A solution approach to the orienteering problem with time windows and synchronisation constraints

Iman Roozbeh *, John W. Hearne, Delaram Pahlavani

School of Science, RMIT University, Melbourne, Australia

**Abstract**

The orienteering problem with time windows and synchronisation constraints, known as the Cooperative Orienteering Problem with Time Windows (COPTW), is a class of problems with some important applications such as in home health care and emergency logistics management, and yet has received relatively little attention. In the COPTW, a certain number of team members are required to collect the associated reward from each node simultaneously and cooperatively. This requirement to have one or more team members simultaneously available at a vertex to collect the reward poses a challenging task. It means that while multiple paths need to be determined as in the team orienteering problem with time-windows (TOPTW), there is the additional requirement that certain paths must meet at some of the vertices. Exact methods are too slow for operational purposes and they are not able to handle large scale instances of the COPTW. In this paper, we address the problem of finding solutions to the COPTW in times that make the approach suitable for use in certain emergency response situations. This is achieved by developing new merit-based heuristics as elements of an Adaptive Large Neighbourhood Search (ALNS) algorithm. We validate the performance of this new approach through an extensive computational study. The computational results show that the proposed method is effective in obtaining high quality solutions in times that are suitable for operational purposes.

**Keywords**
- Mathematics
- Multidisciplinary design optimization
- Mathematical modeling
- Algorithms
- Mathematical optimization
- Industrial engineering
- Adaptive large neighbourhood search
- Orienteering problem with time windows
- Synchronisation constraint
- Vehicle routing problem

**1. Introduction**

The Orienteering Problem (OP) is a well-known problem in combinatorial optimisation, introduced by Golden et al. (1987). The OP emerges from a combination of the travelling salesman problem and the knapsack problem. It is a routing problem where travelling to all vertices is often not feasible due to a time constraint. The objective of the OP is to find the combination of nodes that maximise the total reward collected. The design of tourist itineraries is an example where the OP has been applied (Vansteenwegen and Van Oudheusden (2007), Vansteenwegen et al. (2011a), De Falco et al. (2015)). The tourist trip design problem seeks to select the most interesting combination of attractions to visit within some available time span (Vansteenwegen and Van Oudheusden (2007), Borrás et al. (2014)). An extension of this problem to cycle trip planning was proposed by Verbeeck et al. (2014). Further applications can be found in the review by Vansteenwegen et al. (2011b).

There are several variants of the OP Gunawan et al. (2016). For example, the time-dependent orienteering problem considers the case where the travel time between two vertices depends on the leaving time of the first vertex Abbaspour and Samadzadegan (2011). This situation arises, amongst others, when travel times lengthen during peak traffic times. The time of arrival at a rail station, for example, can have a significant effect on waiting times when using public transport (Garca et al. (2010)). The Team Orienteering Problem with Time Window (TOPTW) has received a great deal of attention (Labadie et al. (2012), Kim et al. (2013), Souffriau et al. (2013), Duque et al. (2015), Gunawan et al. (2015), Zhang et al. (2016)). The TOPTW arises when the sport of orienteering is played by teams of several people. The TOPTW extends the OP to identify multiple paths that maximise the total score while meeting the time window constraint at visited locations or finishing within a fixed time at the final destination.

The Orienteering problem is NP-hard (Golden et al. (1987)). An extensive number of exact and heuristic approaches have been proposed to solve this problem. A few papers have focused on exact methods (Keshk et al. (2016), Poggi et al. (2010), Dang et al. (2013a), Bianchessi et al. (2018)) but generally they are not efficient for large-sized problems. As a result, the main body of the TOPTW literature is dominated by heuristic approaches (Bouly et al. (2010), Liang et al. (2013), Marinakis et al. (2015), Dang et al. (2013b), Gunawan et al. (2015), Labadie et al. (2012), Gunawan et al. (2017), Roozbeh et al. (2020))

* Corresponding author.
E-mail address: iman.roozbeh@rmit.edu.au (I. Roozbeh).

https://doi.org/10.1016/j.heliyon.2020.e04202
Received 25 June 2019; Received in revised form 22 February 2020; Accepted 8 June 2020

2405-8440/© 2020 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
As the TOPTW has become a popular topic, several state-of-the-art papers on the solution methods can be found in the recent literature. Gambardella et al. (2012) enhanced the Ant Colony algorithm to overcome certain drawbacks of the algorithm that they developed earlier (Montemanni and Gambardella (2009)). Vansteenwegen et al. (2009) developed an iterated local search algorithm for the TOPTW, where high quality solutions are attained in much less computational time than exact methods. More complicated benchmark instances for the multi-period orienteering problem, which is a generalisation of the TOPTW, are solved by a variable neighbourhood search algorithm (Tri-coire et al. (2010)). Lin and Vincent (2012) developed the slow and fast simulated annealing algorithms, where the latter performs efficiently in terms of computational effort and the former is more concerned about the solution quality. Gavalas et al. (2013) focused on tourist trip design problems. They introduced two cluster-based algorithms by considering the limitations of the iterative local search algorithm proposed by Vansteenwegen et al. (2009). Gunawan et al. (2015) extended the existing iterative local search algorithm by including more local search operations. Vidal et al. (2015) used variable neighbourhood search with an efficient select algorithm, equipped with iterated local search and a hybrid genetic algorithm, to solve the team orienteering problem. Also, Mei et al. (2016) investigated the multi-objective, time-dependent orienteering problem using a memetic algorithm. Finally, a recent article by Santini (2018) revealed the significance of using an ALNS algorithm for solving OPs.

An extension to TOPTW is the Cooperative Orienteering Problem with Time Windows (COPTW), proposed by Van Der Merwe et al. (2014). This problem arises when some tasks that need to be undertaken at certain locations can only be accomplished by two or more individuals acting cooperatively and simultaneously. Thus, in the COPTW, each vertex has a unique resource requirement. This requirement specifies the number of team members that simultaneously must visit each vertex to collect the reward at that vertex. The reward is also conditional upon the requirement being met within the time windows specified for each vertex. This means that while multiple paths need to be determined as in the team orienteering problem with time-windows (TOPTW), there is the additional requirement that certain paths must meet at some of the nodes.

The COPTW arises in several applications which motivate this study. The first application arose from a problem involving asset protection during escaped wildfires (Roozbeh et al. (2018a)). Valuable assets such as bridges and hospitals are distributed over a landscape with a network of roads. A wildfire sweeps over this landscape in certain directions. For a wildfire that is beyond control, fire-fighters are deployed to visit each asset to undertake tasks that will mitigate the risk of each asset’s destruction. The asset must be serviced before the fire reaches it, but must also not be serviced too early as hosed down structures will dry out again or areas cleared of debris can become littered again. Some assets might require a simultaneous visit by an aerial truck for accessing tall structures and a pump. Others might need a tanker (own water) and a personnel vehicle.

Tour planning for UAV’s is another application of the OP. In this case, rewards at a node correspond to priorities of target locations. Some of these problems require coordination and cooperation typical of the COPTW (Bertuccielli et al., 2004). Various OP applications of UAV’s include operations management in the event of disaster, and for other military and civilian purposes. Issues arising from uncertainties in these problems are also addressed (Bertuccielli et al., 2004, Evers et al., 2015). To handle this complexity, the OP has been extended to include, for example, the OP with Stochastic Weights (Evers et al., 2014c), Robust OP (Evers et al., 2014b) and Online Stochastic OP with time windows and time-sensitive targets (Evers et al., 2014a).

A further application is in the home care services. In such problems, some operations may require more than one staff member. For example, where heavy lifting and specialist medical expertise are required (Bredström and Rönqvist (2008)). A closely related problem is in the home cleaning services where some homes may require one or more services such as basic cleaning, window cleaning, and washing. Access to each house is within a given time-window. Further potential applications of the COPTW could be considered in post-disaster emergency services, where some services require visits from a supply vehicle to be synchronised with visits from personnel to distribute supplies.

For some applications of the COPTW, short solution times are essential. Due to the nature of the COPTW with its synchronisation constraints, the use of exact algorithms, as adopted by Van Der Merwe et al. (2014) is only suitable for relatively small instances. For large problems, exact methods become computationally prohibitive, especially where time-critical operational decisions have to be made. Thus, the aim of this paper is to develop an ALNS algorithm for the COPTW capable of handling large-scale problems and delivering good solutions within times suitable for operational purposes in emergency situations. To achieve this we build on the ideas given in Roozbeh et al. (2018b) and which were used to solve an asset protection problem (APP), a special case of the COPTW. Their method involved adapting a basic merit-based algorithm followed by various ruin and recreate heuristics. In this paper extensions to the proposed algorithm are introduced along with new insertion heuristics, and different problem-specific ruin algorithms are developed to efficiently search the problem space. These heuristics work together in an ALNS framework by detecting ineffective or redundant elements of the ALNS and deleting them. The performance of the approach was evaluated and tested for both small and large-size instances. For small instances, results are compared with optimal solutions obtained using exact methods. However, when it came to large scale instances CPLEX was unable to solve the problem to optimality. We also generate a large set of COPTW instances and report benchmark results for future studies.

In the following section the mathematical formulation of the COPTW problem is described. In section 3 we introduce the general scheme of the proposed approach. Our findings from an extensive experimental study are discussed in section 4. Finally, the conclusion and potential future research directions are stated in section 5.

2. Cooperative orienteering problem with time windows

The COPTW generalises the TOPTW formulation (Labadie et al., 2012), where a certain number of team members are required at vertices \( v_i \) \( (v_1, \ldots, v_N) \) (Van Der Merwe et al., 2014). The location \( v_1 \) is considered served if a required \( r_i \) team members arrive at the vertex within the time windows \( [a_i, c_i] \) and start the service simultaneously at time \( s_i \) for a duration of \( a_i \) time units to collect a reward, \( \psi_i \). In the COPTW, a homogeneous fleet of \( P \) team members start their route from \( v_1 \) and must return to the depot \( v_N \) by time \( T_{\text{max}} \), where both \( v_1 \) and \( v_N \) represent the same location. Unlike the classical OP, the COPTW enforces time windows for service activities and therefore \( T_{\text{max}} \) is represented as the closing time for the depot, \( T_{\text{max}} \). For any two vertices, \( v_i \) and \( v_j \), \( \psi_i \) indicates the required time for each team member to travel from \( i \) to \( j \). The binary decision variable \( y_{ij} \) takes the value \( 1 \) if \( r_i \) team members visit \( v_j \) within the appropriate time windows for the required duration, \( 0 \) otherwise. Another decision variable is \( z_{ij} \), which takes \( 1 \) if \( (i, j) \) is traversed, otherwise \( 0 \). Furthermore, \( x_{ij} \) represents the number of team members travelling from \( i \) to \( j \). Lastly, \( \xi \) defines a set of arcs that can be traversed. For \( (i, j) \in \xi \), if a team member departs from vertex \( i \) at \( a_i + a_j \) and arrives at vertex \( j \) before \( c_j \), the \( (i, j) \) can be traversed. Following the definition of \( \xi \), a set of feasible arcs that can be traversed to and from node \( i \) are shown by \( \Omega^i \) and \( \Omega^* \), respectively.

Van Der Merwe et al. (2014) formulated a mixed integer programming model of the COPTW. For ease of reference, the model is repeated below with a few minor changes in notation.

\[
\text{Maximize} \sum_{n=2}^{N-1} \sum_{i=1}^{P} \psi_i y_{ij} \quad (1)
\]
The objective function (1) maximises the sum of the rewards \( w_i \) collected at each vertex \( t \). Constraint (2) ensures all members depart from and return to a designated depot. Constraint (3) guarantees flow conservation by enforcing the equality of incoming and outgoing arcs to each node. Constraint (4) ensures that the collection of a reward (i.e. a score) at each location is dependent upon the condition of fulfilling its resource requirement. Constraint (5) makes sure that the number of travelling members of a fleet through an arc never exceeds \( P \) and it also ensures that \( x_{ij} \) is zero when the path \( ij \) is not traversed. Constraint (6) ensures that at each vertex the service can only be started when the previously visited location has been served completely and there is sufficient time to travel to the vertex, where \( M \) represents a large constant. Setting \( M = \max(c_i) + \max(t_{ij}) + \max(a_i) - \min(a_i) \) is sufficiently large for this purpose. Constraints (7) and (8) ensure each vertex is visited within its time window. Integer and binary conditions are defined in constraints (9) and (10). Note that the \( T_{\max} \) in the time budget constraint in orienteering problems is assumed to be the closing time of the ending point \( (v, x) \) meaning that there is a time window for all nodes and routes should be finished by returning back to the final depot by \( c_v \).

A graphical representation of a simple solution for COPTW is sketched in Fig. 1 as shown below. Two of the main attributes of each node, the resource requirement \( r_i \) and corresponding reward \( w_i \), are specified while all nodes have a time window constraint. A given number of team members leave the depot and \( r \) members must be at vertex \( i \) before starting the service to collect the associated reward \( w_i \). The numbers over each arc represent the number of members travelling through an arc. It is infeasible to visit all vertices within their time windows. Considering the set of all feasible solutions, a schedule that maximises the score (i.e. the sum of rewards collected) is identified. In Fig. 1, three members leave the depot by travelling through different arcs and attend nodes 4, 1 and 3 individually. As their associated resource requirements are all equal to one, they complete the service and collect the rewards. Thereafter, all three members travel to node number 6 to fulfil its resource requirement and collect the associated reward (10). After collection of maximum amount of possible scores, all members return to the depot. Note that both depots represent the same location.

3. A solution approach for the COPTW

We propose an Adaptive Large Neighbourhood Search (ALNS) that brings together a powerful algorithmic framework to solve the COPTW. We extend the work by Roozbeh et al. (2018b) developed for the asset protection problem. Each of the ruin and recreate heuristics is re-examined for the COPTW and for opportunities for further efficiency enhancements and then modified accordingly. In addition some new insertion and removal algorithms are introduced. Significant changes from the APP algorithm are indicated with an asterisk (*) in Table 2. In this section, we define the ALNS algorithm and describe the heuristics implemented.

3.1. ALNS procedure

The proposed ALNS algorithm comprises the construction of an initial solution followed by ruin and recreate strategies to improve the solution. The latter process continues until some termination conditions are satisfied. Stopping conditions for the ALNS heuristic are based on the total number of iterations and the run time limit. Through every iteration \((i \in N)\) a ruin heuristic \((d \in h_d)\) is selected and partially destroys the current feasible solution \((S^*)\). Thereafter, a recreate heuristic \((r \in h_r)\) inserts unvisited nodes into enough routes to achieve a better solution \((S'^*)\). In the ALNS, a wide range of removal and insertion heuristics are implemented and used dynamically and adaptively according to their past performance. This leads to the exploration of large neighbourhoods and escaping from local optimal solutions. An adaptive weight \(W(h)\) and a score \(s(h)\) is defined for each ruin and recreate heuristic \((h \in h_d \cup h_r)\). Initially, all weights are equal, and scores are set to be zero. The algorithm runs for \(N\) iterations, divided into \(k\) segments and therefore iterates \(n = \frac{N}{k}\) times over each segment. The corresponding score of ruin and recreate heuristics are increased...
by $S_\tau$, $S_\sigma$, and $S_\rho$ defined below, after each iteration. If the new solution improves the current best solution ($S^*$), the value of $S_\tau$ is added to the score of the associated ruin and recreate heuristics. The value of the $S_\sigma$ is added to scores if the new solution improves the best current solution through a segment. Finally, the value of $S_\rho$ is added if the new solution is accepted despite degrading the objective value. After each segment, the weight of algorithm $h \in h_x \cup h_x$ is updated by using the formula $(1 - \rho) \times W(h) + \rho \times Z(h) / \bar{W}(h)$. The parameter $\rho$ is called the reaction factor. It effectively regulates the number of times that ineffective algorithms are utilised when they are not contributing substantially to improving the solution. Also, associated weights, scores and number of times a heuristic is selected are defined by $W(h), z(h)$ and $u(h)$, respectively. The probability of using an algorithm in segment $s + 1$ is defined based on the formula $p_{s+1}(h) = W(h) / \sum_{h_i} W(h_i)$ where the denominator calculates the cumulative weight of ruin and recreate heuristics and the scores are set back to zero when a segment has been completed (Roosbe et al., 2018b).

It has been shown by Santini et al. (2018) that acceptance criteria significantly impact the solution quality. For this reason, we take advantage of Simulated Annealing (SA) (Van Laarhoven and Aarts (1987)) to escape from local optimal solutions by accepting solutions with worse objective values with probability $exp(-\frac{(z(S) - z(S^*)}{\Delta_T})$, while solutions with improved objective values are always accepted. In the SA, $T, z(S)$ and $z(S^*)$ represent temperature, the objective value of the current solution and the best solution, respectively. The initial temperature needs to be defined in such a way as to accept solutions with probability $P_{accept}$ where the objective is $5\%$ worse than the initial solution $z(S_{Initial})$. Therefore, $T_{Initial}$ can be set as $(S_{Initial} + \delta) / \log(1/P_{accept})$. The temperature is updated every $N$ iterations according to the cooling rate, $(T = e \times T)$ where $0 < e < 1$.

### 3.2. Insertion feasibility

A survey of vehicle routing problems with different types of synchronisation constraints can be found in Drexel (2012). Synchronisation constraints exist in various areas such as home health care routing and scheduling (Mankowyska et al., 2014), pick-up and delivery (Clarke et al., 2017)), transportation and logistics, medical delivery and airport ground handling services (Fink et al., 2019)). Our study focuses on synchronised operations in which multiple team members are required to visit a set of nodes to accomplish various tasks. Synchronisation constraints significantly increase complexities in maintaining feasibility during insertion operations. To efficiently handle these complexities in a time efficient manner, the following approaches are implemented.

We first filter out infeasible arcs through the search process; therefore, when $\sigma_i + t_{ij} > c_j$ for all nodes, then $arc_{ij}$ will be removed in the initial solution. Also, due to the requirement of nodes that need to be visited by multiple routes, we filter visiting of node $i$ after $j$ if $j$ has already been inserted before $i$ in one of the existing routes. We finally define a “feasibility matrix” comprising binary elements to facilitate the insertion process. Most importantly, to evaluate the possibility of insertions in constant computational time within a route, we need to define $maxshift_i$. This is to determine the existing amount of spare time that can be invested to meet a node $i$ before arrival. Arrival time at a node can simply be defined as below.

$$arrival_i = departure_{i-1} + traveltime_{i-1,i}$$

Since a service must be accomplished simultaneously by a required number of team members ($T_i$), a synchronised start of service is calculated as below.

$$sync_{start_i} = max_{\tau \in T} \{ max \{arrival_i, o_i \} \}$$

Subsequently the time that a team member waits at node $i$ to start the service is as follows.

$$wait_i = sync_{start_i} - arrive_i$$

For a given route $r$, a visit at node $i$ is defined by $\tau(i)$. The equation (14) defines the time that arrival can be delayed ($maxshift_{r+1}$) while the solution feasibility is satisfied.

$$maxshift_{r+1} = min\{ c_{\tau(i)}, - wait_{\tau(i)+2}, maxshift_{r+1} \}$$

To determine the insertion feasibility of a node $k$ for $i + 1$ in route $r$, calculation of the caused shift ($shift_{r+1}$) is required. Thus:

$$shift_{r+1} = traveltime_{i,k} + wait_{i,k} + service_{i,k}$$

$$+ traveltime_{j,k} - traveltime_{j+1}$$

If the caused shift ($shift_{r+1}$) is less than or equal to the $wait_{i+1}$ then the insertion will be considered as valid.

### 3.3. Insertion algorithms

We introduce new variants to the classical merit-based (MB) heuristics (Roosbe et al., 2016), where their speed and performance assist us in handling complexities of the problem. The MB heuristic is utilised with different approaches including the classical MB heuristic, the noised imposed MB heuristic, the cumulative MB and the noised imposed cumulative MB heuristic. In addition to the variants of the MB heuristics, our algorithm, which can be regarded as a greedy time heuristic is formulated with consideration for the relative travel time. Overall, the proposed algorithm includes five insertion heuristics, which are described below.

#### 3.3.1. Classical merit-based heuristic

The MB algorithm constructs a list of tuples $i, j, M_{ij} > \forall i, j \in V$. The $M_{ij}$ refers to the merit value which is calculated by using the following function:

$$M_{ij} = \delta \frac{\psi_i + \psi_j + d_{ij} + d_{ij} - \lambda d_{ij}}{d_{ij}} + \mu \cos \theta_{ij} \frac{|d_{max} - (d_{ij} + d_{ij})|}{d_{max}}$$

In equation (17), $\theta_{ij}$ is the angle between the vectors from the depot to the nodes $i$ and $j$, respectively. The score for each node and the average score of all vertices are represented by $\psi_i$ and $\psi_j$, respectively. Also, in the above formulation, $d_{ij}$ is the distance between nodes $i$ and $j$ and $d_{max} = max \{d_{ij} : \forall i, j \in V \}$ is used to scale the distance term. To ensure that the assignment of nodes with the highest merit values are prioritised, we implemented parallel route construction in the algorithm. Moreover, best values for the three parameters, $\delta, \lambda$ and $\mu$, are defined in the parameter tuning part of the algorithm. Please note that travel time $t_{ij}$ and travel distance $d_{ij}$ are assumed to be correlated and they can be used interchangeably while having no impact on the order of merit values.

Algorithm 1 represents the MB heuristic. The algorithm starts by ranking unvisited nodes according to the ratio $\frac{\psi_i}{d_{ij}}$ where $r_{ij}$ are the resources (team members) required to collect the reward $\psi_i$ at node $m$. By experimentation the square root in the denominator was found to yield better results than a straight ratio of reward to required resources. Next,
Algorithm 1: Pseudocode for the merit-based heuristic.

```
Input: best collected reward ($\delta$), best routes ($\beta$), unvisited vertices ($U$), Merit Pair List (MPL), score of node $w$, maximum number of travelling team members ($P$), number of sub-routes ($s$), service requirement of node $w$ ($r_m$), distance matrix of nodes ($d_{ij}$), visited node before node $q$ ($q'\beta$)

Output: $\beta$ and $a$

function MB heuristic

1. sort $U$ in descending order of $\frac{d_{ij}}{\sqrt{\beta}}$. $\forall w \in U$
2. generate $d_{ij}$
3. calculate feasibility matrix
4. calculate merit list and sort $(c(i,j), M_{ij})$
5. $M_{ij} \leftarrow \frac{w_i + w_j}{\beta} + d_{ij} - \lambda d_{ij} + \mu \cos \frac{d_{ij} - (d_{ij} + d_{ij} / 2)}{d_{ij}^2}$
6. for $(m \in U)$ do
7.   for $(pair_{ij} \in MPL)$ do
8.     for $(subroutes \in L)$ do
9.       if $(m = w \&\& q \in L \&\& VisitCount_{m} > r_m)$ then
10.      if $(shift_{ij} > |\sum_{i \in L} maxshift_{ij}|)$ then
11.         remove arc $(q', m)$, add arcs $(q', m)$ and $(m, q)$
12.      update maxshift and insertion feasibility matrix
13.     Update $L$
14.     $VisitCount_{m} \leftarrow VisitCount_{m} + 1$
15.     if $(n < P \&\& VisitCount_{m} > r_m)$ then
16.       Create a new route and add $m$
17.       Update $L$
18.       $n \leftarrow n + 1$
19.     $VisitCount_{m} \leftarrow VisitCount_{m} + 1$
20.     if $(VisitCount_{m} = r_m)$ then
21.       update $\beta$
22.       update $a$
23.       update $U$
24. return $\beta$ and $a$
```

the algorithm creates a Euclidean distance matrix and a feasibility matrix as explained in section 3.2. The Merit Pair List (MPL) is constructed and sorted in descending order of $M_{ij}$ (lines 5 and 6). The algorithm iterates over each node in the set $U$ (line 7) for each pair (line 9) as it attempts to assign unvisited nodes into an existing subroute. Otherwise if resources are sufficient a new route is created (line 17). Relevant information is then updated (lines 22-25).

3.3.2. Noise-imposed merit-based heuristic

The noise-imposed merit-based heuristic (NMB) heuristic is used to bring additional diversification to the search. We define a noise parameter $0 < \alpha < 1$, then $\Delta = \alpha \cdot max_{i,j}$, is the amount of noise allowed. The NMB heuristic updates merit values by $M_{ij} = M_{ij} + \xi$ where $\xi \in [-\Delta, \Delta]$.

3.3.3. Cumulative merit-based heuristic

Through the insertion process, using the classical MB heuristic we look at the merit value $M_{ij}$ of inserting the unvisited node $i$ where node $j$ is the succeeding node. However, we may need to consider also the node preceding $i$ to accomplish an efficient insertion. Thus, to incorporate both the succeeding and preceding nodes into the merit values, we make a tuple of four components $(<i, k, j, M_{ijk,j}>)$ where $M_{ijk,j}$ is the merit value that can be achieved by insertion of the unvisited node $k$ between nodes $i$ and $j$.

$$M_{ijk,j} = M_{ijk} + M_{ij} - M_{ij} \quad (18)$$

The Cumulative merit-based (CMB) heuristic uses the calculated merit values for pairs in equation (17) and determines values of $M_{ijk,j}$ by equation (18). Fig. 2 illustrates insertion operation before (Fig. 2(a)) and after (Fig. 2(b)) using equation (18). Insertion by equation (17) may result in a route as defined in Fig. 2(a). In the generated merit pair list using the equation (18), as both succeeding and preceding nodes are considered, the $M_{ijk,j}$ gets a larger value than $M_{ijk}$. Therefore, insertion will take place as shown in Fig. 2(b) which is the optimal assignment.

3.3.4. Noise-imposed cumulative merit-based heuristic

The same approach as in section 3.3.2 is used for the noise-imposed cumulative merit-based (NCMB) heuristic to broaden the search space.

3.3.5. Greedy time heuristic

The amount of travelling may curtail the number of nodes that can be visited. Thus a Greedy Time (GT) heuristic is developed. The GT heuristic focuses on travel time minimisation. A list of tuples $<i,j,k, T_{Rij,k}>$ is generated and sorted in ascending order of time ratio values, $T_{Rij,k}$, defined by equation (19). Note that, although this equation focuses on travel time, it includes other problem specific attributes in the denominator.

$$T_{Rij,k} = \frac{t_{i,k} + t_{k,j} - t_{i,j}}{\sqrt{\frac{\psi_{i,k} + \psi_{j,k}}{\sqrt{\beta}}}} \quad (19)$$

In equation (19), the numerator calculates the increase in the travel time as a result of inserting node $k$ between $i$ and $j$. The $\psi_{ik}$ is the associated score for node $k$ which is divided by the square root of the ratio of the requirements at node $k (r_k)$ to total number of team members ($P$).

In Fig. 3(a) two potential nodes are shown for a insertion between nodes $i$ and $j$. Although visiting node $L$ involves a longer travel time than node $k$, the GT heuristic inserts $L$ into the route. This is because $\psi_{L} > \psi_{k}$ impacts on the total value of $T_{Rij,k}$ and ends up with a higher ranking in the list.

3.4. Ruin methods

The initial solution generated by the classical MB heuristic ($S_0$) is partially destroyed by one of the ruin (removal) algorithms. At each iteration, a ruin heuristic removes $D$ nodes from the solution, where $D$ is called the degree of destruction. The value for $D$ is a random number in the range of $[0.1K, 0.4K]$ when $K$ nodes are visited. In the following, we explain all ruin methods that are implemented in our ALNS in details.

A Random Removal (RR) algorithm randomly removes $D$ customers from the solution. Worst-Distance Removal (WDR) determines the distance of nodes from succeeding and preceding nodes and sorts them in descending order in a list named as $O$. Then, it removes nodes in position $[Y \cdot |O|]$ from the solution and the list. Parameters $k \geq 1$ and $0 < Y < 1$ bring randomness to select new nodes for removal. Worst-Time Removal (WTR) performs the same as the WDR, however the list $O$ is defined based on values of $|sync_{start,i} - v_i|$. Shaw Removal (SR) measures relatedness between nodes using $\Gamma_{ij} = \theta_1 d_{ij} + \theta_2 |o_i - o_j| + \theta_3 \Omega_j$ where $\theta_1$, $\theta_2$, and $\theta_3$ are the Shaw parameters and $\Omega_j$ varies in the range of $[-3, 1]$ based on the number of routes that nodes $i$ and $j$ have in common. The SR heuristic picks a random node and measures relatedness of other nodes with the one selected. Thereafter, position $[Y \cdot |O|]$ will define the node that has to be removed from the related list $O$. In the SR algorithm parameters $\theta_i \geq 1$ is the determinism factor and $1 \geq Y \geq 0$ is a random number. The SR increase the chance of substitution of nodes with the ones that have a high level of similarity in terms of distance, routes in common and travel time window, so that the search can move to a new neighbourhood. Proximity Removal (PR), Time Removal (TR) and Requirement-Based Removal (RBR) algorithms are special cases of the SR heuristic where $\theta_1$, $\theta_2$, and $\theta_3$ are set to a value of 1, while others are set at 0.

Waiting-Time Oriented Removal (WOTOR) is similar to the WDR, however the cost is the waiting time calculated as $W_{T} = sync_{start} - arr_{i,j}$. Worst-Removal Removal (WR) algorithm removes nodes
with highest demand from the solution. Relative-Requirement Removal (RRR1) heuristic is designed for heterogeneous fleet of members (i.e., vehicles). The RRR1 algorithm follows the same logic as the WRR, however, considers available number of members from each type. Therefore, nodes with lowest value of $\omega$ defined as $\omega = r_q * Z_i$ have higher chance of removal, where $r_q$ and $Z_i \in [-1, 0, 1]$ are the number of member type $q$ needs to visit node $i$ and the assigned importance rate of member type $q$, respectively. The parameter $Z_i$ gets a larger value when more of member type $q$ is available. The RRR1 is introduced for sensitivity analysis in Table 2 on the APP’s benchmark instances and is not practical for the COPTW as it deals with a homogeneous fleet of members. Resource Requirement Removal (RRR2) algorithm removes nodes with similar demands to increase the chance of being reinserted in alternative positions.

The Historical-Node Removal (HNR) algorithm uses the cost function $f_i = WT_i + \frac{DC_i}{\gamma_i}$ for removing nodes, where the first two terms are normalised values from WTOR and WDR algorithms. The HNR removes nodes with worst $j^* = \arg\max_j\{f_{j_1} + f_{j_2}\}$ where $f_{j_1}$ is the best position cost of node $j$ before iteration $i$. The Time Windows-Oriented Removal (TWR) algorithm removes nodes with similar values of windows using the same approach as the HNR algorithm. The TWR algorithm divides the total time window length into four zones and removes $\Lambda_i = \frac{\text{Total}\_\text{time}}{4}$ nodes from each zone, where $\text{tu}, \text{zu}$, denote respectively the total nodes and the unvisited nodes in the zone $i$. Every time one of the WDR, WTR, WTOR, and RRR1 algorithms is called, we use a random binary variable to decide whether the calculated cost values should be divided by their associated score or not. The Last removal algorithm is the Cluster Removal (CR) heuristic. We use a density-based algorithm for clustering nodes (Ester et al. (1996)) which have proved to be efficient for classical OPs (Santini (2018)). We select a random cluster and remove $D$ nodes from the chosen cluster in the current solution. The process of random selection of clusters and removal of nodes from the intersection of the cluster and the solution continues until $D$ nodes are being removed.

The overall description of the ALNS is given in Algorithm 2. The ALNS algorithm determines an initial solution ($S_0$) by the classical MB heuristic. Then, probabilities for insertion and removal heuristics are initialised (line 5). The preliminary temperature needs to be initialised for the SA algorithm and is updated (line 23) through the search process. Using the roulette-wheel mechanism as an adaptive layer of the ALNS a recreate and ruin heuristic is selected in every iteration (lines 10 and 11). Note that all variants of the MB heuristic in addition to greedy time heuristic fall within the set of recreate (repair) algorithms (line 9). Depending on the improvements resulting from the selected algorithms, scores are updated for the heuristics (line 18). Finally, the ALNS returns the best solution ($S^*$) found after $N$ iterations (line 24).

### Algorithm 2: The overall structure for the ALNS algorithm with simulated annealing.

**Input:** Number of iterations ($N$), Initial solution ($S_0$), Number of iterations for each segment ($\alpha$), Set of removal heuristics ($h_r$), set of insertion heuristics ($h_i$).

**Output:** Best solution ($S^*$)

```plaintext
1 function ALNS
2 Use the classical MB heuristic to generate an initial solution $S_0$
3 Let $S^* = S_0$
4 Let $\alpha$ be the outmost iteration counter initialised as $\alpha = 1$
5 Initialise $P(i_r), P(i_s)$ for each $i \in h_r \cup h_i$
6 Let $T = T_{\text{init}}$
7 while ($\alpha \leq N$) do
8 Select a removal heuristic $i \in h_r \Rightarrow (S_i^-)$
9 Select a repair heuristic $i \in h_i \Rightarrow (S_i^+)$
10 if $\alpha(S_i^-) \leq \alpha(S_i^+)$ then
11 $\alpha = \alpha(S_i^-)$
12 if $\alpha(S_i^-) > \alpha(S_i^+)$ then
13 $S_i = S_i^+$
14 Let $\beta = \exp\left(\frac{\alpha(S_i^-) - \alpha(S_i^+)}{\text{K}}\right)$
15 Generate a random uniform number $\gamma \in [0, 1]$ where
16 if $\gamma \leq \beta$ then
17 $S_i = S_i^-$
18 Update $x_i$ for the selected heuristics
19 $\alpha = \alpha + 1$
20 if $\alpha \text{mod} \alpha = 0$ then
21 Update all probabilities $P(i_r), P(i_s)$ for each $i \in h_r \cup h_i$, and all adaptive weights
22 Update the SA temperature, $T = \epsilon \times T$
23 return $S^*$
```

### 4. Computational study

Extensive numerical studies were conducted to evaluate the efficacy of the proposed solution approach. A set of benchmark instances was
Table 1  Parameter tuning results.

| Description                  | Parameter Value |
|------------------------------|-----------------|
| Parameters for MB heuristic  | $(\alpha, \mu, \theta)$ (2,1,3) |
| Improving solution score     | $s_2$ 12        |
| Number of iterations         | $N$ 3000        |
| Number of iterations each segment | $n$ 100   |
| Roulette wheel reaction factor | $\rho$ 0.1 |
| Global solution score        | $s_1$ 35        |
| Worst solution score         | $s_1$ 5         |
| Shaw parameters              | $\theta_1, \theta_2, \theta_3$ (3,13,7) |
| SA parameter                 | $\delta$ 0.05   |
| Cooling rate                 | $\epsilon$ 0.9999 |
| Noise parameter              | $\alpha_{max}$ 0.6 |
| WDR determinism factor       | $k$ 8           |
| Shaw determinism factor       | $\eta$ 12       |

generated by adding the problem-specific attribute to the well-known existing benchmark sets (see Vansteenwegen et al. (2009)). The resource requirement attribute was added to each vertex by picking 1, 2 or 3 randomly, which indicates how many members of the team are required to collect the associated reward at each node.

In the first study, we investigate the performance of the proposed removal and insertion algorithms. Thereafter, we evaluate the efficiency of our ALNS approach. In the next study, truncated benchmark sets are designed to solve sufficiently small-sized instances by means of both the CPLEX commercial solver and our algorithm. The number of vertices in the small-sized instances are carefully chosen to obtain the optimal solutions using CPLEX. Furthermore, we explore the trade-off between an increased number of available members for service on the one hand and the computational time and objective value on the other hand. In addition, we demonstrate the performance of the proposed heuristic in terms of time and accuracy on the large-sized benchmark instances and present our results as benchmarks for future studies. All the above computations were performed on a single CPU with 16 GB of RAM on the Australian National Computational Infrastructure using a single thread. Each node is equipped with dual 8-core Intel Xeon (Sandy Bridge 2.6 GHz) processors. The algorithm was programmed in C+++, using a GCC 6.2.0 compiler. Where applicable, MILP models were solved by the CPLEX 12.6 commercial solver in deterministic parallel optimisation mode. All tables show the execution times as elapsed time in seconds.

4.1. Parameter tuning and sensitivity analysis of algorithms

Many aspects of the COPTW are similar to those of the APP. Thus, it seemed logical to use the same parameter values identified in Roozbeh et al. (2018b). This also meant that we could perform a fair comparison on the performance of our new heuristics against the APP developed in that paper. The parameters had been determined using a similar tuning methodology to Emeç et al. (2016), Ropke and Pisinger (2006). Parameters and their tuned values are summarised below. Please note that similar parameter values to Roozbeh et al. (2018b) are used for validation purposes.

We investigate the performance of the insertion and removal heuristics in removing inefficient heuristics. This is done by conducting a sensitivity analysis on the heuristics using the same instances that were used for the parameter tuning: R104, R206, RC104, RC108 and RC206. Among the removal algorithms, WTOE, WDR and HNR are the most frequently used algorithms while CR, TWR and RR rarely contribute through the ALNS algorithm. The results show that all insertion heuristics contribute to the solution.

An algorithm with a minor contribution through the search process might still play a significant role in helping the ALNS to escape a local optimum Emeç et al. (2016). To identify whether omission of an algorithm might improve the final solution, we conduct further experiments omitting just one algorithm. The results in Table 2 indicate that exclusion of the MB heuristic has a negative impact on the worst, average and best values of the objective function in 10 runs. However, exclusion of the NCMB improves the ALNS performance. Thus, this heuristic, and others (indicated in the table with a "Y"), are deleted from the ALNS. The other heuristics (indicated with a "N") make some positive contribution to the objective and remain in the ALNS.

4.2. Experiments on APP instances

We validate the performance of our algorithm against the results published in Roozbeh et al. (2018b) due to its similarity with the COPTW. In this paper an algorithm is proposed to solve an asset protection problem during escaped wildfires where community assets are endangered by the moving fire front. The objective function aims to maximise the total scores collected by protecting at-risk assets. An asset can be protected if a sufficient number of vehicles (i.e. members) from different types accomplish tasks within the time window. Our results are achieved by using the parameters in Table 1 after exclusion of inefficient heuristics. Table 3 shows a comparison of the results of our algorithm and the APP. We define two sets of vehicles as in the original paper named as Set1 and Set2 for the sake of a fair comparison. In order to compare results of each algorithm, the average and best solutions are noted. Results under the label “APP” referring to asset protection problem’s results and ALNS refers to our new algorithm. The proposed ALNS for the COPTW has built on the work by Roozbeh et al. (2018b), and our newly developed heuristics along with the sensitivity analysis improved the ALNS performance significantly as can be seen in Table 3. Our algorithm is implemented to solve the asset protection instances and results compared to the adaptive large neighbourhood search algorithm developed for the APP by Roozbeh et al. (2018b).

In Table 3, $\Delta_{\text{avg}}$ and $\Delta_{\text{best}}$ are the percentage deviation from the reported results in the literature. Our ALNS improves results for all instances and enhances the average results for the $\Delta_{\text{avg}}$ and $\Delta_{\text{best}}$ by around 2% and 3%, respectively. Table 3 shows that our approach performs more efficiently and validates our algorithm on the existing benchmarks. The computation times are quite close for both methods, while the ALNS invests more time on heuristics that bring improvement. It would appear that keeping the best performing heuristics, while discarding others, identified in the sensitivity analysis has yielded a positive improvement in performance.

4.3. Experiments on small-size instances of the COPTW

For further evaluation, a collection of small-sized benchmark instances was generated and each instance was solved by means of both CPLEX and the ALNS algorithm. Note that in this problem set, Euclidean distance is used and rounded down to the first decimal, while results are rounded to two decimal places.

A summary of the tests for 10 and 12 nodes with 3 and 4 team members on instance sets C100, R100 and RC100 is provided in the Table 4. The sizes of truncated instances are chosen in such a way as to investigate the correlation between the increase in problem size and the growth in computational effort. It is worthwhile to mention that infeasible edges are excluded in the MILP formulations to simplify models for the CPLEX implementation. Also, larger problems with more than 12 nodes could not be solved by CPLEX within the time limit of 5 hours.

Computational times for both CPLEX and ALNS are reported in Table 4. The optimality gaps are shown by $\Delta_{\text{avg}}$ and $\Delta_{\text{best}}$ that represent the percentage gap between the CPLEX optimal solution and ALNS algorithm average and best results. The average gap for $\Delta_{\text{avg}}$ and $\Delta_{\text{best}}$ are 0.06% and 0.00%, a promising performance by the pro-

---

1 All the benchmark instances are available via www.sites.google.com/site/imanrzbh/datasets.
Table 2
Statistics for performance of the removal and insertion algorithms. The problem specific heuristics are indicated with an asterisk (*). Numbers in the Table 2 represent the average results for five instances after ten runs (5 instances x 10 runs). The second column denotes the name of algorithms, the third column illustrates the percentage of total iterations that a corresponding algorithm is chosen. The fourth, fifth and sixth columns show the change in the objective values after exclusion of the corresponding algorithm. A positive value indicates an improvement in the objective value. The last column indicates heuristics chosen to omit (Y) or not (N) in subsequent use of the ALNS.

| Type | # | Algorithm | Average % usage | Average % dev in worst obj value | Average % dev in avg obj value | Average % dev in best obj value | Exclusion |
|------|---|-----------|----------------|-------------------------------|-------------------------------|-------------------------------|-----------|
| Insertion | 1 | GT* | 12.98 | -0.36 | -0.33 | 0.69 | N |
| | 2 | NCMB* | 19.00 | 0.06 | 0.08 | 1.00 | Y |
| | 3 | CMB* | 18.73 | -0.31 | -0.37 | -0.24 | N |
| | 4 | NM* | 24.69 | -0.18 | -0.24 | -0.30 | N |
| | 5 | MB* | 24.60 | -0.50 | -0.90 | -0.53 | N |
| Removal | 6 | RR | 3.69 | -0.52 | 1.15 | 1.35 | Y |
| | 7 | WDR | 9.45 | -1.51 | -1.24 | -0.93 | N |
| | 8 | WTR | 6.65 | -1.19 | -0.46 | 0.78 | N |
| | 9 | SR | 8.40 | -0.12 | 0.21 | -0.26 | N |
| | 10 | PR | 8.56 | -1.54 | 0.05 | 0.35 | Y |
| | 11 | TR | 7.67 | -2.03 | -0.27 | 0.15 | N |
| | 12 | RRR | 7.52 | -0.56 | 0.09 | 0.30 | Y |
| | 13 | WTOR | 11.63 | -1.19 | -0.50 | -0.49 | N |
| | 14 | WRR | 6.47 | -1.58 | -0.84 | 0.13 | N |
| | 15 | RRR1 | 8.61 | -0.76 | 0.20 | 0.78 | Y |
| | 16 | RRR2 | 8.02 | -1.00 | -0.22 | 0.49 | N |
| | 17 | HNR | 9.43 | -1.27 | -0.09 | 0.98 | N |
| | 18 | TWR | 2.19 | -0.56 | 0.02 | 0.28 | Y |
| | 19 | CR | 1.71 | -0.06 | 1.16 | 2.47 | Y |

Table 3
A performance comparison of results for 100-nodes. APP represents results for asset protection problem and ALNS shows our results. Vehicle numbers are defined in two categories: Set1 = (V1 = 6, V2 = 5, V3 = 4) and Set2 = (V1 = 7, V2 = 6, V3 = 5). Results in the table are percentage values of scores collected and time is in seconds.

| Instances | # Vehicles | APP | ALNS | Δavg | Δbest |
|-----------|------------|-----|------|------|-------|
|           |            | time | Avg | Best | time | Avg | Best |
| C100      | Set1       | 138.47 | 60.17 | 61.84 | 135.83 | 62.34 | 63.18 | 3.48 | 2.12 |
|           | Set2       | 150.39 | 66.66 | 68.35 | 154.27 | 68.46 | 69.27 | 6.73 | 1.95 |
| C200      | Set1       | 135.48 | 59.04 | 60.72 | 130.64 | 61.06 | 61.92 | 3.31 | 1.94 |
|           | Set2       | 143.92 | 64.87 | 66.58 | 136.67 | 66.73 | 67.73 | 2.79 | 1.70 |
| R100      | Set1       | 134.47 | 61.19 | 62.50 | 125.48 | 63.39 | 64.49 | 3.47 | 3.09 |
|           | Set2       | 138.97 | 68.45 | 69.86 | 131.66 | 70.28 | 71.31 | 2.60 | 2.03 |
| R200      | Set1       | 135.75 | 63.64 | 65.30 | 129.17 | 65.80 | 66.75 | 3.28 | 2.17 |
|           | Set2       | 144.65 | 69.76 | 71.38 | 134.30 | 71.66 | 72.74 | 2.65 | 1.87 |
| RC100     | Set1       | 143.41 | 66.77 | 68.59 | 146.12 | 68.84 | 69.98 | 3.01 | 1.99 |
|           | Set2       | 149.53 | 73.21 | 75.17 | 152.94 | 74.94 | 75.85 | 2.31 | 0.90 |
| RC200     | Set1       | 142.97 | 67.17 | 69.13 | 130.35 | 69.35 | 70.45 | 3.14 | 1.87 |
|           | Set2       | 146.91 | 73.12 | 74.71 | 139.50 | 74.95 | 75.79 | 2.44 | 1.42 |

Table 4
A summary of the ALNS performance for small-size instances for 10 and 12 nodes on C100, R100 and RC100 datasets. All computational times are in seconds.

| Set | # Members | 10 | 12 |
|-----|-----------|---|---|
|     |           | CPLEX | ALNS | Δavg | Δbest | CPLEX | ALNS | Δavg | Δbest |
| C100 | p=3 | 62.15 | 5.48 | 0.00 | 0.00 | 142.37 | 6.86 | 0.00 | 0.00 |
|     | p=4 | 80.16 | 61.24 | 0.00 | 0.00 | 61.75 | 113.21 | 0.00 | 0.00 |
| R100 | p=3 | 244.29 | 4.55 | 0.00 | 0.00 | 5809.95 | 6.91 | 0.00 | 0.00 |
|     | p=4 | 355.91 | 28.78 | -0.02 | 0.00 | 6018.78 | 43.48 | -0.28 | 0.00 |
| RC100 | p=3 | 356.67 | 26.33 | 0.00 | 0.00 | 4502.62 | 21.19 | -0.28 | 0.00 |
|     | p=4 | 277.14 | 60.99 | -0.13 | 0.00 | 3297.12 | 111.34 | -0.10 | 0.00 |

Moreover, it can be seen that the computational time increases significantly with minor changes in the problem size for CPLEX compared with negligible changes for the ALNS.

Table 5, gives a summary of results for 24 and 26 vertices with the same number of available team members. One can see that CPLEX solves larger problems from the sets C200, R200 and RC200 compared to those in Table 4. This is due to the nature of the studied class of problems as the time window intervals are different in length and a larger portion of nodes can be covered by the same number of team members. In Table 5, the average computational time remains around two minutes for all instances, while it takes hours to solve some sets by CPLEX. In Table 5, the average deviation of the ALNS algorithm from optimal solutions is 0.00% for Δavg and just 0.03% for Δaverage which are reasonable. Due to the CPLEX computational time we are unable to solve larger instances in Table 5 while for some instances large amounts of nodes are covered.
Table 5
A summary of the ALNS performance for small-size instances for 24 and 26 nodes on C200, R200 and RC200 dataset. All computational times are in seconds.

| Set   | # Members | 24               | 26               |
|-------|-----------|------------------|------------------|
|       |           | CPLEX  | ALNS  | Δ_{solved} | ALNS  | Δ_{solved} | ALNS  | Δ_{solved} |
| C200  | p = 3     |        |       |            |       |       |       |            |
|       |           | 140.95 | 51.79 | 0.00      | 0.00  | 217.15 | 55.87 | 0.00      |
|       | p = 4     |        |       |            |       |       |       |            |
|       |           | 145.81 | 92.63 | 0.00      | 0.00  | 144.80 | 104.69 | 0.00      |
| R200  | p = 3     |        |       |            |       |       |       |            |
|       |           | 351.03 | 117.21| 0.00      | 0.00  | 473.92 | 128.09 | 0.00      |
|       | p = 4     |        |       |            |       |       |       |            |
|       |           | 224.51 | 132.4 | 0.00      | 0.00  | 286.45 | 134.96 | 0.00      |
| RC200 | p = 3     |        |       |            |       |       |       |            |
|       |           | 509.47 | 130.13| 0.00      | 0.00  | 4516.54| 147.05 | 0.00      |
|       | p = 4     |        |       |            |       |       |       |            |
|       |           | 313.45 | 143.61| 0.00      | 0.00  | 336.61 | 168.52 | 0.00      |

Table 6
Computational results for the largest instances with 100 vertices. Results are shown as the percentage of rewards collected.

| Set   | # Members | 100 worst (%) | average (%) | best (%) | best removal algorithms | best insertion algorithms |
|-------|-----------|---------------|-------------|----------|-------------------------|--------------------------|
|       |           |               |             |          |                         |                          |
| C100  | p = 4     | 35.36         | 35.98       | 36.71    | WDR/HNR/WTR/WTOR        | MB/NMB/GT/NCMB           |
|       | p = 6     | 46.41         | 46.95       | 47.82    | WDR/WTR/TR/SDR          | MB/NMB/NMB/GT            |
| C200  | p = 4     | 72.17         | 73.25       | 74.65    | WDR/HNR/WTR/RTR         | MB/NMB/NMB/GT            |
|       | p = 6     | 85.84         | 87.00       | 88.26    | WDR/HNR/WTR/WTOR        | MB/NMB/NMB/GT            |
| R100  | p = 4     | 32.48         | 33.48       | 34.44    | WDR/WTR/WRR/RRR         | MB/NMB/NMB/GT            |
|       | p = 6     | 42.26         | 43.40       | 44.71    | WDR/RRR1/WTR/WTR        | MB/NMB/NMB/GT            |
| R200  | p = 4     | 74.26         | 75.99       | 78.32    | WDR/WTR/TR/SDR          | MB/NMB/NMB/GT            |
|       | p = 6     | 87.00         | 88.57       | 90.61    | WDR/WTR/RRR1/SDR        | MB/NMB/NMB/GT            |
| RC100 | p = 4     | 30.12         | 30.78       | 31.51    | HNR/WTR/WTOR/WDR        | MB/NMB/NMB/GT            |
|       | p = 6     | 39.96         | 40.74       | 41.51    | WDR/HNR/WTR/RTR         | MB/NMB/GT/NCMB           |
| RC200 | p = 4     | 66.63         | 68.37       | 70.44    | WDR/WTR/WTOR/RRR1/SDR   | MB/NMB/GT/NCMB           |
|       | p = 6     | 81.70         | 83.61       | 86.09    | WTR/WDR/RRR1/SDR        | MB/NMB/GT/NCMB           |

An instance where the ALNS achieved an optimal solution on a small set is demonstrated in Fig. 4. For the sake of better presentation, nodes are assigned into the cells of arrays. The times for starting the service and the corresponding time windows for each succeeding vertex are provided. Consider, for example Fig. 4, three members leave the depot and arrive at node number 5 within its time window [15,67], where they start the service simultaneously to collect the associated score at time 15.1. After that, second and third members leave node 5 toward vertex number 3. Finally, all the team members finish their tour by returning to the depot before T_{\text{max}} = 1236, where T_{\text{max}} is the closing time at the final depot, time budget constraint. It can be seen that the illustrated routes are highly dependent such that any minor changes in sequence of nodes at any tour, requires reconstruction of other routes.

4.4. Experiments on large-size instances of the COPTW

We performed further experiments on larger instances. Table 6 shows the computational results with 4 and 6 team members for 100 nodes on instance sets C100, C200, R100, R200, RC100 and RC200. We utilise the ALNS benchmark since no benchmark exists for the COPTW. Proportional to the problem size more team members are considered in order to cover a substantial percentage of available scores. Our results demonstrate the worst, average and best percentage value of scores collected for each set. In the last two columns of Table 6, we report on the most effective insertion and removal algorithms for each set of instances. Merit-based heuristics and noise-imposed cumulative merit-based are recognised as the best insertion algorithms for all members and sets. It shows that using the merit-based heuristic directly, improves the value of objective function.

5. Conclusion

The COPTW is an important class of the orienteering problem that arises naturally in many important applications, such as home health care problems, the asset protection problem, UAV problems, and in other emergency logistics management. In practical problems many of these applications require solutions as a matter of urgency. Current algorithms are not designed to handle the complexities resulting from the requirements of synchronised visits in the COPTW. Thus, we initially explained handling the synchronous visits within an algorithmic framework in Section 2.2. Then, we developed an ALNS algorithm equipped with variants of the MB heuristic as insertion operators to deal with the various issues that arise from this problem. Together with the MB heuristic 19 heuristics were implemented using the problem specific attributes, where five of them are newly introduced. Thereafter, we evaluated the effectiveness of the removal and insertion algorithms to improve the efficiency of the ALNS by investing more time on effective heuristics. The tailored ALNS algorithm can solve large-sized problems...
in computational times suitable for operational practice. The average time spent to solve each instance with 100 nodes in Table 6 was 193.6 seconds, which is a time suitable for operational purposes.

In order to validate our solution approach, we modified the algorithm to implement tests on the existing benchmarks. We conducted tests on the APP’s instances, a special case of the COPTW. In the APP any minor improvements in the objective function is significant and important as it means that more assets can be protected. Our algorithm improved the existing benchmarks using modest computational resources. For further evaluation, a new benchmark set was generated for the COPTW. The performance of the algorithm was validated successfully for small-sized instances by comparing solutions with the optimal results obtained by the CPLEX solver. Further experiments with large scale problems demonstrated the efficacy as well as accuracy of the ALNS in terms of achieved objective values and computation times.

As the applications grow in the cooperative orienteering class of problems, other developments may be needed to handle complexities such as soft time windows and real-time changes to routes. The ALNS algorithm is a powerful tool which further constraints might be added to deal with such problems.

Declarations

Author contribution statement

I. Roozbeh: Conceived and designed the experiments; Performed the experiments; Analyzed and interpreted the data; Wrote the paper.
J.W. Hearne, D. Pahlavan: Conceived and designed the experiments; Analyzed and interpreted the data; Wrote the paper.

Funding statement

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Competing interest statement

The authors declare no conflict of interest.

Additional information

Data associated with this study has been deposited at https://sites.google.com/site/imanrzrbh/datasets.

Acknowledgements

The authors would like to acknowledge and thank two anonymous referees for the helpful comments which has led to significant enhancement of this paper.

References

Abbaspour, R.A., Samadzadegan, F., 2011. Time-dependent personal tour planning and scheduling in metropolises. Expert Syst. Appl. 38 (10), 12439–12452.
Bertuccelli, L., Alighanbari, M., How, J., 2004. Robust planning for coupled cooperative uav missions. In: 2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No.04CH37601), vol. 3. IEEE, pp. 2917–2922.
Bianchini, N., Manzini, R., Speranza, M.G., 2018. A branch-and-cut algorithm for the team orienteering problem. Int. Trans. Oper. Res. 25 (2), 627–635.
Borràs, J., Moreno, A., Vallà, A., 2014. Intelligent tourism recommender systems: a survey. Expert Syst. Appl. 41 (16), 7370–7389.
Bouly, H., Dang, D.-C., Moukrim, A., 2010. A memetic algorithm for the team orienteering problem. OR 8 (1), 49–70.
Brederstrøm, D., Rönqvist, M., 2008. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. Eur. J. Oper. Res. 191 (1), 19–31.
Clarke, J., Gascón, V., Ferland, J.A., 2017. A capacitated vehicle routing problem with synchronized pick-up and drop-offs: the case of medication delivery and supervision in the dr Congo. IEEE Trans. Eng. Manag. 64 (3), 327–336.
Dang, D.-C., El-Hajj, R., Moukrim, A., 2013a. A branch-and-cut algorithm for solving the team orienteering problem. In: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Springer, pp. 332–339.
Dang, D.-C., Gulbadi, R.N., Moukrim, A., 2013b. An effective pso-inspired algorithm for the team orienteering problem. Eur. J. Oper. Res. 229 (2), 332–344.
De Falco, I., Scafuri, U., Tarantino, E., 2015. A multiobjective evolutionary algorithm for personalized tours in street networks. In: European Conference on the Applications of Evolutionary Computation. Springer, pp. 115–127.
Drexel, M., 2012. Synchronization in vehicle routing—a survey of vrp with multiple synchronization constraints. Transp. Sci. 46 (3), 297–316.
Duque, D., Llozano, L., Madalgalia, A.L., 2015. Solving the orienteering problem with time windows via the pulse framework. Comput. Oper. Res. 54, 168–176.
Emeg, U., Catay, B., Bozkaya, B., 2014. An adaptive large neighborhood search for an e-grocery delivery routing problem. Comput. Oper. Res. 69, 109–125.
Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al., 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd, vol. 96, pp. 226–231.
Evers, L., Barros, A.I., Monsuur, H., Wagelmans, A., 2014a. Online stochastic uav mission planning with time windows and time-sensitive targets. Eur. J. Oper. Res. 238 (1), 348–362.
Evers, L., Barros, A.I., Monsuur, H., Wagelmans, A., 2015. Uav mission planning from robust to agile. In: Military Logistics. Springer, pp. 1–17.
Evers, L., Delevoet, T., Barros, A.I., Monsuur, H., 2014b. Robust uav mission planning. Ann. Oper. Res. 221 (1), 293–315.
Evers, L., Glorie, K., Van Der Sier, S., Barros, A.I., Monsuur, H., 2014c. A two-stage approach to the orienteering problem with stochastic weights. Comput. Oper. Res. 43, 248–260.
Fink, M., Desaulniers, G., Frey, M., Kriemhaar, F., Kolisch, R., Soumis, F., 2019. Column generation for vehicle routing problems with multiple synchronization constraints. Eur. J. Oper. Res. 272 (2), 699–711.
Gambardella, L.M., Montemanni, R., Weyland, D., 2012. Coupling ant colony systems with strong local searches. Eur. J. Oper. Res. 220 (3), 831–843.
Garcia, A., Arbelaitz, O., Vansteenwegen, P., Souffriau, W., Linaza, M.T., 2010. Hybrid approach for the public transportation time dependent orienteering problem with time windows. In: International Conference on Hybrid Artificial Intelligence Systems. Springer, pp. 151–158.
Gavala, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., Tasoulas, Y., 2013. Cluster-based heuristics for the team orienteering problem with time windows. In: International Symposium on Experimental Algorithms. Springer, pp. 390–401.
Golden, B.L., Levy, L., Vohra, R., 1987. The orienteering problem. Nav. Res. Logist. 34 (3), 307–318.
Gunawan, A., Lau, H.C., Lu, K., 2015. An iterated local search algorithm for solving the orienteering problem with time windows. In: European Conference on Evolutionary Computation in Combinatorial Optimization. Springer, pp. 61–73.
Gunawan, A., Lau, H., Vansteenwegen, P., 2016. Orienteering problem: a survey of recent variants, solution approaches and applications. Eur. J. Oper. Res. 255 (2), 315–332.
Gunawan, A., Lau, H.C., Vansteenwegen, P., Lu, K., 2017. Well-tuned algorithms for the team orienteering problem with time windows. J. Oper. Res. Soc. 68 (8), 861–876.
Keshkarkaran, M., Ziarati, K., Bettinelli, A., Vigo, D., 2016. Enhanced exact solution methods for the team orienteering problem. Int. J. Prod. Res. 54 (2), 591–601.
Kim, B.-I., Li, H., Johnson, A.L., 2013. An augmented large neighborhood search method for solving the team orienteering problem. Expert Syst. Appl. 40 (8), 3065–3072.
Labadie, N., Mansini, R., Melechovsky, J., Calvo, R.W., 2012. The team orienteering problem with time windows: an lp-based granular variable neighborhood search. Eur. J. Oper. Res. 220 (1), 15–27.
Li, J., Yang, X., Li, H., 2013. A multi-level variable neighborhood search approach to the orienteering problem. J. Ind. Prod. Eng. 30 (4), 238–247.
Lin, S.W., Vincent, F.Y., 2012. A simulated annealing heuristic for the team orienteering problem with time windows. Eur. J. Oper. Res. 217 (1), 94–107.
Mankowska, D.S., Meisel, F., Bierwirth, C., 2014. The home health care routing and scheduling problem with interdependent services. Health Care Manage. Sci. 17 (1), 15–30.
Marianakis, Y., Politi, M., Marinaki, M., Matsatsinis, N., 2015. A memetic-grasp algorithm for the solution of the orienteering problem. In: Modelling, Computation and Optimization in Information Systems and Management Sciences. Springer, pp. 105–116.
Mei, Y., Salim, P.D., Li, X., 2016. Efficient meta-heuristics for the multi-objective time-dependent orienteering problem. Eur. J. Oper. Res. 254 (2), 443–457.
Montemanni, R., Gambardella, L.M., 2009. An ant colony system for team orienteering problems with time windows. Found. Comput. Decision Sci. 34 (4), 287.
Poggi, M., Viana, H., Uchoa, E., 2010. The team orienteering problem: formulations and branch-and-cut price. In: OASiSs-OpenAccess Series in Informatics, vol. 14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
Roozbeh, I., Hearne, J., Abbasi, B., Oden, M., 2018a. A two-stage stochastic approach for the asset protection problem during escaped wildfires with uncertain timing of a wind change. arXiv preprint arXiv:1811.03272.
Roozbeh, I., Oden, M., Hearne, J.W., 2016. A heuristic scheme for the cooperative team orienteering problem with time windows. arXiv preprint arXiv:1608.05485.
Roozbeh, I., Oden, M., Hearne, J.W., 2018b. An adaptive large neighborhood search for asset protection during escaped wildfires. Comput. Oper. Res. 97, 125–134.
Ropke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp. Sci. 40 (4), 455–472.

Santini, A., 2018. An Adaptive Large Neighbourhood Search Algorithm for the Orienteering Problem.

Ropke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp. Sci. 40 (4), 455–472.

Santini, A., Ropke, S., Hvattum, L.M., 2018. A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic. J. Heuristics 24 (5), 783–815.

Santini, A., 2018. An Adaptive Large Neighbourhood Search Algorithm for the Orienteer Problem.

Santini, A., Ropke, S., Hvattum, L.M., 2018. A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic. J. Heuristics 24 (5), 783–815.

Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., Van Oudheusden, D., 2013. The multiconstraint team orienteering problem with multiple time windows. Transp. Sci. 47 (1), 53–63.

Tricoire, F., Romauch, M., Doerner, K.F., Hartl, R.F., 2010. Heuristics for the multi-period orienteering problem with multiple time windows. Comput. Oper. Res. 37 (2), 351–367.

Van Der Merwe, M., Minas, J., Ozlen, M., Hearne, J., 2014. The cooperative orienteering problem with time windows. Optimization (Online) 7, 11.

VanLaarhoven, P.J., Aarts, E.H., 1987. Simulated annealing. In: Simulated Annealing: Theory and Applications. Springer, pp. 7–15.

Vansteenwegen, P., Souffriau, W., Berge, G.V., Van Oudheusden, D., 2009. Iterated local search for the team orienteering problem with time windows. Comput. Oper. Res. 36 (12), 3281–3290.

Vansteenwegen, P., Souffriau, W., Van Oudheusden, D., 2011a. The city trip planner: an expert system for tourists. Expert Syst. Appl. 38 (6), 6540–6546.

Vansteenwegen, P., Souffriau, W., Van Oudheusden, D., 2011b. The orienteering problem: a survey. Eur. J. Oper. Res. 209 (1), 1–10.

Vansteenwegen, P., Van Oudheusden, D., 2007. The mobile tourist guide: an or opportunity. OR Insight 20 (3), 21–27.

Verbeeck, C., Vansteenwegen, P., Aghezzaf, E.H., 2014. An extension of the arc orienteering problem and its application to cycle trip planning. Transp. Res., Part E, Logist. Transp. Rev. 68, 64–78.

Vidal, T., Maculan, N., Ochi, L.S., Vaz Penna, P.H., 2015. Large neighborhoods with implicit customer selection for vehicle routing problems with profits. Transp. Sci. 50 (2), 720–734.

Zhang, M., Qin, J., Yu, Y., Liang, L., 2018. Traveling salesman problems with profits and stochastic customers. Int. Trans. Oper. Res. 25 (4), 1297–1313.