An Adaptive Large Neighbourhood search for Asset Protection During Escaped Wildfires

Iman Roozbeh*, Melih Ozlen, John W. Hearne

School of Science, RMIT University, Melbourne, Australia

Abstract

The Asset Protection problem is encountered where an uncontrollable fire is sweeping across a landscape comprising important infrastructure assets. Protective activities by teams of firefighters can reduce the risk of losing a particular asset. These activities must be performed during a time-window for each asset determined by the progression of the fire. The nature of some assets is such that they require the simultaneous presence of more than one fire vehicle and its capabilities must meet the requirements of each asset visited. The objective is then to maximise the value of the assets protected subject to constraints on the number and type of fire trucks available. The solution times to this problem using commercial solvers preclude their use for operational purposes. In this work we develop an adaptive large neighbourhood search algorithm (ALNS) based on problem-specific attributes. Several removal and insertion heuristics, including some new algorithms, are applied. A new benchmark set is generated by considering the problem attributes. In tests with small instances the ALNS is shown to achieve optimal, or near optimal, results in a fraction of the time required by CPLEX. In a second set of experiments comprising larger instances the ALNS was able to produce solutions in times suitable for operational purposes. These solutions mean that significantly more assets can be protected than would be the case otherwise.

Keywords: Asset Protection Problem, Wildfire, Adaptive Large Neighbourhood Search, Vehicle Routing Problem, Synchronisation Constraint

1. Introduction

Wildfires are a world-wide natural phenomenon but increasingly pose a threat to life and property. For example, in recent years devastating wildfires have occurred in Alaska and Indonesia (2015), Canada, California and Spain (2016), and Chile, Portugal and again in California (2017). During wildfires Incident Management Teams (IMT's) face complex operational problems. Although there are many studies concerned with the dispatch of resources for direct fire suppression (Duff and Tolhurst (2015)), relatively few studies address the operational challenges of asset protection and defensive tasks (see Donovan and Rideout (2003), Pappis and Rachaniotis (2010), Van Der Merwe et al. (2015), Arrubla et al. (2014) and van der Merwe et al. (2017)).

We consider the problem where an uncontrollable fire is sweeping across a landscape comprising strategically important assets such as bridges, electric substations, and schools. The risk of losing a particular asset can be reduced through protection activities by teams of firefighters operating shortly before the fire reaches that particular asset. This involves activities such as hosing down a structure or removing debris in close proximity to an asset. The number and type of fire-truck teams that need to visit each asset depends on their size and nature. Each asset has a strict time window during which these teams must visit. These time-windows are determined using fire-spread models and the most recent meteorological data. Obviously the teams need to be clear of an asset before the runaway fire reaches it. Conversely, protection effects can be nullified if undertaken too early. For example, debris

*Corresponding author

Email addresses: iman.roozbeh@rmit.edu.au (Iman Roozbeh), melih.ozlen@rmit.edu.au (Melih Ozlen), john.hearne@rmit.edu.au (John W. Hearne)

can accumulate again.

The asset protection problem has similarities to the Vehicle Routing Problem (VRP). Vehicles need to be deployed to visit as many assets as possible during their respective time-windows. The problem differs in that there is some order to the time-windows of the assets imposed by the advancing fire fronts and that it is often not possible to visit every asset. Moreover, some assets have a need for the synchronous visit of more than one truck. Different assets have different requirements. Some are only accessible with a 4x4 vehicle, for example. Trucks visiting a particular asset must have the right capability to meet the requirements of the asset. Also significant is that, unlike most VRP's, deployment decisions need to be made by IMT's under severe time-pressure.

The asset protection problem for wildfires was first introduced by Van Der Merwe et al. (2015). A Mixed Integer Programming model was formulated and solved using CPLEX. Although optimal solutions were achieved in most cases the solution times precluded the method being used for operational purposes. It is the aim of this paper to address this NP-hard problem and find a method of achieving good solutions in times that make it suitable for operational purposes. We propose an Adaptive Large Neighbourhood Search (ALNS) metaheuristic which provides a robust framework for solving large size instances that IMTs may encounter in cases of extensive wildfires.

The remainder of this paper is organised as follows. In the next section we present the problem as a 0-1 mixed-integer linear programming model and describe the problem. In section 3, we present our ALNS algorithm for solving the asset protection problem. In section 4, the results of extensive tests are presented. Finally, the conclusion and potential future research directions are stated in section 5.

2. Problem Description

2.1. Mathematical Model

The asset protection problem considers a set of $N-2$ assets, each of which should be served by a combination of vehicle types $q \in Q$, while only P_q vehicles of type q are available. Let $V = \{v_1, \ldots, v_N\}$ denote the set of nodes. Nodes 1 and N may denote the same depot if every route starts and ends at the same location. The other $N-2$ vertices represent assets. The node i is marked as a visited asset if $R_i =< r_{i1}, r_{i2}, \ldots, r_{iq} >$ vehicles of each type arrive at the vertex i within the associated time window $[o_i, c_i]$ to perform a synchronised visit. The service at node i starts at time s_i for a duration of a_i time units to protect an asset of value ψ_i. To travel from nodes i to j by a vehicle of type q, t_{ijq} units of time are required. The binary decision variable y_i, takes the value 1 if the vehicles assigned to the asset collectively meet the protection requirements R_i, 0 otherwise. The decision variable z_{ijq} takes the value 1 if arc ij is traversed by vehicle type q, otherwise 0. Moreover, x_{ijq} represents the number of vehicles of type q travelling on arc ij. Finally, A_i^{q+} defines a set of arcs $\{i, j\}$ for each node j such that a vehicle of type q departing from node i at time $o_i + a_i$ can reach vertex j by time c_j. Similarly, the set A_i^{q-} represents the set of feasible arcs to traverse to node i. With the notation above, the problem is formulated as the following mixed integer programming model.

$$Maximise \sum_{i=2}^{N-1} \psi_i y_i \tag{1}$$

$$s.t.: \sum_{(1,j)\in A_1^{q+}} x_{1jq} = \sum_{(i,N)\in A_N^{q-}} x_{iNq}, \quad q \in Q, \tag{2}$$

$$\sum_{(i,k)\in A_k^{q-}} x_{ikq} = \sum_{(k,j)\in A_k^{q+}} x_{kjq},\ k = 2, \ldots, N-1, \quad q \in Q, \tag{3}$$

$$r_{kq} y_k = \sum_{(i,k)\in A_k^{q+}} x_{ikq},\ k = 2, \ldots, N-1, \quad q \in Q, \tag{4}$$

$$x_{ijq} \le P_q z_{ijq},\ (i,j) \in A_i^{q+}, \quad q \in Q, \tag{5}$$

$$s_i + t_{ijq} + a_i - s_j \le M(1 - z_{ijq}),\ (i,j) \in A_i^{q+},\ q \in Q, \tag{6}$$

$$o_i \le s_i,\ i = 1, \ldots, N, \tag{7}$$

$$s_i \le c_i,\ i = 1, \ldots, N, \tag{8}$$

$$x_{ijq} \in \{0, 1, \ldots, P_q\},\ (i,j) \in A_i^{q+}, \tag{9}$$

$$y_i, z_{ijq} \in \{0, 1\},\ (i,j) \in A_i^{q+}. \tag{10}$$

The objective function maximises the total value of the protected assets. Constraints (2) ensure all vehicles of each type start and end their route at the depot. Constraint (3) guarantees flow conservation for each vehicle type by enforcing the equality of incoming and outgoing traffic on the arcs to each node. Constraint (4) guarantees an asset is protected only if its protection requirements are satisfied by incoming vehicles. Constraint (5) makes sure that the number of trucks of type q never exceed the number available P_q. Constraint (6) ensures that an asset can be visited when the protection requirements of the previous location has been satisfied and there is enough time to reach the next asset. Constant M is a large number. Setting $M = max(o_i) + max(t_{ij}) + max(a_i) - min(c_i)$ is sufficiently large for our purpose. Terms (7) and (8) ensure that the time window constraints are not violated. Integer and binary conditions are defined in constraints (9) and (10).

2.2. An Illustrative Example

To illustrate the asset protection problem on a small example consider the problem settings in Fig.1. The depot is denoted as "D" and associated protection values are defined in each vertex. Note that, while vehicles depart from the central depot, due to the fire advancement they need to return to another depot, not threatened by fire. There are three types of vehicles, namely tanker, pumper and aerial vehicle each of which are defined by a unique array. Vehicle types are characterised based on the operational fleet of vehicles available. A binary vector is used to represent the capabilities of each vehicle type. The protection requirements of assets are uniformly selected from the set of vectors $R_i = \{< 2,0,0 >, < 0,2,0 >, < 0,0,2 >, < 1,1,0 >, < 0,1,1 >, < 1,0,1 >, < 1,1,1 >\}$ where each member of a vector represents the required number of each vehicle type to protect an asset.

To represent a realistic scenario time windows translate the anticipated remaining time to the fire impact while the fire front spreads in a circular manner as defined in Fig. 2. Therefore, the opening time of each asset is $o_i = \frac{\sqrt{x^2+y^2}}{firevelocity}$ and the latest service time $c_i = o_i + E$, where E is the time duration in which the protection activities have to be carried out. Time windows are correlated with the coordinates in the Euclidean space. Taking that into consideration, the planning horizon (T_{max}) is equal to c_i for the furthest asset from the origin of the fire. Moreover, traversing each arc is a function of distance and vehicle velocity $t_{ij} = \frac{d_{ij}}{vehiclespeed}$. To present a real-life situation we set $E = 2\ hours$, $firevelocity = 10\ km/h$, $vehiclesspeed = 40\ km/h$ and $a = 1\ hour$.

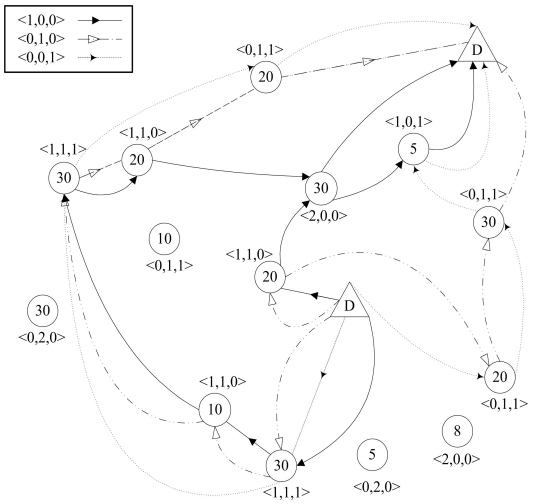

Figure 1: An illustrative example. Assets and depots are defined in circular and triangular shapes, respectively.

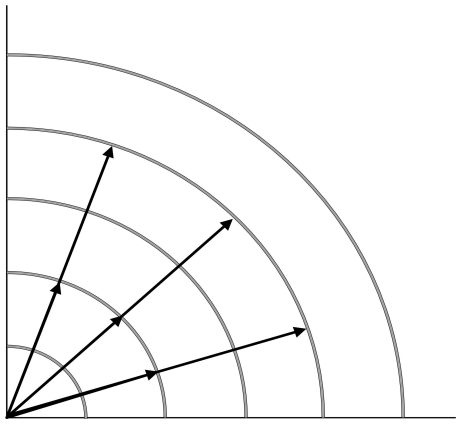

Figure 2: Direction of fire spread

In the graphical representation of the problem (Fig. 1), some assets are not protected. This is because of the time windows imposed by the advancing fire front (see Fig 2). It means that sufficient number of vehicles cannot arrive at those locations by their latest service time. Therefore, a selection of assets must be made that maximises the total value protected. To protect an asset all resource requirements must be present at the asset before the protection operation can commence simultaneously and cooperatively.

Through the asset protection activities, some disruptions or changes to conditions may occur which necessitate rerouting of vehicles. To deal with various disruptions and changing conditions the problem needs to be studied with a dynamic approach. Some of the numerous disruptions that may occur during wildfires are changes in wind speed, wind direction, relative humidity, temperature. Although meteorologist can feed the IMTs with highly reliable data by using advance equipment, changes in weather conditions should never be under-estimated, as they severely impact the speed, intensity and direction of wildfire spread. Other disruptions, such as vehicles breakdowns, change in road conditions and travel times might also take place in wildfire scenarios. Given the time-critical nature of wildfire response, it is important that asset protection plans are updated and implemented as quickly as possible following a disruption. To illustrate a dynamic scenario in an asset protection problem, the following problem is considered and solved by CPLEX. Figures 3 and 4 show the affect of change in the direction of fire spread on assets being impacted and the need for rerouting. When a change in wind direction occurs, rerouting of vehicles take place from the assets last visited before the disruption. The rerouting attempts to cover assets within their updated time windows according to the change in wind direction.

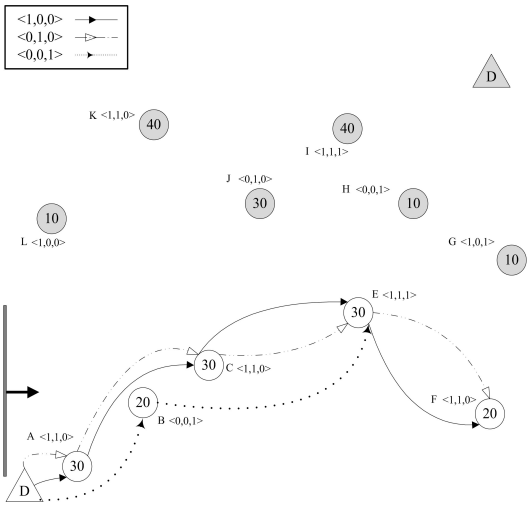

Figure 3: An illustrative example. Assets and depots are defined in circular and triangular shapes, respectively. Assets that are not under threat are shaded as grey, and the bold line shows the direction of fire spread.

In Figure 3, fire spreads at a rate of 10 $km.h^{-1}$

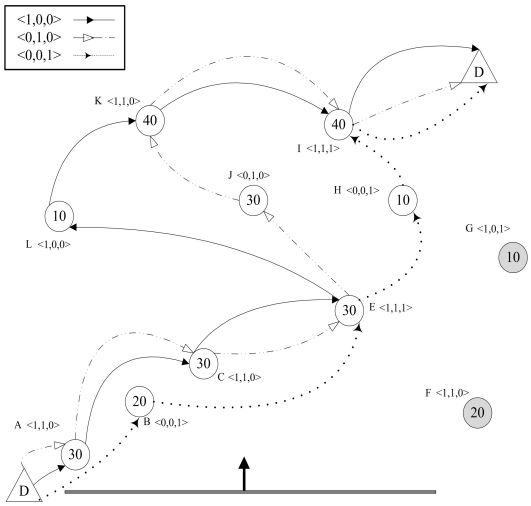

Figure 4: An illustrative example. Assets and depots are defined in circular and triangular shapes, respectively. Assets that are not under threat are shaded as grey, and the bold line shows the direction of fire spread.

in a linear fashion from left to right. As a result of that five out of eleven assets are evaluated at risk. While the primary routes are planned as can be seen in Fig. 3, a disruption occurs when vehicles are at node E and a rerouting is required. In Figure 4, the wind direction and consequently the direction of the fire front changes before asset protection operations being completed, as planned under the primary information. The change in the wind direction necessitates updating assets that need to be protected. In Figure 4, the fire front sweeps over assets in a vertical manner, unlike Figure 3. By comparison of Figure 3 and 4, it can be observed that a new set of assets demand protection while asset F is no longer at risk. Once the status of assets along with their time windows were updated, rerouting was performed and resources were sent to the assets at risk under the new scenario.

The approach developed in this paper can be used to solve the asset protection problem in a dynamic manner in a similar way to that presented in the illustrative example above. Our solution approach can handle changes in conditions that might arise during operation and require rerouting. Although unusual, in some cases rerouting might be required more than once through the course of a protection operation. Considering the computational resources required to solve each problem by commercial solvers, the efficiency of our algorithm is an important and practical tool for IMTs operating

under such circumstances with tight time limits.

3. Proposed Methodology

To solve real size instances within suitable operational times, we propose an Adaptive Large Neighbourhood Search (ALNS) which provides a powerful algorithmic framework. In this section, we describe the general framework of the algorithm followed by details on the problem-specific heuristics.

3.1. Overview of the ALNS metaheuristic

The ALNS paradigm, introduced by Ropke and Pisinger (2006) which extends the large neighbourhood search previously put forward by Shaw (1998). Compared to many local search heuristics by which only minor changes can be applied on the solution, the ALNS brings a larger search space into consideration. Within one iteration, ALNS can rearrange up to 40% of a solution. This attribute is particularly useful with tightly constrained routing problems. Suppose, for example, we have a VRP with 100 nodes where the degree of destruction is 40%. There are $C(100, 40) = 100!/(40!, 60!) = 1.4 \times 10^{28}$ alternative ways to remove the customers. This very specification leads to moving between promising areas in the feasible region and avoiding getting stuck in local optima during the search. The outstanding performance of ALNS in solving various scheduling and routing problems has been demonstrated. In a subsequent study, Pisinger and Ropke (2007) showed that the improved ALNS algorithm gives promising results for different VRP variants. Since then, ALNS has been used to solve variants of routing problems, e.g. the periodic inventory routing problem (Aksen et al. (2014)), VRP with multiple routes (Azi et al. (2014)), distribution problem of perishable products (Belo-Filho et al. (2015)), e-grocery delivery routing problem (Emeç et al. (2016)), share-a-ride problem (Li et al. (2016)), railway line planning problem (Canca et al. (2017)), and cross-dock selection (Maknoon and Laporte (2017)). Our developed algorithm brings in a set of destroy (h_d) and repair (h_r) heuristics. These heuristics are either introduced by authors for efficiently handling the problem side constraints or are adapted versions of the existing heuristics, mostly proposed by Demir et al. (2012); Ropke and Pisinger (2006); Emeç et al. (2016). The problem specific heuristics are indicated with an asterisk (*) when they are introduced. Please note that even the heuristics used by applying modifications to the existing algorithms in the literature incorporate new terms and ideas. We now describe the general framework of our proposed ALNS approach below.

3.1.1. Initial solution construction

Roozbeh et al. (2016) proposed a heuristic to solve the cooperative orienteering problem with time windows named as the Modified Clarke and Wright heuristic (MCW). The MCW is adapted to construct the initial solution in an efficient manner. MCW combines the strengths of classical CW heuristic with a sweep algorithm while trying to maximise the total award by an additional term in the saving function. The pseudo-code of the construction of the initial solution is described in Algorithm 1.

In Algorithm 1, at first, a saving pair list is initialised. As far as the parameters in the saving function are concerned, constant values for $(\lambda, \mu, \vartheta)$ triplets are defined, based on Roozbeh et al. (2016) study. The first term of the savings function enhances the reshaping ability of the classical Clarke and Wright heuristic and its circumference characteristic. The second term aims to protect assets with higher values earlier than the rest. Motivation of the last term is to give early placement to pairs in vicinity of the depot by including $\cos\theta_{ij}$, which is the value of constructed angles between pairs. After that, transitive closures are computed by considering the time windows and asset protection requirements of assets. So that there is an arc connecting any two vertices that have resource requirements in common and can be reached within their time windows. Algorithm 1 returns α_{best} and β which are the total value of the protected assets and the best set of routes. Note that, *velocity* and T_{max} refer to the vehicle speed and the planning horizon.

Synchronisation constraints in vehicle routing problems have been investigated by Afifi et al. (2016) and Drexl (2012) developed heuristic solutions. The asset protection problem is an interdependence problem due to the service synchronisation. It means that routes are highly dependent in the sense that any minor change in orientation of the nodes in any tour necessitates a check of every single constraint for all routes. This is because insertion and removal of any node within the routes may impact on the arrival and service start time in all the tours. Therefore, to evaluate the feasibility of insertion at any point in a constant time, a

Algorithm 1: Pseudocode for the initial solution

Input: vector of temporary routes $\tau = (\tau_1, \tau_2, \tau_3)$, best routes (β), best collected reward (α_{best}), set of all nodes (ν), Saving Pair List *(SPL)*, unvisited vertices (U), vector of resource requirement $R_i = (r_{i1}, r_{i2}, r_{i3})$, distance matrix $(d_{\nu\times\nu})$, number of routes by vehicle type $q \in Q$ (n_q)

Output: α_{best} *and* β

```
function MCWheuristic
    forall ((i, j) ∈ ν) do
        S_{i,j} ← (d_{i0} + d_{0j} − λ d_{ij}) / d^max + ϑ * (ψ_i + ψ_j) / ψ̄ +
                  μ * (cos θ_{ij} |d^max − (d_{i0} + d_{0j})/2|) / d^max
    insert S_{i,j} to a vector of tuples (i, j, S_{i,j})
     //initialising SPL
    Sort SPL in descending order of S_{i,j} values
    forall (q ∈ Q) do
        forall ((i, j) ∈ ν) do
            if ((o_j + a_j + d_{j0}/velocity ≤ T_max)
             && (o_i + a_i + d_{ij}/velocity ≤ c_j)
             && (r_{iq} ≠ 0 && r_{jq} ≠ 0)) then
                feas_{q,i,j} ← 1 else
                feas_{q,i,j} ← 0
    forall (q ∈ Q) do
        forall (pairs ∈ SPL) do
            if (feas_{q,i,j} == 1) then
                assign (j ∈ U) to
                 (subroutes ∈ τ_q)
                VisitCount_{jq} ←
                 VisitCount_{jq} + 1
            else
                if (n_q < P_q) then
                    Open a new route and add
                     (j ∈ U)
                    n_q ← n_q + 1
                    VisitCount_{jq} ←
                     VisitCount_{jq} + 1
            if (R_i == satisfied) then
                Update U, α_best and β
    return α_best and β
```

calculation needs to be performed initially and then updated after each insertion of visits. For each node i, we define $maxshift_i$ to memorise the allowed delay in arrival to node i where an unvisited node get inserted before i. To find $maxshift_i$ the following variables are defined and should be calculated beforehand.

$$arrive_i = departure_{i-1} + traveltime_{i-1,i} \quad (11)$$

Due to the synchronised visit, each node may need to be visited within multiple routes and $i \in \mathrm{T}$ represent the routes that node i belongs to. o_i in the following equation refers to the opening of the time window.

$$start_i^{sync} = max\{\max_{i\in \mathrm{T}} arrive_i, o_i\} \quad (12)$$

By having the synchronised start at node i the departure time is as below.

$$departure_i = start_i^{sync} + a_i \quad (13)$$

Subsequently the waiting time at node i can be calculated as follow.

$$wait_i = start_i^{sync} - arrive_i \quad (14)$$

For a given route τ, a visit at node i is defined by $\tau(i)$. The value of $maxshift_{\tau(i)}$ in the equation 15 is equal to the time that the arrival at point i can be delayed while the feasibility conditions are met. This amount of delay is equal to the summation of $wait_{\tau(i+1)}$ and $maxshift_{\tau(p+1)}$ unless it violates the time window bound.

$$maxshift_{\tau(i)} = min\{c_{\tau(i)} - start_{\tau(i)}, \; wait_{\tau(i+1)} + maxshift_{\tau(i+1)}\} \quad (15)$$

The value of the $maxshift_{\tau(i)}$ in the equation 15 must be calculated in a backward manner. It means that we start our calculation from the last visit in each route where $maxshift$ for the subsequent visit (depot) can be calculated independent of other nodes. On the other hand, as visits need to be synchronised, the minimum value of $maxshift$ for each node in existing routes has to be taken. Therefore, for the $maxshift_{\tau(i)}$ if there exist $i+1 \in V$ such that $\{\tau(i), \tau(i+1)\} \in \mathrm{T}$ we have:

$$maxshift_{\tau(i)}^{sync} = min\{maxshift_{\tau(i)}, \; \min_{i\in \mathrm{T}} maxshift_{\tau(i+1)}\} \quad (16)$$

To define whether an insertion of a node z between i and $i+1$ in route r is feasible, we need to calculate the generated shift ($shift_z^{\tau,i}$).

$$shift_z^{\tau,i} = traveltime_{i,z} + wait_z + servicetime_z + traveltime_{z,i+1} - traveltime_{i,i+1} \quad (17)$$

If the value of $shift_z^{\tau,i}$ is less than or equal to the $wait_{i+1} + maxshift_{\tau(i+1)}^{sync}$, the insertion will be considered valid.

Since the visits have to be synchronised, an update is required through all routes after each insertion. Transitive closures (Aho et al. (1972)) are used in order to filter infeasible arcs to avoid infinite loops. Arcs that connect nodes with no resource requirements in common are infeasible to traverse. For example, travelling from i to j should be marked as infeasible where i only needs two visits by vehicle type1 and j two visits by vehicle type2 for protection. Moreover, cross synchronisation needs to be filtered out, e.g. when node j is visited after i by the first vehicle, the visit of i after j should be prohibited in other routes. Also, we filter out the arc_{ij} when $o_i + a_i + t_{ij} > c_j$, at the preprocessing step of the algorithm.

3.1.2. General flow

Initially, a feasible solution $S_0 \leftarrow \beta$ is formed by using the MCW heuristic. At iteration i, a removal heuristic $d \in h_d$ is selected dynamically and adaptively to destroy the the current feasible solution partially. Then the resulting solution S_i^- undergoes for reconstruction with the hope of improving the objective function by choosing a repair heuristic $r \in h_r$ based on a calculated probability. The new solution S_i^+ is a temporary feasible solution which can be discarded or replaced with the best current solution according to the change in the objective function. The performance of the heuristics will be recorded to use in the next iterations for dynamic and adaptive updates of the selection probabilities.

3.1.3. Adaptive weight adjustment procedure

There is no heuristic that can perform efficiently for all types of problems. Since the asset protection problem is new in nature, it may be difficult to anticipate the performance of a heuristics according to the problem and instance class. The ALNS enables us to pick as many destroy and repair heuristics as we want. Assuming that the past success of the chosen heuristics indicates their future performance, the algorithm assigns a weight to each heuristic based on how they impact the objective function.

The algorithm runs for N number of iterations, divided into k segments. Therefore, the algorithm iterates over each segment for $n = \frac{N}{k}$. Each heuristic $s \in h_d \cup h_r$ is associated with a weight $W(s)$ and a score π_s. Initially, equal weights and score of zero are assigned to all heuristics. We reset the value of π_s to zero before starting each segment. After a solution goes through the destroy and repair process the result will drop into one of the following cases. (1) If the new solution is the best one found so far, the corresponding scores of the repair and destroy heuristics are increased by σ_1. (2) If the new solution improves the current best one but not the best known so far then the scores are increased by σ_2. (3) If the new solution is accepted, even though it is worse than current best one, the scores are incremented by σ_3.

The probability to select a heuristic at each iteration is as below.

$$p(r_s) = \frac{W(r_s)}{\sum_{j=1}^{R} W(r_j)}, p(d_s) = \frac{W(d_s)}{\sum_{j=1}^{D} W(r_j)} \quad (18)$$

In equation 18, weights dynamically and adaptively are adjusted after n iterations according to their performance. At the end of each segment weights are updated as

$$W(h) \begin{cases} (1-\rho)W(h) + \rho\frac{\pi(h)}{u(h)}, & \text{if } u(h) > 0 \\ (1-\rho)W(h), & \text{if } u(h) = 0 \end{cases} \quad (19)$$

,where ρ is a parameter called reaction factor. This parameter can regulate about after how many iterations most ineffective heuristic should not play any substantial role. Thus, for example, if we want to have a 0.01% of the $W_{initial}$ for ineffective heuristics after 1000 iterations, when $N = 10000$ and $n = 100$, the minimum value for ρ can be calculated by equation 20 as, $\frac{1}{1000} > (1-\rho)^{\frac{1000}{100}} \rightarrow \rho \geq 0.602$.

$$W(h) \approx W_{initial}(1-\rho)^{[\frac{N}{n}]} \quad (20)$$

In equation 19, $\pi(h)$ and $u(h)$ record the number of times a heuristic is selected by a roulette-wheel mechanism and associate weight of the heuristic.

3.1.4. Acceptance and stopping criteria

At the master level of the ALNS algorithm, we use an acceptance criterion based on a Simulated Annealing (SA) local search framework (see Van Laarhoven and Aarts (1987)). Let $z(S)$ and $z(S^*)$ denote the objective value of the current solution and the best known solution, respectively. The initial temperature $T_{initial}$ should be set in a way to accept solutions with δ% worse objective value compare to $z(S_{initial})$ with the probability of P_{accept}.

$$T_{initial} = \frac{(S_{initial} * \delta)}{\log(1/P_{accept})} \quad (21)$$

The achieved initial temperature by equation 21 cools down with a fixed cooling rate $0 < \epsilon < 1$, $(T = \epsilon * T)$. Following the SA framework, solutions with worse objective values would be accepted with probability of $\exp(\frac{(z(S)-z(S^*)}{T})$ and those improving the objective value will always be accepted.

3.1.5. Applying noise

Some heuristics may insert each node at its best place iteratively, but locally best moves can increase the chance of getting stuck in local optimum (Ropke and Pisinger (2006)). To apply diversification to the search, we use a noise-imposed insertion heuristic beside the clean insertion which uses the original saving list. Additionally, there is a random removal heuristic among the destroy heuristics which derives a significant amount of randomisation. It is worthwhile to mention that the implementation of noise and randomisation may not always result in a better solution; however it increases the chance of exploring new parts of the search space with the hope of improving the objective function.

3.2. Removal Algorithms

Before a removal heuristic can be used to destroy a solution partially, the algorithm needs to determine the degree of destruction, D. Large values for D can assist the algorithm towards overcoming the tightly constrained search space of the problem and give more freedom to the repair function. The number of nodes D to be removed is a random number from $[0.1K, 0.4K]$, where K is the number of nodes covered by all constructed routes while their resource requirements are satisfied.

3.2.1. Random Removal

The random removal randomly selects D nodes and removes them from all existing routes. This heuristic is important as it performs randomly regardless of any cost function or criteria which creates diversification.

3.2.2. Worst-Distance Removal (WDR)

We employ two classes of WDR, the classic WDR that considers the cumulative distance, and the relative distance to the protection value which looks at the cost/benefit ratio. A binary random variable is used to pick either the classical or new WDR each time we iterate over the algorithm. For each node i, the distance-cost can be calculated as $DC_i = d_{li} + d_{ij}$, where l and j are preceding and succeeding nodes on different routes for the vertex i. The algorithm sorts nodes in descending order based on the distance-cost, sorted list O, and removes the node in position $\lfloor \Upsilon^{\kappa} |O| \rfloor$ from the list. Parameters $0 < \Upsilon < 1$ and $\kappa \geq 1$ introduce randomness to avoid repeated removal of the same nodes. Alternatively, the $\frac{DC_i}{\psi}$ ratio can be used before sorting the list. We name the process of node selection for removal and using the $cost/benefit$ logic as the *selection mechanism*.

3.2.3. Worst-Time Removal (WTR)

The WTR is similar to WDR when we look at the general flow; however it considers the $TC_i = |start_i^{sync} - o_i|$, where $start_i^{sync}$ is the synchronised service starting time and o_i is the earliest service time. Note that we use the same mechanism as in WDR to choose between classical WTR or $cost/benefit$ WTR.

3.2.4. Shaw Removal (SR)

Many heuristics have been developed in attempt to measure the relatedness between nodes, but SR (Shaw (1998)) has got more attention since it integrates several criteria. The SR algorithm is modified for asset protection problem as below:

$$\Gamma_{ij} = \theta_1 d_{ij} + \theta_2 |o_i - o_j| + \theta_3 \Omega_{ij} \qquad (22)$$

where $\theta_1 - \theta_3$ are the Shaw parameters and Ω_{ij} can get any value from the set $\beta = \{-3, -2, -1, 1\}$ depending on number of same routes that node i and j belong to. The Ω_{ij} gets value of 1 when i and j are not assigned to any mutual route at all. This value decreases as the number of mutual routes for i and j increases (e.g., -3 for three mutual tours). Γ_{ij} decreases as the relatedness of two nodes increases. The algorithm starts with a random node and calculates the relatedness value of other nodes with the selected one by using equation 22. The node in position $\lfloor \Upsilon^{\eta} |O| \rfloor$ will be removed from the relatedness list $|O|$, where $\eta \geq 1$ is the Shaw removal determinism factor and $1 \geq \Upsilon \geq 0$ is a random number.

3.2.5. Proximity-Based Removal (PR)

A special case of SR algorithm where θ_1 takes value 1 and θ_2 and θ_3 are 0.

3.2.6. Time-Based Removal (TR)

A special case of SR algorithm we set $\theta_2 = 1$ and $\theta_2 = \theta_3 = 0$.

3.2.7. Requirement-Based Removal (RR)*

A special case of SR algorithm where θ_3 takes value 1 and $\theta_1 = \theta_2 = 0$.

3.2.8. Waiting Time-Oriented Removal (WTOR)*

This heuristic considers removing nodes with highest waiting time $WT_i = start_i^{sync} - arrive_i$, caused by the synchronised start time. Same procedure as WDR is used for picking a node for removal and choosing between classic WTOR or cost/benefit WTOR.

3.2.9. Worst-Requirements Removal (WRR)*

In the asset protection problem, multiple resources are required to satisfy the protection requirements of an asset. The WRR removes nodes with highest cumulative resource requirements. The *selection mechanism* is used to take advantage of the $cost/benefit$ approach.

3.2.10. Relative-Requirement Removal (RRR)*

Let r_{iq} and $Z_q \in \{-1, 0, 1\}$ denote the number of vehicle type q required to satisfy the requirements of node i and the score of vehicle type q, respectively. Vehicles that are low in number have lower scores, e.g. when number of vehicle types are $V1 < V2 < V3$ we have $Z_1 = -1$, $Z_2 = 0$ and $Z_3 = 1$. Therefore, for each node, we can compute ω such that $\omega = r_{iq} * Z_i$. After all, values can be sorted in ascending order and we remove nodes with applying the same logic as WDR selection mechanism.

3.2.11. Cluster Removal (CR)*

This algorithm categorises nodes based on their resource requirements. Then, the CR heuristic removes nodes that are in the same cluster with the hope of possible exchanges and finding better solutions.

3.2.12. Historical-Node Removal (HR)*

The HR heuristic takes advantage of the historical records when removing nodes. To achieve this purpose we developed a cost function for the asset protection problem as below.

$$f_i = \overline{\mathrm{WT_i}} + \overline{\mathrm{DC_i}} - \overline{\psi_\mathrm{i}} \tag{23}$$

The first two terms can be calculated by normalisation, $\overline{x} = (x - x_{min})/(x_{max} - x_{min})$, of the achieved values in WTOR and WDR heuristics, and the last term is the normalised protection value of the associated asset. Let $f_j^* = min_{m=1,\ldots,i-1}\{f_{jm}\}$ be the best position cost of node j before iteration i. The HR heuristic removes, $D = degreeofdestruction$, nodes which have the worst $j^* = argmax_{j \in V}\{f_{ji} - f_j^*\}$.

3.2.13. Time Windows-Oriented Removal (TWR)*

The TWR heuristic is another problem-specific algorithm trying to make room for nodes with limited insertion possibilities. In the asset protection problem, time windows are defined based on the cartesian coordinates. The TWR heuristic divides the whole area to four different zones (see Figure 5).

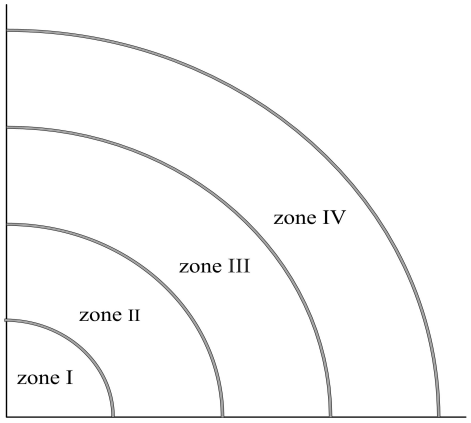

Figure 5: An illustrative example for TWR

The number of nodes that should be removed Λ_i from zones $Z_{\mathrm{I},\ldots,i-1}$, when D, tu, zu_i denote the degree of destruction, total unvisited nodes and number of the unvisited nodes in the zone i, is $\Lambda_i = \frac{D * zu_i}{tu}$. For instance, when $\Lambda_{\mathrm{III}} = 10$ it means that total number of 10 nodes must be removed from zone I, zone II and zone III. While Λ_i defines the number of nodes that has to be removed, the removal at each iteration will follow the same procedure as HR.

3.3. Insertion Algorithms

In the final repair phase of the algorithm, partially destroyed solutions will evolve into complete feasible solutions.

*3.3.1. Classical MCW Heuristic**

This heuristic attempts to insert unvisited nodes with highest value in vicinity of vertices with maximum possible value of S_{ij}. Although the algorithm

seeks to find the best possible position for insertion with highest saving value, it may increase the chance of getting trapped at local optimum.

3.3.2. Noise-Imposed MCW Heuristic (NMCW)*

To apply further diversification to the search, we use the NMCW algorithm beside the clean insertion. let $0 < \alpha_{noise} < 1$ denote a noise parameter, then $\Delta = \alpha_{noise} * max\{S_{ij}\}$ is the allowed amount of noise. In the NMCW heuristic we consider $S_{ij} = S_{ij} + \xi$ where $\xi \in [-\Delta, \Delta]$.

The master-level overview of the presented ALNS algorithm is provided in the following pseudocode, Algorithm. 2.

Algorithm 2: Pseudocode for the ALNS Algorithm

Input: N, n, h_d, h_r
Output: S^*

```
function ALNS algorithm with simulated annealing
    Generate initial solution S_0 using MCW
    i ← 1
    Let S* ← S_i ← S_0
    Initialise P(r_s), P(d_s) for each s ∈ h_d ∪ h_r
    Initialise T_initial by equation 21
    while (i ≤ N) do
        j ← 1
        while (j ≤ n) do
            Select a removal heuristic d ∈ h_d → (S_i^-) Select a repair heuristic r ∈ h_r → (S_i^+)
            if z(S*) ≤ z(S_i^+) then
                S* ← S_i^+
            if z(S_i) ≤ z(S_i^+) then
                S_i ← S_i^+
            if z(S_i) ≥ z(S_i^+) then
                Using SA criterion to accept/reject S_i^+
            Update π_s for the selected heuristics
            i ← i + 1
            j ← j + 1
        Update adaptive weights of heuristics, s ∈ h_d ∪ h_r
        Update temperature
    return S*
```

The general framework of our algorithm when certain parameters, such as predefined vehicle speed, fire velocity and time windows are involved have been explained. When dynamic routing is concerned as explained in section 2.2, the algorithm can also be implemented to solve the problem at multiple stages. Therefore, once a condition changes (like, time windows, wind speed, road accessibility and so forth) the algorithm can start rerouting. One way of doing so is to start a new routing problem with the updated time-windows. Other than this the main difference is that instead of starting at the depot, the vehicles are re-routed from their current first-phase location.

4. Computational Study

We carried out a set of computational experiments to validate the performance of the proposed ALNS approach. We perform further tests on the large set of generated benchmark instances[1]. As the focus is on large-scale problems, the problem-specific attributes are added to extended VRPTW benchmarks of Gehring and Homberger (1999). The problem attributes are added to the benchmark sets as described in section 2.2. The sixty problems of each size are divided into R1, C1, RC1, R2, C2 and RC2 classes based on their spatial distribution over a 140×140 grid and solved with two sets of vehicle numbers. In other words, there are ten instances under each class and a total of six classes exist $(6 * 10 = 60)$, which are solved with two different sets of vehicle numbers $(60 * 2 = 120)$ at three various sizes $(120 * 3 = 360)$.

In the first study, truncated benchmark sets are designed to solve sufficiently small-size instances by means of both the CPLEX commercial solver and the ALNS algorithm. We further show the efficacy of the ALNS on large-size instances, where they are compared to the CPLEX best bound. All the above computational work is performed on a node of 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 and 32GB of RAM. The algorithm was coded in C++, using a GCC 6.2.0 compiler. Where applicable, MILP models were solved by the CPLEX 12.7 commercial solver in deterministic mode. All tables show the execution times as CPU time in seconds.

[1] All the benchmark instances are available for future studies via www.sites.google.com/site/imanrzbh/datasets

4.1. Parameter Tuning

Our tuning methodology has been carried out by following the literature (Ropke and Pisinger (2006); Demir et al. (2012); Emeç et al. (2016)). To get the most information about the parameters contributions we omitted C1 and C2 problem classes as they mostly converge to optimal solutions. Subsequently, R104, R206, RC104, RC108 and RC206 were selected to determine the value for following parameters.

Table 1: Parameters used in the proposed algorithm

Description	Parameter	Value
Parameters for MCW heuristic	$(\lambda, \mu, \vartheta)$	(2,1,3)
Improving solution score	σ_2	12
Number of iterations	N	3000
Number of iterations over each segment	n	100
Roulette wheel reaction factor	ρ	0.1
Global solution score	σ_1	35
Worse solution score	σ_3	5
Shaw parameters	θ_1, θ_2, θ_3	(3,13,7)
SA parameter	δ	0.05
Cooling rate	ϵ	0.9999
Noise parameter	α_{noise}	0.6
WDR determinism factor	κ	8
Shaw determinism factor	η	12

The initial value of parameters are set in line with those by Ropke and Pisinger (2006) and Emeç et al. (2016). We perform five runs on tuning instances considering ten different values for each parameter. Thereafter we set each parameter on the value that yield the least average deviation from the best achieved solution. The results of the parameter tuning are available at "www.sites.google.com/site/imanrzbh/datasets".

4.2. Experiments on Asset Protection Problem

In this section, we generate instances with 35, 100 and 200 nodes to solve them by using the proposed methodology. For validation and performance evaluation of the ALNS, we solve truncated benchmark sets by means of both the CPLEX and the ALNS algorithm. For larger instances, we present our results as benchmarks for future research.

4.2.1. Numerical Results for Small-Size Instances

We solve small instances (35 nodes) with two different set of vehicle numbers($(V1 = 4, V2 = 3, V3 = 2)$ and $(V1 = 5, V2 = 4, V3 = 3)$). This is to verify the reliability of ALNS in different scenarios. We demonstrate the results of the performed tests in Table 2. The average and best results achieved in 10 runs of ALNS algorithm are compared to those by CPLEX. The proposed algorithm performs similarly in all examined cases which assures its reliability for further runs on larger problems. Note that instances with more than 35 nodes cannot be solved to optimality within the time limit of 48 hours for each class of instances.

In Table 2 computation times are reported in seconds and the optimality gap is defined by "OPT Gap %" and reported for the both average and best run of the ALNS. Furthermore, initial solutions that feed the algorithm are reported under the tag of "MCW %". This value shows how much the ALNS improves over the initial solution found by MCW. Our algorithm improves the initial solution ($MCW\%$) by 20% of the total value of assets to be protected. Also, the average gap of 1.9% and 0.12% in the last two columns of the table from optimal solution prove the efficacy of the ALNS algorithm. Comparing the achieved results by MCW and ALNS to the optimal solutions reveals that the ALNS improves the high quality initial solution significantly and often converges to the optimal solution. The average run time of ALNS is only 9.67 seconds whereas CPLEX spent 6,083.97 seconds on average. Among 120 instances solved by both CPLEX and ALNS, the proposed algorithm achieves optimal solution for about 95% of the problem instances. The ALNS achieves optimal solution for all instances in which nodes are displaced in cluster manner (C) and most of the random clustered class (RC). The deviation from optimal solution mostly occurs when vertices are randomly distributed.

4.2.2. Numerical Results for Large-Size Instances

To test the utility of our algorithm for operational purposes we solve large instances with 100 and 200 nodes in our experimental study. Since there is no benchmark to compare our results with, we run CPLEX for nine hours and report the best upper bound and best integer solution. Note that for the sake of better comparison, results are presented as a percentage of the total value of assets that required protection.

The results for 100-node instances are illustrated in Table 3. The number of vehicles are considered proportional to the problem size in order to cover a substantial portion of available assets. The second column defines the set of vehicle numbers which are either set1=(V1=6, V2=5, V3=4) or set2=(V1=7, V2=6, V3=5). In Table 3 CPLEX covers about

Table 2: A summary of results for 35-node. Vehicle numbers are defined in two categories: Set1=(V1=4,V2=3,V3=2) and Set2=(V1=5,V2=4,V3=3).

Instances	#Vehicles	CPLEX		MCW (%)	ALNS			OPT Gap (%)	
		Asset Value Protected(%)	Time (sec)		Asset Value Protected(%)		Time (sec)		
					Avg	Best		Avg	Best
C100	Set1	77.56	2,632.71	55.47	76.41	77.56	9.37	-1.49	0.00
	Set2	89.82	6,747.80	67.59	87.98	89.82	10.08	-2.03	0.00
C200	Set1	71.23	2,065.90	51.67	70.17	71.23	9.29	-1.49	0.00
	Set2	83.64	10,641.48	61.08	81.63	83.64	9.42	-2.38	0.00
R100	Set1	76.20	24.01	57.87	74.79	75.91	9.08	-1.85	-0.38
	Set2	89.19	128.90	71.33	87.12	89.04	9.48	-2.31	-0.17
R200	Set1	84.58	3,076.49	65.62	82.73	84.38	9.44	-2.20	-0.23
	Set2	95.28	5,057.62	76.75	93.52	95.00	9.90	-1.85	-0.29
RC100	Set1	86.11	4,511.13	64.03	84.95	86.11	9.84	-1.34	0.00
	Set2	96.58	10,774.72	75.53	95.02	96.58	10.37	-1.62	0.00
RC200	Set1	83.07	18,181.56	62.56	81.33	82.74	9.59	-2.10	-0.39
	Set2	95.71	9,164.91	73.25	93.58	95.71	10.23	-2.22	0.00

Table 3: A summary of results for 100 nodes. Vehicle numbers are defined in two categories: Set1=(V1=6,V2=5,V3=4) and Set2=(V1=7,V2=6,V3=5).

	#Vehicles	100					
		Time (sec)	CPLEX		MCW (%)	ALNS(%)	
			LB(%)	UB(%)		Avg	Best
C100	Set1	138.47	47.21	95.78	40.87	60.17	61.84
	Set2	150.39	65.73	99.52	45.42	66.66	68.35
C200	Set1	133.48	48.51	94.64	38.82	59.04	60.72
	Set2	143.92	61.24	99.29	43.15	64.87	66.58
R100	Set1	134.47	53.96	96.20	46.19	61.19	62.50
	Set2	138.97	56.68	99.74	52.19	68.45	69.86
R200	Set1	135.75	59.36	99.71	46.78	63.64	65.30
	Set2	144.65	64.94	99.68	51.86	69.76	71.38
RC100	Set1	143.41	60.59	98.52	49.43	66.77	68.59
	Set2	149.53	67.72	99.89	53.06	73.21	75.17
RC200	Set1	142.97	63.87	98.92	47.35	67.17	69.13
	Set2	146.91	55.48	99.90	52.88	73.12	74.71

57% of the total value of assets, while the ALNS covers 67.84%, on average. It can be seen that CPLEX is unable to handle the complexity of the asset protection problem when it comes to large

Table 4: A summary of results for 200-node. Vehicle numbers are defined in two categories: Set1=(V1=9, V2=8, V3=7) and Set2=(V1=12, V2=11, V3=10).

Instances	#Vehicles	200					
		Time (sec)	CPLEX(%)		MCW (%)	ALNS(%)	
			LB	UB		Avg	Best
C100	Set1	589.60	21.37	100	32.21	56.13	57.68
	Set2	619.33	31.73	100	38.90	65.11	66.57
C200	Set1	542.64	15.33	100	29.23	51.06	52.60
	Set2	566.36	19.96	100	35.99	60.34	61.56
R100	Set1	539.19	19.64	100	39.53	58.23	59.60
	Set2	585.49	27.62	100	46.57	69.04	70.29
R200	Set1	542.78	17.59	100	36.82	57.75	59.27
	Set2	589.17	21.94	100	43.13	68.74	73.58
RC100	Set1	561.80	18.87	100	37.16	60.90	62.18
	Set2	607.04	32.62	100	43.92	71.21	72.46
RC200	Set1	570.06	21.84	100	36.86	61.45	62.66
	Set2	633.17	23.63	100	44.29	72.14	73.58

scale instances, while the ALNS achieves better solutions than CPLEX in a shorter computation time. This is more evident when we increase the problem size by 100 nodes in Table 4. However, a better solution by ALNS does not guarantee its quality as the gap between the best integer and the CPLEX bound is still large. It is important to note that running CPLEX for a longer time to achieve a better upper bound would not be helpful. To investigate this claim, we performed a few experiments by running CPLEX for 48 hours to find a better upper bound. The results showed a slight improvement of about 0.5% in the upper bound. Therefore, as the results need to be achieved in operational time and no benchmark exists for the same type of problem, the quality of the results are validated by comparing to the optimal solution for small instances and the Lower Bound (LB) for larger instances.

In Table 4 two sets of vehicle numbers are defined, namely set1=(V1=9, V2=8, V3=7) and set2=(V1=12, V2=11, V3=10). Based on the results presented, in all instances the ALNS performs much better than CPLEX in terms of computational time or solution quality. The ALNS covers on average 40% more value of assets among the large instances with 200 nodes compared to the best solution by CPLEX, while improving the initial solution by 23.96%. The computation time for 100 and 200 node instances are 2.3 and 9.6 minutes, respectively. This is considered to be within the times suitable for operational purposes.

5. Conclusion

The loss of an infrastructure asset can cause major disruption to daily life and for an extended period. When these assets are threatened by runaway wildfires the deployment of resources to reduce their vulnerability is very important. It is therefore desirable to optimally deploy resources to try to save as many assets as possible. The optimal deployment problem for asset protection, however, is NP-hard and beyond human ability to solve especially under severe pressure ot time. Moreover, we found that using one of the most advanced commercial solvers available, in general, did not produce the results

required quickly enough for operational purposes.

In this study, we developed a solution scheme for solving the asset protection problem within times that make it suitable for operational purposes. The efficacy of the solution procedure was validated through extensive computational experiments. To evaluate the solution approach, new benchmark instances were generated based on problem-specific attributes. Our solution approach is inspired by methods in the literature (see Ropke and Pisinger (2006), Emeç et al. (2016)). We have, however, designed new removal and insertion heuristics and modified existing ones to assist us toward finding high quality solutions. We believe these heuristics can be implemented for solving other routing problems particularly those with synchronisation constraints.

Our computational experiments reveal the efficacy of the solution procedure under tight time limits. The results show that for problems up to 35 nodes the ALNS heuristic can generate near-optimal solutions in computational times of a few seconds. For larger problems in several minutes the ALNS can generate solutions that in most cases enable a three-fold increase in the number of assets treated compared with the best solutions CPLEX can achieve in nine hours. Thus, in the context of the asset protection problem the ALNS offers incident-management controllers a tool that may lead to significant reduction in losses during extreme fire events.

Acknowledgement

We wish to thank all anonymous reviewers and the associate editor for their constructive and valuable comments which have helped to improve this paper substantially. The second author is supported by the Australian Research Council under the Discovery Projects funding scheme (project DP140104246).

References

Afifi, S., Dang, D.-C., Moukrim, A., 2016. Heuristic solutions for the vehicle routing problem with time windows and synchronized visits. Optimization Letters 10 (3), 511–525.

Aho, A. V., Garey, M. R., Ullman, J. D., 1972. The transitive reduction of a directed graph. SIAM Journal on Computing 1 (2), 131–137.

Aksen, D., Kaya, O., Salman, F. S., Tüncel, Ö., 2014. An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem. European Journal of Operational Research 239 (2), 413–426.

Arrubla, J. A. G., Ntaimo, L., Stripling, C., 2014. Wildfire initial response planning using probabilistically constrained stochastic integer programming. International Journal of Wildland Fire 23 (6), 825–838.

Azi, N., Gendreau, M., Potvin, J.-Y., 2014. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. Computers & Operations Research 41, 167–173.

Belo-Filho, M., Amorim, P., Almada-Lobo, B., 2015. An adaptive large neighbourhood search for the operational integrated production and distribution problem of perishable products. International Journal of Production Research 53 (20), 6040–6058.

Canca, D., De-Los-Santos, A., Laporte, G., Mesa, J. A., 2017. An adaptive neighborhood search metaheuristic for the integrated railway rapid transit network design and line planning problem. Computers & Operations Research 78, 1–14.

Demir, E., Bektaş, T., Laporte, G., 2012. An adaptive large neighborhood search heuristic for the pollution-routing problem. European Journal of Operational Research 223 (2), 346–359.

Donovan, G. H., Rideout, D. B., 2003. An integer programming model to optimize resource allocation for wildfire containment. Forest Science 49 (2), 331–335.

Drexl, M., 2012. Synchronization in vehicle routinga survey of vrps with multiple synchronization constraints. Transportation Science 46 (3), 297–316.

Duff, T. J., Tolhurst, K. G., 2015. Operational wildfire suppression modelling: a review evaluating development, state of the art and future directions. International Journal of Wildland Fire 24 (6), 735–748.

Emeç, U., Çatay, B., Bozkaya, B., 2016. An adaptive large neighborhood search for an e-grocery delivery routing problem. Computers & Operations Research 69, 109–125.

Gehring, H., Homberger, J., 1999. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In: Proceedings of EUROGEN99. Vol. 2. Citeseer, pp. 57–64.

Li, B., Krushinsky, D., Van Woensel, T., Reijers, H. A., 2016. An adaptive large neighborhood search heuristic for the share-a-ride problem. Computers & Operations Research 66, 170–180.

Maknoon, Y., Laporte, G., 2017. Vehicle routing with cross-dock selection. Computers & Operations Research 77, 254–266.

Pappis, C. P., Rachaniotis, N. P., 2010. Scheduling a single fire fighting resource with deteriorating fire suppression times and set-up times. Operational Research 10 (1), 27–42.

Pisinger, D., Ropke, S., 2007. A general heuristic for vehicle routing problems. Computers & operations research 34 (8), 2403–2435.

Roozbeh, I., Ozlen, M., Hearne, J. W., 2016. A heuristic scheme for the cooperative team orienteering problem with time windows. arXiv preprint arXiv:1608.05485.

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

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems. In: International Conference on Principles and Practice of Constraint Programming. Springer, pp. 417–431.

Van Der Merwe, M., Minas, J. P., Ozlen, M., Hearne, J. W.,

2015. A mixed integer programming approach for asset protection during escaped wildfires. Canadian Journal of Forest Research 45 (4), 444–451.

van der Merwe, M., Ozlen, M., Hearne, J. W., Minas, J. P., 2017. Dynamic rerouting of vehicles during cooperative wildfire response operations. Annals of Operations Research 254 (1-2), 467–480.

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