{3}$$

$$mv_l(u,r) = \begin{cases} 0 & \text{if } wl(r)+d_u \le Q; \\ (l_0 - e_0) \times \dfrac{(wl(r)+d_u - Q)}{wl(r)+d_u} & \text{otherwise.} \end{cases} \tag{4}$$

$$wl(r) = \sum_{v \in r} d_v$$

$$mv_t(u,r) = \min_{0 \le i \le |r|} c(u, v_i, r)$$

$$c(u, v_i, r) = \max\{ea'_u - l_u, 0\} + \max\{e_u - la'_u, 0\} + \max\left\{ea'_{v_{i+1}} - la_{v_{i+1}}, 0\right\} \tag{5}$$

The cost of inserting supplier u into route r, denoted by $mv(u,r)$, is computed based on the extent of violating the workload limit and the time-window constraint. The amount of workload is translated into time unit by expression (4), where $wl(r)$ is the cumulative workload in route r. If the inspector has enough capability to serve supplier u, namely $wl(r)+d_u \le Q$, then no workload penalty is incurred. Otherwise, the penalty, denoted by $mv_l(u,r)$, equals the length of the depot time window multiplied by the workload violation percentage.

The penalty for time-window violation, denoted by $mv_t(u,r)$, considers all possible insertions. For each $v_i \in r$, we can easily find its earliest arrival time ea_{v_i} when $(0, v_1, \ldots, v_i)$ is feasible, and its latest arrival time la_{v_i} that does not affect the feasibility of $(v_{i+1}, v_{i+2}, \ldots, v_{|r|}, 0)$. Inserting u into r at the position immediately after v_i creates a new route r', which may be infeasible. Under the condition that $(0, v_1, \ldots, v_i)$ is feasible, we can find the earliest arrival times at u and v_{i+1} in r', denoted by ea'_u and $ea'_{v_{i+1}}$, respectively. The partial route $(0, v_1, \ldots, v_i, u, v_{i+1})$ may be infeasible, i.e., $ea'_u > l_u$ and/or $ea'_{v_{i+1}} > l_{v_{i+1}}$. Furthermore, we can also find the latest arrival time at u, denoted by la'_u, that makes $(v_{i+1}, v_{i+2}, \ldots, v_{|r|}, 0)$ feasible. The penalty for time-window violation incurred by inserting u between v_i and v_{i+1} is calculated by summing up the violations of l_u, e_u and $la_{v_{i+1}}$ (see expression (5)). As shown in expression (3), the cost of inserting u into r takes into account both $mv_l(u,r)$ and $mv_t(u,r)$ whose relative weights are controlled by a parameter η. After sorting the $mv(u,S)$ values of all unserved suppliers in ascending order, we can obtain a sequence $(mv_1, \ldots, mv_{|U|})$, where $|U|$ is the cardinality of U. The value of $D(S)$ is calculated by $\sum_{i=1}^{|U|} mv_i/i$. We believe that the solution S with smaller $D(S)$ has more chance to be improved by including the unserved suppliers.

The third component $F(S)$ is the summation of the maximal free times of all routes in S. The maximal free time of route r is defined as $mft(r) = \max_{0 \le i \le |r|+1}\{la_i - ea_i\}$ and accordingly $F(S) = \sum_{r \in S} mft(r)$.

5.3. Initialization

We obtain an initial feasible solution for the tabu search algorithm using Algorithm 3. This algorithm first generates N_{init} feasible solutions using the function *init* (see Algorithm 4) and then chooses the best one as the initial solution. In each iteration of *init*, we begin with computing the shortest transit time st_i^r from the tail of each route r to each unserved supplier v_i. If v_i cannot be feasibly appended at the tail of r, we set $st_i^r = +\infty$. Next, we calculate the ratio of st_i^r to d_i and set ρ_i to be the minimal ratio of v_i over all routes (see Algorithm 4, line 11). If the value of ρ_i is positive infinity, i.e., v_i cannot be appended at the tail of any route, we remove v_i from U. Finally, we sort all suppliers in U by increasing value of ρ_i and relocate the kth supplier v_s from U to the tail of the route r who has $\rho_s = st_s^r/d_s$. The value of k is a random number generated by $k = \lfloor \text{random}(0,1)^{\alpha_1} \times |U| \rfloor$, where the controlling parameter $\alpha_1 > 1$. This process is repeated until U becomes empty.

Algorithm 3. Function *best_init*.

```
Initialize S_0 = ∅;
while i ≤ N_init do
    S = init();
    if S is better than S_0 then
        S_0 ← S;
    end if
    i = i + 1;
end while
return S_0.
```

Algorithm 4. Function *init*.

```
1:  INPUT: the set U of unserved suppliers and m empty
    routes;
2:  while U is not empty do
3:    for each v_i in U do
4:      for r = 1, ..., m do
5:        if v_i can be feasibly appended to the tail of r then
6:          st_i^r ← the shortest transit time from the last
    supplier of r to v_i;
7:        else
8:          st_i^r ← +∞;
9:        end if
10:     end for
11:     ρ_i = min_{r=1}^{m} {st_i^r / d_i};
12:     if ρ_i = +∞ then
13:       Remove v_i from U;
14:     end if
15:   end for
16:   Sort all suppliers in U by increasing value of ρ_i;
17:   v_s ← the kth supplier in the sorted supplier list, where
    k = ⌊random(0,1)^{α_1} × |U|⌋ and α_1 > 1;
18:   Append v_s at the tail of r with st_s^r / d_s = ρ_s;
19:   Remove v_s from U;
20: end while
```

5.4. Local search with tabu moves

The pseudocode of the local search procedure with tabu moves is provided in Algorithm 5. The following context of this subsection presents all main components of this procedure, including neighborhood structure, tabu list, aspiration and termination criteria.

Algorithm 5. The local search procedure with tabu moves (*local_search*).

```
INPUT: the initial solution S;
The current best solution S' ← S and Iter ← 0;
while Iter ≤ maxLocalIter do
  Apply the 2-opt, Or-opt, 2-opt*, relocate and exchange
  operators on S ;
  S ← the best allowable solution found by the above
  operators;
  if S is better than S' then
    S' ← S and Iter ← 0;
  else
    Iter ← Iter + 1;
  end if
  Update the tabu list;
```

end while
return S'.

5.4.1. Neighborhood structure

The neighborhood structure is one of the most important components that determine the size of the search space and the quality of the final solution. Our tabu search algorithm employs five neighborhood operations adapted from classical operations for the VRPTW [5], namely 2-*opt*, *Or-opt*, 2-*opt**, *Relocate* and *Exchange*. We treat the ejection pool as a dummy route that includes all unserved suppliers. Compared with their classical counterparts, these adapted operations require more computational efforts to check the feasibility of the resultant solution, and to update the earliest and latest arrival times at the affected suppliers.

Fig. 8 illustrates the 2-opt and Or-opt operations. Assume that we are given the earliest and latest arrival times (ea_i and la_i) at each supplier i in route r. The earliest departure time (ed_i) of each supplier can be easily derived by

$$ed_i = \begin{cases} ea_i + s_i & \text{if } \mathrm{ceil}(ea_i) - ea_i \geq s_i; \\ \mathrm{ceil}(ea_i) + s_i & \text{otherwise.} \end{cases}$$

The 2-opt operation replaces edges $(i, i+1)$ and $(j, j+1)$ with edges (i, j) and $(i+1, j+1)$, and then reverses the directions of all edges between $i+1$ and j. The resultant route r' shown in Fig. 8(b) must be feasible if its subroute $(j, j-1, \ldots, i+1, j+1, \ldots, 0)$ is feasible. To check the feasibility of r', we need to re-calculate the earliest arrival time (ea'_k) at each supplier k in subroute $(j, j-1, \ldots, i+1, j+1)$. If ea'_k is less than e_k or within $[e_k, l_k]$ for each supplier, this subroute must be feasible. If ea'_{j+1} in r' is less than or equal to la_{j+1} in r, subroute $(j+1, \ldots, 0)$ must be feasible. All ea'_k can be obtained in $O(n_s \log n)$ time using the procedure described in Section 4, where n_s is the number of suppliers in subroute $(j, j-1, \ldots, i+1, j+1)$. Therefore, it requires $O(n_s \log n)$ time to check the feasibility of route r'. By contrast, when dealing with the VRPTW, a 2-opt operation only requires $O(n_s)$ time to accomplish the feasibility check. In addition, updating the values of ea'_i and la'_i for all suppliers in r' requires $O(|r| \log n)$ time and computing the fitness of the resultant solution requires $O(|r||U| \log n)$ time.

The Or-opt operation replaces three edges $(i-1)$, $(i+1, i+2)$ and $(j, j+1)$ with edges $(i-1, i+2)$, (j, i) and $(i+1, j+1)$; the resultant route is illustrated in Fig. 8(c). After an Or-opt operation, we can also derive the time complexity for checking the feasibility of the resultant route, updating the earliest and latest arrival times at each supplier, and computing the fitness of the resultant solution in a manner similar to that used for the 2-opt operation.

Fig. 9 illustrates the 2-opt* operation which exchanges the latter subroutes of r_1 and r_2 by replacing edges $(i, i+1)$ and $(j, j+1)$ with edges $(i, j+1)$ and $(j, i+1)$. The feasibility of the resultant routes can be checked by simply comparing ea'_{j+1} (resp. ea'_{i+1}) with la_{j+1} (resp. la_{i+1}) in $O(\log n)$ time. After this operation, we need $O((|r_1| + |r_2|) \log n)$ time to update ea'_i and la'_i in r'_1 and r'_2 and $O((|r_1| + |r_2|)|U| \log n)$ time to update the fitness of the resultant solution.

The relocate operation can either relocate supplier j in route r_1 to another position in the same route or to route r_2, which is illustrated in Fig. 10. In the former case, the feasibility of the resultant route shown in Fig. 10(b) can be checked by calculating ea'_i for all suppliers in subroute $(j, i+1, \ldots, j-1, j+1)$. In the latter case, we only need to check the feasibility of r'_2 shown in Fig. 10(c), which can be done in $O(\log n)$ time. The relocation operation can also relocate a supplier in the ejection pool to a certain route or vice versa.

The exchange operation exchanges positions of two suppliers. Fig. 11(b) shows the resultant route after exchanging the positions of two suppliers in the same route. The feasibility of this route can be checked by calculating ea'_i for all suppliers in subroute $(j, i+1, \ldots, j-1, i, j+1)$. The resultant routes created by exchanging two suppliers from two different routes are shown in Fig. 11(c). The feasibility check can be done in $O(\log n)$ time. This operation can also exchange a supplier in some route with a supplier in the ejection pool.

5.4.2. Tabu list, aspiration and termination

Tabu search algorithm employs one or more tabu lists to prevent the search process from being trapped in local optima. In our implementation, the tabu list stores edges that have been created within the previous ξ iterations. A move is considered as tabu if it attempts to remove the edges in the tabu list. The tabu restriction can be overridden if the aspiration criterion is satisfied. Specifically, we allow the tabu moves to be performed if the solutions they result in are better than the current best solution S'. The solutions that are created by non-tabu moves or by aspiration are called *allowable* neighbors. All allowable moves are stored in a

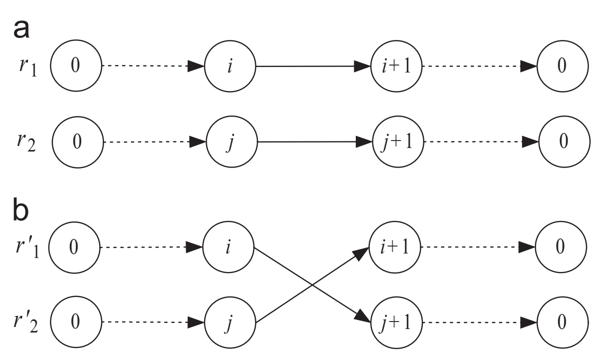

Fig. 9. (a) The original routes r_1 and r_2. (b) The resultant routes r'_1 and r'_2 after a 2-opt* operation.

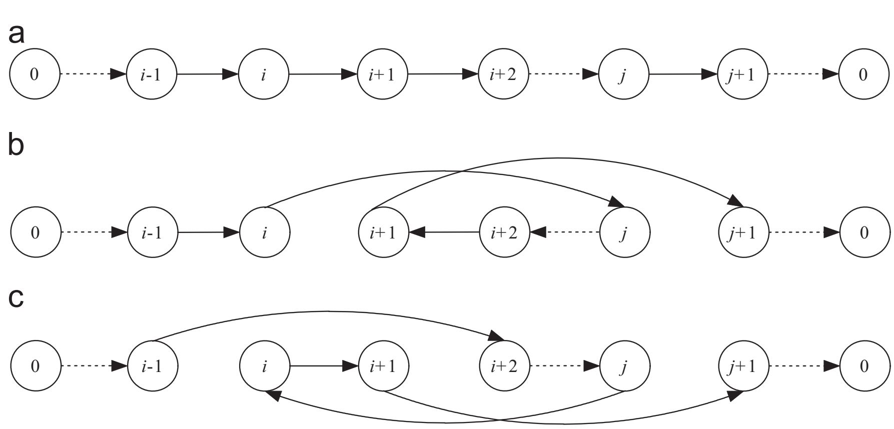

Fig. 8. (a) The original route r. (b) The resultant route r' after a 2-opt operation. (c) The resultant route r' after an Or-opt operation.

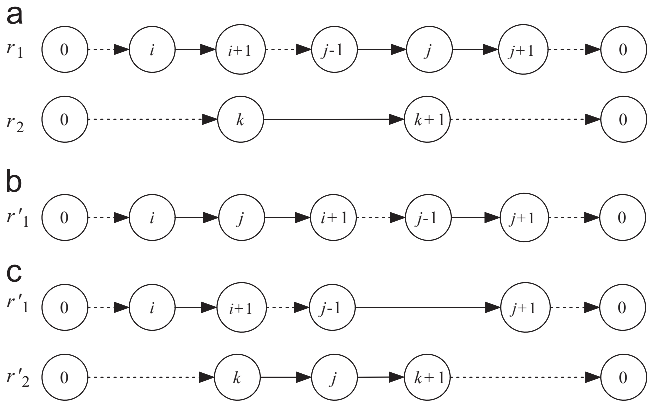

Fig. 10. (a) The original routes r_1 and r_2. (b) The resultant routes r'_1 after relocating supplier j between suppliers i and $i+1$. (c) The resultant routes r'_1 and r'_2 after relocating supplier j between suppliers k and $k+1$.

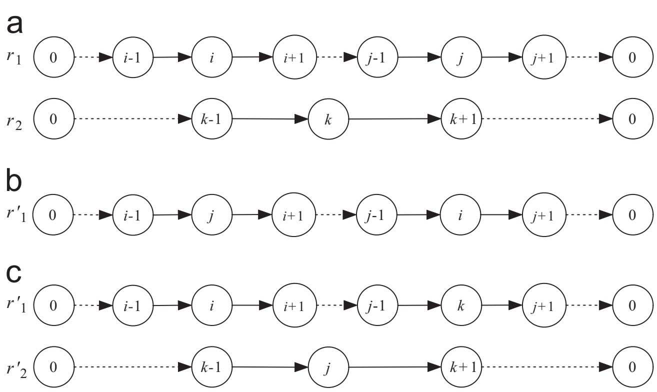

Fig. 11. (a) The original routes r_1 and r_2. (b) The resultant routes r'_1 after exchanging suppliers i and j. (c) The resultants routes r'_1 and r'_2 after exchanging suppliers j and k.

candidate list and sorted according to the fitness values of their resultant solutions. The best candidate is performed to generate the next incumbent solution. We terminate the local search procedure when *maxLocalIter* consecutive iterations are unable to improve on S'.

5.5. Ejection pool

Ejection pool has been previously used in the algorithms for reducing the number of routes for some routing problems (see for example [28,32,9]). Our ejection pool algorithm (EPA) is presented in Algorithm 6. The initial solution S' of this algorithm is the best solution found by function *local_search*. Since S' is a local optimum, no supplier in the ejection pool can be feasibly inserted into S'. The EPA generates a candidate solution based on S' for each of the unserved suppliers by an insertion-ejection procedure. If the best candidate solution is superior to S', then S' is updated.

Algorithm 6. The ejection pool algorithm (*EPA*).

INPUT: the initial feasible solution S';
for each $u \in U$ **do**
  Evaluate all insertion positions using the function $c(i, u, j)$;
  $S_u \leftarrow$ the resultant solution after inserting u into S' at the best position;
  Eject suppliers one by one using the function $c(i)$ until S_u becomes feasible;
  Improve S_u by function *local_search* with $\xi = 0$;
end for
if the best candidate solution S_u is better than S' **then**
  $S' \leftarrow S_u$;
end if
return S'.

For each $u \in U$, we evaluate its insertion positions using function $c(i, u, j)$, namely the cost of inserting u between two consecutively served suppliers i and j, which is defined as

$$c(i, u, j) = \beta_1 \times d_u - \beta_2 \times \left(\max\{0, ea'_u - l_u\} + \max\left\{0, ea'_j - l_j\right\}\right),$$

where β_1 and β_2 are controlling parameters, and ea'_u and ea'_j are the earliest arrival times at suppliers u and j after inserting u between i and j. The position with the smallest value of $c(i, u, j)$ is selected for insertion.

The target route r_t becomes infeasible after the insertion. Thus, some of its suppliers (except the newly inserted one) need to be ejected one by one until its feasibility is restored. The supplier i to be ejected is determined based on the value of $c(i)$, which is defined as

$$c(i) = \beta_3 \times d_i + \beta_4 \times \max\{wl(r'_t) - Q, 0\} + \beta_5 \times \text{violation}_{tw}(r'_t),$$

where β_3, β_4 and β_5 are controlling parameters, r'_t is the resultant route after removing i from r_t, and $\text{violation}_{tw}(r'_t)$ is the total time-window violation of all suppliers in route r'_t, defined as

$$\text{violation}_{tw}(r'_t) = \sum_{j \in r'_t} \max\left\{0, ea'_j - l_j\right\}.$$

The supplier i with minimal $c(i)$ is ejected from the route r_t.

After performing the insertion-ejection procedure on each $u \in U$, we obtain $|U|$ candidate solutions, which are further improved by *local_search* without tabu moves, namely $\xi = 0$. The solution S' is updated by the best candidate solution if possible.

5.6. Perturbation

The perturbation procedure is a diversification scheme that helps the search process escape from local optima. Our perturbation procedure (function *perturb*) randomly removes some suppliers from the solution S' following the rule that the suppliers with smaller workloads have more chances to be removed. Given a solution S', we sort the served suppliers in non-decreasing order of workloads, generating a supplier list $(v_1, v_2, \ldots, v_{n'})$. The probability of removing the kth supplier is determined by

$$p_{\min} + (p_{\max} - p_{\min}) \times \frac{k}{n'}, \tag{6}$$

where $p_{\min}$ and $p_{\max}$ are controlling parameters satisfying $0 \le p_{\min} \le p_{\max} \le 1$. It implies that the supplier with larger workload has smaller probability to be kicked out.

The tabu search algorithm performs at least *maxPerturbation* iterations (see line 4–13, Algorithm 2). We store the best solution S' found within each iteration in a solution list. The solutions with equal values of $P(S)$, $D(S)$ and $F(S)$ are regarded as the same. The number of times that the current solution S' appears in the solution list is represented by N_{rep}. When N_{rep} grows large, we expect that the perturbed solution deviates far from the current solution S'. To this end, we replace p_{min} and p_{max} in expression (6) by

$$p_{\min} \leftarrow p_{\min} + p_{\Delta} \times \min\{N_{rep}, N_{\max}\},$$

$$p_{\max} \leftarrow p_{\max} + p_{\Delta} \times \min\{N_{rep}, N_{\max}\},$$

where p_{Δ} is a controlling parameter, and the parameter $N_{\max}$ is used to set an upper bound for the probability. The introduction of $N_{\max}$ can help avoid the overly large probability, which would cause the process to degenerate into an ineffective multi-start method.

6. Upper bound

The solutions generated by our tabu search algorithm are lower bounds to the MPISP. In this section, we construct a constrained knapsack model to produce an upper bound for the MPISP; this model is motivated by Lau et al. [26].

The time windows of all suppliers can be adjusted based on the following straightforward observations. For a supplier i, if the opening time of its time window lies within period p, i.e., $a_p \le e_i \le b_p$, and it is impossible to complete its service during that period, i.e., $b_p - e_i < s_i$, then the real earliest service starting time for supplier i should be the opening time a_{p+1} of the next period. Consequently, in this situation e_i can be updated by $e_i = a_{p+1}$. Analogously, if $a_p \le l_i \le b_p$ and $b_p - l_i < s_i$, we can update l_i by $l_i = b_p - s_i$. Let $\lambda_i = \text{ceil}(l_i + s_i) + \hat{t}_{i,0}$ denote the time of returning to the depot immediately after severing supplier i with the service starting time l_i. We construct a supplier set $V_G = \{g_1, g_2, \ldots, g_m\} \subseteq V_C$ that contains m distinct suppliers satisfying $\lambda_i \ge \lambda_j$ for all $i \in V_G$ and $j \in V_C \backslash V_G$.

Obviously, if $e_i > l_i$, supplier i cannot be served by any inspector and vertex i can only be used as a waypoint. We define $f_1(i) = 0$ if $e_i > l_i$ and $f_1(i) = 1$ otherwise. Suppose e_i and l_i lie within periods p_1 and p_2, respectively. We define $f_2(i,p) = 1$ if $p_1 \le p \le p_2$ and $f_2(i,p) = 0$ otherwise, where $f_2(i,p) = 1$ indicates that supplier i could probably be served during period p. Furthermore, we use $f_3(i,j) = 1$ to indicate that it is possible for an inspector to serve both suppliers i and j when supplier time windows, workload capacity and working periods are not considered. Thus, the definition of $f_3(i,j)$ is

$$f_3(i,j) = \begin{cases} 1 & \text{if } e_i + s_i + t_{i,j} \le l_j \text{ or } e_j + s_j + t_{j,i} \le l_i; \\ 0 & \text{otherwise.} \end{cases}$$

We denote by r_i^p the time required to directly travel from supplier i to its nearest neighbor who could probably be served by the same inspector during period p. Define set $V_i^p = \{j \in V \mid j \ne i, f_2(j,p) = 1, f_3(i,j) = 1\}$. We set $r_i^p = +\infty$ if V_i^p is empty, and otherwise $r_i^p = \min_{j \in V_i}\{t_{i,j}\}$. Let $x_{i,k,p}$ be a binary decision variable that equals 1 if supplier i is served by inspector k during period p, and 0 otherwise. The optimal solution value of the following integer programming model gives an upper bound to the MPISP:

$$\max \sum_{i \in V_C} \sum_{k \in K} \sum_{p \in P} d_i x_{i,k,p} \tag{7}$$

$$\text{s.t.} \quad \sum_{k \in K} \sum_{p \in P} x_{i,k,p} \le f_1(i), \quad \forall i \in V_C \tag{8}$$

$$\sum_{k \in K} x_{i,k,p} \le f_2(i,p), \quad \forall i \in V_C, \ p \in P \tag{9}$$

$$\sum_{p \in P} (x_{i,k,p} + x_{j,k,p}) \le 1 + f_3(i,j), \quad \forall i, \ j \in V_C, \ i \ne j, \ k \in K \tag{10}$$

$$\sum_{i \in V_C} \sum_{p \in P} d_i x_{i,k,p} \le Q, \quad \forall k \in K \tag{11}$$

$$\sum_{i \in V_C} \sum_{p \in P} \left(s_i + \min\{r_i^p, r_i^{p+1}, \ldots, r_i^w\} \right) x_{i,k,p} + r_0^1 \le \lambda_{g_k}, \quad \forall k \in K \tag{12}$$

$$\sum_{i \in V_C} x_{i,k,p}(s_i + r_i^p) - \max_{i \in V_C} r_i^p \le T, \quad \forall k \in K, \ p \in P \tag{13}$$

$$x_{i,k,p} \in \{0,1\}, \forall i \in V_C, \ k \in K, \ p \in P \tag{14}$$

The objective (7) is to maximize the total completed workload. Constraints (8) state that each supplier must be assigned to at most one inspector and be served in at most one period. Constraints (9) guarantee that if supplier i cannot be served by any inspector during period p, all relative variables must be set to zero. Constraints (10) ensure that if $f_3(i,j) = 0$, suppliers i and j cannot be served by the same inspector. As the inspector workload capacity cannot be exceeded, Constraints (11) apply. When supplier i is served by some inspector, the time it consumes must be at least the sum of s_i and the traveling time to its nearest neighbor. Any feasible solution to the MPISP must have m routes, each of which has an earliest time of returning to the depot. It is easy to observe that the kth largest return time must be less than or equal to the kth largest λ_{g_k}. Obviously, the sum of $s_i + \min\{r_i^p, r_i^{p+1}, \ldots, r_i^w\}$ associated with all suppliers covered by a route should be less than or equal to the length of that route, which is capped by λ_{g_k}. Therefore, Constraints (12) hold. After completing the service of supplier i, the inspector may stay at vertex i until the start of the next period. The difference between the total $s_i + r_i^p$ of all suppliers served in each period and the largest r_i^p must be less than or equal to the period length, which is ensured by Constraints (13).

Any feasible solution to the MPISP must be also feasible to the constrained knapsack model (7)–(14). The knapsack problem and many of its variants have been well-studied and can be efficiently handled by several commercial mathematical programming solvers.

7. Computational experiments

Our tabu search (TS) algorithm was coded in C++ and compiled using the gcc 4.6.1 compiler, and was tested on a Dell server with an Intel Xeon E5430 2.66 GHz CPU, 8 GB RAM and running Linux-CentOS-5.0 64-bit operating system. The algorithm was configured with determined parameter settings: $\eta = 1.0$, $N_{init} = 100$, $\alpha_1 = 5$, maxPerturbation=4, maxLocalIter=200, $\xi = 100$, $\beta_1 = 0.6$, $\beta_2 = \beta_3 = \beta_4 = 0.4$, $\beta_5 = 0.2$, $p_{min} = 0.05$, $p_{\max} = 0.30$, $p_\Delta = 0.1$ and $N_{\max} = 5$. The MPISP reduces to the traditional TOPTW when w=1 and $Q = +\infty$. In our experiments, we first applied the TS algorithm to the TOPTW instances and compared the results with the recent results reported in Vansteenwegen et al. [41], Lin and Yu [29], Labadie et al. [25], Hu and Lim [22]. Next, we conducted experiments on the MPISP instances generated from the Solomon's VRPTW instances [35]. Since our TS algorithm is not deterministic, we solved each instance 10 times. Finally, we achieved an upper bound for each MPISP instance by solving the model (7)–(14) using ILOG CPLEX 12.1 with default settings. Computation times reported are in CPU seconds on this server. All instances and detailed results can be found online at: http://www.computational-logistics.org/orlib/mpisp/.

7.1. Test instances

We considered the TOPTW instances used in Vansteenwegen et al. [41] and Hu and Lim [22], which can be accessed at http://www.mech.kuleuven.be/en/cib/op. Hu and Lim [22] classified these instances into two categories, namely "INST-M" and "OPT"; the instances in category

Table 1
Programming languages and experimental environments.

Algorithm	Language	Experimental environment
ILS	N/A	Intel Core 2 2.5 GHz CPU, 3.45 GB RAM
SSA	C	Intel Core 2 2.5 GHz CPU
GVNS	Embarcadero Delphi 2010	Intel Pentium (R) IV 3 GHz CPU
I3CH	Java	Intel Xeon E5430 CPU clocked at 2.66 GHz, 8 GB RAM
TS	C++	Intel Xeon E5430 CPU clocked at 2.66 GHz, 8 GB RAM

INST-M have unknown optimal solution values while the optimal solution values of the instances in category OPT are given.

The TOPTW instances in category INST-M were constructed by Montemanni and Gambardella [31] based on the OPTW instances by considering the number of vehicles taken from set {1, 2, 3, 4}. These OPTW instances were designed by Righini and Salani [33] using 56 Solomon's VRPTW instances [35] and 20 Cordeau's multi-depot VRP (MDVRP) instances [12]. The Solomon's VRPTW instances, each containing 100 customers, are divided into six groups, namely C1 (c101–c109), C2 (c201–c208), R1 (r101–r112), R2 (r201–r211), RC1 (rc101–rc108) and RC2 (rc201–rc208). The numbers of customers in the MDVRP instances (pr01–pr20) range from 48 to 288. The TOPTW

Table 2
The summarized results for the TOPTW instances.

Instance group	ILS (1 run)		SSA (1 run)		GVNS (5 runs)		I3CH (1 run)		TS (10 runs)	
	Ratio (%)	Time (s)	Ratio (%)	Time (s)	Ratio (%)	Time (s)	Ratio (%)	Time (s)	Ratio (%)	Time (s)
$m=1$										
C1	0.9889	0.3	**1.0000**	21.1	0.9944	166.5	**1.0000**	25.2	**1.0000**	7.2
C2	0.9772	1.7	0.9987	37.5	0.9945	192.4	0.9960	84.4	**1.0000**	68.2
R1	0.9815	0.2	0.9995	23.3	0.9834	29.4	0.9950	28.6	**1.0000**	4.0
R2	0.9731	1.7	0.9891	45.8	0.9776	33.8	0.9916	176.2	**0.9970**	110.7
RC1	0.9708	0.2	**1.0000**	22.2	0.9812	9.8	0.9834	25.5	0.9958	2.9
RC2	0.9699	1.6	0.9947	50.3	0.9789	16.0	0.9774	119.3	**0.9974**	157.2
pr01–pr20	0.9318	1.9	0.9801	137.3	0.9850	18.3	0.9768	119.6	**0.9938**	536.8
$m=2$										
C1	0.9906	1.1	**1.0000**	26.4	0.9953	139.5	**1.0000**	87.0	**1.0000**	21.8
C2	0.9746	3.5	0.9882	53.7	0.9975	33.8	0.9933	401.2	**1.0000**	63.8
R1	0.9777	0.9	**0.9991**	36.6	0.9895	60.3	0.9955	63.0	0.9970	14.1
R2	0.9702	2.3	0.9917	91.4	0.9909	14.7	0.9955	526.8	**0.9992**	91.5
RC1	0.9771	0.7	**1.0000**	40.5	0.9940	20.3	0.9928	58.9	0.9952	11.6
RC2	0.9593	2.2	0.9883	80.1	0.9839	12.8	0.9945	439.7	**0.9995**	243.2
pr01–pr20	0.9311	5.0	0.9699	187.8	0.9915	60.8	0.9825	275.8	**0.9935**	1,809.5
$m=3$										
C1	0.9745	1.5	0.9967	35.3	0.9955	165.0	0.9989	190.2	**1.0000**	31.1
C2	0.9807	2.2	0.9876	59.7	0.9993	7.7	**1.0000**	12.3	**1.0000**	6.8
R1	0.9832	1.7	0.9972	56.1	0.9889	73.9	**0.9990**	118.3	0.9983	16.6
R2	0.9970	1.4	0.9992	41.9	0.9989	7.0	0.9999	90.8	**1.0000**	14.8
RC1	0.9695	1.1	0.9946	42.8	0.9918	33.7	0.9982	101.0	0.9944	14.3
RC2	0.9856	1.7	0.9973	59.0	0.9968	7.4	0.9996	164.1	**0.9998**	52.1
pr01–pr20	0.9213	9.5	0.9695	224.4	**0.9933**	118.3	0.9928	460.5	0.9895	1,680.5
$m=4$										
C1	0.9689	2.6	0.9945	49.5	0.9897	133.2	0.9990	261.8	**1.0000**	38.6
C2	**1.0000**	1.0	**1.0000**	41.8	**1.0000**	0.6	**1.0000**	0.1	**1.0000**	6.5
R1	0.9670	2.6	0.9929	58.4	0.9880	84.7	**0.9986**	184.3	0.9956	21.0
R2	**1.0000**	0.9	**1.0000**	39.7	**1.0000**	0.3	**1.0000**	0.2	**1.0000**	7.7
RC1	0.9693	2.0	0.9974	68.1	0.9916	36.9	**0.9988**	152.4	0.9975	17.7
RC2	**1.0000**	1.2	**1.0000**	40.2	**1.0000**	0.9	**1.0000**	0.2	**1.0000**	15.5
pr01–pr20	0.9249	13.9	0.9719	269.8	0.9872	180.0	**0.9988**	647.6	0.9940	1,542.4
$m=$opt										
C1	0.9859	3.0	0.9896	77.7	N/A	7.8	**1.0000**	47.6	**1.0000**	61.8
C2	**1.0000**	1.1	**1.0000**	41.9	N/A	0.5	**1.0000**	0.6	**1.0000**	113.3
R1	0.9807	3.0	0.9958	104.7	N/A	39.5	0.9993	877.7	**0.9999**	58.2
R2	0.9938	1.7	0.9984	58.3	N/A	5.5	0.9993	173.4	**1.0000**	265.2
RC1	0.9794	3.8	0.9965	84.6	N/A	39.5	**1.0000**	57.4	0.9994	54.3
RC2	0.9953	1.7	0.9993	41.4	N/A	2.8	0.9996	190.2	**1.0000**	194.9
pr01–pr10	0.9768	30.4	0.9896	566.0	N/A	51.3	0.9922	326.7	**0.9933**	420.5
Overall average	0.9751	3.2	0.9933	83.3	0.9914	51.6	0.9957	185.4	**0.9980**	222.2

$m=$opt implies that these instances belong to category OPT.

Table 3
Average performance of four combinations on the 50 TOPTW instances generated from pr01 to pr10.

m	EP + PER		LS + EP		LS + PER		LS + EP + PER	
	Avg. ratio (%)	Avg. time	Avg. ratio (%)	Avg. time	Avg. ratio (%)	Avg. time	Avg. ratio (%)	Avg. time
1	0.9742	515.4	0.9755	184.9	0.9796	63.2	0.9895	347.3
2	0.9771	1985.8	0.9743	734.7	0.9689	123.6	0.9835	1046.7
3	0.9824	2883.0	0.9799	841.5	0.9688	148.1	0.9832	1122.6
4	0.9860	3956.5	0.9807	892.3	0.9697	179.9	0.9800	978.9
opt	0.9977	2169.2	0.9954	196.1	0.9979	401.2	0.9987	420.5
Overall average	0.9835	2302.0	0.9811	569.9	0.9770	183.2	0.9870	783.2

$m=$opt implies that these instances belong to category OPT.

instances were obtained from the VRPTW or MDVRP instances by the following two steps: (1) set the profit collected at each customer to be the demand of this customer, and (2) remove the vehicle capacity restriction. The instances in category OPT are the same as the instances in category INST-M except for the number of vehicles available. Vansteenwegen et al. [41] designed the instances in category OPT by setting the number of vehicles in each aforementioned TOPTW instance except for pr11–pr20 to the number of vehicles appearing in the solution of the corresponding VRPTW instance. This implies that with such number of vehicles, all customers can be visited and the optimal objective value must be equal to the total profits of all customers. Therefore, we have 76 $\times 4 = 304$ instances in category INST-M and 66 instances in category OPT, for a total of 370 TOPTW instances.

We generated 12 MPISP instances from each Solomon's VRPTW instance by taking the values of w and m from {1, 3, 5} and {7, 9, 11, 13}, respectively, for a total of 72 instance groups (each instance group is identified by the name of Solomon's instance group, w and m) and 672

Table 4
Computational results for the MPISP instances with $m = 7$.

Instance	$w=1$				$w=3$				$w=5$			
	UB	Max. workload	Ave. workload	Ave. time	UB	Max. workload	Ave. workload	Ave. time	UB	Max. workload	Ave. workload	Ave. time
c101	1400	1400	1400.0	4.9	1400	1400	1399.0	25.4	1350	1240	1239.0	26.0
c102	1400	1400	1400.0	7.1	1400	1400	1400.0	7.7	1400	1400	1400.0	11.4
c103	1400	1400	1400.0	11.0	1400	1400	1400.0	12.0	1400	1400	1400.0	22.5
c104	1400	1400	1400.0	20.5	1400	1400	1400.0	23.5	1400	1400	1400.0	33.4
c105	1400	1400	1400.0	5.8	1400	1400	1400.0	7.0	1400	1400	1400.0	12.5
c106	1400	1400	1400.0	6.6	1400	1400	1400.0	6.6	1400	1370	1368.0	53.3
c107	1400	1400	1400.0	7.6	1400	1400	1400.0	7.8	1400	1400	1400.0	12.8
c108	1400	1400	1400.0	8.9	1400	1400	1400.0	10.7	1400	1400	1400.0	14.6
c109	1400	1400	1400.0	11.4	1400	1400	1400.0	12.5	1400	1400	1400.0	21.1
c201	1400	1400	1400.0	7.0	1400	1400	1400.0	7.4	1400	1400	1400.0	7.8
c202	1400	1400	1400.0	13.3	1400	1400	1400.0	12.2	1400	1400	1400.0	14.5
c203	1400	1400	1400.0	19.9	1400	1400	1400.0	23.1	1400	1400	1400.0	32.0
c204	1400	1400	1400.0	34.9	1400	1400	1400.0	35.8	1400	1400	1400.0	47.6
c205	1400	1400	1400.0	9.1	1400	1400	1400.0	10.0	1400	1400	1400.0	10.0
c206	1400	1400	1400.0	13.0	1400	1400	1400.0	13.2	1400	1400	1400.0	14.4
c207	1400	1400	1400.0	13.9	1400	1400	1400.0	15.2	1400	1400	1400.0	18.2
c208	1400	1400	1400.0	14.8	1400	1400	1400.0	15.8	1400	1400	1400.0	17.8
r101	1001	941	934.7	27.3	1001	891	889.1	36.0	1001	885	879.1	51.7
r102	1222	1146	1140.6	47.1	1222	1109	1103.4	106.7	1222	1122	1113.7	116.2
r103	1374	1277	1264.9	64.6	1374	1243	1233.8	102.0	1374	1231	1223.5	77.1
r104	1400	1335	1323.1	75.2	1400	1307	1288.4	84.2	1400	1290	1277.2	88.7
r105	1400	1128	1115.9	46.3	1400	1095	1086.6	53.5	1400	1071	1062.8	66.5
r106	1400	1247	1236.2	43.7	1400	1230	1218.8	68.9	1400	1210	1198.4	109.5
r107	1400	1302	1290.1	53.6	1400	1284	1268.0	86.1	1400	1262	1250.9	103.6
r108	1400	1342	1330.2	51.7	1400	1331	1317.6	72.8	1400	1311	1301.6	131.1
r109	1400	1235	1222.9	43.4	1400	1218	1204.1	60.3	1400	1190	1175.7	83.9
r110	1400	1295	1279.8	54.5	1400	1253	1241.2	56.6	1400	1247	1235.4	74.2
r111	1400	1295	1285.5	46.7	1400	1274	1263.5	79.3	1400	1259	1246.4	90.4
r112	1400	1343	1329.1	59.8	1400	1318	1306.9	90.6	1400	1319	1300.7	73.4
r201	1400	1400	1400.0	5.8	1400	1400	1400.0	6.1	1400	1400	1400.0	7.4
r202	1400	1400	1400.0	6.4	1400	1400	1400.0	6.5	1400	1400	1400.0	7.8
r203	1400	1400	1400.0	8.4	1400	1400	1400.0	7.3	1400	1400	1400.0	9.1
r204	1400	1400	1400.0	9.5	1400	1400	1400.0	10.4	1400	1400	1400.0	11.5
r205	1400	1400	1400.0	7.6	1400	1400	1400.0	9.6	1400	1400	1400.0	9.0
r206	1400	1400	1400.0	8.5	1400	1400	1400.0	9.8	1400	1400	1400.0	11.0
r207	1400	1400	1400.0	9.0	1400	1400	1400.0	11.3	1400	1400	1400.0	12.0
r208	1400	1400	1400.0	12.0	1400	1400	1400.0	14.0	1400	1400	1400.0	15.4
r209	1400	1400	1400.0	8.8	1400	1400	1400.0	10.7	1400	1400	1400.0	11.5
r210	1400	1400	1400.0	7.5	1400	1400	1400.0	8.9	1400	1400	1400.0	9.5
r211	1400	1400	1400.0	12.9	1400	1400	1400.0	16.2	1400	1400	1400.0	18.7
rc101	1400	1228	1211.1	33.0	1400	1174	1166.5	40.6	1400	1161	1149.7	66.2
rc102	1400	1359	1350.0	50.7	1400	1339	1326.6	56.5	1400	1331	1312.6	69.4
rc103	1400	1399	1383.1	61.0	1400	1393	1377.0	75.5	1400	1368	1359.0	86.7
rc104	1400	1400	1396.6	61.1	1400	1400	1395.6	79.4	1400	1398	1387.8	106.8
rc105	1400	1331	1321.2	43.5	1400	1309	1296.3	49.8	1400	1282	1275.2	57.3
rc106	1400	1369	1344.5	60.2	1400	1329	1315.2	76.7	1400	1310	1296.1	89.5
rc107	1400	1393	1379.2	61.5	1400	1361	1348.4	91.6	1400	1369	1347.9	198.6
rc108	1400	1399	1390.0	38.1	1400	1388	1375.4	95.4	1400	1370	1359.3	165.1
rc201	1400	1400	1400.0	8.6	1400	1400	1400.0	10.5	1400	1400	1400.0	12.1
rc202	1400	1400	1400.0	9.1	1400	1400	1400.0	11.0	1400	1400	1400.0	11.7
rc203	1400	1400	1400.0	12.3	1400	1400	1400.0	13.9	1400	1400	1400.0	16.2
rc204	1400	1400	1400.0	18.2	1400	1400	1400.0	28.7	1400	1400	1400.0	27.9
rc205	1400	1400	1400.0	10.4	1400	1400	1400.0	11.0	1400	1400	1400.0	10.8
rc206	1400	1400	1400.0	13.2	1400	1400	1400.0	16.3	1400	1400	1400.0	18.1
rc207	1400	1400	1400.0	16.0	1400	1400	1400.0	18.1	1400	1400	1400.0	19.8
rc208	1400	1400	1400.0	22.9	1400	1400	1400.0	28.1	1400	1400	1400.0	29.2

instances. The workload of each supplier is set to be the demand of the corresponding vertex. The duration of each period is set to $T=(l_0-e_0)/w$ and the workload limit of each inspector is set to 200. The total profits in the MPISP instances related to the Solomon's instance groups C1, C2, R1, R2, RC1 and RC2 are 1810, 1810, 1458, 1458, 1724 and 1724, respectively.

Table 5
Computational results for the MPISP instances with $m=9$.

Instance	$w=1$				$w=3$				$w=5$			
	UB	Max. workload	Ave. workload	Ave. time	UB	Max. workload	Ave. workload	Ave. time	UB	Max. workload	Ave. workload	Ave. time
c101	1730	1710	1707.0	22.5	1670	1630	1621.0	18.3	1480	1380	1380.0	6.2
c102	1800	1800	1799.0	24.6	1760	1750	1748.0	30.3	1640	1620	1620.0	9.8
c103	1800	1800	1800.0	9.9	1780	1770	1770.0	11.5	1680	1680	1680.0	8.7
c104	1800	1800	1800.0	5.5	1800	1800	1800.0	16.2	1680	1680	1680.0	13.9
c105	1800	1730	1728.0	31.6	1800	1680	1677.0	22.5	1710	1590	1590.0	11.1
c106	1800	1740	1738.0	32.6	1780	1690	1690.0	23.3	1650	1530	1530.0	10.1
c107	1800	1750	1748.0	52.5	1800	1720	1718.0	46.7	1710	1600	1600.0	12.0
c108	1800	1780	1770.0	34.0	1800	1740	1739.0	28.1	1710	1610	1610.0	14.6
c109	1800	1800	1800.0	22.8	1800	1800	1796.0	39.2	1710	1650	1650.0	15.5
c201	1800	1800	1800.0	4.3	1800	1800	1800.0	4.5	1800	1800	1800.0	4.7
c202	1800	1800	1800.0	5.0	1800	1800	1800.0	5.1	1800	1800	1800.0	4.8
c203	1800	1800	1800.0	5.6	1800	1800	1800.0	5.8	1800	1800	1800.0	5.9
c204	1800	1800	1800.0	6.5	1800	1800	1800.0	6.7	1800	1800	1800.0	6.9
c205	1800	1800	1800.0	4.6	1800	1800	1800.0	5.0	1800	1800	1800.0	4.6
c206	1800	1800	1800.0	4.9	1800	1800	1800.0	5.0	1800	1800	1800.0	5.1
c207	1800	1800	1800.0	5.5	1800	1800	1800.0	5.3	1800	1800	1800.0	5.4
c208	1800	1800	1800.0	5.7	1800	1800	1800.0	5.3	1800	1800	1800.0	5.5
r101	1171	1109	1098.3	37.4	1171	1045	1042.6	86.3	1171	1053	1048.8	62.7
r102	1321	1292	1287.7	81.1	1321	1247	1239.3	111.9	1321	1264	1259.9	92.4
r103	1413	1395	1388.2	69.6	1413	1370	1360.4	99.4	1413	1364	1359.6	103.6
r104	1458	1449	1445.4	65.0	1458	1414	1408.8	75.4	1458	1426	1418.9	89.3
r105	1458	1290	1273.8	42.0	1458	1270	1245.8	57.4	1458	1228	1218.3	95.8
r106	1458	1386	1371.5	43.8	1458	1359	1347.9	65.6	1458	1350	1340.0	112.0
r107	1458	1429	1419.8	46.2	1458	1419	1409.0	84.3	1458	1397	1390.8	99.3
r108	1458	1458	1457.5	71.5	1458	1455	1446.5	62.4	1458	1445	1436.4	119.7
r109	1458	1378	1371.1	33.0	1458	1367	1358.0	45.8	1458	1340	1330.7	97.1
r110	1458	1424	1416.0	46.3	1458	1407	1395.0	90.9	1458	1388	1373.4	81.5
r111	1458	1433	1426.3	69.2	1458	1411	1399.7	51.9	1458	1401	1394.1	61.4
r112	1458	1458	1455.4	65.4	1458	1452	1446.1	96.3	1458	1440	1430.3	86.5
r201	1458	1458	1458.0	6.1	1458	1458	1458.0	5.9	1458	1458	1458.0	6.5
r202	1458	1458	1458.0	6.5	1458	1458	1458.0	6.6	1458	1458	1458.0	7.0
r203	1458	1458	1458.0	7.4	1458	1458	1458.0	7.4	1458	1458	1458.0	7.8
r204	1458	1458	1458.0	8.7	1458	1458	1458.0	8.8	1458	1458	1458.0	9.3
r205	1458	1458	1458.0	7.1	1458	1458	1458.0	7.4	1458	1458	1458.0	7.9
r206	1458	1458	1458.0	7.4	1458	1458	1458.0	7.8	1458	1458	1458.0	8.2
r207	1458	1458	1458.0	9.1	1458	1458	1458.0	8.9	1458	1458	1458.0	9.6
r208	1458	1458	1458.0	10.9	1458	1458	1458.0	11.0	1458	1458	1458.0	11.7
r209	1458	1458	1458.0	7.3	1458	1458	1458.0	7.9	1458	1458	1458.0	8.1
r210	1458	1458	1458.0	6.9	1458	1458	1458.0	7.9	1458	1458	1458.0	8.5
r211	1458	1458	1458.0	10.0	1458	1458	1458.0	9.7	1458	1458	1458.0	10.8
rc101	1724	1456	1435.9	51.9	1724	1391	1380.2	78.3	1724	1371	1362.4	91.6
rc102	1724	1585	1575.2	41.4	1724	1567	1556.4	75.9	1724	1535	1523.1	86.6
rc103	1724	1672	1655.3	52.9	1724	1651	1630.6	90.6	1724	1635	1612.0	79.6
rc104	1724	1702	1689.1	47.7	1724	1673	1663.0	58.4	1724	1668	1647.7	116.9
rc105	1696	1542	1533.8	45.3	1696	1527	1515.3	54.2	1696	1486	1477.9	68.5
rc106	1724	1606	1586.7	40.4	1724	1577	1563.2	85.8	1724	1552	1539.3	104.6
rc107	1724	1651	1633.5	59.0	1724	1637	1614.2	87.4	1724	1613	1596.5	92.2
rc108	1724	1687	1666.1	41.2	1724	1661	1642.4	52.4	1724	1658	1643.4	114.3
rc201	1724	1724	1724.0	5.1	1724	1724	1724.0	5.1	1724	1724	1724.0	5.5
rc202	1724	1724	1724.0	5.6	1724	1724	1724.0	5.7	1724	1724	1724.0	4.5
rc203	1724	1724	1724.0	5.9	1724	1724	1724.0	5.7	1724	1724	1724.0	5.6
rc204	1724	1724	1724.0	7.4	1724	1724	1724.0	7.1	1724	1724	1724.0	7.5
rc205	1724	1724	1724.0	5.4	1724	1724	1724.0	5.0	1724	1724	1724.0	4.1
rc206	1724	1724	1724.0	5.9	1724	1724	1724.0	5.6	1724	1724	1724.0	5.2
rc207	1724	1724	1724.0	6.4	1724	1724	1724.0	5.7	1724	1724	1724.0	5.6
rc208	1724	1724	1724.0	8.1	1724	1724	1724.0	7.6	1724	1724	1724.0	6.3

7.2. Results for the TOPTW instances

To evaluate the performance of our algorithm based on the TOPTW instances, we considered the following four state-of-the-art existing algorithms in our comparisons:

- ILS: the iterated local search algorithm by Vansteenwegen et al. [41].
- SSA: the slow version of the simulated annealing algorithm by Lin and Yu [29].
- GVNS: the LP-based granular variable neighborhood search algorithm by Labadie et al. [25].
- I3CH: the iterative three-component heuristic (I3CH) by Hu and Lim [22].

Lin and Yu [29] proposed two versions of simulated annealing algorithm for the TOPTW, namely a fast version and a slow one. Compared with the fast version, the slow simulated annealing algorithm (SSA) is able to find better solutions at the expense of more computation time. As we are more concerned with solution

Table 6
Computational results for the MPISP instances with $m=11$.

Instance	$w=1$				$w=3$				$w=5$			
	UB	Max. workload	Ave. workload	Ave. time	UB	Max. workload	Ave. workload	Ave. time	UB	Max. workload	Ave. workload	Ave. time
c101	1810	1810	1810.0	4.0	1740	1740	1740.0	18.2	1540	1480	1480.0	4.2
c102	1810	1810	1810.0	4.9	1760	1760	1760.0	4.5	1670	1670	1670.0	11.3
c103	1810	1810	1810.0	5.4	1780	1780	1780.0	5.2	1730	1730	1730.0	6.3
c104	1810	1810	1810.0	6.9	1800	1800	1800.0	6.7	1750	1750	1750.0	8.2
c105	1810	1810	1810.0	4.0	1810	1810	1810.0	8.4	1810	1730	1730.0	7.2
c106	1810	1810	1810.0	4.7	1780	1780	1780.0	4.7	1710	1670	1670.0	7.0
c107	1810	1810	1810.0	4.8	1810	1810	1810.0	4.6	1810	1760	1760.0	8.7
c108	1810	1810	1810.0	5.1	1810	1810	1810.0	4.8	1810	1770	1770.0	14.7
c109	1810	1810	1810.0	6.7	1810	1810	1810.0	5.9	1810	1810	1810.0	7.7
c201	1810	1810	1810.0	4.8	1810	1810	1810.0	5.0	1810	1810	1810.0	5.0
c202	1810	1810	1810.0	5.4	1810	1810	1810.0	5.4	1810	1810	1810.0	5.6
c203	1810	1810	1810.0	6.5	1810	1810	1810.0	6.5	1810	1810	1810.0	6.3
c204	1810	1810	1810.0	8.2	1810	1810	1810.0	7.8	1810	1810	1810.0	8.3
c205	1810	1810	1810.0	5.2	1810	1810	1810.0	5.2	1810	1810	1810.0	5.3
c206	1810	1810	1810.0	5.3	1810	1810	1810.0	5.6	1810	1810	1810.0	5.6
c207	1810	1810	1810.0	5.9	1810	1810	1810.0	6.3	1810	1810	1810.0	5.9
c208	1810	1810	1810.0	6.0	1810	1810	1810.0	6.0	1810	1810	1810.0	6.5
r101	1307	1243	1225.5	28.2	1307	1170	1158.2	58.8	1307	1191	1181.3	51.3
r102	1388	1367	1358.3	38.9	1388	1328	1321.6	62.0	1388	1338	1333.5	80.2
r103	1441	1441	1433.6	22.2	1441	1423	1420.0	46.9	1441	1425	1421.9	77.3
r104	1458	1458	1458.0	5.3	1458	1458	1452.8	29.1	1458	1458	1458.0	27.4
r105	1458	1391	1379.0	25.0	1458	1378	1363.4	57.8	1458	1351	1339.2	61.7
r106	1458	1453	1445.5	53.4	1458	1435	1426.7	54.9	1458	1432	1422.0	83.7
r107	1458	1458	1458.0	5.4	1458	1458	1458.0	14.0	1458	1458	1458.0	37.0
r108	1458	1458	1458.0	5.2	1458	1458	1458.0	6.4	1458	1458	1458.0	6.5
r109	1458	1458	1455.7	39.5	1458	1456	1445.4	36.3	1458	1441	1432.4	71.8
r110	1458	1458	1458.0	7.3	1458	1458	1458.0	20.8	1458	1458	1457.4	54.0
r111	1458	1458	1458.0	6.4	1458	1458	1456.8	14.7	1458	1458	1457.2	39.8
r112	1458	1458	1458.0	5.1	1458	1458	1458.0	5.7	1458	1458	1458.0	7.1
r201	1458	1458	1458.0	6.2	1458	1458	1458.0	6.2	1458	1458	1458.0	6.7
r202	1458	1458	1458.0	6.5	1458	1458	1458.0	6.9	1458	1458	1458.0	7.1
r203	1458	1458	1458.0	7.4	1458	1458	1458.0	7.4	1458	1458	1458.0	8.2
r204	1458	1458	1458.0	8.6	1458	1458	1458.0	9.4	1458	1458	1458.0	10.1
r205	1458	1458	1458.0	7.2	1458	1458	1458.0	7.7	1458	1458	1458.0	8.0
r206	1458	1458	1458.0	7.4	1458	1458	1458.0	8.1	1458	1458	1458.0	8.7
r207	1458	1458	1458.0	9.8	1458	1458	1458.0	9.5	1458	1458	1458.0	10.0
r208	1458	1458	1458.0	10.8	1458	1458	1458.0	11.3	1458	1458	1458.0	11.9
r209	1458	1458	1458.0	8.3	1458	1458	1458.0	8.6	1458	1458	1458.0	8.7
r210	1458	1458	1458.0	8.1	1458	1458	1458.0	8.0	1458	1458	1458.0	8.7
r211	1458	1458	1458.0	10.3	1458	1458	1458.0	10.1	1458	1458	1458.0	10.7
rc101	1724	1621	1609.4	41.3	1724	1563	1548.6	46.2	1724	1538	1518.5	88.9
rc102	1724	1690	1677.0	33.5	1724	1674	1653.7	34.3	1724	1656	1644.6	39.7
rc103	1724	1724	1723.0	21.3	1724	1724	1717.2	50.5	1724	1714	1707.2	25.8
rc104	1724	1724	1724.0	7.3	1724	1724	1724.0	7.0	1724	1724	1724.0	17.5
rc105	1719	1681	1673.1	48.2	1719	1661	1642.6	49.2	1719	1645	1630.2	80.3
rc106	1724	1724	1714.5	55.2	1724	1707	1691.6	60.1	1724	1687	1671.8	59.2
rc107	1724	1724	1724.0	14.3	1724	1724	1722.7	37.1	1724	1724	1715.7	51.6
rc108	1724	1724	1724.0	4.6	1724	1724	1724.0	9.3	1724	1724	1723.4	22.3
rc201	1724	1724	1724.0	5.5	1724	1724	1724.0	5.5	1724	1724	1724.0	5.3
rc202	1724	1724	1724.0	6.2	1724	1724	1724.0	6.0	1724	1724	1724.0	5.0
rc203	1724	1724	1724.0	6.1	1724	1724	1724.0	6.0	1724	1724	1724.0	6.5
rc204	1724	1724	1724.0	8.4	1724	1724	1724.0	8.0	1724	1724	1724.0	8.1
rc205	1724	1724	1724.0	5.4	1724	1724	1724.0	5.7	1724	1724	1724.0	4.9
rc206	1724	1724	1724.0	6.0	1724	1724	1724.0	5.9	1724	1724	1724.0	5.4
rc207	1724	1724	1724.0	7.1	1724	1724	1724.0	5.9	1724	1724	1724.0	6.2
rc208	1724	1724	1724.0	8.4	1724	1724	1724.0	7.8	1724	1724	1724.0	6.9

quality, we used the SSA rather than the fast version in the comparisons. For each TOPTW instance, the ILS, SSA and I3CH were executed only once while the GVNS was performed five times. Although these algorithms were coded in different programming languages and executed on different computational environments (see Table 1), we believe that there is no dramatic difference between the speeds of these algorithms and it is acceptable to directly compare their computation times.

Table 2 summarizes the results of the TS, ILS, SSA, GVNS and I3CH for the TOPTW instances. We first identified the best solution value (BSV) obtained by these five algorithms and then computed the ratio of the best solution value produced by each algorithm to the BSV. The columns "Ratio (%)" and "Time (s)" show the average ratios and average computation times of all instance groups. Since Labadie et al. [25] did not report the best solution value in their article, we filled the corresponding cells with "N/A" and ignored these cells when calculating the overall average ratio. Beside the name of each algorithm, we give the number of times it was run for each instance. The overall average values of "Ratio (%)" and "Time (s)" are presented in the last row and the best ratios in each

Table 7
Computational results for the MPISP instances with $m=13$.

Instance	$w=1$				$w=3$				$w=5$			
	UB	Max. workload	Ave. workload	Ave. time	UB	Max. workload	Ave. workload	Ave. time	UB	Max. workload	Ave. workload	Ave. time
c101	1810	1810	1810.0	4.2	1740	1740	1740.0	3.6	1540	1530	1530.0	3.3
c102	1810	1810	1810.0	5.0	1760	1760	1760.0	4.2	1670	1670	1670.0	4.2
c103	1810	1810	1810.0	5.7	1780	1780	1780.0	5.1	1730	1730	1730.0	6.4
c104	1810	1810	1810.0	7.1	1800	1800	1800.0	7.1	1750	1750	1750.0	9.4
c105	1810	1810	1810.0	4.7	1810	1810	1810.0	4.7	1810	1810	1810.0	4.6
c106	1810	1810	1810.0	4.6	1780	1780	1780.0	4.4	1710	1710	1710.0	4.0
c107	1810	1810	1810.0	5.2	1810	1810	1810.0	4.9	1810	1810	1810.0	4.4
c108	1810	1810	1810.0	5.8	1810	1810	1810.0	5.4	1810	1810	1810.0	5.3
c109	1810	1810	1810.0	7.2	1810	1810	1810.0	6.5	1810	1810	1810.0	6.1
c201	1810	1810	1810.0	5.1	1810	1810	1810.0	5.1	1810	1810	1810.0	5.1
c202	1810	1810	1810.0	5.4	1810	1810	1810.0	5.8	1810	1810	1810.0	5.9
c203	1810	1810	1810.0	6.8	1810	1810	1810.0	6.6	1810	1810	1810.0	6.7
c204	1810	1810	1810.0	8.8	1810	1810	1810.0	7.8	1810	1810	1810.0	8.6
c205	1810	1810	1810.0	5.7	1810	1810	1810.0	5.8	1810	1810	1810.0	5.4
c206	1810	1810	1810.0	6.0	1810	1810	1810.0	6.3	1810	1810	1810.0	5.7
c207	1810	1810	1810.0	6.3	1810	1810	1810.0	6.0	1810	1810	1810.0	6.0
c208	1810	1810	1810.0	6.6	1810	1810	1810.0	6.2	1810	1810	1810.0	6.0
r101	1386	1345	1329.0	36.0	1386	1266	1255.4	45.8	1386	1302	1286.2	30.3
r102	1420	1412	1407.8	31.5	1420	1371	1367.5	43.9	1420	1387	1383.0	37.1
r103	1458	1458	1455.7	31.5	1458	1444	1444.0	25.3	1458	1451	1449.6	70.8
r104	1458	1458	1458.0	5.3	1458	1458	1458.0	5.9	1458	1458	1458.0	7.2
r105	1458	1450	1441.3	33.1	1458	1438	1429.0	37.3	1458	1419	1410.8	38.9
r106	1458	1458	1458.0	5.4	1458	1458	1458.0	22.1	1458	1458	1456.2	14.3
r107	1458	1458	1458.0	5.2	1458	1458	1458.0	5.9	1458	1458	1458.0	6.3
r108	1458	1458	1458.0	6.1	1458	1458	1458.0	6.8	1458	1458	1458.0	7.3
r109	1458	1458	1458.0	4.2	1458	1458	1458.0	6.3	1458	1458	1458.0	9.4
r110	1458	1458	1458.0	4.5	1458	1458	1458.0	5.7	1458	1458	1458.0	5.9
r111	1458	1458	1458.0	5.3	1458	1458	1458.0	6.1	1458	1458	1458.0	6.2
r112	1458	1458	1458.0	5.9	1458	1458	1458.0	6.7	1458	1458	1458.0	8.0
r201	1458	1458	1458.0	6.3	1458	1458	1458.0	6.7	1458	1458	1458.0	6.8
r202	1458	1458	1458.0	7.1	1458	1458	1458.0	7.1	1458	1458	1458.0	7.4
r203	1458	1458	1458.0	7.6	1458	1458	1458.0	8.0	1458	1458	1458.0	8.4
r204	1458	1458	1458.0	9.2	1458	1458	1458.0	9.3	1458	1458	1458.0	10.3
r205	1458	1458	1458.0	7.7	1458	1458	1458.0	8.4	1458	1458	1458.0	8.4
r206	1458	1458	1458.0	8.0	1458	1458	1458.0	9.1	1458	1458	1458.0	8.8
r207	1458	1458	1458.0	9.7	1458	1458	1458.0	9.9	1458	1458	1458.0	10.1
r208	1458	1458	1458.0	11.3	1458	1458	1458.0	11.2	1458	1458	1458.0	12.3
r209	1458	1458	1458.0	8.6	1458	1458	1458.0	8.5	1458	1458	1458.0	9.1
r210	1458	1458	1458.0	7.9	1458	1458	1458.0	8.4	1458	1458	1458.0	8.6
r211	1458	1458	1458.0	11.3	1458	1458	1458.0	11.5	1458	1458	1458.0	11.9
rc101	1724	1703	1691.1	29.2	1724	1677	1654.5	40.8	1724	1649	1637.7	39.1
rc102	1724	1724	1720.0	27.0	1724	1724	1712.3	26.3	1724	1721	1709.9	24.3
rc103	1724	1724	1724.0	4.4	1724	1724	1724.0	5.5	1724	1724	1724.0	6.0
rc104	1724	1724	1724.0	5.0	1724	1724	1724.0	6.0	1724	1724	1724.0	6.4
rc105	1724	1724	1717.8	22.8	1724	1709	1702.9	28.2	1724	1701	1692.8	36.0
rc106	1724	1724	1724.0	5.9	1724	1724	1724.0	7.5	1724	1724	1724.0	25.6
rc107	1724	1724	1724.0	4.4	1724	1724	1724.0	6.2	1724	1724	1724.0	5.7
rc108	1724	1724	1724.0	4.9	1724	1724	1724.0	5.9	1724	1724	1724.0	6.1
rc201	1724	1724	1724.0	5.8	1724	1724	1724.0	6.2	1724	1724	1724.0	5.9
rc202	1724	1724	1724.0	6.4	1724	1724	1724.0	6.5	1724	1724	1724.0	5.4
rc203	1724	1724	1724.0	6.4	1724	1724	1724.0	6.6	1724	1724	1724.0	6.5
rc204	1724	1724	1724.0	8.2	1724	1724	1724.0	7.8	1724	1724	1724.0	9.1
rc205	1724	1724	1724.0	5.9	1724	1724	1724.0	5.8	1724	1724	1724.0	4.9
rc206	1724	1724	1724.0	6.5	1724	1724	1724.0	6.9	1724	1724	1724.0	5.7
rc207	1724	1724	1724.0	7.3	1724	1724	1724.0	6.3	1724	1724	1724.0	6.2
rc208	1724	1724	1724.0	8.9	1724	1724	1724.0	8.5	1724	1724	1724.0	7.3

Table 8
The average computation time of each MPISP instance group.

$m=7$	$w=1$	$w=3$	$w=5$
(a)			
c1	9.31	12.58	23.07
c2	15.74	16.59	20.29
r1	51.16	74.75	88.86
r2	8.76	10.07	11.17
rc1	51.14	70.69	104.95
rc2	13.84	17.20	18.23
$m=9$	$w=1$	$w=3$	$w=5$
(b)			
c1	26.22	26.23	11.32
c2	5.26	5.34	5.36
r1	55.88	77.30	91.78
r2	7.95	8.12	8.67
rc1	47.48	72.88	94.29
rc2	6.23	5.94	5.54
$m=11$	$w=1$	$w=3$	$w=5$
(c)			
c1	5.17	7.00	8.37
c2	5.91	5.98	6.06
r1	20.16	33.95	49.82
r2	8.24	8.47	8.98
rc1	28.21	36.71	48.16
rc2	6.64	6.35	6.04
$m=13$	$w=1$	$w=3$	$w=5$
(d)			
c1	5.50	5.10	5.30
c2	6.34	6.20	6.18
r1	14.50	18.15	20.14
r2	8.61	8.92	9.28
rc1	12.95	15.80	18.65
rc2	6.93	6.83	6.38

row are marked in bold. All of the detailed solutions can be found in Appendix B.

The numbers of the best solution values achieved by ILS, SSA, GVNS, I3CH and TS are 85, 191, 138, 247 and 272, respectively (see Appendix B). Although TS produced the largest number of the best solution values and the largest overall average ratio (i.e., 0.9980), we cannot conclude that this algorithm is superior to the rest since it was executed more times and consumed more computation time. We can only say that the results generated by our TS algorithm are comparable to those generated by the best existing approaches for the TOPTW.

7.3. Analysis of components

As our TS algorithm consists of three main components, namely local search with tabu moves (LS), ejection pool (EP) algorithm and perturbation (PER) procedure, it is important to investigate the performance of these components. In the experiments, we considered the combinations LS + EP, LS + PER, EP + PER and LS + EP + PER and 50 TOPTW instances generated from pr01 – pr10. Table 3 shows the average performance of these four combinations. For each test instance, we calculated a ratio that is equal to the average profit over ten runs divided by the best solution value generated by these four combinations (for the detailed results, see Appendix C). The column "Avg. Ratio (%)" shows the average values of the ratios of the instances grouped by m. From this table, we can see that on average, LS + EP + PER performed best while LS + PER generated the worst results. Moreover, EP + PER and LS + EP performed slightly worse than LS + EP + PER. These observations imply that the ejection pool algorithm plays a critical role in improving the solution quality of our proposed approach.

7.4. Results for the MPISP instances

The computational results for the 672 MPISP instances are reported in Tables 4–7. The column "UB" shows the upper bound of each instance obtained from the constrained knapsack model (see Section 6). Each block corresponds to a value of w and includes the maximum workload *Max. workload*, the average workload *Ave. workload* and the average computation time *Ave. time* over the 10 executions. Since the MPISP is a new problem, there is no existing algorithm tailored for it. As a consequence, we cannot compare our tabu search algorithm with other approaches. The test instances and computational results reported in this paper can serve as benchmarks for future researchers on this problem.

Theoretically speaking, the optimal solution value of some instance with $w=d$ must be greater than or equal to that of the same instance with $w=kd$, where k is an integral number. This is because we can always construct a feasible solution to an instance with $w=d$ from any solution to this instance with $w=kd$. For example, the optimal solution value of an instance with $w=1$ must be greater than those of this instance with $w=3$ and $w=5$. Since our tabu search algorithm is a stochastic approach, it is possible that the maximum workload of some instance with $w=3$ or $w=5$ is larger than that of this instance with $w=1$. Fortunately, we did not encounter such phenomenon in our experiments. However, when w_2/w_1 is not an integer, an instance with w_2 may have larger optimal solution value than this instance with w_1. For example, the optimal solution value of an instance with $w=5$ may be larger than this instance with $w=3$. In these tables, we can find several instances with $w=3$ have larger maximum workload than their counterparts with $w=5$. Since these maximum workloads may not be optimal, we cannot judge whether these phenomena were resulted from the randomness of the tabu search algorithm or the nature of the instances. Obviously, for each instance with a certain w, the maximum workload increases as the number of vehicles.

For each instance group, we calculated the average of all "Ave. Time" and show the statistical results in Table 8. From this table, we can observe that in most cases (those are not marked in bold), the average computation times increase as the value of w.

8. Conclusion

This paper introduces an inspector scheduling problem in which a set of inspectors are dispatched to complete a set of inspection requests at different locations in a multi-period planning horizon. At the end of each period, each inspector is not required to return to the depot but has to stay at one of the inspection locations for recuperation. We first studied the way of computing the shortest transit time between any pair of locations when the working time periods are considered. Next, we introduced several local search operators that were adapted from classical VRPTW operators and integrated these adapted operators in a tabu search framework. Moreover, we presented a constrained knapsack model that is able to produce an upper bound for the MPISP. Finally, we evaluated the algorithm based on 370 TOPTW instances and 672 MPISP instances. The experimental results reported in this study show the effectiveness of our algorithm and can serve as benchmarks for future researchers. Since the working time windows of the scheduling subjects and the use of waypoint are very common considerations in practice, a possible research direction can focus on studying the variants of other existing vehicle routing models that involves these two factors.

Acknowledgements

This research was partially supported by the Fundamental Research Funds for the Central Universities, HUST (Grant No.

2014QN206), Sun Yat-Sen University (Grant No. 15lgpy37) and National Natural Science Foundation of China (Grant No. 71201065, 71131004).

Appendix A. Supplementary data

Supplementary data associated with this paper can be found in the online version at http://dx.doi.org/10.1016/j.cor.2015.01.003.

References

[1] Ahuja RK, Magnanti TL, Orlin JB. Network flowstheory, algorithms, and applications. Englewood Cliffs, N.J: Prentice-Hall; 1993.
[2] Azi N, Gendreau M, Potvin J-Y. An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. Eur J Oper Res 2010;202(3):756–63.
[3] Barnes JW, Chambers JB. Solving the job shop scheduling problem with tabu search. IIE Trans 1995;27(2):257–63.
[4] Battarra M, Monaci M, Vigo D. An adaptive guidance approach for the heuristic solution of a minimum multiple trip vehicle routing problem. Comput Oper Res 2009;36(11):3041–50.
[5] Bräysy O, Gendreau M. Vehicle routing problem with time windows. Part I: route construction and local search algorithms. Transp Sci 2005;39(1):104–18.
[6] Burke EK, Kendall G, Soubeiga E. A tabu-search hyperheuristic for timetabling and rostering. J Heuristics 2003;9(6):451–70.
[7] Cai Y, Zhang Z, Guo S, Qin H, Lim A. A tree-based tabu search algorithm for the manpower allocation problem with time windows and job-teaming constraints. In: Proceedings of the twenty-third international joint conference on artificial intelligence; 2013. p. 496–502.
[8] Castro M, Sörensen K, Vansteenwegen P, Goos P. A memetic algorithm for the travelling salesperson problem with hotel selection. Comput Oper Res 2013;40(7):1716–28.
[9] Cheang B, Gao X, Lim A, Qin H, Zhu W. Multiple pickup and delivery traveling salesman problem with last-in-first-out loading and distance constraints. Eur J Oper Res 2012;223(1):60–75.
[10] Cheang B, Lim A, Rodrigues B. Nurse rostering problems—a bibliographic survey. Eur J Oper Res 2003;151(3):447–60.
[11] Chiang W-C, Russell RA. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. INFORMS J Comput 1997;9(4):417–30.
[12] Cordeau J-F, Gendreau M, Laporte G. A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks 1997;30(2):105–19.
[13] Divsalar A, Vansteenwegen P, Cattryssea D. A variable neighborhood search method for the orienteering problem with hotel selection. Int J Product Econ 2013;145(1):150–60.
[14] Divsalar A, Vansteenwegen P, Cattryssea D. A memetic algorithm for the orienteering problem with hotel selection. Eur J Oper Res 2014;237(1):29–49.
[15] Ernst AT, Jiang H, Krishnamoorthy M, Sier D. Staff scheduling and rostering: a review of applications, methods and models. Eur J Oper Res 2004;153(1):3–27.
[16] Gaudioso M, Paletta G. A heuristic for the periodic vehicle routing problem. Transp Sci 1992;26(2):86–92.
[17] Gendreau M, Hertz A, Laporte G. A tabu search heuristic for the vehicle routing problem. Manag Sci 1994;40(10):1276–90.
[18] Goel A. Vehicle scheduling and routing with drivers' working hours. Transp Sci 2009;43(1):17–26.
[19] Goel A. Truck driver scheduling in the European union. Transp Sci 2010;44(4):429–41.
[20] Gordeau J-F, Laporte G, Mercier A. A unified tabu search heuristic for vehicle routing problems with time windows. J Oper Res Soc 52(8);2001:928–36.
[21] Hemmelmayr VC, Doerner KF, Hartl RF. A variable neighborhood search heuristic for periodic routing problems. Eur J Oper Res 2009;195(3):791–802.
[22] Hu Q, Lim A. An iterative three-component heuristic for the team orienteering problem with time windows. Eur J Oper Res 2014;232(2):276–86.
[23] Kohl N, Karisch SE. Airline crew rostering: problem types, modeling, and optimization. Ann Oper Res 2004;127(1–4):223–57.
[24] Kok AL, Meyer CM, Kopfer H, Schutten JMJ. A dynamic programming heuristic for the vehicle routing problem with time windows and European community social legislation. Transp Sci 2010;44(4):442–54.
[25] Labadie N, Mansini R, Melechovský J, Wolfler Calvo R. The team orienteering problem with time windows: an LP-based granular variable neighborhood search. Eur J Oper Res 2012;220(1):15–27.
[26] Lau HC, Sim M, Teo KM. Vehicle routing problem with time windows and a limited number of vehicles. Eur J Oper Res 2003;148(3):559–69.
[27] Li Y, Lim A, Rodrigues B. Manpower allocation with time windows and job-teaming constraints. Naval Res Logist 2005;52(4):302–11.
[28] Lim A, Zhang X. A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. INFORMS J Comput 2007;19(3):443–57.
[29] Lin S-W, Yu VF. A simulated annealing heuristic for the team orienteering problem with time windows. Eur J Oper Res 2012;217(1):94–107.
[30] Macedo R, Alves C, Valério de Carvalho JM, Clautiaux F, Hanafi S. Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. Eur J Oper Res 2011;214(3):536–45.
[31] Montemanni R, Gambardella LM. An ant colony system for team orienteering problems with time windows. Found Comput Dec Sci 2009;34(4):287–306.
[32] Nagata Y, Bräysy O. A powerful route minimization heuristic for the vehicle routing problem with time windows. Oper Res Lett 2009;37(5):333–8.
[33] Righini G, Salani M. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. Comput Oper Res 2009;36(4):1191–203.
[34] Savelsbergh M, Sol M. Drive: dynamic routing of independent vehicles. Oper Res 1998;46(4):474–90.
[35] Solomon MM. Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper Res 1987;35(2):254–65.
[36] Tang H, Miller-Hooks E, Tomastik R. Scheduling technicians for planned maintenance of geographically distributed equipment. Transp Res Part E: Logist Transp Rev 2007;43(5):591–609.
[37] Toth P, Vigo D. The vehicle routing problem, vol. 9. Philadelphia, PA: Siam; 2002.
[38] Toth P, Vigo D. The granular tabu search and its application to the vehicle-routing problem. INFORMS J Comput 2003;15(4):333–46.
[39] Vansteenwegen P, Souffriau W, Oudheusden DV. The orienteering problem: a survey. Eur J Oper Res 2011;209(1):1–10.
[40] Vansteenwegen P, Souffriau W, Sörensen K. The travelling salesperson problem with hotel selection. J Oper Res Soc 2012;63(2):207–17.
[41] Vansteenwegen P, Souffriau W, Vanden Berghe G, van Oudheusden D. Iterated local search for the team orienteering problem with time windows. Comput Oper Res 2009;36(12):3281–90.
[42] Xu H, Chen Z-L, Rajagopal S, Arunapuram S. Solving a practical pickup and delivery problem. Transp Sci 2003;37(3):347–64.
[43] Zäpfel G, Bögl M. Multi-period vehicle routing and crew scheduling with outsourcing options. Int J Prod Econ 2008;113(2):980–96.
[44] Zhang Z, Che O, Cheang B, Lim A, Qin H. A memetic algorithm for the multiperiod vehicle routing problem with profit. Eur J Oper Res 2013;229(3):573–84.
[45] Zhu W, Qin H, Lim A, Wang L. A two-stage tabu search algorithm with enhanced packing heuristics for the 3L-CVRP and M3L-CVRP. Comput Oper Res 2012;39(9):2178–95.