# A scatter search algorithm for time-dependent prize-collecting arc routing problems

Computers & Operations Research 134 (2021) 105392

Contents lists available at ScienceDirect

Computers and Operations Research

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

# A scatter search algorithm for time-dependent prize-collecting arc routing problems

Vahid Riahi [a,1], M.A. Hakim Newton [b], Abdul Sattar [b]

[a] The Australian e-Health Research Centre, CSIRO, Melbourne, VIC 3052, Australia
[b] Institute for Integrated and Intelligent Systems (IIIS), Griffith University, Brisbane, QLD 4111, Australia

ARTICLE INFO

*Keyword:*
Arc routing problems
Time-dependent
Heuristics
Single vehicle
Scatter search

ABSTRACT

Time-dependent prize-collecting arc routing problems (TD-PARPs) generalise the regular prize-collecting arc routing problems (PARPs). PARPs have arcs associated with collectable prizes along with travelling costs. TD-PARPs allow travel times to vary at the travelling horizon so that real-life uncertain factors such as traffic and weather conditions can be taken into account. A TD-PARP is to find a travelling route that maximises the profit i.e. total collected prizes minus total travelling costs. TD-PARPs have two facets: selecting a subset of arcs to be travelled and scheduling the selected arcs in the travelling route. TD-PARPs have not been studied much although they are more realistic and generic. In this paper, we first propose a set of deterministic heuristic search algorithms that range from a simple procedure producing quite good results in a fraction of a CPU second to a more extensive procedure producing high-quality results but at the expense of slightly extra CPU time. In this paper, we then propose a meta-heuristic based scatter search (SS) algorithm for TD-PARPs. For the improvement method in the SS algorithm, we propose a *multi-operator algorithm* that incorporates various neighbourhood operators to diversify the local exploration. For the combination method in the SS algorithm, we propose a *2-level path relinking* procedure, which explores combinations of visited and unvisited arcs using two different operators. To control the diversity of the solutions in the SS algorithm, we propose a new effective distance measurement. The experimental results on standard benchmark problems indicate that the proposed SS algorithm significantly outperforms the state-of-the-art existing methods.

## 1. Introduction

Arc routing problems (ARPs) are well-known among vehicle routing and scheduling problems. ARPs have been studied for a long time (Christofides, 1973; Assad and Golden, 1995; Benavent et al., 2007; Dror, 2012) because of their wide range of applications such as waste collection, newspaper delivery, postal service, milk delivery, electric meter reading, road maintenance, and street cleaning (Corberán et al., 2020). ARPs are modelled as directed graphs where arcs represent demand customers incurring non-negative costs and nodes represent pickup and delivery locations for the demand customers. Mourão and Pinto (2017) provides a comprehensive review of the ARP literature. ARPs have many variants. In one variant, all arcs must be visited (Aráoz et al., 2013). In another one, called profit-based ARPs, arcs are associated with prizes and so with profits (prizes minus costs). In yet other variants, a subset of arcs can be visited or profits can be collected multiple times for the same arc (Pearn and Wang, 2003; Archetti et al., 2010).

One of the main variants of the profit-based ARPs is the prize-collecting arc routing problems (PARPs) (Aráoz et al., 2006). In PARPs, the prize associated with each arc is collected at most once: the first time the arc is visited. PARPs have several practical applications that include freight industry (Black et al., 2013). PARPs belong to the class of NP-Hard problems (Aráoz et al., 2006). PARPs have been further studied in the literature. Aráoz et al. (2009b) proposed a two-phase heuristic algorithm for PARPs based on constraint relaxation. Akbari and Salman (2017) studied PARPs with arcs being blocked to simulate the logistics of a road clearing operation after a disaster and proposed a relaxed mixed integer programming model to solve the problems.

PARPs have their variants as well. Clustered PARPs (CPARPs) Aráoz et al. (2009a) have arcs clustered and all arcs of a cluster must be visited when any of the arcs of that cluster is visited. A banch-and-bound algorithm has been presented for CPARPs by Aráoz et al. (2009a). Windy CPARPs (WCPARPs) are CPARPs having asymmetric costs on arcs and a

*E-mail addresses:* vahid.riahi@csiro.au (V. Riahi), mahakim.newton@griffith.edu.au (M.A. Hakim Newton), a.sattar@griffith.edu.au (A. Sattar).
[1] 343 Royal Parade, Parkville VIC 3052.

https://doi.org/10.1016/j.cor.2021.105392
Received 18 September 2020; Received in revised form 17 May 2021; Accepted 17 May 2021
Available online 1 June 2021

branch-and-cut algorithm has been presented for WCPARPs by Corberán et al. (2011). Capacitated PARPs (CaPARPs) have fleets of capacitated vehicles, and heuristic and branch-and-bound algorithms exist for them (Archetti et al., 2010). Moreover, a hierarchical decomposition based algorithm (Tang et al., 2016) and a two-phase exact algorithm based on a branch-and-cut algorithm (Archetti et al., 2017) have been developed for CaPARPs. Later, Montero et al. (2019) modeled a milk delivery problem in Chile as CaPARPs and proposed an integer programming (IP) formulation and a search algorithm to solve the problem.

One of the key assumptions made by all of the above-mentioned studies is that the travel time associated with any arc is a constant. However, in real-life scenarios, travel times can be affected by traffic, weather, or other factors. When the actual travel times differ from the originally planned ones, the changes may have direct impact on already scheduled routes and the profits. More accurate travel times might help provide better travelling routes, which might further result into more profit by serving more customer demands and reducing costs. Time-dependent travel times allow travelling costs of arcs to be varying based on the time when the travelling takes place. Time-dependent travel times have been studied in the vehicle routing domain (Malandraki and Daskin, 1992; Hill and Benton, 1992; Ichoua et al., 2003; Fleischmann et al., 2004; Eglese et al., 2006; Maden et al., 2010; Huang et al., 2017). Moreover, time-dependent service times have been studied for ARPs (Tagmouti et al., 2010) and for CaPARPs (Tagmouti et al., 2011).

Time-dependent PARPs (TD-PARPs) have been proposed by Black et al. (2013). They modelled TD-PARPs using mixed integer linear programming (MILP), and then proposed a variable neighbourhood search (VNS) algorithm to solve the problem. They compared their VNS algorithm with LANTIME algorithm (Maden et al., 2010) that was originally proposed for time-dependent vehicle routing problems. Later, Vincent and Lin (2015) proposed an iterated greedy (IG) search algorithm to solve TD-PARPs. The IG algorithm has a destruction-construction method in which a number of arcs wih prizes are removed, and then reinserted into their best possible positions. The IG algorithm performs significantly better than the VNS and the LANTIME algorithm.

In this paper, we propose four constructive heuristics for TD-PARPs. Considering the NP-hard nature of the TD-PARPs, it is understandable that researchers mainly focus on designing metaheuristic search algorithms to solve the problem. However, constructive heuristics are still important because they are fast and easy to implement for practical applications. To our best knowledge, this paper is the first to propose constructive heuristics for TD-PARPs. The proposed heuristics range from a very simple procedure that provides quite good results in a fraction of a CPU second to a more extensive procedure that provides high-quality results but at the expense of slightly extra CPU time. In this paper, we also propose a very effective Scatter Search (SS) algorithm for TD-PARPs. The main concept of an SS algorithm is to keep a balance between diversification and intensification by retaining a set of good quality solutions and a set of diverse solutions, and then to combine them intelligently to reach better solutions (Sánchez-Oro et al., 2015). To ensure the diversity among the potential diverse solutions, we propose a new distance measurement that captures the characteristics of the problem. The proposed SS algorithm also benefits from a specialised *2-level path relinking procedure* which is a combination method to explore visited and unvisited arcs using two different operators. The proposed SS algorithm is further armed with a *multi-operator algorithm*, which is an improvement method as it helps SS explore more areas around a solution and escape from strong local optima. The experimental and statistical results indicate that the proposed SS algorithm is better than the existing state-of-the-art algorithms with the mean results at least 1% better than those obtained by other methods compared.

The rest of the paper is organised as follows: Section 2 provides formal definitions of TD-PARPs; Section 3 describes our proposed constructive heuristic algorithms for TD-PARPs; Section 4 presents in detail our proposed SS algorithm for TD-PARPs; Section 5 reports our experimental results, and finally, Section 6 concludes this paper.

## 2. Problem definition

We provide the exact TD-PARP definition used in this paper. A TD-

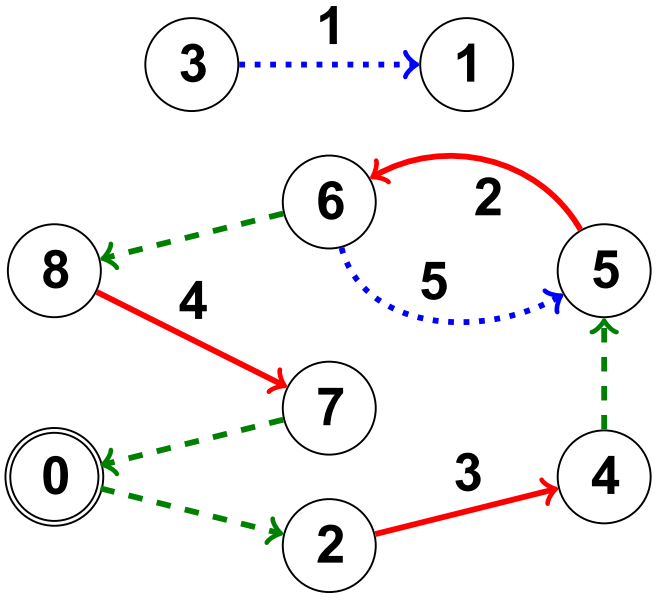


**Fig. 1.** A TD-PARP example having (i) nodes $0, \ldots, 8$ in circles, (ii) prize arcs with labels $1, \ldots, 5$ over red solid or dotted blue lines, (iii) non-prize arcs with green dashed lines without any labels, and (*iv*) a travelling route through non-prized dashed green arcs and selected prized red arcs starting and ending at double circled depot node 0. The collectable prizes for the red solid or blue dotted prize arcs have not been shown in the figure. Moreover, neither the arcs between pairs of nodes nor their travelling costs have been shown in the figure.

PARP is a directed graph $G = \langle V, A\rangle$ where $V$ is a set of $|V|$ nodes and $A$ is a set of $|A|$ arcs. The arcs represent the demand customers and the nodes represent the pickup and delivery locations for the demand customers. As such, each arc $k \in A$ has an origin node $i_k \in V$ and a destination node $j_k$, and so can also be denoted by $\langle i_k, j_k\rangle$. There is a set $P \subseteq A$ of $|P|$ non-loop arcs such that each $k \in P$ has a prize $p_k > 0$. These are *prize arcs* and can be visited more than once but the prize is collected only one time. The travelling cost of an arc $\langle i_k, j_k\rangle$ is defined by a time-dependent function $c(\langle i_k, j_k\rangle, t) > 0$, where $t$ is the time when the travelling takes place from the origin node $i_k$. Essentially $G$ has arcs between all pairs of nodes but the self-loops have cost 0. Note that, in this paper, the travelling times and the travelling costs are considered the same. So $c(\langle i_k, j_k\rangle, t)$ essentially gives the travelling time needed to travel through the arc $\langle i_k, j_k\rangle$ starting at time $t$ from $i_k$. Also, note that for convenience of modelling, $c(\langle i_k, j_k\rangle, t)$ and $p_k$ are in the same units. Ideally, there would be some problem-dependent transformation functions to bring cost, time, and prize in the same unit. However, those transformation functions are left for the problem modeller. In the benchmark datasets, they are provided in the same unit. For time-dependent travel times, $c(\langle i_k, j_k\rangle, t)$ function must maintain the First In Out First (FIFO) property (Eglese et al., 2006). The FIFO property states that if two identical vehicles (could be the same vehicle, too, but under two different potential travelling conditions) travel along an arc $k$ but starting from node $i_k$ at two different times $t_i < t_i^{'}$ respectively, then they will reach node $j_k$ at times $t_j < t_j^{'}$ respectively, where $t_j = t_i + c(\langle i_k, j_k\rangle, t_i)$ and $t_j^{'} = t_i^{'} + c(\langle i_k, j_k\rangle, t_i^{'})$ (Ichoua et al., 2003). Note that the benchmark instances used in this work are generated by Black et al. (2013), who essentially have maintained the FIFO property as they have defined the $c(\langle i_k, j_k\rangle, t)$ function for each instance.

While a TD-PARP in general might have multiple vehicles to serve the demand customers, in this paper, we assume there is just one vehicle, which can serve at most one demand customer at a time. We also assume the travelling route in a TD-PARP always starts and ends at the same designated depot node identified by node 0, no prize arc starts or ends at node 0, and there is no waiting time at any node. Without loss of generality, we further assume the travelling route will start at time 0 and must be completed within a given time limit $T$. This is called the *route duration constraint*. The aim in TD-PARP is to find a travelling route that maximises the total profit i.e. total collected prizes minus the total travelling costs.

Fig. 1 shows an example TD-PARP. The travelling route starts from the depot node 0 and includes prize arcs 3,2, and 4 denoted respectively by $\langle 2,4\rangle, \langle 5,6\rangle$, and $\langle 8,7\rangle$. To complete the route, non-prize arcs $\langle 0,2\rangle, \langle 4,5\rangle, \langle 6,8\rangle$ and $\langle 7,0\rangle$ have also been visited. There are two other prize arcs 1 and 5 denoted respectively by $\langle 3,1\rangle$ and $\langle 6,5\rangle$ and these arcs have not been included in the travelling route. To avoid cluttering of the figure, the time-dependent travelling costs have not been shown in the figure, but they exist for each pair of nodes.

We represent a TD-PARP solution $S$ by a sequence $\langle s_1, s_2, \ldots, s_{|S|}\rangle$ of $|S|$ prize arcs i.e. $s_l \in P$ for each $l \in [1, |S|]$. For convenience, we denote $s_l$ by $[l]$ for a given $S$ or by $[l]_S$. This representation essentially implies a travelling route via nodes $0, i_{[1]}, j_{[1]}, i_{[2]}, j_{[2]}, \ldots, i_{[|S|]}, j_{[|S|]}, 0$ in sequence. So the travelling route includes non-prize arcs $\langle 0, i_{[1]}\rangle, \langle j_{[1]}, i_{[2]}\rangle, \ldots, \langle j_{[|S|]}, 0\rangle$. For a given solution $S$, by $\overline{S}$, we denote the set of $\left|\overline{S}\right|$ prize arcs that are not in $S$. Given a TD-PARP solution $S$, for each selected arc $[l]$, we compute pickup time $\mathrm{PT}[l]$ and delivery time $\mathrm{DT}[l]$ using (1)–(3). We then compute total travelling cost (TTC), total collected prize (TCP), and total profit (TP) using (4)–(6) respectively. Using these equations, computing the total profit is at most $\mathcal{O}(|P|)$ as each prize arc can appear at most once in a solution $S$ and there are $|P|$ prize arcs at most. The route duration constraint is provided by (7). We use $TP(S)$ to denote the total profit of a solution $S$.

$$\mathrm{PT}[1] = c(\langle 0, i_{[1]}\rangle, 0) \quad l = 1 \tag{1}$$

$$\mathrm{PT}[l] = \mathrm{DT}[l-1] + c(\langle j_{[l-1]}, i_{[l]}\rangle, \mathrm{DT}[l-1]) \quad 2 \leqslant l \leqslant |S| \tag{2}$$

$$\mathrm{DT}[l] = \mathrm{PT}[l] + c([l], \mathrm{PT}[l]) \quad 1 \leqslant l \leqslant |S| \tag{3}$$

$$\mathrm{TTC} = \mathrm{DT}[|S|] + c(\langle j_{[|l|]}, 0\rangle, \mathrm{DT}[|S|]) \tag{4}$$

$$\mathrm{TCP} = \sum_{k=1}^{|S|} p_{[k]} \tag{5}$$

$$\mathrm{TP} = \mathrm{TCP} - \mathrm{TTC} \tag{6}$$

$$\mathrm{TTC} \leqslant T \quad \text{route duration constraint} \tag{7}$$

## 3. Proposed Constructive Heuristic Algorithms

We propose four constructive heuristic algorithms for TD-PARPs. Algorithm 1 shows the procedure of one of them. The other three are simplified versions of that and are not shown.

### 3.1. Highest Prize-based Constructive (HPC) Heuristic

Because of the nature of the objective function, HPC heuristic prioritises and selects the arcs with the highest prizes. Starting from an empty solution $S$, until the route duration constraint is violated, HPC keeps removing the highest prize arc from $\overline{S}$ and adding to the end of $S$. At the worst case, HPC orders all prize arcs in $P$. Thus, its complexity is $\mathcal{O}(|P|\log|P|)$.

### 3.2. Maximum Profit-based Constructive (MPC) Heuristic

MPC also starts from an empty solution $S$, and within a loop, adds one prize arc at a time, until the route duration constraint is violated. In this process, at step $l$, all arcs in $\overline{S}$ are examined for the $l$th position of $S$ and the one that leads to the highest total profit is selected. At each step, the objective function TP can be calculated in $\mathcal{O}(1)$ time, since the travel times of already planned arcs in $S$ need not be calculated again. At the worst case, all prize arcs in $P$, which will be all in $\overline{S}$ initially, are added to $S$. So the time complexity is $\mathcal{O}(\left|P\right|^2)$.

### 3.3. Exploration-based Constructive (EC) Heuristic

EC heuristic is an extension of MPC heuristic. Starting from an empty solution $S$, like MPC, at each step $l$, EC examines all arcs $k \in \overline{S}$ for the $l$th position in $S$ and selects the ark $k^{'}$ that leads to the highest total profit. However, unlike MPC, within each step of the procedure, EC further attempts to relocate the selected arc $k^{'}$ within $S$ to a position $l^{'}$ that leads to the highest total profit. Since each of the two loops runs for $\mathcal{O}(|P|)$ times and the objective computation is $\mathcal{O}(|P|)$, EC heuristic has a time complexity of $\mathcal{O}(\left|P\right|^3)$.

### 3.4. Exploration-based constructive 2 (EC2) Heuristic

This heuristic is a combination of MPC and EC heuristics. In EC2, at each step $l$, all arcs in $\overline{S}$ are examined separately as candidates at all positions $l^{'}$ in $S$ to produce candidate solutions. Among all the candidate solutions generated, the best solution $S^{'}$ is selected and becomes the current solution $S$ for the next step. EC has three nested loops, each having time complexity $\mathcal{O}(|P|)$ and the objective computation is also $\mathcal{O}(|P|)$. So EC2 has a time complexity of $\mathcal{O}(\left|P\right|^4)$.

**Algorithm 1**. Exploration-based Constructive 2 (EC2) Heuristic

1: $S \leftarrow \emptyset, \overline{S} \leftarrow P$ // *S is empty solution, and* $\overline{S}$ *holds prize arcs*

(*continued on next page*)

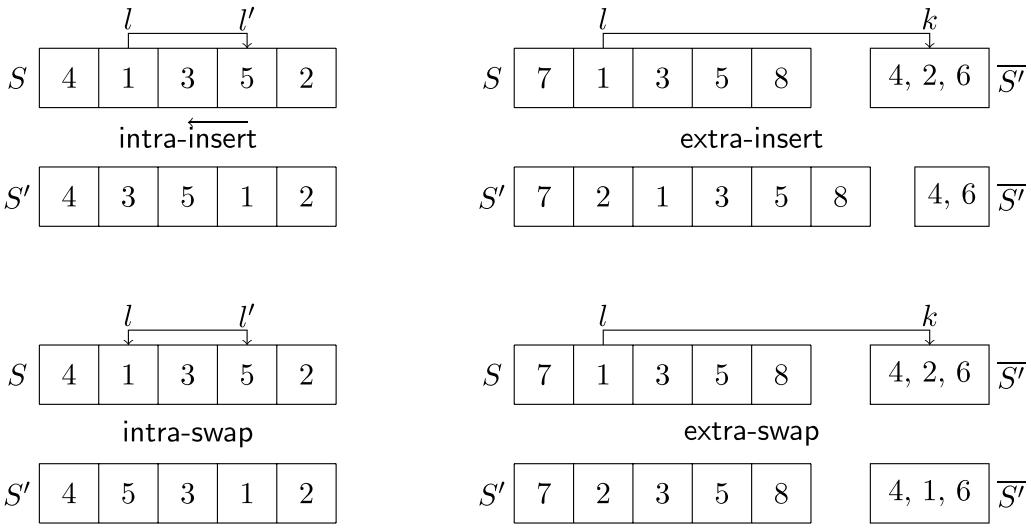


**Fig. 2.** One solution created by each neighbourhood operator. Actually, a set of solutions is created by each operator considering all possible values of $l$, $l'$, and $k$.

(*continued*)

2: **for** $l = 1$ to $|\overline{S}|$ **do** // *lth arc to be added to S*
3: $S' = \emptyset$ with TP($S'$) $= -\infty, k' \leftarrow \emptyset$ // *S′ is best solution after adding arc k′*
4: **for** $k \in \overline{S}$ **do** // *for each arc in* $\overline{S}$
5: **for** $l' = l$ to 1 **do** // for each position in $S$
6: $S'' \leftarrow$ insert $k$ in $l'$th position in $S$ shifting arcs at positions $k$
7: and onward towards the end
8: **if** TP($S''$) > TP($S'$) and route duration constraint not violated by $S''$ **then**
9: $S' \leftarrow S'', k' = k$ // *hold the best solution and the arc*
10: **if** $k'$ is $\emptyset$ **then** // *if better not found*
11: **break**
12: $S \leftarrow S', \overline{S} \leftarrow$ remove $k'$ from $\overline{S}$
**return** $S$

## 4. Proposed Scatter Search Algorithm

SS algorithms (Glover et al., 2000) are evolutionary meta-heuristic algorithms that are used in solving many optimisation problems that include nurse rostering problem (Burke et al., 2010), flowshop scheduling (Riahi et al., 2017), facility location problem (Hakli and Ortacay, 2019), and vehicle routing problem (Soman and Patil, 2020). SS strives to reach good quality solutions by keeping a balance between diversification and intensification, thus steering away from the randomness of other evolutionary algorithms. The basic procedure of SS is in Algorithm 2. First, a number of initial solutions are generated. These solutions undergo improvements. Then, two reference sets are created from the improved solutions. One reference set stores the high quality solutions while the other one stores diverse solutions based on a given distance measurement. Next, within a loop, for each high quality solution, for each diverse solution, a combined solution is created, the combined solution is improved and stored in a temporary set of solutions. Then, the two reference sets are updated using the temporary set. Lastly, the best solution found so far is returned. Although SS has a generic procedure for solving optimisation problems, the main steps shown in bold in Algorithm 2 are problem-specific, and thus are designed for the problem at hand in this paper. The comments in Algorithm 2 show where we have proposed new methods. In the rest of this section, we describe the proposed methods for each step of SS.

**Algorithm 2**. Proposed SS algorithm

1: **Generate an initial population** PopSet. // a constructive heuristic + random construction
2: Apply $S \leftarrow$ **ImprovementMethod**($S$) to each $S$ in PopSet. // a multi-operator algorithm
3: Create two reference sets of solutions:

(*continued on next column*)

(*continued*)

QualitySet ← $b_1$ **high quality solutions** from Pop after improvement.
DiverseSet ← $b_2$ diverse solutions using a **distance measurement**. // new measurement
4: **while** stopping criteria not satisfied **do**
5: TempSet ← an empty list of solutions
6: **for** each $S_q \in$ QualitySet **do** // *subset generation*
7: **for** each $S_d \in$ DiverseSet **do** // *subset generation*
8: $S_c \leftarrow$ **CombinationMethod**($S_q, S_d$) // *2-level path relinking*
9: $S_i \leftarrow$ **ImprovementMethod**($S_c$) //*a multi-operator algorithm*
10: Add solution $S_i$ to the list TempSet of solutions.
11: **UpdateReferenceSets**(QualitySet, DiverseSet, TempSet)
12: **return** the best solution found so far.

### 4.1. Initial population

We have proposed four constructive heuristics in Section 3. These heuristics slightly differ from each other. Moreover, SS is sensitive when solutions are not diverse. So in this paper, we do not use more than one constructive heuristic at the same time. However, we do create versions of our proposed algorithm using each constructive heuristic separately. Nevertheless, our proposed heuristics are deterministic algorithms. So using the selected heuristic, we just generate one solution. The rest of the solutions to be in the initial population are generated by choosing random prize arcs from $P$ and ensuring the route duration constraint is satisfied.

### 4.2. Improvement method

The improvement method is applied on the initial solutions and also on the combined solutions. In this paper, we propose a multi-operator improvement method (MOIM). In any call of the improvement method, MOIM picks randomly one of the proposed neighbourhood operators and applies it to the given solution. We use the following neighbourhood operators:

- intra-insert($S$): For each position $l \in \{1, \ldots, |S|\}$ and for each position $l' \in \{1, \ldots, |S|\}$ such that $l \neq l'$, insert the arc $[l]$ at position $l'$ to get a new solution $S'$ shifting the arcs at the intermediate positions accordingly. This operator creates $|S|(|S|-1)$ new solutions. The best feasible solution in terms of the objective values is returned.
- intra-swap($S$): For each position $l \in \{1, \ldots, |S|-1\}$ and for each position $l' \in \{l+1, \ldots, |S|\}$, swap the arcs $[l]$ and $[l']$ in $S$ to get a new solution $S'$. This operator creates $|S|(|S|-1)/2$ new solutions. The best feasible solution in terms of the objective value is returned.

- extra-insert($S$): For each arc $k \in \overline{S}$ and for each position $l \in \{1,\ldots,|S|+1\}$, add arc $k$ to solution $S$ at position $l$ to get a new solution $S'$. This operator creates $\left|\overline{S}\right|(\left|S\right|+1)$ new solutions. The best feasible solution in terms of the objective value is returned.
- extra-swap($S$): For each arc $k \in \overline{S}$ and for each position $l \in \{1,\ldots,|S|\}$, swap arc $k$ with the arc at position $l$ in $S$ to get a new solution $S'$. This operator creates $\left|\overline{S}\right|\left|S\right|$ new solutions. The best feasible solution in terms of the objective value is returned.

The four operators have been depicted in Fig. 2. The operators might produce infeasible new solutions. We use repair strategies described in Section 4.4 to remove infeasibility from the solutions. As mentioned already, only the best solutions among the feasibles one are returned.

The procedure of the proposed improvement method is given in Algorithm 3. Based on the total profit, it returns the improved solution if possible, or the solution given to it for improvement.

**Algorithm 3**. Improvement Method

1: Let $S$ be solution to undergo improvement
2: selected-operator ← random(intra-insert, intra-swap, extra-insert, extra-swap)
3: $S' =$ selected $-$operator($S$)
4: **if** TP($S'$) > TP($S$) **then** $S \leftarrow S'$
5: **return** $S$ as the resultant solution.

### 4.3. Combination method

In this paper, we use the path relinking (PR) algorithm (Glover et al., 2000) as a combination method. The main idea of the PR method is to explore intermediate solutions between two given solutions. From a starting solution $S^s$, the PR method gradually moves towards the target solution $S^t$ using neighbourhood operators. We use the following two neighbourhood operators:

- inner-swap($S,l,l'$): swap the arcs at positions $l$ and $l'$ in $S$ to get a new solution $S'$. This operator is used in the first step of 2LPR.
- outer-swap($S,l,k$): an arc $k$ in $\overline{S}$ is swapped with the arc at position $l$ in $S$ to get a new solution $S'$. This operator is used in the second step of 2LPR.

In terms of solution generation style, operators inner-swap and outer-swap are like intra-swap and extra-swap respectively but the former two generate one solution each while the latter two generate a set of solutions each and returns the best solutions.

The two operators described above might produce infeasible new solutions. We use repair strategies described in Section 4.4 to remove infeasibility from the solutions.

There are many alternative paths to explore and move from $S^s$ towards $S^t$. We pick a random path to create new intermediate solutions. To be more specific, we propose a 2-level path relinking (2LPR) procedure that first relocates the arcs already in $S^s$ and then relocates the arcs needed to be moved from $\overline{S^s}$ to $S^s$. Algorithm 4 describes our the 2LPR procedure, which returns the intermediate solution that has the best total profit.

**Algorithm 4**. 2LPR algorithm

1: Let $S^s$ and $S^t$ be the starting and the target solutions respectively
2: $S^i \leftarrow S^s$ // *the intermediate solution starts from* $S^s$
3: $\mathscr{L} \leftarrow \emptyset$ // *a list of explored intermediate solutions*
4: **while** exists $l^i \neq l^t$ such that $[l^i]_{S^i} = [l^t]_{S^t}$ **do** // *first phase*
5: $l^i, l^t \leftarrow$ a random pair if multiple exists
6: $S^i \leftarrow$ inner $-$swap($S^i, l^i, l^t$)
7: add $S^i$ to $\mathscr{L}$
8: **while** exists $l^t$ such that not exists $l^i$ such that $[l^t]_{S^i} = [l^i_{S^t}$ **do** // *second phase*
9: $l^t \leftarrow$ a random one if multiple exists
10: $S^i \leftarrow$ outer $-$swap($S^i, l^t, [l^t]_{S^i}$)

(*continued on next column*)

(*continued*)

11: add $S^i$ to $\mathscr{L}$
12: **return** $S$ from $\mathscr{L}$ such that TP($S$) is the largest. // *select the best*

Consider an example with 10 prize arcs and $S^s = \langle 5,3,8,4,7\rangle$ and $S^t = \langle 7,1,10,8,6\rangle$. In the first step, 2LPR focuses on arcs 7 and 8 since these are in both the starting and target solutions. The arc 7 is at position 5 in $S^s$ and should be moved to position 1 as it is at that position in $S^t$. Similarly 8 is to be moved from position 3 to 4. Assume 7 is randomly selected for movement. Applying inner-swap($S^s, 5, 1$) , we get an intermediate solution $S^i = \langle 7,3,8,4,5\rangle$. In the intermediate solution $S^i$, we have identified again the arcs that are to be moved in order to match their position with the position in $S^t$. Clearly, 8 is the only candidate arc for moving and after the move by using the inner-swap operator, we get $S^i = \langle 7,3,4,8,5\rangle$. In the second step, 2LPR focuses on arcs that are in the target solution but not in the starting solution, e.g., arcs 1, 10 and 6. Arc 1 is at position 2 of $S^t$, while it is not in the starting solution. Similarly, arcs 10 and 6 are in positions 3 and 5 in $S^t$. Assume arc 10 is randomly chosen for movement. Applying outer-swap($S^i, 3, 10$), we get an intermediate solution $S^i = \langle 7,3,10,8,5\rangle$. Next, assume arc 6 is chosen. After applying outer-swap, we get $S^i = \langle 7,3,10,8,6\rangle$. We have to repeat this process until the second phase is finished.

After creating a sequence of intermediate solutions by 2LPR, we are to choose one of them as a combined solution and then use it for the improvement method. To avoid the chance of returning to a previously visited local optimum, we remove the first and the last intermediate solutions, and then the best intermediate solution among the remaining ones is selected as the combined solution.

### 4.4. Repair strategies

The neighbourhood operators used in both the improvement and the combination methods might produce infeasible solutions, i.e. the route duration constraint might be violated. To overcome this issue, we examine three different repair methods:

- **Last-Drop (LD) Method:** Keep dropping the last arc from a solution until the route duration constraint is satisfied. This is used in IG algorithms (Vincent and Lin, 2015).
- **Random-Drop (RD) Method:** Keep dropping a random arc from a solution until the route duration constraint is satisfied.
- **Best-Drop (BD) Method:** Until the route duration constraint is satisfied, keep dropping one selected arc at a time from a solution. The selection is made from all the solutions that can be produced by dropping just one arc and then taking the best solution produced in terms of total profit, although these solutions might still be infeasible.

As mentioned before, any of the above three methods are applied immediately after applying a neighbourhood operator within the improvement or the combination methods. So while considering arcs for dropping, we ignore the arcs that are involved in the application of the neighbourhood operators. For example, if using inner-swap($S,l,l'$) leads to an infeasible solution $S'$, the arcs $[l]$ and $[l']$ in $S'$ will not be dropped by a repair method. Note that when an arc is dropped from a new solution $S'$, it is added to $\overline{S'}$. Nevertheless, among the three repair methods, the first two are very simple, but the third method is computationally more expensive than the other two. This is because the first two methods just drop an arc, while the BD method selects the best arc to be dropped.

**Table 1**
The benchmark set. The numbers of arcs include self-loops. The numbers of prize arcs are followed by parentheses enclosed the number of instances having that number of arcs.

| Instance category | No. nodes | No arcs | No. prize arcs (No. instances) | Total No. instances |
|---|---|---|---|---|
| NW-A | 25 | 625 | 50 (2), 100 (2), 150 (1) | 5 |
| NW-B | 25 | 625 | 50 (2), 100 (2), 150 (1) | 5 |
| NW-C | 25 | 625 | 50 (2), 100 (2), 150 (1) | 5 |
| NW-D | 25 | 625 | 50 (2), 100 (1) | 3 |
| NW-E | 25 | 625 | 50 (2), 100 (1) | 3 |
| NW-F | 100 | 10000 | 300 (1), 400 (1), 500 (2), 600 (1) | 5 |
| London-B | 50 | 2500 | 75 (10) | 10 |
| London-L | 50 | 2500 | 350 (5) | 5 |

### 4.5. Reference sets and subset generation

The reference sets are one of the main aspects of the SS algorithm and helps the algorithm balance the diversification and the intensification in the search process. The reference sets maintain $b$ solutions that are divided in two groups: QualitySet that contains $b_1 = b/2$ best solutions in terms of the objective values and DiverseSet that contains $b_2 = b/2$ diverse solutions.

After generating the initial population of solutions and evaluating their objective values, $b_1$ solutions with the highest total profits are stored in QualitySet. To find $b_2$ diverse solutions for DiverseSet, we consider a typical distance measure for permutation-based representation (Guo and Tang, 2015; Riahi et al., 2017) defined as $\mathrm{Dist}(S, S')$ of two solutions $S$ and $S'$ as the number of arcs in $S$ that differ from $S'$ in either direction. Based on this measurement, if $[l]_S$ and $[l+1]_S$ neither match respectively with any $[l']_{S'}$ and $[l'+1]_{S'}$ nor do so with any $[l'+1]_{S'}$ and $[l']_{S'}$, then a mismatch is counted.

$$\mathrm{Dist}\left(S, S'\right) = \sum_{l=1}^{n-1} d\left([l]_S, [l+1]_S, S'\right) \text{ where}$$

$$d(k, k', S') = 0 \text{ if } \exists_{l'}\left(([l']_{S'} = k \Lambda [l'+1]_{S'} = k') \vee ([l'+1]_{S'} = k \Lambda [l']_{S'} = k')\right) \text{ else } 1$$

Assume a problem instance has 20 prize arcs labelled by $1, \ldots, 20$. Also, assume 4 solutions as $S_1 = \langle 5,6,13,18,20,7,11,15\rangle, S_2 = \langle 13,5,6,15,11,20,7,18\rangle, S_3 = \langle 18,15,20,5,13,11,6,7\rangle$, and $S_4 = \langle 19,4,17,16,8,2,12,1\rangle$. So $\mathrm{Dist}(S_1, S_2) = 4$ because the consecutive arc pairs $(6,13)$, $(13,18)$, $(18,20)$, and $(7,11)$ or their inverse pairs are in $S_1$ but not in $S_2$. Using the above definition, $\mathrm{Dist}(S_1, S_3) = 7$ and $\mathrm{Dist}(S_1, S_4) = 7$ because each pair in $S_1$ is neither in $S_3$ nor in $S_4$. However, we notice that each arc in $S_1$ is also in $S_3$ but not in $S_4$. So one should perhaps expect that $\mathrm{Dist}(S_1, S_4)$ be larger than $\mathrm{Dist}(S_1, S_3)$. Clearly, this is a drawback of the above definition.

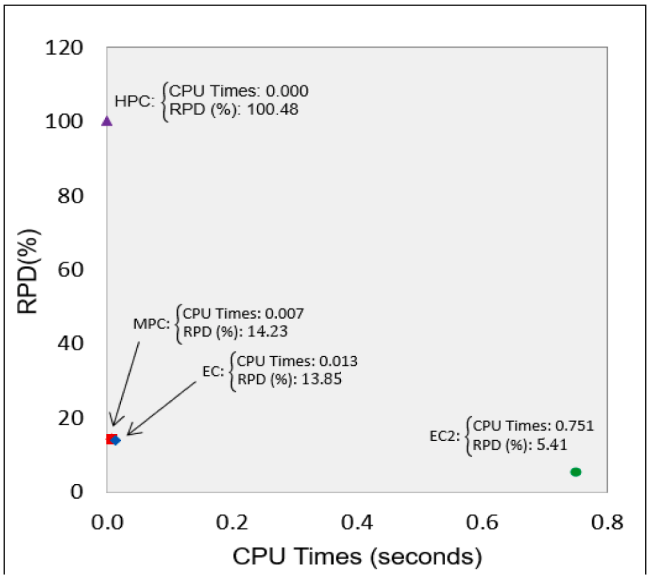


**Fig. 3.** ARPDs versus CPU times for the proposed constructive heuristics.

To address the above mentioned drawback, we propose a new distance measurement $\widehat{\mathrm{Dist}}(S, S')$ that includes the number of arcs that are in $S$ but in $S'$. The definition is as follows:

$$\widehat{\mathrm{Dist}}\left(S, S'\right) = \sum_{l=1}^{n-1} d\left([l]_S, [l+1]_S, S'\right) + \sum_{l=1}^{n} d'\left([l]_S, S'\right) \text{ where} d(k, k', S')$$

$$= 0 \text{ if } \exists_{l'}\left([l']_{S'} = k \Lambda [l'+1]_{S'} = k') \vee ([l'+1]_{S'} = k \Lambda [l']_{S'} = k')\right) \text{ else} 1,\ d'(k, S') = 0 \text{ if } \exists_{l'}\left([l']_{S'} = k\right) \text{ else } 1.$$

Based on the new measurement, $\widehat{\mathrm{Dist}}(S_1, S_3)$ is still 7 while $\widehat{\mathrm{Dist}}(S_1, S_4)$ is 15 because each of the 8 arcs in $S_1$ is not in $S_4$. The $\widehat{\mathrm{Dist}}$ measurement lies between 0 and $2n-1$ and the larger values imply more diversity. Using $\widehat{\mathrm{Dist}}$, the solutions that are at least $n-1$ away from the best solution so far are allowed to be in DiverseSet. If there are not enough solutions with at least $n-1$ distance from the best solution so far, we generate new solutions randomly but satisfying the route duration

**Table 2**
The total profits and ARPDs of the proposed constructive heuristics. Note BKS stands for best known solution even after using our proposed SS algorithms. The bold numbers represent the best found solutions for each instance category.

| Instances | Total Profit | | | | ARPD(%) among Heuristics | | | | ARPD(%) w.r.t. BKS | |
|---|---|---|---|---|---|---|---|---|---|---|
| | HPC | MPC | EC | EC2 | HPC | MPC | EC | EC2 | EC | EC2 |
| NW-A | 736.924 | 1309.342 | 1309.342 | 1339.006 | 44.29 | 13.59 | 13.59 | **4.98** | 53.80 | 47.96 |
| NW-B | −268.106 | 184.366 | 184.366 | 193.488 | 285.05 | 14.16 | 14.16 | **11.70** | 63.55 | 58.91 |
| NW-C | 718.248 | 2465.944 | 2486.038 | 2303.188 | 72.67 | 2.19 | **0.94** | 9.31 | 33.03 | 38.92 |
| NW-D | 460.3967 | 812.5033 | 901.7967 | 738.3467 | 51.93 | 8.53 | **0.00** | 15.91 | 26.95 | 37.94 |
| NW-E | 456.4933 | 1036.753 | 1036.753 | 1148.007 | 65.14 | 12.27 | 12.27 | **2.14** | 38.52 | 32.56 |
| NW-F | 1115.346 | 2582.672 | 2504.67 | 2594.828 | 58.49 | 4.12 | 7.06 | **3.86** | 55.66 | 54.09 |
| London-B | 14.033 | 562.868 | 562.133 | 726.809 | 97.66 | 23.17 | 23.73 | **1.85** | 60.87 | 49.11 |
| London-L | 58.606 | 729.374 | 739.584 | 960.152 | 94.33 | 23.84 | 22.98 | **0.00** | 73.80 | 66.05 |
| Average | 358.4412 | 1159.389 | 1159.926 | 1216.597 | 100.05 | 14.23 | 13.85 | **5.41** | 53.76 | 49.57 |

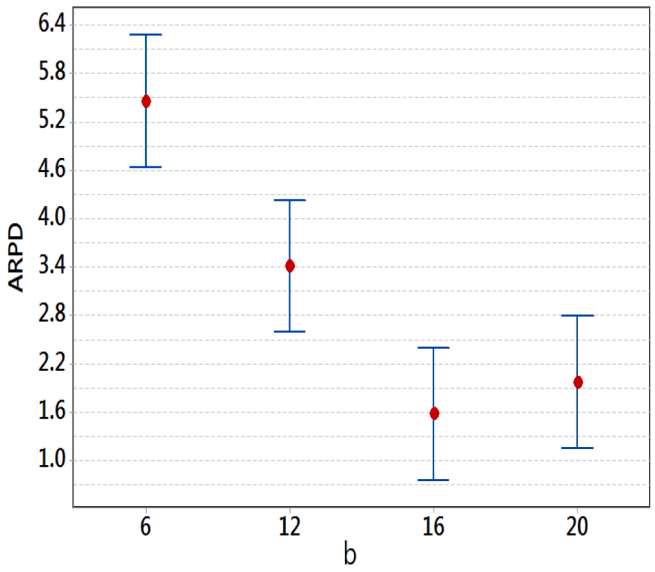


**Fig. 4.** ARPD and 95% HSD confidence intervals for different values of parameter $b$.

constraint to obtain $b_2$ solutions in total in DiverseSet.

For subset generation, the proposed SS considers all possible combinations of solutions in QualitySet with the solutions in DiverseSet. For example, assume that $b = 6$ so that $b_1 = b_2 = 3$, QualitySet $= \{S_1, S_2, S_3\}$ and DiverseSet $= \{S_4, S_5, S_6\}$. So the proposed SS generates 9 subsets as $\{S_1, S_4\}, \{S_1, S_5\}, \{S_1, S_6\}, \{S_2, S_4\}, \{S_2, S_5\}, \{S_2, S_6\}, \{S_3, S_4\}, \{S_3, S_5\}, \{S_3, S_6\}$.

As shown in Algorithm 2, after subset generation in the proposed SS algorithm, the combination method produces a solution for each subset and the improvement method further improves the produced solution. The improved solution is then added to QualitySet if the improved solution is better than the worst solution in QualitySet. Since QualitySet always keeps $b_1$ solutions, when a new solution is added, the worst solution would be removed.

## 5. Computational experiments

We evaluate the performance of the proposed heuristics and the SS algorithm. All methods are implemented in C programming language and are compiled on a computer with an Intel Core 2 Duo processor, operating at a frequency of 2.5 GHz with 3.58 GB RAM, running on Windows 7. The experiments are conducted across benchmarks of common use for the TD-PARPs. The details of the benchmark instances used for comparison are given in Section 5.1. In Section 5.2, we compare the proposed constructive heuristics in terms of both objective values and the required computational times. Next, a parameter tuning experiment has been done to find the best possible values for the SS parameters and the results are reported in Section 5.3. In Section 5.4, we analyse the impact using the proposed multi-operator improvement method, the new distance measurement, the repair strategies, and various heuristics in the initial population. Finally, we compare the proposed SS algorithm with the state-of-the-art algorithms for the TD-PARPs in Section 5.5.

In each subsection, after running the experiments, we compute the relative percentage deviation (RPD) as $(TP^{ref} - TP^{algo})/TP^{ref} * 100$, where $TP^{algo}$ is the total profit value obtained by a given run for a given instance, and $TP^{ref}$ is the reference total profit value. Since the SS algorithm is a stochastic algorithm, we run each SS variant on each instance 10 times and calculate the average RPDs (ARPDs) over the runs and over the instances.

### 5.1. Benchmark instances

In this paper, the 41 instances proposed by Black et al. (2013) are used to evaluate the proposed heuristics and the SS algorithm. This benchmark set is generated based on two regions of England: the north-west of England (denoted by NW) with 26 instances, and the London area (denoted by London) with 15 instances. The NW data is divided into 6 different categories A, B, C, D, E, and F, and the London data is divided into 2 categories B and L. The number of prize arcs are varied from 50 to 600. Based on the number of prize arcs, NW-F and London-L are considered as the hard instances. These datasets include time-dependent travel time information for each 15-min interval over 24 h for each pair of nodes to travel from one node to another. Table 1 provides a detailed description of the instances based on the instance category, the number of nodes, the number of arcs, the number of prize arcs, and the total numbers of instances in the groups.

**Table 3**
The ARPD results of the eleven SS variants. The bold numbers represent the best found solutions for each instance category.

| Instance | ◇Base | ◇EC | ◇MPC | ◇HPC | ◇II | ◇EI | ◇IS | ◇ES | ◇D | ◇LD | ◇BD |
|---|---|---|---|---|---|---|---|---|---|---|---|
| NW-A | **0.132** | 0.133 | 0.134 | 0.135 | 0.214 | 0.190 | 0.201 | 0.195 | 0.154 | 0.174 | 0.168 |
| NW-B | **0.048** | 0.050 | 0.052 | 0.048 | 0.068 | 0.074 | 0.070 | 0.069 | 0.053 | 0.055 | 0.068 |
| NW-C | **0.740** | 0.745 | 0.743 | 0.790 | 1.052 | 1.063 | 1.078 | 1.072 | 0.866 | 0.971 | 0.909 |
| NW-F | **1.814** | 1.816 | 1.817 | 2.025 | 2.564 | 2.596 | 2.571 | 2.633 | 2.104 | 2.213 | 2.344 |
| London-B | **0.266** | 0.268 | 0.271 | 0.287 | 0.381 | 0.383 | 0.382 | 0.393 | 0.301 | 0.324 | 0.322 |
| London-L | **1.204** | 1.205 | 1.209 | 1.262 | 1.719 | 1.721 | 1.718 | 1.773 | 1.385 | 1.419 | 1.582 |
| Average | **0.639** | 0.641 | 0.642 | 0.690 | 0.911 | 0.914 | 0.917 | 0.933 | 0.738 | 0.783 | 0.817 |

**Table 4**
The average ranking of each SS variant obtained by the Friedman test.

| Algorithm | Ranking |
|---|---|
| ◇Base | **2.929** |
| ◇EC | 3.214 |
| ◇MPC | 3.243 |
| ◇HPC | 4.671 |
| ◇D | 5.514 |
| ◇LD | 6.000 |
| ◇BD | 6.400 |
| ◇II | 8.071 |
| ◇EI | 8.200 |
| ◇IS | 8.457 |
| ◇ES | 9.300 |

**Table 5**
The Nemenyi test results. The bold numbers represent the best found solutions for each instance category.

| Control algorithm | Others | p-value |
|---|---|---|
| ◇Base | ◇EC | 1.000 |
| | ◇MPC | 1.000 |
| | ◇HPC | 0.905 |
| | ◇D | 0.469 |
| | ◇LD | 0.265 |
| | ◇BD | 0.159 |
| | ◇II | **0.019** |
| | ◇EI | **0.016** |
| | ◇IS | **0.012** |
| | ◇ES | **0.005** |

### 5.2. Comparing constructive heuristics

In this section, we compare the performance of the proposed constructive heuristics. The comparison is done on 41 instances explained in Section 5.1. Table 2 reports the results of the heuristics in terms of both the total profit as the objective function and the ARPD for each instance category considering the best solution found by the heuristics as $TP^{ref}$ (see column ARPD(%) among Heuristics). As can be seen, the EC2 obtains the best results for 6 instance categories out of 8, while for the other two categories, the best results are obtained by EC. An interesting note is that the total profit obtained by HPC is less than zero for instance category NW-B while traversing no prize arc leads to a total profit of zero. This point indicates putting attention only on prize arcs and ignoring the required travel times is not a decent approach. Nevertheless, a paired t-test with a significance level $\alpha = 0.05$ is carried out on the ARPD of heuristics. The results show a significant difference between EC2 and EC ($\text{p-value} = 0.022 < 0.05$).

Since the proposed heuristics have different complexities, we also compared them in terms of CPU times. The ARPDs versus the average computational times for the compared heuristics are shown in Fig. 3. Except for the EC2 that spent 0.75 s on average, the other three heuristics spent less than 0.014 s. Given 1 min and 10 min computational times that are used as the cutoff for the search algorithms, these running times are negligible. Although the use of intensive exploration in EC2 increases the required CPU times, it also yields significantly better results.

To see how the constructive heuristics perform compared to the best known solutions (BKS), we compute ARPDs for EC and EC2 w.r.t. the BKS and show them in the last two columns of Table 2. The BKS are obtained by running any known algorithm for TD-PARPs with 10 min timeout and the known algorithms include our proposed SS variants. The heuristics achieve results with about 50% ARPDs, but only take a fraction of a second compared to the 10 min allowed for the metaheuristics algorithms.

### 5.3. SS parameter tuning

Before conducting the main experiments, proper values should be found for the SS parameters. The proposed SS has two parameters: the size of the initial population PopSet, and the size of the reference set $b$. In this paper, we just focus on parameter $b$, and consider $\text{PopSet} = 100$. The reason is that, unlike other evolutionary algorithms like genetic algorithms, the SS algorithm mainly focuses on solutions stored in the reference set instead of the whole generated population. Moreover, based on the procedure of the proposed SS algorithm, to find $b_2$ diverse solutions for DiverseSet, if there are not enough solutions that can satisfy the predetermined diversity conditions, SS generates new solutions randomly to find such solutions. Therefore, the SS algorithm does not only rely on the initial or existing population and would generate more random solutions if needed. Our initial experiments with other PopSet sizes 50 and 150 also confirms that point. For $b$, we consider four values $\{6, 12, 16, 20\}$.

For the tuning of the parameter $b$, from each instance category, one instance is selected randomly. Then, each such instance is tested 10 times with a given value of $b$. Ten minutes of CPU time per run is used as the stopping criterion. Also, ARPD is calculated where $TP^{ref}$ is the best total profit found by the SS algorithms in this section. Fig. 4 shows the interval plots considering 95% confidence interval with Tukey's Honest Significant Difference (HSD). From the figure, SS with $b = 6$ obtains the worst results, and the performance of the SS improves with the increase of $b$, but until $b = 16$. The possible reason is the more the size of $b$, the less the number of reference set updates, which reduces the chance of escaping from local optima. Although $b = 16$ and $b = 20$ are not statistically different, we select 16 for parameter $b$ because it has a lower average error.

### 5.4. Analysis of the proposed SS components

This experiment evaluates the effectiveness of the proposed SS components. We evaluate our initial populations using proposed heuristics, the multi-operator algorithm in the improvement method, the newly proposed distance measurement, and the repair strategies. To that end, we create 10 further versions of the proposed SS algorithm. All SS variants are described below:

- sep0ex
- ◇Base: Using EC2 (Algorithm 1) in the initial population, multiple neighbourhood operators (Algorithm 3) as the improvement method, the new distance measurement $\widehat{\text{Dist}}(S, S')$ in the reference set update, and the RD method as the repair method.
- ◇EC: Using EC instead of EC2 in the initial population of ◇Base
- ◇MPC: Using MPC instead of EC2 in the initial population of ◇Base
- ◇HPC: Using HPC instead of EC2 in the initial population of ◇Base
- ◇II: Using only intra-insert operator in the improvement method of ◇Base

**Table 6**
The ARPD of competing algorithms for both 1 min and 10 min computational times. The bold numbers represent the best found solutions for each instance category.

| CPU | Instance | LANTIME | | | VNS | | | IG | | | IG+LS1 | | | IG+LS2 | | | proposed SS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| times | category | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min |
| 1 min | NW-A | 10.15 | 4.65 | 14.83 | 6.16 | 2.75 | 10.06 | 2.06 | 0.07 | 4.34 | 0.63 | 0.09 | 1.54 | 0.93 | 0.40 | 2.23 | **0.59** | 0.13 | 1.34 |
| | NW-B | 9.18 | 4.89 | 13.69 | 6.30 | 2.22 | 9.83 | 2.33 | 0.47 | 5.23 | 0.40 | 0.00 | 1.41 | 0.96 | 0.10 | 2.33 | **0.35** | 0.00 | 1.01 |
| | NW-C | 12.69 | 8.92 | 17.77 | 11.41 | 8.32 | 15.66 | 6.38 | 2.46 | 11.60 | 3.73 | 0.70 | 7.87 | 4.87 | 2.24 | 7.87 | **2.18** | 1.30 | 2.88 |
| | NW-D | 3.74 | 1.23 | 7.11 | 0.44 | 0.00 | 0.74 | 0.00 | 0.00 | 0.00 | 0.02 | 0.00 | 0.74 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | NW-E | 4.50 | 2.17 | 7.72 | 0.87 | 0.00 | 1.62 | 0.24 | 0.00 | 0.64 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | NW-F | 26.36 | 19.15 | 33.02 | 15.07 | 9.01 | 23.34 | 7.93 | 5.22 | 10.94 | 11.72 | 7.13 | 15.93 | 9.07 | 5.61 | 11.60 | **6.11** | 3.83 | 8.24 |
| | London-B | 11.98 | 8.46 | 15.55 | 7.33 | 2.78 | 15.70 | 2.41 | 0.97 | 5.08 | 1.24 | 0.26 | 2.26 | 1.54 | 0.75 | 2.29 | **1.12** | 0.42 | 1.71 |
| | London-L | 42.05 | 21.20 | 62.15 | 36.38 | 16.87 | 54.19 | 11.08 | 7.48 | 16.97 | 4.64 | 1.94 | 10.36 | 6.18 | 2.85 | 12.23 | **3.26** | 1.85 | 4.51 |
| | Average | 15.08 | 8.83 | 21.48 | 10.49 | 5.24 | 16.39 | 4.05 | 2.09 | 6.85 | 2.80 | 1.27 | 5.01 | 2.95 | 1.49 | 4.82 | **1.70** | 0.94 | 2.46 |
| 10 min | NW-A | 6.15 | 3.55 | 8.45 | 4.58 | 2.01 | 8.67 | 0.57 | 0.07 | 1.21 | 0.29 | 0.09 | 0.48 | 0.26 | 0.09 | 0.54 | **0.13** | 0.00 | 0.26 |
| | NW-B | 3.56 | 1.55 | 5.64 | 4.38 | 0.84 | 8.14 | 0.51 | 0.06 | 1.17 | **0.00** | 0.00 | 0.00 | 0.08 | 0.00 | 0.35 | 0.05 | 0.00 | 0.14 |
| | NW-C | 8.25 | 5.23 | 10.96 | 11.27 | 6.59 | 14.01 | 3.48 | 1.30 | 7.29 | 1.57 | 0.33 | 3.81 | 2.43 | 0.50 | 4.95 | **0.74** | 0.00 | 1.42 |
| | NW-D | 0.58 | 0.00 | 1.23 | 0.37 | 0.00 | 0.74 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | NW-E | 2.37 | 1.21 | 3.22 | 0.35 | 0.00 | 0.79 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | NW-F | 26.36 | 19.15 | 33.02 | 10.31 | 2.40 | 20.22 | 4.53 | 3.09 | 5.91 | 8.00 | 5.56 | 9.95 | 5.32 | 3.31 | 7.56 | **1.81** | 0.00 | 3.29 |
| | London-B | 8.99 | 6.32 | 11.03 | 7.16 | 2.80 | 14.15 | 0.93 | 0.31 | 1.75 | 0.52 | 0.11 | 0.91 | 0.73 | 0.36 | 1.08 | **0.27** | 0.00 | 0.60 |
| | London-L | 27.49 | 16.67 | 38.31 | 33.21 | 17.67 | 49.59 | 6.27 | 5.14 | 7.65 | 1.95 | 0.90 | 2.92 | 1.83 | 0.89 | 2.95 | **1.21** | 0.00 | 2.13 |
| | Average | 10.47 | 6.71 | 13.98 | 8.95 | 4.04 | 14.54 | 2.04 | 1.25 | 3.12 | 1.54 | 0.87 | 2.26 | 1.33 | 0.64 | 2.18 | **0.53** | 0.00 | 0.98 |

- ◇EI: Using only extra-insert operator in the improvement method of ◇Base
- ◇IS: Using only intra-swap operator in the improvement method of ◇Base
- ◇ES: Using only extra-swap operator in the improvement method of ◇Base
- ◇D: Using $\mathrm{Dist}(S_s, S_{s'})$ in the reference set update of ◇Base
- ◇LD: Using LD method as the repair strategy of ◇Base
- ◇BD: Using BD method as the repair strategy of ◇Base

In this experiment, we use 35 instances described in Section 5.1 excluding NW-D and NW-E instances. We did not include the NW-D and NW-E in these experiments since all SS variants could obtain the best found solutions in all runs for these two categories. We use 10 min as the timeout. Table 3 shows the ARPD results over 6 instance categories. The best results for each category are emboldened. Before making observations from Table 3, we conduct Friedman statistical test at 95% confidence interval (significance level of $\alpha = 0.05$). The p-value of Friedman test is 0.000 ($< 0.05$) indicating that there is a significant difference between at least two SS variants. Then, a post hoc test is used to calculate the p-value of each comparison between the control algorithm (the best-performing SS variant) and the rest of the SS variants. First, to find the control algorithm, Table 4 reports the average ranking (the lower the better) of each method obtained by the Friedman test. Based on this table, ◇Base version, which is our main version, has the best ranking followed by ◇EC, ◇MPC, ◇HPC, ◇D, ◇LD, ◇BD, ◇II, ◇EI, ◇IS, and ◇ES in the given order. Next, we use Nemenyi test as a post hoc test to find out the p-values between ◇Base (the control algorithm) and other SS variants. The results are given in Table 5. In this table, a p-value less than the critical level 0.05 indicates a significant difference between ◇Base (the control algorithm) and the corresponding SS version. From the results reported in Tables 3–5, we can make the following observations:

- Comparison of the results of ◇Base with ◇EC, ◇MPC, and ◇HPC, shows that the proposed SS algorithm is independent of the constructive heuristic used in the initial population. Although better constructive heuristics lead to better solutions on average, their impact on the proposed SS algorithm is statistically equivalent as the p-values are larger than 0.05.
- Comparison of the results of ◇Base and ◇D shows the efficiency of the proposed distance measurement, $\widehat{Dist}(S_s, S_{s'})$. Although Table 5 reports no significant difference between these two versions, results taken from Table 3 show that the ◇Base obtains almost 0.1% lower ARPD than what ◇D does; which is remarkable. It emphasises the importance of having an effective method to keep a balance between quality and diversity in the proposed SS algorithm.
- Comparison of the results of ◇Base with ◇LD and ◇BD indicates the effectiveness of RD method as the repair strategy. Using RD method improves the results of the SS algorithm more than 0.15% in ARPD compared to those with LD and BD methods. As we discuss in Section 4.4, BD method is computationally more expensive than the other two methods. Knowing this helps understand better why BD unexpectedly leads to worse solutions compared to other two methods despite having a more greedy approach.
- Comparison of the results of ◇Base (using multiple neighbourhood operators) with those of ◇II, ◇EI, ◇IS, and ◇ES (using a single neighbourhood operator) demonstrates the efficiency of using the proposed multi-operator improvement method. From Table 3, ◇Base outperforms ◇II, ◇EI, ◇IS, and ◇ES in all instance categories. Table 5 confirms that the differences between ◇Base and all

**Table 7**
Computational results of 1 min computational time.

| Instance | LANTIME | | | VNS | | | IG | | | IG+LS1 | | | IG+LS2 | | | proposed SS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min |
| NW25-A1 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | 1244.34 |
| NW25-A2 | 2046.81 | 2168.18 | 1984.96 | 2130.6 | 2168.18 | 2046.87 | 2128.29 | 2168.18 | 2099.61 | 2168.18 | 2168.18 | 2168.18 | 2168.18 | 2168.18 | 2168.18 | 2168.18 | 2168.18 | 2168.18 |
| NW25-A3 | 2180.88 | 2278.84 | 2047.56 | 2311.5 | 2430.92 | 2200.11 | 2363.34 | 2443.04 | 2213.44 | 2439.86 | 2443.04 | 2430.92 | 2431.88 | 2443.04 | 2365.05 | 2443.04 | 2443.04 | 2443.04 |
| NW25-A4 | 2795.59 | 3120.74 | 2509.35 | 2970.18 | 3135.96 | 2785.18 | 3216.15 | 3289.99 | 3151.92 | 3276.62 | 3289.99 | 3211.16 | 3258.68 | 3289.99 | 3206.4 | 3264.801 | 3284.01 | 3209.85 |
| NW25-A5 | 3349.91 | 3682.9 | 3085.4 | 3575.84 | 3799.11 | 3350.18 | 4033.82 | 4141.66 | 3950.45 | 4048.07 | 4136.64 | 3956.55 | 4021.32 | 4073.15 | 3931.97 | 4066.081 | 4136.25 | 3978.33 |
| NW25-B1 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 | 130.34 |
| NW25-B2 | 307.95 | 310.62 | 302.64 | 309.23 | 309.23 | 309.23 | 311.87 | 314.18 | 302.61 | 314.18 | 314.18 | 314.18 | 314.18 | 314.18 | 314.18 | 314.18 | 314.18 | 314.18 |
| NW25-B3 | 495.25 | 521.27 | 467.36 | 484.59 | 501.58 | 443.62 | 513.15 | 520.04 | 483.37 | 521.15 | 521.27 | 520.04 | 518.9 | 520.04 | 510.91 | 521.27 | 521.27 | 521.27 |
| NW25-B4 | 675.88 | 715.55 | 627.7 | 709.32 | 757.26 | 678.17 | 752.44 | 781.66 | 732.09 | 781.36 | 781.66 | 772.52 | 777.9 | 781.66 | 762.29 | 781.66 | 781.66 | 781.66 |
| NW25-B5 | 692.82 | 790.46 | 606.1 | 802.2 | 904.26 | 748.22 | 876.37 | 908.73 | 846.47 | 910.74 | 928.59 | 876.18 | 892.65 | 926.01 | 861.8 | 912.457 | 928.59 | 881.78 |
| NW25-C1 | 1821.63 | 1830.23 | 1744.59 | 1830.21 | 1830.23 | 1830.09 | 1773.57 | 1830.23 | 1699.55 | 1799.84 | 1830.23 | 1699.55 | 1800.21 | 1830.23 | 1722.01 | 1830.23 | 1830.23 | 1830.23 |
| NW25-C2 | 2216.18 | 2296.42 | 2149.6 | 2173.61 | 2180.97 | 2168.7 | 2305.37 | 2403.45 | 2215.62 | 2361.13 | 2403.45 | 2293.31 | 2358.12 | 2403.45 | 2304.71 | 2403.45 | 2403.45 | 2403.45 |
| NW25-C3 | 3003.26 | 3129.37 | 2785.53 | 3015.77 | 3357.04 | 2841.99 | 3408.97 | 3564.97 | 3142.7 | 3554.41 | 3724.41 | 3402.87 | 3460.51 | 3583.44 | 3337.83 | 3606.814 | 3651.31 | 3558.98 |
| NW25-C4 | 4006.06 | 4256.57 | 3635.96 | 4030.89 | 4219.69 | 3548.58 | 4346.68 | 4604.9 | 4013.69 | 4527.1 | 4712.74 | 4302.1 | 4442.47 | 4581.05 | 4332.51 | 4642.232 | 4682.39 | 4616.74 |
| NW25-C5 | 5151.55 | 5563.63 | 4844.51 | 5593.5 | 5722.76 | 5191.31 | 5950.5 | 6132.09 | 5682.42 | 6067.01 | 6279.27 | 5834.71 | 5982.57 | 6203.44 | 5797.02 | 6099.962 | 6252.38 | 5991.29 |
| NW25-D1 | 772.82 | 772.82 | 772.82 | 762.53 | 772.82 | 755.67 | 772.82 | 772.82 | 772.82 | 772.25 | 772.82 | 755.67 | 772.82 | 772.82 | 772.82 | 772.82 | 772.82 | 772.82 |
| NW25-D2 | 1417.37 | 1426.94 | 1370.05 | 1426.94 | 1426.94 | 1426.94 | 1426.94 | 1426.94 | 1426.94 | 1426.94 | 1426.94 | 1426.94 | 1426.94 | 1426.94 | 1426.94 | 1426.94 | 1426.94 | 1426.94 |
| NW25-D3 | 1444.01 | 1554.68 | 1333.92 | 1614.08 | 1614.08 | 1614.08 | 1614.08 | 1614.08 | 1614.08 | 1614.08 | 1614.08 | 1614.08 | 1614.08 | 1614.08 | 1614.08 | 1614.08 | 1614.08 | 1614.08 |
| NW25-E1 | 1023.43 | 1026.03 | 1000.07 | 1026.03 | 1026.03 | 1026.03 | 1026.03 | 1026.03 | 1026.03 | 1026.03 | 1026.03 | 1026.03 | 1026.03 | 1026.03 | 1026.03 | 1026.03 | 1026.03 | 1026.03 |
| NW25-E2 | 1865.86 | 1904.37 | 1822.01 | 1953.88 | 1953.88 | 1953.88 | 1953.88 | 1953.88 | 1953.88 | 1953.88 | 1953.88 | 1953.88 | 1953.88 | 1953.88 | 1953.88 | 1953.88 | 1953.88 | 1953.88 |
| NW25-E3 | 2119.47 | 2229.93 | 2000.51 | 2261.69 | 2322.59 | 2209.69 | 2306.13 | 2322.59 | 2277.93 | 2322.59 | 2322.59 | 2322.59 | 2322.59 | 2322.59 | 2322.59 | 2322.59 | 2322.59 | 2322.59 |
| NW100-F1 | 3987.21 | 4439.3 | 3794.98 | 4776.99 | 4856.56 | 4489.29 | 4825.9 | 4942.17 | 4684.43 | 4693.13 | 4864.24 | 4547.26 | 4723.54 | 4865.5 | 4631.49 | 4871.921 | 4926.16 | 4802.06 |
| NW100-F2 | 4075.42 | 4306.1 | 3451.66 | 4421.67 | 4929.52 | 3878.3 | 5081.3 | 5222.9 | 4915.61 | 4609.31 | 4833 | 4399.49 | 4827.39 | 5030.46 | 4636.87 | 5039.407 | 5159.87 | 4923.54 |
| NW100-F3 | 4509.27 | 4862.05 | 4064.91 | 5158.37 | 5587.36 | 4370.63 | 5764.55 | 5940.17 | 5556.52 | 5697.9 | 5979.24 | 5431 | 5854.34 | 6061.31 | 5680.12 | 5920.087 | 6169.76 | 5755.56 |
| NW100-F4 | 4379.1 | 5025.9 | 3884.29 | 5298.89 | 5706.04 | 4987.7 | 5727.75 | 5840.15 | 5561.17 | 5551 | 5878.15 | 5186.71 | 5681.51 | 5872.83 | 5523.52 | 5855.063 | 6007.77 | 5688.85 |
| NW100-F5 | 4023.48 | 4404.67 | 3850.58 | 4544.91 | 4878.57 | 4088.81 | 4882.4 | 5106.64 | 4703.96 | 4675.32 | 4991.24 | 4443 | 4898.28 | 5142.89 | 4787.17 | 5128.474 | 5221.67 | 5025.71 |
| London-B1 | 1242.72 | 1274.13 | 1196.64 | 1354.41 | 1370.15 | 1307.22 | 1355.84 | 1370.35 | 1343.99 | 1370.27 | 1382.52 | 1364.15 | 1368.07 | 1372.43 | 1364.15 | 1374.163 | 1382.34 | 1369.49 |
| London-B2 | 1171.49 | 1194.37 | 1149.89 | 1220.64 | 1294.27 | 1135.38 | 1281.33 | 1300.52 | 1190.7 | 1304.66 | 1317.22 | 1293.57 | 1304.38 | 1306.99 | 1301.47 | 1301.506 | 1313.32 | 1283.26 |
| London-B3 | 1280.48 | 1336.45 | 1236.25 | 1398.22 | 1432.34 | 1294.19 | 1409.15 | 1452.95 | 1317.49 | 1441.6 | 1475.19 | 1391.45 | 1433.4 | 1448.63 | 1403.45 | 1449.077 | 1454.38 | 1441.72 |
| London-B4 | 961.84 | 998.42 | 903.17 | 1041.29 | 1089.37 | 999.18 | 1099.82 | 1114.31 | 1079.57 | 1104.02 | 1115.68 | 1095.02 | 1098.53 | 1103.91 | 1093.15 | 1100.34 | 1111.38 | 1093.35 |
| London-B5 | 1162.23 | 1192.67 | 1108.14 | 1160.45 | 1272.48 | 990.27 | 1263.45 | 1278.52 | 1241.58 | 1274.92 | 1284.1 | 1265.69 | 1267.93 | 1282.29 | 1251.51 | 1280.102 | 1296.37 | 1272.83 |
| London-B6 | 1292.97 | 1348.46 | 1240.56 | 1357.81 | 1357.81 | 1357.81 | 1445.43 | 1471.4 | 1425.21 | 1456.8 | 1472.42 | 1442.68 | 1454.45 | 1471.35 | 1444.86 | 1454.432 | 1471.34 | 1446.85 |
| London-B7 | 1340.51 | 1402.01 | 1299.14 | 1332.77 | 1451.5 | 1016.11 | 1491.26 | 1499.13 | 1486.53 | 1501.33 | 1508.61 | 1494.85 | 1496.35 | 1506.65 | 1487.7 | 1499.467 | 1505.39 | 1496.09 |
| London-B8 | 1443.1 | 1546.4 | 1359.45 | 1564.51 | 1632.45 | 1518.75 | 1642.05 | 1658.12 | 1616.83 | 1653.85 | 1661.81 | 1640.41 | 1649.98 | 1660.86 | 1635.39 | 1656.255 | 1660.42 | 1643.91 |
| London-B9 | 1278.4 | 1330 | 1250 | 1359.42 | 1398.89 | 1275.94 | 1390.98 | 1400.04 | 1381.91 | 1402.98 | 1427.03 | 1383.16 | 1397.09 | 1427.03 | 1384.13 | 1412.916 | 1427.03 | 1401.34 |
| London-B10 | 1370.85 | 1435.87 | 1290.36 | 1416.11 | 1556.31 | 1095.95 | 1532.12 | 1572.92 | 1446.62 | 1570.62 | 1574.72 | 1564.13 | 1568.87 | 1572.94 | 1565.69 | 1570.515 | 1573.07 | 1567.35 |
| London-L1 | 1630.07 | 2314.07 | 765.41 | 2021.16 | 2612.68 | 1178.64 | 2683.1 | 2800.32 | 2589.86 | 2770.23 | 2872.39 | 2555.98 | 2696.14 | 2863.29 | 2472.05 | 2811.065 | 2856.58 | 2772.17 |
| London-L2 | 1547.01 | 1992.53 | 1010.09 | 1334.42 | 2070.98 | 1069.38 | 2326.34 | 2395.85 | 2131.32 | 2586.85 | 2652.49 | 2407.64 | 2553.71 | 2629.02 | 2323.15 | 2588.337 | 2609.21 | 2563.03 |
| London-L3 | 1945.01 | 2158.27 | 1713.52 | 1619.26 | 2076.73 | 1192.94 | 2588.92 | 2720.57 | 2295.27 | 2821.42 | 2885.1 | 2699.49 | 2729.99 | 2841.8 | 2501.62 | 2859.042 | 2899.83 | 2806.06 |
| London-L4 | 1462.68 | 2205.88 | 816.93 | 1642.37 | 2231.58 | 1011.29 | 2340.7 | 2414.28 | 2269.69 | 2470.35 | 2534.58 | 2406.12 | 2454.35 | 2515.84 | 2373.21 | 2510.54 | 2569.06 | 2471.3 |
| London-L5 | 1584.54 | 2417.81 | 1049.46 | 2379.79 | 2721.68 | 2038.22 | 2587.09 | 2707.1 | 2401.82 | 2781.03 | 2867.42 | 2547.05 | 2775.41 | 2833.52 | 2683.13 | 2858.323 | 2889.71 | 2838.24 |

**Table 8**
Computational results of 10 min computational time. The bold numbers represent the best found solutions for each instance category.

| Instance | LANTIME | | | VNS | | | IG | | | IG+LS1 | | | IG+LS2 | | | Proposed SS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min |
| NW25-A1 | 1244.34 | 1244.34 | 1244.34 | 1244.34 | **1244.34** | **1244.34** | **1244.34** | **1244.34** | **1244.34** | **1244.34** | **1244.34** | **1244.34** | **1244.34** | **1244.34** | **1244.34** | **1244.34** | **1244.34** | **1244.34** |
| NW25-A2 | 2165.78 | **2168.18** | 2162.68 | 2167.76 | **2168.18** | 2163.92 | **2168.18** | **2168.18** | **2168.18** | **2168.18** | **2168.18** | **2168.18** | **2168.18** | **2168.18** | **2168.18** | **2168.18** | **2168.18** | **2168.18** |
| NW25-A3 | 2291.39 | 2397.46 | 2211.78 | 2407.12 | **2443.04** | 2308.11 | 2421.08 | **2443.04** | 2365.05 | **2443.04** | **2443.04** | **2443.04** | **2443.04** | **2443.04** | **2443.04** | **2443.04** | **2443.04** | **2443.04** |
| NW25-A4 | 2988.39 | 3120.74 | 2845.9 | 2970.76 | 3117.45 | 2721.16 | 3265.66 | **3289.99** | 3248.36 | **3289.99** | **3289.99** | **3289.99** | 3288.19 | **3289.99** | 3284.01 | **3289.99** | **3289.99** | **3289.99** |
| NW25-A5 | 3522.67 | 3710.07 | 3365.67 | 3669.57 | 3956.74 | 3310.2 | 4106.49 | 4141.66 | 4089.34 | 4095.67 | 4136.93 | 4056.42 | 4103.59 | 4136.93 | 4051 | 4128.77 | **4156.26** | 4103.13 |
| NW25-B1 | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** | **130.34** |
| NW25-B2 | 312.41 | **314.18** | 309.23 | 309.3 | 309.92 | 309.23 | **314.18** | **314.18** | **314.18** | **314.18** | **314.18** | **314.18** | **314.18** | **314.18** | **314.18** | **314.18** | **314.18** | **314.18** |
| NW25-B3 | 517.87 | **521.27** | 514.04 | 506.84 | **521.27** | 487.51 | 520.78 | 521.27 | 520.04 | **521.27** | **521.27** | **521.27** | **521.27** | **521.27** | **521.27** | **521.27** | **521.27** | **521.27** |
| NW25-B4 | 761.22 | **781.66** | 727.01 | 732.95 | **781.66** | 678.17 | 780.75 | **781.66** | 772.52 | **781.66** | **781.66** | **781.66** | **781.66** | **781.66** | **781.66** | **781.66** | **781.66** | **781.66** |
| NW25-B5 | 798.71 | 856.45 | 759.14 | 823.39 | 902.01 | 748.22 | 906.98 | 926.01 | 887.23 | **928.59** | **928.59** | **928.59** | 924.79 | **928.59** | 912.55 | 926.36 | **928.59** | 922.19 |
| NW25-C1 | **1830.23** | **1830.23** | **1830.23** | **1830.23** | **1830.23** | **1830.23** | 1813.53 | **1830.23** | 1802.42 | 1824.61 | **1830.23** | 1802.42 | 1824.61 | **1830.23** | 1802.42 | **1830.23** | **1830.23** | **1830.23** |
| NW25-C2 | 2333.04 | 2396.68 | 2270.87 | 2178.15 | 2238.63 | 2168.7 | 2369.9 | **2403.45** | 2314.25 | 2395.89 | **2403.45** | 2372.44 | 2380.84 | **2403.45** | 2345.94 | **2403.45** | **2403.45** | **2403.45** |
| NW25-C3 | 3294.94 | 3504.26 | 3142.09 | 2975.45 | 3422.22 | 2777.18 | 3522.15 | 3636.78 | 3343.53 | 3686.55 | **3724.41** | 3521 | 3591.32 | **3724.41** | 3468.31 | 3690.68 | **3724.41** | 3643.49 |
| NW25-C4 | 4221.06 | 4440.18 | 4049.3 | 4131.05 | 4413.13 | 3970.05 | 4494.31 | 4638.14 | 4138.17 | 4617.31 | 4767.56 | 4473.42 | 4566.22 | 4704.8 | 4399.74 | 4731.06 | **4767.56** | 4699.8 |
| NW25-C5 | 5442.55 | 5587.37 | 5235.25 | 5560.84 | 5752.05 | 5265.49 | 6175.11 | 6337.41 | 5928.06 | 6232.71 | 6322.28 | 6133.78 | 6229.06 | 6354.38 | 6028.48 | 6298.50 | **6429.01** | 6204.93 |
| NW25-D1 | **772.82** | **772.82** | **772.82** | 764.25 | **772.82** | 755.67 | **772.82** | **772.82** | **772.82** | **772.82** | **772.82** | **772.82** | **772.82** | **772.82** | **772.82** | **772.82** | **772.82** | **772.82** |
| NW25-D2 | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** | **1426.94** |
| NW25-D3 | 1585.76 | **1614.08** | 1554.68 | **1614.08** | **1614.08** | **1614.08** | **1614.08** | **1614.08** | **1614.08** | **1614.08** | **1614.08** | **1614.08** | **1614.08** | **1614.08** | **1614.08** | **1614.08** | **1614.08** | **1614.08** |
| NW25-E1 | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** | **1026.03** |
| NW25-E2 | 1909.29 | **1953.88** | 1883.87 | **1953.88** | **1953.88** | **1953.88** | **1953.88** | **1953.88** | **1953.88** | **1953.88** | **1953.88** | **1953.88** | **1953.88** | **1953.88** | **1953.88** | **1953.88** | **1953.88** | **1953.88** |
| NW25-E3 | 2210.39 | 2237.96 | 2181.49 | 2298.1 | 2322.59 | 2267.43 | **2322.59** | **2322.59** | **2322.59** | **2322.59** | **2322.59** | **2322.59** | **2322.59** | **2322.59** | **2322.59** | **2322.59** | **2322.59** | **2322.59** |
| NW100-F1 | 3987.21 | 4439.3 | 3794.98 | 4685.8 | 4984.03 | 4342.35 | 4908.07 | 4971 | 4869.59 | 4816.01 | 4863.06 | 4746.33 | 4846.78 | 4943.6 | 4743.01 | 4984.55 | **5075.37** | 4942.19 |
| NW100-F2 | 4075.42 | 4306.1 | 3451.66 | 4826.09 | 5312.66 | 4232.73 | 5259.06 | 5353.02 | 5177.91 | 4850.88 | 5108.49 | 4737.25 | 5001.98 | 5121.64 | 4868.26 | 5334.03 | **5455.49** | 5200.37 |
| NW100-F3 | 4509.27 | 4862.05 | 4064.91 | 5660.62 | 6147.93 | 5194.85 | 6055.99 | 6150.55 | 5989.3 | 6029.26 | 6186.72 | 5890.79 | 6149.93 | 6327.03 | 6034.87 | 6228.29 | **6335.43** | 6161.13 |
| NW100-F4 | 4379.1 | 5025.9 | 3884.29 | 5427.56 | 6130.1 | 4417.59 | 5979.84 | 6091.56 | 5840.15 | 5703.62 | 5798.14 | 5598.07 | 5923.78 | 6002.05 | 5800.31 | 6149.65 | **6222.43** | 6072.26 |
| NW100-F5 | 4023.48 | 4404.67 | 3850.58 | 4993.64 | 5309.17 | 4536.36 | 5073.5 | 5125.16 | 4998.03 | 4897.99 | 5037.45 | 4765.06 | 5150.65 | 5252.42 | 4989.59 | 5361.12 | **5481.32** | 5263.03 |
| London-B1 | 1292.09 | 1317.15 | 1275.39 | 1299.82 | 1372.03 | 1253.2 | 1379.1 | 1382.52 | 1372.8 | 1380.79 | 1382.52 | 1373.41 | 1379.09 | **1382.75** | 1371.02 | 1382.52 | **1382.75** | 1382.28 |
| London-B2 | 1205.22 | 1251.79 | 1175.46 | 1219.52 | 1274.38 | 1140.03 | 1311.64 | 1313.73 | 1306.99 | 1316.47 | **1317.22** | 1313.73 | 1315.97 | **1317.22** | 1313.37 | 1312.61 | **1317.22** | 1303.56 |
| London-B3 | 1316.39 | 1359.07 | 1286.31 | 1405.29 | 1463.77 | 1294.19 | 1442.58 | 1455.24 | 1389.63 | 1462.46 | **1475.19** | 1451.52 | 1453.97 | 1474.52 | 1446.94 | 1471.13 | **1475.19** | 1463.77 |
| London-B4 | 996.02 | 1043.2 | 975.18 | 1065.37 | 1092.75 | 1014.15 | 1115.77 | **1123.9** | 1111.55 | 1114.87 | 1117.4 | 1106.93 | 1114.64 | 1116.34 | 1110.51 | 1114.34 | **1123.9** | 1107.29 |
| London-B5 | 1187.36 | 1214.3 | 1177.63 | 1160.44 | 1250.48 | 1013.46 | 1280.86 | 1294.8 | 1275.34 | 1288.8 | **1297.27** | 1282.88 | 1280.89 | 1285.81 | 1274.58 | 1295.56 | **1297.27** | 1294.26 |
| London-B6 | 1336.51 | 1378.35 | 1298.05 | 1357.81 | 1357.81 | 1357.81 | 1459.47 | 1472.19 | 1448.59 | 1468.26 | 1472.63 | 1461.75 | 1464.77 | 1472.63 | 1452.07 | 1470.77 | **1475.86** | 1461.36 |
| London-B7 | 1387.29 | 1423.01 | 1350.64 | 1406.95 | 1487.41 | 1297.4 | 1498.23 | 1506.16 | 1488.74 | 1507.44 | **1508.61** | 1505.37 | 1505.34 | 1507.39 | 1500.22 | 1507.11 | 1508.57 | 1505.44 |
| London-B8 | 1503.55 | 1546.4 | 1461.31 | 1572.89 | 1629.99 | 1485.04 | 1656.28 | **1661.81** | 1639.12 | 1660.76 | **1661.81** | 1659.01 | 1660.72 | **1661.81** | 1659.27 | 1659.63 | **1661.81** | 1656.59 |
| London-B9 | 1325.4 | 1358 | 1306 | 1360.28 | 1400.39 | 1288.53 | 1402.68 | 1417.32 | 1400.59 | 1407 | **1427.03** | 1401.48 | 1401.87 | 1402.33 | 1401.34 | 1426.77 | **1427.03** | 1426.5 |

(*continued on next page*)

**Table 8** (*continued*)

| Instance | LANTIME | | | VNS | | | IG | | | IG+LS1 | | | IG+LS2 | | | Proposed SS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min |
| London-B10 | 1423.58 | 1457.88 | 1370.88 | 1379.03 | 1526.76 | 1070.66 | 1574.22 | 1581 | 1567.7 | 1574.07 | 1579.94 | 1571.97 | 1574.54 | **1584.04** | 1572.76 | 1577.00 | **1584.04** | 1569.42 |
| London-L1 | 2188.57 | 2452.84 | 1711.06 | 2144.04 | 2564.45 | 1528.81 | 2806.55 | 2868.5 | 2718.63 | 2872.44 | 2907.96 | 2836.64 | 2882.67 | 2913.87 | 2834.68 | 2899.00 | **2935.79** | 2875.24 |
| London-L2 | 1838.56 | 2011.91 | 1622.53 | 1184.78 | 1573.59 | 1027.13 | 2466.86 | 2496.04 | 2444.36 | 2645.52 | 2672.54 | 2631.99 | 2643.42 | 2656.89 | 2617.88 | 2652.03 | **2687.22** | 2634.2 |
| London-L3 | 2201.76 | 2630.41 | 1881.09 | 2038.09 | 2517.97 | 1608.92 | 2787.24 | 2802.55 | 2762.18 | 2890.51 | 2913.89 | 2863.53 | 2886.76 | 2923.59 | 2838.9 | 2906.47 | **2941.4** | 2881.52 |
| London-L4 | 1847.74 | 2205.88 | 1595.37 | 1690.75 | 2309.65 | 1072.99 | 2422.34 | 2454.94 | 2399.91 | 2529.19 | 2554.91 | 2508.17 | 2535.81 | 2563.04 | 2517.58 | 2559.67 | **2587.33** | 2529.4 |
| London-L5 | 2151.08 | 2451.34 | 1884.03 | 2411.85 | 2666.3 | 1924.04 | 2725.32 | 2745.89 | 2686.56 | 2873.51 | 2911.07 | 2832.99 | 2880.74 | 2905.25 | 2859.51 | 2899.35 | **2934.79** | 2866.06 |

four versions are statistically significant. Moreover, these results show that for the proposed SS algorithm, the insertion operators (i.e. intra-insert and extra-insert) lead to better results. The main possible reason could be the fact that the improvement method is being applied after a swap-based combination method (a 2-level path relinking) leading to exploration of more diverse solutions than when using swap-based moves in the improvement methods.

Since ◇Base is the best SS version, from henceforth, we use this as the final proposed SS algorithm.

### 5.5. Comparison with the existing algorithms

As we already mentioned, the body of research literature on this problem is limited. The existing algorithms include LANTIME and VNS algorithms proposed by Black et al. (2013), and three IG based algorithms (i.e. IG, IG+LS1, and IG+LS2) proposed by Vincent and Lin (2015). We choose these methods to compare with the proposed SS algorithm. Note that we make a meticulous effort to re-implement as close as possible the original implementation of the IG algorithms (i.e. IG, IG+LS1, and IG+LS2) reported in Vincent and Lin (2015). Unfortunately, our re-implemented versions of the IG algorithms could not reproduce the reported results. The potential reasons could be different factors such as details of the methods missing in the published articles, or variation in the operating systems or compilers. Considering this background, we compare the results of the proposed SS algorithm against the reported results of the competing algorithms. Originally, the LANTIME and VNS algorithms were implemented by the respective authors in C++ and the experiments were carried out on a PC with an Intel Core 2 CPU at 2.40 GHz, and 2.97 GB of RAM. The IG algorithms were implemented in C programming language and run on a personal computer with an Intel Core 2 CPU at 2.26 GHz with 2 GB of RAM, running in the Windows XP. We implemented our algorithms in C programming language and compiled the programs on a computer with an Intel Core 2 Duo processor, operating at a frequency of 2.5 GHz with 3.58 GB RAM, running on Windows 7. The TD-PARPs algorithms do not need much memory (less than 100 MB) and the computers described above are of very similar performance levels. As such, the best possible way for us is to take the reported results of the competing algorithms and compare our results with them.

Table 6 reports the ARPD results obtained by the competing algorithms as well as the proposed SS algorithm for all 8 instance categories. To have a better idea about the performance of the algorithms, two different computational times are considered as the cutoff: 1 min and 10 min. Also, the best solution found in both 1-min and 10-min experiments for each instance is considered as the $TP^{ref}$ of ARPD. The column Instance represents the names of the problem instances. For each algorithm and for each problem instance, we compute the average results over 10 runs (column Mean), the best solution found over 10 runs (column Max) and the worst solution over 10 runs (column Min). Besides ARPD results, to have a better picture of the performance of the algorithms for all 41 instances, Tables 7 and 8 report the results for 1 min and 10 min respectively.

From tables, it can be seen that the proposed SS algorithm obtained better results than all other competing algorithms in both 1-min and 10-min experiments; The ARPDs across all instances for SS algorithm are 1.70% and 0.53% respectively, while ARPDs for IG+LS2 the second-best algorithm are 2.95% and 1.33%. Obviously, the differences between the ARPDs of the proposed SS and the other competing algorithms are even greater. The same results can be seen for both Min and Max. For 10-min experiments, from Table 6, the proposed SS algorithm obtained the best results for Mean for 7 out of 8 instance categories (5 out of 8 strictly better). Also, the results given in Table 8 show that the proposed SS algorithm yields the best solutions found for 40 out of 41 instances, while it is 24 instances and 22 instances for IG+LS1 and IG+LS2 respectively. Note that among 8 instance categories, the NW-F category

**Table 9**
Paired t-tests results on 10-min running time. The bold numbers represent the best found solutions for each instance category.

| SS vs. | IG | | | IG+LS1 | | | IG+LS2 | | |
|---|---|---|---|---|---|---|---|---|---|
| RPD Metric | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min |
| Difference | 1.5556 | 1.2527 | 2.231848 | 1.022561 | 0.865697 | 1.28534 | 0.841734 | 0.672292 | 1.229701 |
| Degree of Freedom | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| t-value | 5.1254 | 3.9858 | 5.2171 | 2.9815 | 2.8387 | 3.4474 | 3.9488 | 3.318 | 4.3404 |
| p-value | <**0.0001** | **0.0003** | <**0.0001** | **0.0049** | **0.0071** | **0.0014** | **0.0003** | **0.0019** | <**0.0001** |

**Table 10**
Results of the best solutions found by the proposed SS and the family of IG algorithms (IG, IG+LS1 and IG+LS2) using 10 min and 240 min CPU time. The underlined values are the best among within the same timeout and the emboldened values are the best among both types of timeout. The bold numbers represent the best found solutions for each instance category.

| Instance | SS-10 | IG-10 | SS-240 | IG-240 |
|---|---|---|---|---|
| NW25-A5 | 4156.26 | 4141.66 | **4156.57** | 4142.61 |
| NW25-C5 | 6429.01 | 6356.38 | **6485.13** | 6370.07 |
| NW100-F1 | **5075.37** | 4984.03 | **5057.37** | **5057.37** |
| NW100-F2 | 5455.49 | 5353.02 | **5468.06** | 5455.49 |
| NW100-F3 | 6335.43 | 6327.03 | **6347.70** | 6331.29 |
| NW100-F4 | 6222.43 | 6130.10 | **6231.34** | 6219.73 |
| NW100-F5 | 5481.32 | 5309.17 | **5566.00** | 5448.98 |
| London-L1 | 2935.79 | 2913.87 | **3000.11** | 2933.34 |
| London-L2 | 2687.22 | 2672.54 | **2702.07** | 2687.65 |
| London-L3 | 2941.40 | 2923.59 | **2980.67** | 2924.86 |
| London-L4 | 2587.33 | 2563.04 | **2616.22** | 2573.57 |
| London-L5 | 2934.79 | 2911.07 | **2984.00** | 2930.29 |

contains the hardest problem instances in terms of problem sizes (i.e. the number of prize arcs). Table 8 shows that for this instance category, the worst solution obtained by the proposed SS algorithm (reported in column Min) is still better than the Mean of IG+LS1 and IG+LS2 in all 5 instances of this category, and even better than the Max of IG+LS1 and IG+LS2 in some instances of this category.

To verify the effectiveness of the proposed SS algorithm statistically, we perform a paired t-test at the 95% confidence level (significance level $\alpha = 0.05$) for the proposed SS algorithm versus IG, IG+LS1, and IG+LS2 algorithms for 10 min computational times and for all Mean, Max, and Min metrics. We did not compare the proposed SS algorithm with VNS and LANTIME since Vincent and Lin (2015) already showed that all three IG variants are statistically significantly better than VNS and LANTIME. From Table 9, the p-value is less than the significance level 0.05 for all 9 scenarios. This indicates that the proposed SS algorithm statistically outperformed all three IG algorithms (i.e. IG, IG+LS1, and IG+LS2).

As a final experiment, we have tested the performance of the proposed SS algorithm for a very long computational time of 240 min. We run the SS algorithm 5 times for this long CPU time and record the best solution found out of 5 runs. For this experiment, 12 large instances are picked. The results are reported in the Table 10. In this table, columns SS-10 and IG-10 gives the best solution found by the proposed SS algorithm and the family of IG algorithms proposed by (Vincent and Lin, 2015) (i.e. IG, IG+LS1, and IG+LS2), respectively, using 10 min CPU time. Moreover, columns SS-240 and IG-240 report the results using 240 min (4 h) CPU time. From Table 10, the proposed SS-240 improves the quality of the solutions by a large margin. For example, for the problem instance NW100-F5, which is the hardest instance, the solution in SS-240 is over 2% better than that in SS-10 and IG-240. It is also worth mentioning that SS-10 values are better than IG-240 value in 10 out of 12 instances; which show the quality of the proposed SS algorithm.

## 6. Conclusion

In this paper, we study a realistic variant of the prize-collecting arc routing problems (PARPs) called time-dependent PARPs (TD-PARPs). In contrast to the regular PARPs where the travel times are constant, in the TD-PARPs, the travel times between locations depend on the time of travel. This point is really important since in real-life situations, travel times can be affected by traffic conditions, weather conditions or other reasons. More accurate travel times help provide better travelling routes, leading to more revenue by serving more customers and reducing costs. TD-PARPs have a wide range of applications such as waste collection, newspaper delivery, postal service, milk delivery, electric meter reading, road maintenance, and street cleaning. Despite being realistic and interesting, the body of research literature on this problem is limited. In this paper, we propose five algorithms: four constructive heuristics and a metaheuristic based on the scatter search (SS) algorithm. All the proposed algorithms have specialised procedures to capture the characteristics of the problem. While the constructive heuristics returns solutions very quickly, the SS algorithm improves them further by maintaining a balance between intensification and diversification during search. The experimental and statistical results indicate the effectiveness of the proposed heuristics and the SS algorithm. The proposed SS algorithm is better than the existing algorithms in the literature, where the average results are at least 1% better than those obtained by other methods. Moreover, new best-known solutions have been found by the proposed SS algorithm for 12 out of 12 large-sized problem instances. These results indicate that we achieve our goal of pushing the boundaries and setting up a new standard in the solution of the TD-PARPs.

## CRediT authorship contribution statement

**Vahid Riahi:** Conceptualization, Methodology, Software, Validation, Formal analysis, Resources, Writing - original draft, Writing - review & editing. **M.A. Hakim Newton:** Formal analysis, Writing - review & editing. **Abdul Sattar:** Supervision, Writing - review & editing.

## References

Akbari, V., Salman, F.S., 2017. Multi-vehicle prize collecting arc routing for connectivity problem. Computers & Operations Research 82, 52–68.
Aráoz, J., Fernández, E., Franquesa, C., 2009a. The clustered prize-collecting arc routing problem. Transportation Science 43 (3), 287–300.
Aráoz, J., Fernández, E., Franquesa, C., 2013. Grasp and path relinking for the clustered prize-collecting arc routing problem. Journal of Heuristics 19 (2), 343–371.
Aráoz, J., Fernández, E., Meza, O., 2009b. Solving the prize-collecting rural postman problem. European Journal of Operational Research 196 (3), 886–896.
Aráoz, J., Fernández, E., Zoltan, C., 2006. Privatized rural postman problems. Computers & Operations Research 33 (12), 3432–3449.
Archetti, C., Bertazzi, L., Laganà, D., Vocaturo, F., 2017. The undirected capacitated general routing problem with profits. European Journal of Operational Research 257 (3), 822–833.
Archetti, C., Feillet, D., Hertz, A., Speranza, M.G., 2010. The undirected capacitated arc routing problem with profits. Computers & Operations Research 37 (11), 1860–1869.
Assad, A.A., Golden, B.L., 1995. Arc routing methods and applications. Handbooks in operations research and management science 8, 375–483.

Benavent, E., Carrotta, A., Corberán, A., Sanchis, J.M., Vigo, D., 2007. Lower bounds and heuristics for the windy rural postman problem. European Journal of Operational Research 176 (2), 855–869.
Black, D., Eglese, R., Wøhlk, S., 2013. The time-dependent prize-collecting arc routing problem. Computers & Operations Research 40 (2), 526–535.
Burke, E.K., Curtois, T., Qu, R., Vanden Berghe, G., 2010. A scatter search methodology for the nurse rostering problem. Journal of the Operational Research Society 61 (11), 1667–1679.
Christofides, N., 1973. The optimum traversal of a graph. Omega 1 (6), 719–732.
Corberán, Á., Eglese, R., Hasle, G., Plana, I., Sanchis, J.M., 2020. Arc routing problems: A review of the past, present, and future. Networks.
Corberán, Á., Fernández, E., Franquesa, C., Sanchis, J.M., 2011. The windy clustered prize-collecting arc-routing problem. Transportation Science 45 (3), 317–334.
Dror, M., 2012. Arc Routing: Theory, Solutions and Applications. Springer Science & Business Media.
Eglese, R., Maden, W., Slater, A., 2006. A road timetabletm to aid vehicle routing and scheduling. Computers & Operations Research 33 (12), 3508–3519.
Fleischmann, B., Gnutzmann, S., Sandvoß, E., 2004. Dynamic vehicle routing based on online traffic information. Transportation Science 38 (4), 420–433.
Glover, F., Laguna, M., Martí, R., 2000. Fundamentals of scatter search and path relinking. Control and Cybernetics 29 (3), 653–684.
Guo, Q., Tang, L., 2015. An improved scatter search algorithm for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. Applied Soft Computing 29, 184–195.
Hakli, H., Ortacay, Z., 2019. An improved scatter search algorithm for the uncapacitated facility location problem. Computers & Industrial Engineering 135, 855–867.
Hill, A.V., Benton, W.C., 1992. Modelling intra-city time-dependent travel speeds for vehicle scheduling problems. Journal of the Operational Research Society 43 (4), 343–351.
Huang, Y., Zhao, L., Van Woensel, T., Gross, J.-P., 2017. Time-dependent vehicle routing problem with path flexibility. Transportation Research Part B: Methodological 95, 169–195.
Ichoua, S., Gendreau, M., Potvin, J.-Y., 2003. Vehicle dispatching with time-dependent travel times. European Journal of Operational Research 144 (2), 379–396.
Maden, W., Eglese, R., Black, D., 2010. Vehicle routing and scheduling with time-varying data: A case study. Journal of the Operational Research Society 61 (3), 515–522.
Malandraki, C., Daskin, M.S., 1992. Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. Transportation Science 26 (3), 185–200.
Montero, E., Canales, D., Paredes-Belmar, G., Soto, R., 2019. A prize collecting problem applied to a real milk collection problem in chile. In: 2019 IEEE Congress on Evolutionary Computation (CEC). IEEE, pp. 1415–1422.
Mourão, M.C., Pinto, L.S., 2017. An updated annotated bibliography on arc routing problems. Networks 70 (3), 144–194.
Pearn, W., Wang, K., 2003. On the maximum benefit chinese postman problem. Omega 31 (4), 269–273.
Riahi, V., Khorramizadeh, M., Newton, M.H., Sattar, A., 2017. Scatter search for mixed blocking flowshop scheduling. Expert Systems with Applications 79, 20–32.
Sánchez-Oro, J., Laguna, M., Duarte, A., Martí, R., 2015. Scatter search for the profile minimization problem. Networks 65 (1), 10–21.
Soman, J.T., Patil, R.J., 2020. A scatter search method for heterogeneous fleet vehicle routing problem with release dates under lateness dependent tardiness costs. Expert Systems with Applications 150, 113302.
Tagmouti, M., Gendreau, M., Potvin, J.-Y., 2010. A variable neighborhood descent heuristic for arc routing problems with time-dependent service costs. Computers & Industrial Engineering 59 (4), 954–963.
Tagmouti, M., Gendreau, M., Potvin, J.-Y., 2011. A dynamic capacitated arc routing problem with time-dependent service costs. Transportation Research Part C: Emerging Technologies 19 (1), 20–28.
Tang, K., Wang, J., Li, X., Yao, X., 2016. A scalable approach to capacitated arc routing problems based on hierarchical decomposition. IEEE Transactions on Cybernetics 47 (11), 3928–3940.
Vincent, F.Y., Lin, S.-W., 2015. Iterated greedy heuristic for the time-dependent prize-collecting arc routing problem. Computers & Industrial Engineering 90, 54–66.