Optimisation approaches for an orienteering problem with applications to wildfire management

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

Iman Roozbeh

MSc in Industrial Engineering | Eastern Mediterranean University

BSc in Industrial Engineering | Sadjad University of Technology

School of Science

College of Science, Engineering and Health

RMIT University

September 2019

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed. I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Iman Roozbeh

02 September 2019

Acknowledgments

I would like to thank my supervisors Prof. John Hearne and Babak Abbasi for their supervision, encouragement and giving me the opportunity to take this path. It has been a pleasure and a great experience to work with them and I have cherished every moment. My special appreciation is for Prof. John Hearne for his support and guidance in all the time of research and writing of this thesis.

Finally, yet importantly, I would like to give my greatest appreciation to my parents, Azar, Reza and my brother and sister Ashkan and Golbarg for their endless support, encouragement and love that made me believe in myself and pursue my dreams. Without you all this would not be achievable.

Abstract

During uncontrollable wildfires, Incident Management Teams (ITMs) dispatch vehicles for tasks aimed at reducing the hazard to key assets. The deployment plan is complicated by the need for vehicle capabilities to match asset requirements within time-windows determined by the progression of the fire. Assignment of the response vehicles to undertake protection activities at different assets is known as the asset protection problem. The asset protection problem is one of the real-life applications of the Cooperative Orienteering Problem with Time Windows (COPTW).

The COPTW is a class of problems with some important applications 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.

This thesis addresses the problem of finding solutions to COPTW in times that make the approaches suitable for use in certain emergency response situations. Computation of exact solutions within a reasonable time is impossible due to the nature of the COPTW. Thus, the thesis introduces an efficient heuristic approach to achieve reliable solutions in short computation times. Thereafter, a new set of algorithms are developed to work together as components of an adaptive large neighbourhood

search algorithm. The proposed solution approaches in this work are the first algorithms that can achieve promising solutions for realistic sizes of the COPTW in a time efficient manner.

In addition to the COPTW, this thesis presents an algorithmic approach to solve the asset protection problem. The complexities involved in the asset protection problem are handled by a metaheuristic algorithm. The asset protection problem is often further complicated by a wind change that is expected but with uncertainty in its timing. For this situation a two stage stochastic model is introduced for the optimal deployment of response vehicles. The model addresses uncertainty in the timing of changes in the problem conditions for the first time in the literature. It is shown that deployment plans, which improve on current practices, can be generated in operational times thus providing useful decision support in time-pressured environments. The performance of the proposed approaches are validated through extensive computational studies. The computational results show that the proposed methods are effective in obtaining good quality solutions in times that are suitable for operational purposes. This is particularly useful for increasing the tools available to IMT's faced with making deployment decisions crucial to savings lives and critical assets.

Contents

List of Figures

List of Tables

List of Abbreviations

ALNS Adaptive Large Neighbourhood Search

APP Asset Protection Problem

COPTW Cooperative Orienteering Problem with Time Windows

IMTs Incident Management Teams

MBH Merit Based Heuristic

OP Orienteering Problem

TOPTW Team Orienteering Problem

VRP Vehicle Routing Problem

Chapter 1

Introduction

Wildfires (bushfires) is an unplanned, uncontrolled and free-moving combustion reaction spreading across the landscape (Cruz et al., 2018). Wildfires constitute a recurrent and seasonal phenomenon with key roles in maintaining a balance in environmental ecosystems. Devastation and destruction caused by wildfires claim lives and damage economies. The Attica wildfires of Greece in July 2018, for instance, cost more than €33.7 million in terms of the damage done (Newsroom, 2018). Later in November 2018, the largest complex wildfires were recorded in California and considered to be the costliest natural disaster in the world, causing over US$16.5 billion worth of financial losses (Amadeo, 2019). The most destructive wildfires in the last decade were the so-called "Black Saturday" bushfires of 2009 in Australia, resulting in over 2000 homes being lost and 173 people killed (Haynes et al., 2010). Australia has a long history of wildfires and four of the five most devastating bushfires on record have occurred in Victoria and particularly in regions around Melbourne. Three mega-fires in Victoria over the period 2002-2009 burnt some three million hectares, or 40% of the state's public land (Attiwill and Adams, 2013). The Black Saturday bushfires were the worst on record with financial losses estimated to be $4.5 billion and destruction of over 3500 structures (Whittaker et al., 2009).

The Victorian Bushfires Royal Commission (VBRC) made 67 recommendations to the Victorian Government in July 2010, after the stark reminder of the Black Sat-

urday bushfires, for highlighting the need to amend and improve policies and procedures. The VBRC underscored the importance of immediate firefighting and protection of critical assets (Commission et al., 2009). In May 2017, the VBRC invested $23 million in developing scientific evidence for bushfire preparedness and response. Efficient planning of resources for suppressing fires is one of the nine projects that the fund has announced which proves the importance of this study (Commission et al., 2009).

Decisions in fire management are made at operational, tactical and strategic levels. Operational decisions are made is short time spans for immediate reactions to fire events, examples of which are displacement of fire crews and assignment of resources in fire suppression. Longer term decisions over days and seasons are known as tactical decisions which mostly involve seasonal and daily resource deployment, fire prevention planning and fuel treatment. Finally, decision over a horizon of years and decades are strategic decisions and focus on substantial improvements including budget allocation and facility location. This thesis mostly focuses on utilising operational research applications to assist decision-makers in operational decisions.

Bushfires are capable of destroying natural resources, environmental services and capital stock, impacting on both consumption and production aspects of people's lives (Ganewatta and Handmer, 2009). Asset protection and fire suppression aim to minimise the aftermath of bushfires by employing capital and human resources. To derive maximum benefit from such activities at operational level, scarce resources need to be implemented in the most efficient and effective way. In this thesis we aim to take realistic operational challenges into account and achieve optimal assignment of resources.

Lessons learnt from the Black Saturday bushfires highlight the key role of speedy assignment of resources to prevent the fire from escaping and incurring substantial damage costs. Along with efficient assignment of resources, change in problem conditions should be brought into our decision making process, an example of which is change in wind direction. In the Black Saturday bushfire following a wind change the long narrow fire-front become a wide fire-front that burned through a number

of townships with tragic consequences (Cruz et al. (2012)). There are similar events that highlight the importance of existing uncertainties during bushfires where some of which are investigated in this thesis.

Incident Management Teams (IMTs) must deploy personnel and equipment to wildfire response-related operations, such as fire suppression and asset protection. Wildfires at various levels of intensity require different suppression responses. For example, a deep-seated wildfire requires additional steps to assure it is completely extinguished. Given the complexities of fire dynamics and the physical and chemical interactions involved, ultimate extinguishment of a wildfire should be achieved by undertaking and completing cooperative tasks. Fire suppression and asset protection operations may involve single and/or multiple resources cooperating simultaneously or sequentially. Along with fire suppression requirements, other operational attributes such as limited resources and objectives need to be accounted by incident managers working under severe time pressures.

The asset protection and suppression operations can be considered as special cases of the COPTW which has many applications from health care problems (Cissé et al., 2017) to disaster management problems (Van Der Merwe et al., 2015). Although a wide range of real-life applications are identified for the COPTW, no solution approach is developed in the literature to solve large instances of such problems in operational times. Consideration of existing issues regarding the Asset Protection Problem (APP) and Cooperative Orienteering Problem with Time Windows (COPTW) shapes the following research questions.

Question 1: *How can we develop an efficient method to solve the COPTW?*

Question 2: *How can we handle the complexity of a real-world APP to include synchronous operations?*

Question 3: *How metaheuristic algorithms can help us to employ the COPTW in operational applications?*

Question 4: *How can we deal with existing uncertainties in the timing of wind changes during asset protection operations?*

1.1 Thesis structure

Chapter 2 presents a review of the relevant literature. This chapter covers some of the most recent operational research literature in wildfire and disaster management. Then it further explains variants of the vehicle routing problem and orienteering problem. The APP is a special case of the COPTW and will be discussed in this thesis. The APP which is the focus of this thesis is explained and literature gaps are identified. Thereafter, the stochastic orienteering problem is explored. Then, since development of efficient solution approaches are not investigated for the APP, a literature review is conducted to highlight some of the solution approaches for the orienteering problem.

In chapter 3 the orienteering problem with time windows and synchronisation constraints known as the COPTW is studied. Real world applications of the COPTW are investigated and a heuristic algorithm is developed to solve the COPTW for the first time in literature. Synchronous visits are handled efficiently by designing new operators for the merit-based heuristic. A new set of benchmark instances are designed to evaluate the performance of the algorithm and they are solved both by the means of an exact method and the newly developed heuristic. The work brought into this chapter is the first heuristic solution for the COPTW.

In chapter 4 the APP as a practical application of the COPTW is studied. Applications of the APP in emergency situations reveal the necessity of developing an efficient solution approach. An adaptive large neighbourhood search algorithm developed to solve the APP in a time efficient manner. The efficacy of the solution procedure is validated through extensive computational experiments. The illustrated metaheuristic algorithm in this chapter is the first heuristic solution for the problem which solves large instances that make it suitable for operational purposes.

From the experience of the real application in chapter 4, further development of the meta-heuristic for a general COPTW was made in chapter 5. In chapter 5 new insertion and removal heuristics are designed to solve the COPTW using an adaptive large neighbourhood search. Thereafter, efficiency of the designed heuristics

are evaluated and inefficient ones are filtered out. The final metaheuristic algorithm could beat existing benchmarks and achieve close to optimal solutions in short computational times.

Applications of the COPTW in real-life problems led us to consider the stochastic environment of such problems in chapter 6. The APP, for instance, involves uncertainties as do many other natural disasters. In chapter 6 a two-stage stochastic approach for the APP is presented that encompasses multiple characteristics of the problem. In the mixed integer programming model presented, possible wind changes are considered that have impacts on the time windows that assets need to be serviced. The proposed mathematical model for the first time considers uncertainty in the timing of changes in a problem's condition.

The findings of the thesis are summarised in chapter 7 followed by future research needs.

Chapter 2

Literature Review

In this chapter a review of the literature relevant to this work is presented. The literature review first covers the most recent operational research literature in wildfire and disaster management. From this the importance of VRP in this area becomes clear. Thereafter, comprehensive review of literature in vehicle routing problem and orienteering problem are presented.

2.1 Operation research in wildfire and disaster management

Although emergency logistics for natural disasters including floods, earthquakes, hurricanes and volcanic eruptions have been extensively studied (Özdamar and Ertem, 2015), applications of operation research in the wildfire management have received relatively less attention (Minas et al., 2012). Given the substantial investments in wildland fire management systems throughout Europe, Australia and North America many research have emerged since the early 21st century and some of which are reviewed by Minas et al. (2012) and Martell (2015). Fire management systems focus on prevention (Prestemon et al., 2010), detection (Ko et al., 2012; Benkraouda et al., 2014) suppression resource acquisition, deployment, dispatch and use (Martell, 2015).

Published research to date mostly cover strategic planning in regard of wildfire management. Minas et al. (2014) focused on reducing potential wildfires at the mitigation stage utilising a spatial fuel management method in Victoria, Australia. They developed a deterministic MILP model to minimise connectivity of old fuel cells in the belief that fragmentation of the landscape fuel complex will inhibit fire spread. Rachmawati et al. (2018) similarly introduced a MILP for fuel treatment planning to fragment high fuel load areas while inhibited connectivity is taken into account. More recently, Matsypura et al. (2018) proposed an optimisation model to identify the optimal spatial allocation of prescribed burning activities over a finite planning horizon. A rather simple myopic heuristic algorithm was developed to solve large scenarios. Krasko and Rebennack (2017) addressed a post-wildfire debris flow hazard management system by developing a deterministic model for allocating a budget to reduce flow of debris. In addition, they incorporate uncertain travel time and number of wounded people in trying to solve the routing problem in a two-stage stochastic programming model. This involves picking up injured people and delivering them to the hospitals. Wei et al. (2014) developed a model to station and dispatch hand crews and fire trucks for initial attack on bushfires. They introduced a chance-constrained two-stage stochastic programming approach.

Van Der Merwe et al. (2015) addressed the APP during escaped wildfires. Their study found that unplanned and uncontrollable wildfires sweep across valuable structures such as hospitals, bridges and schools that are connected by a network of roads and some of which may be remotely located. They assumed a fleet of resources where each has a vector of capabilities. Resource capabilities should match the protection requirements that assets require so that they are protected. They applied the developed deterministic MILP model for bushfires in Hobart, Australia. However, they were unable to solve realistic sized problems with exact methods.

2.2 Vehicle routing problems

The vehicle routing problem (VRP) is well-known integer programming problem that emerges from a combination of travelling salesman problem (TSP) and bin packing problem. In other words, VRP is a TSP where the capacity constraint is taken into account and aims to minimise the total travel distance (Dantzig and Ramser, 1959). VRP and its variants have been intensively studied and future research directions are set in the literature (Cordeau et al. (2007); Toth and Vigo (2014); Laporte (2009)). Feillet et al. (2005) introduced the VRP with profit (VRPP) where multiple vehicles cannot visit all vertices of the given graph. Archetti et al. (2014) provide a survey on the VRP with profit in which the orienteering problem (OP) was considered as the basic problem of this class. Many of the developed solution approaches for existing VRPs cannot be replicated for OPs due to the problem-specific attributes.

2.2.1 Orienteering problem and its variants

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); Borràs 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 a number of 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 (Garcia et al. (2010)). The *Team Orienteering Problem with Time Window* (TOPTW) is one of the widely studied OPs by scholars (Labadie et al. (2012); Kim et al. (2013); Souffriau et al. (2013); Duque et al. (2015); Gunawan et al. (2015); Zhang et al. (2018)). 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.

2.2.2 Synchronisation constraints

The VRP has been studied extensively over more than half a century. More recently, synchronisation constraints between vehicles became a hot topic and has received a lot of attention from scholars (Gansterer and Hartl (2018); Liu et al. (2019)). This is inspired from real-life routing problems that involve spatial, temporal and load synchronisation between vehicles Drexl (2012). Vehicles Synchronisation are concerned with tasks, operations, movements, loads and resources associated to vehicles Drexl (2012). An example in which synchronisation exist is the home care services. In such services, some operations may require more than one staff member. For instance, where heavy lifting and specialist medical expertise are required (Bredström and Rönnqvist (2008)). A closely related problem is a home cleaning service where some homes may require one or more services such as the following: basic cleaning, window cleaning, and washing. Access to each house is within a given time-window.

2.2.3 Cooperative orienteering problem

An extension to TOPTW is the *Cooperative Orienteering Problem with Time Windows* (COPTW), proposed by Van Der Merwe et al. (2015). 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 TOPTW, there is the additional requirement that certain paths must meet at some of the nodes.

The COPTW arises in several applications. The first application arose from a problem involving asset protection during escaped wildfires (Van Der Merwe et al. (2015)). Valuable assets such as bridges and hospitals are distributed over a landscape with a network of roads. A wildfire sweeps over this landscape in a certain direction. 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 pumper. Others might need a tanker (own water) and a personnel vehicle. 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.

2.2.4 The asset protection problem

The APP is analogous to the OP(Van Der Merwe et al. (2015)). Constrained by time, both problems involve choosing a subset of all nodes to visit to maximise an objective. In the case of the OP a reward is on offer at each node and the objective is to maximise the total rewards collected. The objective of the APP is to maximise some weighted total of the number of sites serviced. Sites are often weighted by a measure of the economic consequence of the facility being destroyed or damaged. Unlike the OP, the APP may require synchronous visits to some nodes (assets) by more than one vehicle type. As in APP multiple vehicle types are involved we may

call APP is an extension to the COPTW. Dynamic fire fronts further impose time window constraints on the service time of each node as well as on the accessibility of roads. Therefore, the APP may deal with some stochastic parameters.

2.2.5 Stochastic orienteering problem

Recently, research interest on stochastic variants of routing problems has increased significantly. Advances in technology have enabled larger, more complex problems to be solved to support decision makers. For further details, we refer the interested readers to the surveys by Gendreau et al. (2016) and Ritzinger et al. (2016). When dealing with stochastic problems, a wide range of approaches can be utilised, e.g. discrete event simulation or robust optimisation (Hoyos et al. (2015)). One of the most frequently used techniques that has attracted substantial attention in stochastic VRP and disaster management problems is the two-stage stochastic programming (Falasca and Zobel (2011); Grass and Fischer (2016); Krasko and Rebennack (2017); Badri et al. (2017); Van Hui et al. (2014)). In brief, the two-stage stochastic programming make decisions in the first-stage prior to realisation of uncertain events, while taking prospective second-stage decisions into account. There are various stochastic characteristics of the orienteering problem that have been investigated. Ilhan et al. (2008) introduced the OP with stochastic profit for the first time. Other uncertainties, such as stochastic travel and service times (Papapanagiotou et al. (2014)), stochastic time-dependent travel times (Varakantham and Kumar (2013)) and stochastic waiting time (Zhang et al., 2014) have been studied. While to the best of our knowledge, no formulations have been reported in the literature of a two-stage stochastic programming model with an uncertain time of change in the problem condition, i.e. staging time. Staging time is the moment when transition from one stage to another occurs. This is a situation that is commonly faced in emergency and logistic problems.

2.3 Solution approaches for the OP

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 (Keshtkaran et al. (2016); Poggi et al. (2010); Dang et al. (2013a); Bianchessi et al. (2018)) but in general exact methods are not efficient in dealing with 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)). 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 the generalisation of the TOPTW, are solved by a variable neighbourhood search algorithm (Tricoire 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 and introduced two cluster-based algorithms by considering the limitation 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. Vincent et al. (2019) studied the team orienteering problem with time windows and time-dependent scores and developed a hybrid artificial bee colony algorithm to solve it. Finally, A recent article

by Santini (2019) revealed the significance of using Adaptive Large Neighbourhood Search (ALNS) algorithm for solving OPs.

Chapter 3

A heuristic solution for the COPTW

In this chapter, the COPTW is studied. The existing applications of the COPTW require an efficient solution method. The problem of finding solutions to the COPTW in times that make the approach suitable for use in certain emergency response situations is addressed. This is achieved by developing a new merit based heuristic. The algorithm tested extensively and compared with exact solutions to validate its performance. The results prove the efficacy of the solution approach to generate high quality solutions in short computation times.

3.1 The Cooperative Orienteering Problem with Time Windows

The orienteering problem with time windows and synchronisation constraints known as the Cooperative Orienteering Problem with Time Windows (COPTW) generalises the TOPTW formulation (Van Der Merwe et al. (2015)), where a certain number of team members are required at vertices $N = \{v_1, \ldots, v_n\}$. The location v_i is considered served if r_i team members, $i \in N$, arrive at the vertex within the time windows $[o_i, c_i]$ and start the service simultaneously at time s_i for a duration of a_i units of time to collect a reward, ψ_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_{max}, where both v_1 and v_N represent the same location. For any two vertices, t_{ij} and d_{ij} indicate the required time and distance for each team member to travel from i to j. The binary decision variable y_i takes the value 1 if r_i team members visit v_i 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, A defines a set of arcs that can be traversed. For $(i, j) \in A$, if a team member departs from vertex i at $o_i + a_i$ and arrives at vertex j before c_j the (i, j) can be traversed. Following the definition of A, a set of feasible arcs that can be traversed to and from node i are shown by Ω_i^- and Ω_i^+, respectively.

The mathematical formulation of the COPTW as a mixed-integer program is as follows(Van Der Merwe et al. (2015)):

$$Maximise \sum_{i=2}^{N-1} \psi_i y_i \tag{3.1}$$

$$s.t. : \sum_{(1,j)\in\Omega_1^+} x_{1j} = \sum_{(i,N)\in\Omega_N^-} x_{iN} \leq P, \tag{3.2}$$

$$\sum_{(i,k)\in\Omega_k^-} x_{ik} = \sum_{(k,j)\in\Omega_k^+} x_{kj},\ k = 2, \ldots, N-1, \tag{3.3}$$

$$r_k y_k \leq \sum_{(k,j)\in\Omega_k^+} x_{kj},\ k = 2, \ldots, N-1, \tag{3.4}$$

$$x_{ij} \leq P z_{ij},\ (i, j) \in A, \tag{3.5}$$

$$s_i + t_{ij} + a_i - s_j \leq M(1 - z_{ij}),\ (i, j) \in A, \tag{3.6}$$

$$o_i \leq s_i,\ i = 1, \ldots, N, \tag{3.7}$$

$$s_i \leq c_i,\ i = 1, \ldots, N, \tag{3.8}$$

$$x_{ij} \in \{0, 1, \ldots, P\},\ (i, j) \in A, \tag{3.9}$$

$$y_i, z_{ij} \in \{0, 1\},\ i \in N,\ (i, j) \in A. \tag{3.10}$$

The objective function (3.1) maximises the sum of the rewards ψ_i collected at each

vertex i. Constraint (3.2) ensures all members depart from and return to a designated depot. Constraint (3.3) guarantees flow conservation by enforcing the equality of incoming and outgoing arcs to each node. Constraint (3.4) 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 (3.5) 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 (3.6) 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 $M = max(c_i) + max(t_{ij}) + max(a_i) - min(o_i)$ is sufficiently large for this purpose (Van Der Merwe et al. (2015)). Constraints (3.7) and (3.8) ensure each vertex is visited within its time window. Integer and binary conditions are defined in constraints (3.9) and (3.10). In the above model the subtour elimination implemented similar to VRPs.

A graphical representation of a simple solution for the COPTW is sketched in Figure 3.1 as shown below. Two of the main attributes of each node, the resource require-

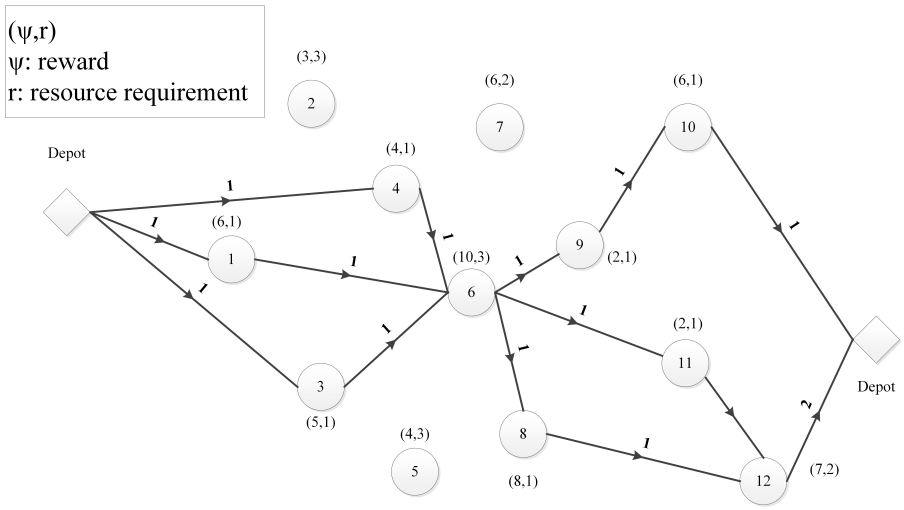

Figure 3.1: A sample solution of the COPTW

ment (r_i) and corresponding reward (ψ_i), are specified while time window constraint exists for all nodes. A number of team members leave the depot and r_i members must be at vertex i before starting the service to collect the associated reward (ψ_i). The numbers over each arc represent the number of travelling members 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 Figure 3.1, 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 its reward (10). After collection of maximum amount of possible scores, all members return to the depot. Note that both depots represent the same location.

3.2 A heuristic approach for the COPTW

Starting with the well-known Clarke and Wright (CW) algorithm (Clarke and Wright (1964)), we make several changes to address the complexities of the COPTW problem. The classical CW algorithm uses saving values by merging pairs of customers in the same route. The CW Saving Algorithm is one of the most implemented heuristic methods for solving routing problems due to its efficient computation time and reliability. As we want to maximise the rewards collected from visiting a node we replace the "savings list" in the CW algorithm with a "merit list" as discussed below. The next obvious problem to address is that the reward at some nodes can only be collected if there is a synchronous visit by more than one team member (or vehicle). Furthermore these synchronous visits must occur within a specified time-window. Afterwards, we rank each unvisited node according to its reward and the resources it requires. The initial solution of the algorithm can be any feasible combination of two nodes with highest merit value. Working through this ranked list we consider the possibility of inserting unvisited nodes within existing routes. This action may have feasibility consequences which also need to be considered. In some cases an insertion may lead to the removal of a node that is already in a route. Further details of all these considerations are given below. For ease of reference we will refer to our algorithm as the MB (merit based) heuristic.

3.2.1 Merit list

The COPTW aims to maximise the collected score by meeting the resource requirements of nodes within their associated time windows. Thus pairs of nodes are ranked according to their combined reward. However, a greedy algorithm approach to this may lead to too much time consumed in collecting the largest rewards at the expense of several smaller rewards. To compensate for this we add to the merit function a "savings pair" term as used in the CW algorithm for finding the shortest routes. This is further enhanced with another term based on the sweep algorithm introduced by Doyuran and Çatay (2011). Sweep Algorithm is another well-studied heuristic that constructs routes based on the angles of customers with depot and another arbitrary line. To dynamically adjust the impact of the terms in equation 3.11, we introduced three associated weights. Our algorithm iterates over different values of these weights within an interval pre-determined by experimentation. Intervals and step-sizes of weights are defined in such a way to compromise between solution qualities and computation times. Therefore, the proposed merit function for the MB heuristic is given by:

$$M_{i,j} = \vartheta \, \frac{\psi_i + \psi_j}{\overline{\psi}} + \frac{d_{i0} + d_{0j} - \lambda \, d_{ij}}{d^{max}} + \mu \, \cos\theta_{ij} \, \frac{|d^{max} - (d_{i0} + d_{0j})/2|}{d^{max}}. \quad (3.11)$$

In equation 11, θ_{ij} is the angle between the vector from the depot to node i and the vector from the depot to node j, ψ_i and $\overline{\psi}$ represent the score for each node and the average score of all vertices. Also, in the above formulation, d_{ij} is the euclidean distance between nodes i and j and $d^{max} = \max\{d_{i,j};\ \forall i, j \in N\}$. It is assumed that travel time and travel distance are correlated. Moreover, iterations of the three parameters, ϑ, λ and mu, are used to search a broader solution space. To ensure the assignment of nodes with highest merit values prior to others, we implemented parallel route construction in the MB algorithm.

3.2.2 Synchronous visits

Some nodes in the COPTW require synchronous visits from more than one team member (or vehicle). We first illustrate how to deal with the multiple visit without concern for the timing. Suppose node 3 requires two members to visit. Given a merit pairing list as follows: (2,3), (3,4), (4,5), (3,6), and (6,7) with a depot at node 1 we construct routes initially as in the CW algorithm. This gives: 1-2-3-4-5-1 based on the first three pairs in the list. The next pair in the list is (3,6) and would normally be deleted as node 3 is internal to a route. In this case, however, node 3 still requires another visit so we introduce a new route 1-3-6-1. With the final pair this second route becomes 1-3-6-7-1. So we have two routes with both routes visiting node 3 as shown in Figure 3.2. If the timing of these visits is such that both team members arrive at the node in time to service the node within the required time-window then the reward is collected. Some issues arise when constructing routes to

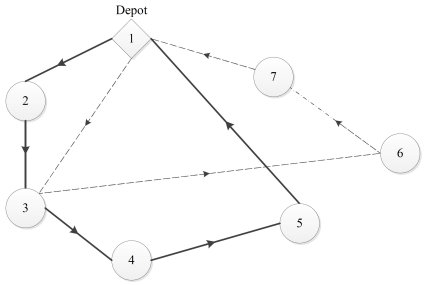

Figure 3.2: Illustration of the synchronous visits handling

meet the requirements at a node. One is that it might not be feasible to reach the node within its time window. The second is that after working through the pairs in the merit list a particular node is visited by insufficient number of team members to collect its reward. This means that time is possibly wasted travelling to that particular node and the node should be removed from all routes. Another option is to consider removing a node that has already been assigned to the routes. This would help us to save sufficient time and resources to meet the demand of the other node. The change would be accepted if the changes yielded greater rewards in total than previously.

3.2.3 Feasibility

Before adding a node to a route the feasibility of arriving at the node before its closing time must be calculated. The "closing time" is the latest possible arrival time at a node to be able to commence and complete servicing it within the time window required. The required number of team members must all be present before synchronous servicing can commence. Thus the start time of the service is determined by the latest arriving team member and this may change as routes are modified after insertion of a new node. It is also important to track the completion time. The departure time from a node may affect the feasibility of visiting nodes with later time windows.

Another feasibility constraint is the number of team members available. This together with the time window constraints means that, in general, it will not be possible to service all nodes.

3.2.4 Prioritising nodes for insertion

After working through the merit list, a feasible solution is achieved but a number of nodes may be unvisited. We performed extensive experiments to find the best ranking methods of the unvisited nodes. As a result of experiments we rank these nodes according to the ratio

$$\frac{\psi_m}{\sqrt{r_m}}$$

where ψ_m and r_m are the rewards available and resources (visits) required 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.

3.2.5 Insertion

We first illustrate the basic process of insertion with the following simple example. Consider the pairs in the following merit list: (2, 3), (3, 4), (4, 6), (6, 7), (4, 5), (3,

5), (5, 6). Working down the list we end up with the route 1-2-3-4-6-7-1. Node 5 would be excluded from the route as when the pairs involving node 5 are considered, in each case the other element of the pair is internal to the route. We see, however, that node 5 can be inserted between nodes 3 and 5 as shown in Figure 3.3. As

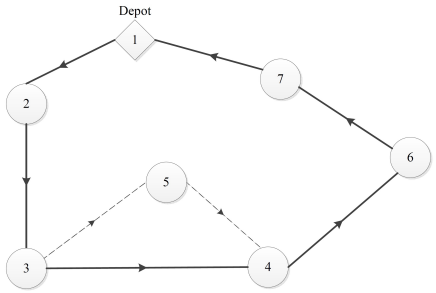

Figure 3.3: Illustration of the insertion procedure

previously mentioned feasibility has to be checked before an insertion such as the one above can be accepted. Suppose that the route proposed in the example above is not feasible because the arrival at node 6 from node 4 is too late. If the path segment 3-5-6 is feasible (see Figure 3.4) we have two choices. Keep the original path, with node 5 omitted, or remove node 4 from the solution. The obvious choice is the one that maximises the rewards. There is a further difficulty with insertion

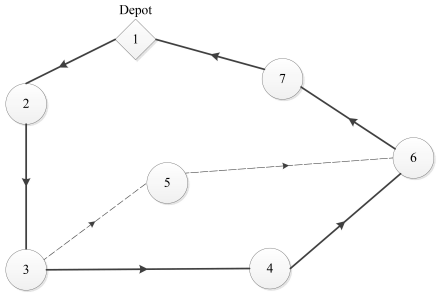

Figure 3.4: Illustration of the insertion procedure

when synchronous visits are involved. Suppose that node 4 required a visit by two members for the reward to be collected and a solution satisfying this had already been achieved as shown in Figure 3.5. In removing node 4 from the first route there is no longer any point in the second route including node 4. In fact, the route segment 8-10-11 might now be feasible or a slightly earlier departure time from node 11 (after servicing within the required time window) might just tip the difference between arriving at node 12 in time or not.

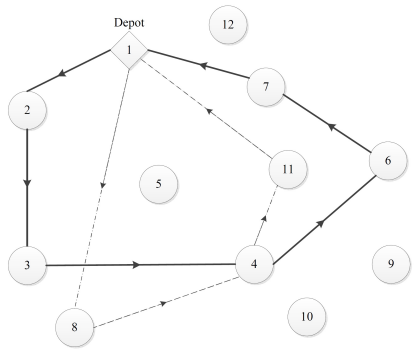

Figure 3.5: A sample solution of the COPTW

3.2.6 The MB Heuristic

All the operations discussed above are included in the pseudo-codes of our heuristic as given in Algorithms 1-4. To start the algorithm, the CW algorithm modified to enable synchronous visits as discussed in section 3.2.2, is employed to generate an initial solution. The main body of the algorithm follows.

Iterating over values of ϑ, λ, and μ, a new merit list is compiled at each iteration by calling the function "$CalcMeritList$". Then considering the set U of all unvisited nodes, we try to assign each node to a route based on the pairs that they belong to, their merit values and their resource requirements. Specifically, at line 11 of Algorithm 3.1, an attempt is made to assign node $j \in U$, which belongs to $pair_{ij}$ into the existing routes, Λ. For each subroute belonging to Λ in which the node i is located, the algorithm seeks to insert the arc ij to gain the score at j. Other possibilities for insertion are investigated by the $improve()$ function in Algorithm 3.4. Nodes that cannot be added to the existing routes in Λ will be assigned to a new subroute, if the number of subroutes (n) in the set of temporary routes (Λ) is less than the total number of team members (P). Note that newly assigned nodes that do not satisfy the resource requirements will be ignored when the algorithm returns to line 7 without updating τ. The algorithm investigates further improvements by calling the improve function at line 29.

Algorithm 3.1: Pseudocode for the MB heuristic

Input: temporary routes (τ), best routes (β), collected reward (α), set of all nodes (N), set of assigned nodes (γ), Merit Pair List (MPL), unvisited vertices (U), service requirement of node j (r_j), parameters' intervals ($\vartheta = [0, 3.2]$, $\lambda = [0, 1.2]$, $\mu = [0, 1.2]$, step size $= 0.4$), available team members (P), number of sub-routes (n)

Output: α_{best} *and* β

```
1  function MBheuristic
2      generate distance matrix; generate initial solution by CW heuristic
3      forall (ϑ, λ, μ) do
4          Call the CalcMeritPairs(ϑ, λ, μ) function
5          forall (pair_{i,j} ∈ MPL) do
6              Λ ← τ        //make a copy of τ
7              if (j ∈ U) then
8                  VisitCount_j ← 0
9                  forall (subroutes ∈ Λ) do
10                     try to add arc (i, j)
11                     Call the FeasMatrix function
12                     if (feas_ij == true) then
13                         VisitCount_j ← VisitCount_j + 1
14                         Update Λ
15                     else
16                         if (n < P) then
17                             Create a new route and add j
18                             Update Λ
19                             n ← n + 1
20                             VisitCount_j ← VisitCount_j + 1
21                     if (VisitCount_j == r_j) then
22                         α_current ← α_current + ψ_j
23                         γ ← γ ∪ {j}
24                         Update U
25                         τ ← Λ
26                         if (α_current > α_best) then
27                             α_best ← α_current, β ← τ
28     Call the Improve function
29     return α_best and β
```

The $CalcMeritList()$ function is represented in Algorithm 3.2. As far as the parameters in the merit function are concerned, the computational effort is positively correlated to the tuning of the (ϑ, λ, μ) tuples. After much experimentation, and considering the compromise between the search effort and solution quality, the intervals and incremental size for the coefficients shown in Algorithm 3.1. were chosen.

Algorithm 3.2: Pseudocode for the CalcMeritPairs

Input: merit function parameters (ϑ, λ, μ), Merit Pair List (MPL), score of node i (ψ_i), travel velocity ($velocity$), set of all arcs (G), latest arrival time to the depot (T_{max})

Output: MPL

1 **function** CalcMeritPairs
2 **for** $((i,j) \in G)$ **do**
3 **if** $((o_j + a_j + d_{j0}/velocity \leq T_{max})$ *and* $(o_i + a_i + d_{ij}/velocity \leq c_j))$ **then**
4 $M_{i,j} \leftarrow \vartheta \dfrac{\psi_i + \psi_j}{\overline{\psi}} + \dfrac{d_{i0} + d_{0j} - \lambda\, d_{ij}}{d^{max}} + \mu \cos\theta_{ij} \dfrac{|d^{max} - (d_{i0} + d_{0j})/2|}{d^{max}}$
5 insert $M_{i,j}$ to a vector of tuples $(i, j, M_{i,j})$ // *initialising* MPL
6 Sort MPL in descending order of $M_{i,j}$ values
7 **return** MPL

In the MB heuristic, Algorithm 3.3. handles frequent updates of the feasibility matrix, and Algorithm 3.4. enhances the solution quality, as defined below.

In Algorithm 3.4, the $improve()$ function checks whether further score can be achieved by adding more nodes to the routes or whether any of the inserted nodes can be substituted for unvisited ones. The algorithm sorts unvisited vertices based on the associated resource requirements and rewards. Those with higher reward and less resource requirements climb within the list. This is done according to the value of $\psi_m/\sqrt{r_m}$, which represents the relative attractiveness of each vertex (line 2). Thus, nodes that are ranked better in U (line 2), have a higher chance of assignment to the routes, based on the pairs they belong to (lines 5-20). Nodes in set U will be assigned at locations where the highest merit values can be achieved while still maintaining all feasibility conditions. This is done in lines 5-8, where for each node the algorithm iterates over MPL to add the node m where the highest merit value can be achieved. Lines 9-20 consider the situation where the insertion of the node m brings infeasibility for another node named as j. The insertion would be taken

Algorithm 3.3: Pseudocode for the FeasMatrix

Input: set of temporary routes (Λ), set of all nodes (N), earliest start time o_i, latest possible start time c_i, service start time at node i (φ_i), existing nodes in sub-tours (κ), latest arrival time of members to node i (T_i), feasibility to travel from i to j ($feas_{i,j}$)

Output: $FeasMatrix$

```
1  function FeasMatrix
2      for (i ∈ N) do
3          φ_i ← o_i
4      for (κ ∈ Λ) do
5          for (i ∈ κ) do
6              compute T_i
7              φ_i ← max(φ_i, T_i)
8      for (κ ∈ Λ) do
9          for (i ∈ κ) do
10             if (φ_i ≥ c_i) then
11                 feas_{i−1,i} ← false
12             else
13                 feas_{i−1,i} ← true
14     return FeasMatrix;
```

into account if it could improve the objective function despite the loss of score from the removal of the other node (lines 9-17).

Algorithm 3.4: Pseudocode for the improve function

Input: best collected reward (α), best routes (β), unvisited vertices (U), Merit Pair List (MPL), score of node i (ψ_i), resource requirements of node i (r_m)

Output: β and α

```
1  function improve
2      sort set of U members        // ψ_m/√r_m
3      for (m ∈ U) do
4          L ← β
5          for (pair_{p,q} ∈ MPL) do
6              if (m == p) then
7                  Seek to insert arc (m, q)
8                  Call the FeasMatrix function
9                  if (insertion causes infeasibility of node j) then
10                     if (ψ_j ≤ ψ_m) then
11                         insert m and remove j
12                         Call the FeasMatrix function
13                         if (insertion is feasible) then
14                             update β
15                             update α
16                         else
17                             β ← L
18                 if (insertion is feasibe) then
19                     update β
20                     update α
21     return β and α
```

3.3 Computational results

Extensive numerical studies were conducted to evaluate the efficacy of the proposed solution approach. A set of benchmark instances was 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[1].

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 MB algorithm. The number of vertices in the small-size instances are chosen to obtain the optimal solutions using CPLEX. Furthermore, we explored the trade-off between an increased number of available members for service on the one hand and the computation time and objective value on the other hand. We furthermore showed the efficient performance of the proposed heuristic in terms of time and accuracy on the large-size benchmark instances. All the above computational work was 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 programmed in C++, using a GCC 5.2.0 compiler. Where applicable, MILP models were solved by the CPLEX 12.7 commercial solver in deterministic parallel optimisation mode. All tables show the execution times as elapsed time in seconds.

For the parameter studies, after running 720 instances with various parameter settings, we tuned them in a way to define the best possible trade-off between runtime and solution quality. It was decided to change the parameters within $[0, 1.2]$ for both λ and μ and $[0, 3.2]$ for ϑ. Based on the authors' observations the value of ϑ plays a significant role in the solution quality, thus a broader interval is considered for the newly introduced term in the merit function. Additionally, an incremental size of 0.4 is large enough to search the feasible region sufficiently and to avoid redundant iterations. The large interval and step-size assist the heuristic to explore a broader area of the solution space.

For validation and performance evaluation a collection of 456 small-size benchmark instances were generated and solved by means of both CPLEX and the MB heuristic. A summary of the tests for $10 - 12$ nodes with 3 and 4 team members on instance sets c100, r100 and rc100 is provided in Table 1. The benchmark instances have cluster (c), random (r) and random cluster (rc) distribution, where the last to are

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

Table 3.1: A summary of MB Heuristic (MBH) performance for small-size instances for 10, 11 and 12 nodes on c100, r100 and rc100 datasets. All computational times are in seconds.

Set	# Vehicles	10		Opt Gap %	11		Opt Gap %	12		Opt Gap %
		CPLEX	MBH		CPLEX	MBH		CPLEX	MBH	
c100	p=3	1.54	0.05	0.00	0.90	0.07	0.00	2.82	0.12	0.00
	p=4	1.70	0.04	0.00	0.75	0.06	0.00	1.18	0.11	0.00
r100	p=3	9.23	0.13	-2.23	40.07	0.16	-1.83	1740.62	0.22	-0.16
	p=4	12.88	0.12	-3.43	59.92	0.15	-3.46	1455.69	0.22	-3.50
rc100	p=3	15.97	0.09	0.00	43.38	0.14	-0.54	304.96	0.20	-1.02
	p=4	8.12	0.09	-0.60	503.42	0.13	-1.09	1908.95	0.18	-0.98

Table 3.2: A summary of MB heuristic performance for small-size instances for 24, 25 and 26 nodes on c200, r200 and rc200 dataset. All computational times are in seconds.

Set	# Vehicles	24		Opt Gap %	25		Opt Gap %	26		Opt Gap %
		CPLEX	MBH		CPLEX	MBH		CPLEX	MBH	
c200	p=3	3.11	0.46	0.00	4.00	0.61	0.00	5.44	0.73	0.00
	p=4	2.89	0.48	0.00	3.93	0.49	0.00	3.15	0.56	0.00
r200	p=3	11.84	0.92	-0.03	12.17	1.19	-0.08	19.13	1.40	-0.54
	p=4	5.24	0.71	0.00	6.88	0.93	0.00	7.71	1.08	0.00
rc200	p=3	22.75	0.81	0.00	478.37	1.04	0.00	13585.62	1.40	-1.15
	p=4	7.78	0.61	0.00	8.31	0.81	0.00	9.83	1.04	0.00

harder to solve to optimality. The sizes of truncated instances are chosen in a way to investigate the correlation between the increase in problem size and exponential growth in computational effort. It is worthwhile to mention that infeasible edges are excluded in MILP formulations to simplify models for the CPLEX implementation.

In Table 3.1 computation times and the optimality gap are reported for all problems. The "optimality gap" (OPT Gap %) represents the percentage gap between the optimal solutions obtained by the CPLEX and MB heuristic, where is negative meaning MB heuristic solution is not optimal. It is seen in Table 3.1 that the average gap is -1.04% which shows the promising performance of the proposed heuristic. Moreover, it can be seen that the computation time increases significantly with minor changes in the problem size for CPLEX compared with the negligible changes for the MB heuristic.

Table 3.2 gives the summary of results for $24 - 26$ 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 3.1. This is due to the nature of the studied class of problems as the time window intervals are different in

Table 3.3: A summary of MB heuristic performance for small-size instances for 19, 20 and 21 nodes on pr01-10 dataset. All computational times are in seconds.

Set	# Vehicles	19		Opt Gap %	20		Opt Gap %	21		Opt Gap %
		CPLEX	MBH		CPLEX	MBH		CPLEX	MBH	
pr01-10	p=3	55.36	0.45	-2.08	84.27	0.50	-1.93	259.63	0.54	-1.93
	p=4	64.98	0.47	-1.87	254.00	0.51	-2.41	854.03	0.56	-3.03

Table 3.4: A summary of MB heuristic performance for small-size instances for 10, 11 and 12 nodes on pr11-20 dataset. All computational times are in seconds.

Set	# Vehicles	10		Opt Gap %	11		Opt Gap %	12		Opt Gap %
		CPLEX	MBH		CPLEX	MBH		CPLEX	MBH	
pr11-20	p=3	10.92	0.14	-2.09	113.72	0.19	-2.50	1497	0.25	-3.95
	p=4	19.50	0.12	-1.27	131.95	0.18	-1.11	6917.79	0.24	-3.05

length and a larger portion of nodes can be covered by the same number of team members. In Table 3.2, the heuristic average computational time remains around one second for all instances, while it takes hours to solve some sets by CPLEX. In Table 3.2, the average deviation of the MB heuristic from optimal solutions is just 0.10% which is reasonable for a heuristic solution.

To further verify the reliability of MBH, more truncated instances from (pr01-pr10) and (pr11-pr20) sets were tested and results are demonstrated in Table 3.3 and 3.4. The proposed algorithm performs similarly in all examined cases which assures its reliability for further runs on larger problems.

An instance where the MB heuristic achieved an optimal solution on a small set is demonstrated in Fig. 3.6. 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, three

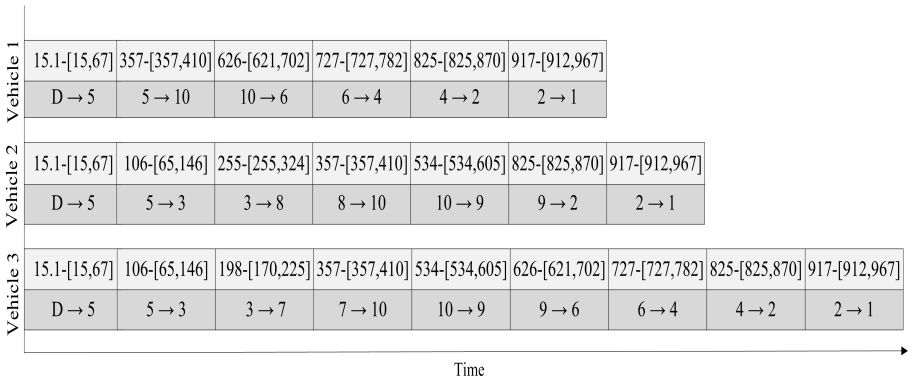

Figure 3.6: A sample scheduled tour by MBH

members leave the depot and arrive at node number 5 within its time windows [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 $T_{max} = 1236$. One can see that the illustrated routes are highly dependent such that any minor change in sequence of nodes at any tour, requires reconstruction of other routes.

We performed further experiments on larger instances. Proportional to the problem size more team members are considered in order to cover a substantial percentage of available nodes. As the objective function we report our results in the percentage value of rewards collected. Since there is no benchmark for our tests, we compare with the best CPLEX bound after an one hour run (CPLEX Best Solution%). An analysis of our results indicated that on average 1.45, 1.12, and 0.85 were the values assigned to ϑ, μ and λ, respectively. The magnitude of μ, the second largest value, implies that minimising the distance travelled, as in the CW algorithm, is an important factor in maximising the rewards collected. Perhaps, this is not surprising given the time-window aspect of the problem and that travelling time is directly proportional to distance travelled. This provided the motivation for looking at the simpler CW* algorithm with its focus on minimising distance travelled. The CW*, initialises the merit list by the following equation, $S_{i,j} = d_{0i} + d_{j0} - d_{ij}$. Having the results of the CW* allows us to evaluate our contribution through the efficacy of the applied logic in the merit function, improve function and the way the MB heuristic sorts unvisited node. Finally, in Table 3.5, 3.6 and 3.7 we report a summary of tests for large instances.

It is important to note that the MB heuristic produces better results than the CPLEX best solution in all of the sets. Furthermore, the MB heuristic collects on average 26% and 36% more of the total rewards available than those achieved by CPLEX and CW*, respectively. In instances with more than a hundred vertices the MB heurstic performs around 40% and 30% better than CPLEX and CW*, respectively. As can be observed, the performance of our algorithm gets better with

Table 3.5: Computational results for the large-size sets with 50 vertices. The last three columns show the percentages of rewards collected.

Set	#Vehicles	50				
		t_{MBH}(sec)	t_{CW^*}(sec)	MBH%	CPLEX Best Solution%	CW*%
c100	p=3	10.69	0.16	48.45	43.15	22.22
	p=4	11.25	0.17	59.04	52.20	28.29
c200	p=3	10.47	0.11	88.08	80.81	55.81
	p=4	10.72	0.38	96.37	90.26	61.92
r100	p=3	13.03	0.26	37.39	27.27	17.60
	p=4	13.75	0.26	44.58	37.85	20.98
r200	p=3	15.36	0.21	86.79	66.8	46.24
	p=4	16.23	0.15	93.01	69.34	49.56
rc100	p=3	9.81	0.20	34.15	26.93	11.98
	p=4	9.93	0.23	41.75	36.34	13.92
rc200	p=3	14.17	0.16	77.45	58.38	36.08
	p=4	15.44	0.21	84.92	65.98	40.08

Table 3.6: Computational results for the large-size sets with 100 vertices.The last three columns show the percentages of rewards collected.

Set	#Vehicles	100				
		t_{MBH}(sec)	t_{CW^*}(sec)	MBH%	CPLEX Best Solution%	CW*%
c100	p=4	60.11	1.08	36.10	23.88	11.05
	p=6	73.76	1.15	47.64	29.83	19.46
c200	p=4	88.02	1.06	74.31	47.31	35.36
	p=6	94.94	1.11	88.40	58.56	45.30
r100	p=4	88.41	1.70	33.77	18.12	11.29
	p=6	94.60	1.51	44.00	22.84	16.10
r200	p=4	133.26	1.76	76.01	38.73	28.99
	p=6	138.27	1.59	89.85	44.94	33.68
rc100	p=4	74.90	1.47	29.86	14.99	9.02
	p=6	82.97	1.49	39.53	22.29	13.26
rc200	p=4	128.52	1.65	67.04	36.96	24.93
	p=6	130.33	1.52	82.85	40.98	28.24

Table 3.7: Computational results for the large-size instances in the set of pr01-20.The last three columns show the percentages of rewards collected.

Set	#Vertices	#Vehicles	t_{MBH}(sec)	t_{CW^*}(sec)	MBH%	CPLEX Best Solution%	CW*%
pr01&11	48	p=4	15.40	0.25	64.31	54.34	32.72
		p=5	16.11	0.25	71.99	59.28	35.31
pr07&17	72	p=4	27.87	0.40	52.22	20.78	16.88
		p=5	30.31	0.35	62.08	36.08	23.95
pr02&12	96	p=4	98.74	1.60	52.05	24.71	13.73
		p=6	104.62	1.45	63.28	25.04	20.41
pr03&08 &13&18	144	p=5	237.82	3.52	42.41	5.14	12.74
		p=7	256.93	3.60	51.67	11.28	18.16
pr04&14	192	p=6	396.03	5.95	40.45	4.86	10.74
		p=8	436.35	6.15	49.41	11.36	14.47
pr09&19	216	p=6	786.69	16.40	41.21	5.63	17.23
		p=8	939.35	15.10	49.20	2.72	20.32
pr05&15	240	p=8	1116.10	37.25	43.48	6.61	11.79
		p=10	1558.30	39.45	50.98	9.33	14.49
pr06&10 &16&20	288	p=8	965.52	46.35	38.97	2.56	12.81
		p=12	1538.40	44.77	50.27	7.13	18.45

the increasing size of the problem instances.

The proposed algorithm achieves optimal solutions for 75% of instances for which the optimal results are known (342 out of 456 small instances). our implementation attains an optimality gap of 1.09% on small instances and solves benchmarks with realistic size efficiently.

3.4 Summary and discussion

In this chapter, the COPTW as an important class of the orienteering problem that arises naturally in many important applications has studied. In practical problems many of these applications require solutions as a matter of urgency. Before the work presented here, only exact methods had been used to solve the COPTW. While exact methods have been good for illustrative purposes involving small-sized instances, they are not useful for operational needs. Furthermore current heuristic algorithms are not designed to handle the complexities resulting from the requirements of synchronised visits in the COPTW.

In this chapter, we developed a new heuristic algorithm to deal with the various issues that arise from this problem. The proposed approach can solve large-sized problems in times that are appropriate for operational uses. To evaluate the solution approach, a new benchmark set was generated for the COPTW problem. The performance of the algorithm was validated 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 of the MB heuristic in terms of various metrics. The significance of the results should be seen in the light of an application like that originally introduced by Van Der Merwe et al. (2015). Where the heuristic solution produces more than seven times the rewards achieved by CPLEX this could mean seven times more structures (or even lives) saved.

Chapter 4

An adaptive large neighbourhood search for the APP

In chapter 3 the COPTW was investigated. The Asset Protection Problem (APP) is an extension to the COPTW where multiple team members (resources) with unique capabilities are involved during bushfires. In this chapter, the mixed integer linear programming model from chapeter 3 is used to solve small instances with CPLEX. Although optimal solutions were achieved in most cases the solution times precluded the method being used for operational purposes. The following chapter aims 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. Computational experiments show the efficacy of the implemented approach to achieve solutions close to optimal in time efficient manner.

4.1 An Illustrative Example

To illustrate the APP on a small example consider the problem settings in Fig.4.1. 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.

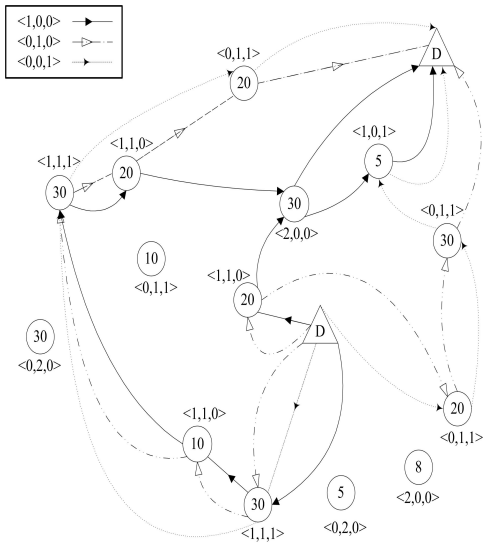

Figure 4.1: An illustrative example. Assets and depots are defined in circular and triangular shapes, respectively.

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. 4.2. 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 , x and y are the time duration in which the protection activities have to be carried out and coordinates of an asset. 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 $a = 1\ hour$.

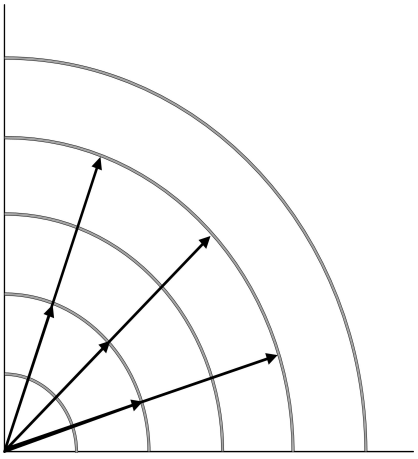

Figure 4.2: Direction of fire spread

In the graphical representation of the problem (Fig. 4.1), some assets are not protected. This is because of the time windows imposed by the advancing fire front (see Fig 4.2). 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 APP, the following problem is considered and solved by CPLEX. Figures 4.3 and 4.4 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.

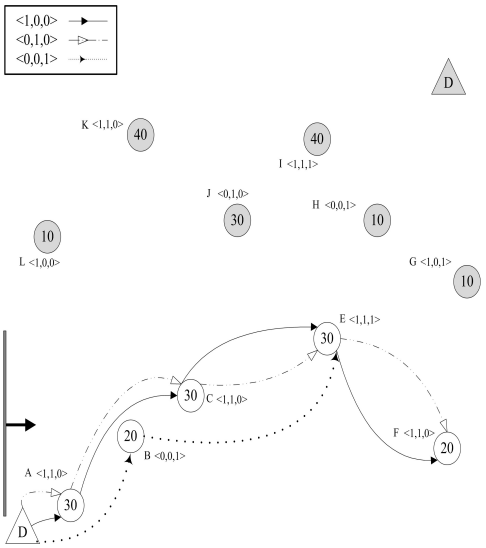

Figure 4.3: An illustrative example. Assets and depots are defined in circular and triangular shapes, respectively. Assets that are not under threat are shaded as grey, and the bold line shows the direction of fire spread.

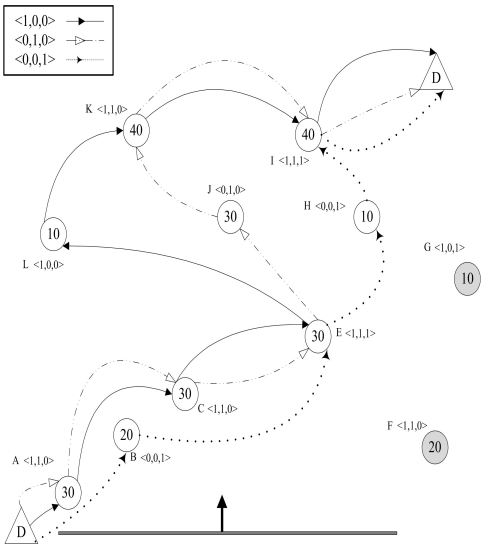

Figure 4.4: An illustrative example. Assets and depots are defined in circular and triangular shapes, respectively. Assets that are not under threat are shaded as grey, and the bold line shows the direction of fire spread.

In Figure 4.3, fire spreads at a rate of 10 $km.h^{-1}$ 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. 4.3, a disruption occurs when vehicles are at node E and a rerouting is required. In Figure 4.4, 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 4.4, the fire front sweeps over assets in a vertical manner, unlike Figure 4.3. By comparison of Figure 4.3 and 4.4, 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 thesis can be used to solve the APP 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.

4.2 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.

4.2.1 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.

4.2.1.1 Initial solution construction

In chapter 3 we proposed a heuristic to solve the COPTW named as the Merit Based (MB) heuristic. The heuristic is adapted to construct the initial solution in an efficient manner. MB heuristic 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 4.1.

Algorithm 4.1: 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 (N), Merit Pair List *(MPL)*, unvisited vertices (U), vector of resource requirement $R_i = (r_{i1}, r_{i2}, r_{i3})$, distance matrix $(d_{N\times N})$, number of routes by vehicle type $q \in Q$ (n_q)

Output: $\alpha_{best}\, and\, \beta$

```
function MB heuristic heuristic
    forall ((i, j) ∈ N) 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 MPL
    Sort MPL in descending order of S_{i,j} values
    forall (q ∈ Q) do
        forall ((i, j) ∈ N) 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 ∈ MPL) 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
```

In Algorithm 4.1, at first, a merit 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 our study in chapter 3. 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 4.1 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 APP 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 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.

$$arrive_i = departure_{i-1} + traveltime_{i-1,i} \tag{4.1}$$

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.

$$start_i^{sync} = max\{\max_{i\in \mathrm{T}} arrive_i, o_i\} \tag{4.2}$$

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

$$departure_i = start_i^{sync} + a_i \tag{4.3}$$

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

$$wait_i = start_i^{sync} - arrive_i \tag{4.4}$$

For a given route τ, a visit at node i is defined by $\tau(i)$. The value of $maxshift_{\tau(i)}$ in the equation 4.5 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.

$$\begin{aligned} maxshift_{\tau(i)} = min\{&c_{\tau(i)} - start_{\tau(i)}, \\ &wait_{\tau(i+1)} + maxshift_{\tau(i+1)}\} \end{aligned} \tag{4.5}$$

The value of the $maxshift_{\tau(i)}$ in the equation 4.5 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:

$$\begin{aligned} maxshift_{\tau(i)}^{sync} = min\{&maxshift_{\tau(i)}, \\ &\min_{i \in \mathrm{T}} maxshift_{\tau(i+1)}\} \end{aligned} \tag{4.6}$$

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

$$\begin{aligned} shift_z^{\tau,i} = &traveltime_{i,z} + wait_z + \\ &servicetime_z + traveltime_{z,i+1} - traveltime_{i,i+1} \end{aligned} \tag{4.7}$$

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.

4.2.1.2 General flow

Initially, a feasible solution $S_0 \leftarrow \beta$ is formed by using the MB 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.

4.2.1.3 Adaptive weight adjustment procedure

There is no heuristic that can perform efficiently for all types of problems. Since the APP 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.

$$p(r_s) = \frac{W(r_s)}{\sum_{j=1}^{R} W(r_j)}, p(d_s) = \frac{W(d_s)}{\sum_{j=1}^{D} W(r_j)} \tag{4.8}$$

In equation 4.8, weights dynamically and adaptively are adjusted after n iterations according to their performance. At the end of each segment weights are updated as

$$W(h) \begin{cases} (1-\rho)W(h) + \rho\frac{\pi(h)}{u(h)}, & \text{if } u(h) > 0 \\ (1-\rho)W(h), & \text{if } u(h) = 0 \end{cases} \tag{4.9}$$

,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 4.10 as, $\frac{1}{1000} > (1-\rho)^{\frac{1000}{100}} \rightarrow \rho \geq 0.602$.

$$W(h) \approx W_{initial}(1-\rho)^{[\frac{N}{n}]} \tag{4.10}$$

In equation 4.9, $\pi(h)$ and $u(h)$ record the number of times a heuristic is selected by a roulette-wheel mechanism and associate weight of the heuristic.

4.2.1.4 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 $\delta\%$ worse objective value compare to $z(S_{initial})$ with the probability of P_{accept}.

$$T_{initial} = \frac{(S_{initial} * \delta)}{\log(1/P_{accept})} \tag{4.11}$$

The achieved initial temperature by equation 4.11 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.

4.2.1.5 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.

4.2.2 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.

4.2.2.1 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.

4.2.2.2 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*.

4.2.2.3 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.

4.2.2.4 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 the APP as below:

$$\Gamma_{ij} = \theta_1 d_{ij} + \theta_2 |o_i - o_j| + \theta_3 \Omega_{ij} \tag{4.12}$$

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 4.12. 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.

4.2.2.5 Proximity-Based Removal (PR)

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

4.2.2.6 Time-Based Removal (TR)

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

4.2.2.7 Requirement-Based Removal (RR*)

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

4.2.2.8 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.

4.2.2.9 Worst-Requirements Removal (WRR*)

In the APP, 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.

4.2.2.10 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.

4.2.2.11 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.

4.2.2.12 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 APP as below.

$$f_i = \overline{\mathrm{WT_i}} + \overline{\mathrm{DC_i}} - \overline{\psi_\mathrm{i}} \tag{4.13}$$

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 $j^* = argmax_{j \in V}\{f_{ji} - f_j^*\}$.

4.2.2.13 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 APP, time windows are defined based on the cartesian coordinates. The TWR heuristic divides the whole area to four different zones (see Figure 4.5).

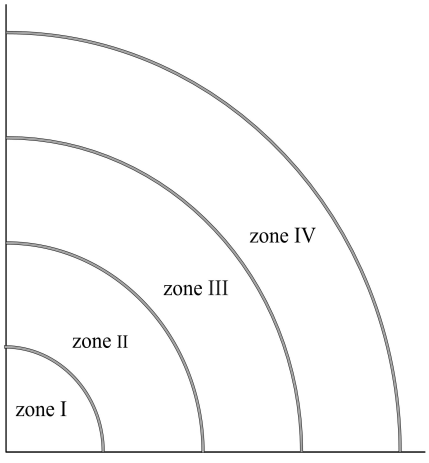

Figure 4.5: An illustrative example for TWR

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 $\Lambda_{\text{III}} = 10$ 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.

4.2.3 Insertion Algorithms

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

4.2.3.1 Classical MB 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.

4.2.3.2 Noise-Imposed MB Heuristic (NMBH*)

To apply further diversification to the search, we use the NMBH 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 NMBH 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 4.2.

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 4.1, 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

Algorithm 4.2: 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 MBheuristic
    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*
```

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.

4.3 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. 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 4.1. 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.

4.3.1 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.

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.

Table 4.1: Parameters used in the proposed algorithm

Description	**Parameter**	**Value**
Parameters for MB 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

4.3.2 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.

4.3.2.1 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 4.2. 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.

Table 4.2: 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		MB heuristic (%)	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 4.3: 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		MB heuristic (%)	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

In Table 4.2 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 "MB heuristic %". This value shows how much the ALNS improves over the initial solution found by MB heuristic. Our algorithm improves the initial solution ($MBheuristic\%$) by 20% of the total value of assets to be protected. Also, the

Table 4.4: 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(%)		MB heuristic (%)	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

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 MB heuristic 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.

4.3.2.2 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 4.3. 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 4.3 CPLEX covers about 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 APP when it comes to large 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 4.4. 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 4.4 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.

4.4 Summary and discussion

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 chapter, we developed a solution scheme for solving the APP 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 APP the ALNS offers incident-management controllers a tool that may lead to significant reduction in losses during extreme fire events.

Chapter 5

An adaptive large neighbourhood search for the COPTW

Following the proposed solution approaches in chapters 3 and 4, we further improve the proposed Adaptive Large Neighbourhood search (ALNS), in chapter 4, to implement on the COPTW . The COPTW is a broader class of problems of which the APP is a special case. In this chapter we propose an ALNS algorithm that provides a powerful framework for solving the COPTW. Newly designed removal and insertion heuristics are integrated in the body of the ALNS and evaluated based on their performance. Achieved improvements are reported after exclusion of the inefficient heuristics. In this section, we define the ALNS algorithm and explain the implemented heuristics.

5.1 ALNS procedure

The ALNS paradigm, introduced by Ropke and Pisinger (2006) which extends the large neighbourhood search previously put forward by Shaw (1998). The proposed ALNS algorithm consists of node removal and node insertion heuristics. The ALNS applies removal and insertion heuristics on the initial solution until the termination conditions are met. Stopping conditions for the ALNS heuristic are the total

number of iterations and the run-time limit. Through every iteration ($i \in N$) a removal heuristic ($d \in h_d$) is selected and partially destroys the feasible solution (S_i^-). Thereafter, a repair heuristic ($r \in h_r$) inserts unvisited nodes into a sufficient number of routes in order to achieve a better solution (S_i^+). In the ALNS, wide ranges 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 enables escape from local optimal solutions. An adaptive weight $W(h)$ and a score $\pi(h)$ is defined for each 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 $n = \frac{N}{k}$ time over each segment. The corresponding score of removal and repair heuristics are increased by σ_1, σ_2 and σ_3 after each iteration. If the new solution improves the best solution (S^*) so far, the value of σ_1 is added to the score of the associated removal and repair heuristics. The value of the σ_2 is added to scores if the resulted solution improves the best current solution through a segment. Finally, a value of σ_3 is added if the solution is accepted despite the degraded value of the achieved solution. After each segment, weights of algorithm $h \in h_d \cup h_r$ updates by using the formula $(1-\rho)W(h) + \rho \ \times \pi(h)/u(h)$ where parameter ρ is called the reaction factor regulating the number of iterations that ineffective algorithms should not substantially contribute in the solution. Also, associate weights and number of times a heuristic is selected are defined by $\pi(h)$ and $u(h)$ .The probability of using an algorithm in segment $s+1$ is defined based on the formula $prob^{s+1}(h) = W^s(h)/\sum_{j=1}^{R} W^s(h_j)$ where the denominator calculates the cumulative weight of destroy/repair heuristics and the scores are set back to zero when a segment has been completed.

It has shown by Santini et al. (2018) that acceptance criterion can significantly impact the solution quality. For this reason, we take advantage of the Simulated Annealing (SA) (Van Laarhoven and Aarts (1987)) local search framework to escape from local optimal solutions by accepting solutions with worse objective values with probability $\exp(\frac{(z(S)-z(S^*)}{T})$, while those with improved objective values will always be accepted. In the SA, $z(S)$ and $z(S^*)$ represent the objective value of the current and best solution. The initial temperature $T_{initial}$ in a way to accept solutions with

$\delta\%$ worse objective value compare to $z(S_{initial})$ with probability of P_{accept}. The initial temperature can be set as $(S_{initial} * \delta)/\log(1/P_{accept})$ where $0 < \epsilon < 1$ is the cooling rate, $(T = \epsilon * T)$.

5.2 Insertion feasibility

To efficiently conduct insertion operations, we filter out infeasible arcs through the search process, as discussed in chapter 4. We initially filter arc_{ij} when $o_i+a_i+t_{ij} > c_j$ for all nodes. 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 binary matrix named as the "feasibility matrix" to facilitate the insertion process. Most importantly, to evaluate the possibility of an insertion in time efficient manner, 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 (see chapter 4).

5.3 Insertion algorithms

To handle The complexities of synchronous visits we implement the proposed Merit Based (MB) heuristic as explained in chapter 3. The MB heuristic is utilised with different approaches to get the most out of it, namely classical MB heuristic, noised imposed MB heuristic, cumulative MB and noised imposed cumulative MB heuristic. In addition to variants of the MB heuristics, an algorithm designed to take relative travel time into account and named as greedy time heuristic. A total of five insertion algorithms were designed and are illustrated below.

5.3.1 Classical merit based heuristic

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

$$M_{i,j} = \vartheta \, \frac{\psi_i + \psi_j}{\overline{\psi}} + \frac{d_{i0} + d_{0j} - \lambda \, d_{ij}}{d^{max}} + \mu \, \cos\theta_{ij} \, \frac{|d^{max} - (d_{i0} + d_{0j})/2|}{d^{max}}. \quad (5.1)$$

In equation 5.1, θ_{ij} is the angle between the vector from the depot to node i and the vector from the depot to node j, ψ_i and $\overline{\psi}$ represent the score for each node and the average score of all vertices. Also, in the above formulation, d_{ij} is the distance between nodes i and j and $d^{max} = \max\{d_{i,j};\ \forall i, j \in N\}$ used to scale. To ensure the assignment of nodes with highest merit values prior to others, we implemented parallel route construction in the algorithm. Moreover, best values for three parameters, ϑ, λ and μ, are defined in the parameter tunning of the algorithm.

The Algorithm 5.1. represents the MB heuristic. The algorithm starts with ranking unvisited nodes according to the ratio $\frac{\psi_m}{\sqrt{r_m}}$ where ψ_m and r_m are the rewards (scores) available and resources (visits) required 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. Then the algorithm creates the euclidean distance matrix and feasibility matrix as explained in section 5.2. A Merit Pair List (MPL) is constructed and sorted in descending order of $M_{i,j}$ in lines 5 and 6. The algorithm iterates over each node in set U (line 7) for every pair (line 9) to assign unvisited nodes into the existing subroutes. If resources allow a new route would be opened in line 17. If the algorithm accomplishes a successful insertion, the relevant information is updated in lines 22-25.

Algorithm 5.1: Pseudocode for the merit based heuristic

Input: best collected reward (α), best routes (β), unvisited vertices (U), Merit Pair List (MPL), score of node m (ψ_m), maximum number of travelling team members (P), number of sub-routes (n), service requirement of node m (r_m), distance matrix of nodes($d_{N\times N}$) , node visited before node q (q')

Output: β and α

function MB heuristic
- sort U in descending order of $\frac{\psi_m}{\sqrt{r_m}}, \quad \forall m \in U$
- generate $d_{N\times N}$
- calculate $feasibility\ matrix$
- calculate merit list and sort($< i, j, M_{i,j} >$)
- $M_{i,j} \leftarrow \vartheta \dfrac{\psi_i + \psi_j}{\overline{\psi}} + \dfrac{d_{i0} + d_{0j} - \lambda\, d_{ij}}{d^{max}} + \mu\, \cos\theta_{ij}\, \dfrac{|d^{max} - (d_{i0} + d_{0j})/2|}{d^{max}}$
- **for** *($m \in U$)* **do**
  - $L \leftarrow \beta$
  - **for** *($pair_{p,q} \in MPL$)* **do**
    - **for** *($subroutes \in L$)* **do**
      - **if** *($m == p$ && $q \in L$ && $VisitCount_m < r_m$)* **then**
        - **if** *($shift_k^{\tau,i+1} \leq wait_{i+1} + maxshift_{\tau(i+1)}^{sync}$)* **then**
          - remove arc (q', q), add arcs (q', m) and (m, q)
          - update $maxshifts$ and $insertion\ feasibility\ matrix$
          - Update L
          - $VisitCount_m \leftarrow VisitCount_m + 1$
      - **if** *($n < P$ && $VisitCount_m < r_m$)* **then**
        - Create a new route and add m
        - Update L
        - $n \leftarrow n + 1$
        - $VisitCount_m \leftarrow VisitCount_m + 1$
    - **if** *($VisitCount_m == r_m$)* **then**
      - update β
      - update α
      - update U
- **return** β and α

5.3.1.1 Noise-imposed merit based heuristic

The noise-imposed merit based heuristic (NMB) heuristic is used to bring additional diversification to the search (Emeç et al. (2016)). We define a noise parameter $0 < \alpha_{noise} < 1$, then $\Delta = \alpha_{noise} * max\{M_{ij}\}$ is the allowed amount of noise. The

NMB heuristic updates merit values by $M_{ij} = M_{ij} + \xi$ where $\xi \in [-\Delta, \Delta]$.

5.3.1.2 Cumulative merit based heuristic

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

$$M_{i,j,k} = M_{i,k} + M_{k,j} - M_{i,j} \tag{5.2}$$

The Cumulative Merit Based (CMB) heuristic uses the calculated merit values for pairs in equation 5.1 and determines values of $M_{i,j,k}$ by equation 5.2. Figure 5.1 illustrates insertion operation before (figure 5.1(a)) and after (figure 5.1(b)) using equation 5.2. Insertion by equation 5.1 may result a route as defined in Figure 5.1(a). In the generated merit pair list using the equation 5.2, as both succeeding and preceding nodes are considered, the $M_{i,j,k}$ gets a larger value than $M_{i,L,k}$. Therefore, insertion will take place as shown in Figure 5.1(b) which is the optimal assignment.

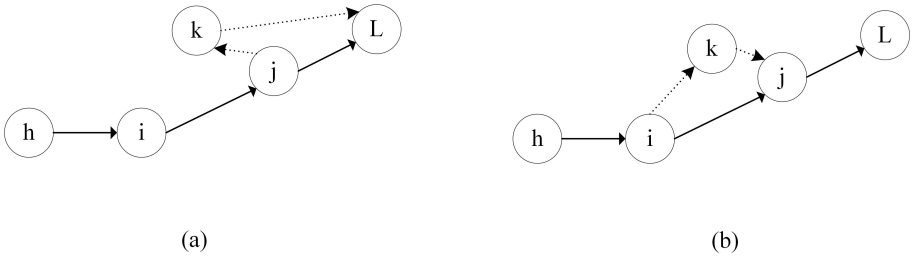

Figure 5.1: A sample improvement by cumulative merit-based heuristic.

5.3.1.3 Noise-imposed cumulative merit based heuristic

The same approach as in section 5.3.1.1 is used for the noise-imposed cumulative merit based (NCMB) heuristic to explore a broader search space.

5.3.1.4 Greedy time heuristic

As travel time minimisation may result in an increase in the number of nodes visited, and hence in the rewards collected, the Greedy Time (GT) heuristic is developed. The GT heuristic focuses on travel time minimisation, we generate a list of tuples $< i, j, k, TR_{ijk} >$ and sort them in ascending order of TR_{ijk} values. The TR_{ijk} is the time ratio of node k for insertion between nodes i and j. Note that, although equation 5.3 focuses more on travel time, it includes other problem specific attributes in the denominator.

$$TR_{ijk} = \frac{t_{i,k} + t_{k,j} - t_{i,j}}{\frac{\psi_k}{\sqrt{r_k/P}}} \tag{5.3}$$

In equation 5.3, the numerator calculates the increase in the travel time as a result of inserting node k between i and j. The ψ_k is the associated score for node k which is divided by the square root of the ratio between node k requirements (r_k) and total number of team members available(P).

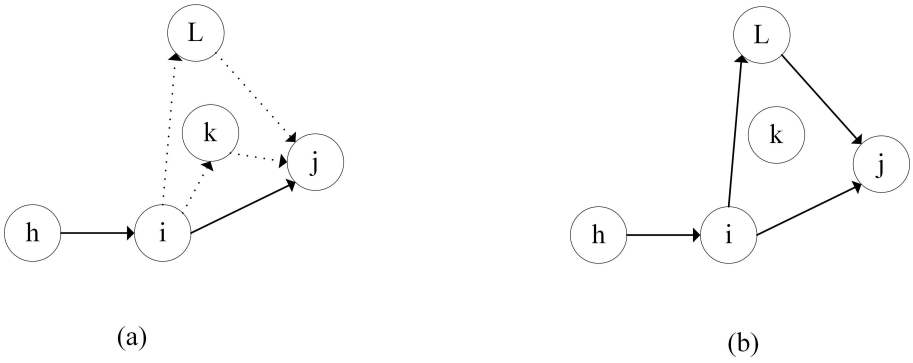

Figure 5.2: A sample insertion by greedy time heuristic.

In Figure 5.2(a) two alternatives are shown for insertion between nodes i and j. Although visiting node L demands a longer travel time, the GT heuristic inserts L into the route. This is because $\psi_L > \psi_k$ which impacts on the total value of the TR_{ijL} and hence will be ranked higher in the list.

5.3.2 Destroy methods

Using one of the removal (destruction) algorithms discussed below, the initial solution generated by the classical MB heuristic (S_0)is partially destroyed. At each iteration, a removal 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. We utilise removal algorithms presented in chapter 4 and Santini (2019) as discussed in detail below.

The Random Removal (RR) algorithm randomly removes D nodes from the solution. Worst-Distance Removal (WDR) determines the distance of nodes from succeeding and preceding nodes and sorts them in a descending order in a list named as O. Then, it removes nodes in position $\lfloor \Upsilon^{\kappa} |O| \rfloor$ from the solution and the list. Parameters $\kappa \geq 1$ and $0 < \Upsilon < 1$ bring randomness to select new nodes for removal. Worst-Time Removal (WTR) performs same as the WDR, however the list O is defined based on values of $|sync\_start_i - o_i|$ (see chapter 4 for more details). Shaw Removal (SR) measures relatedness between nodes using $\Gamma_{ij} = \theta_1 d_{ij} + \theta_2 |o_i - o_j| + \theta_3 \Omega_{ij}$ where θ_1, θ_2 and θ_3 are the Shaw parameters and Ω_{ij} 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 $\lfloor \Upsilon^{\eta} |O| \rfloor$ will define the node that has to be removed from the relatedness list $|O|$. In the SR algorithm parameters $\eta \geq 1$ is the determinism factor and $1 \geq \Upsilon \geq 0$ is a random number. The SR increase the chance of substitution of nodes with the ones that have high level of similarities in terms of distance, routes in common and time window, so that the algorithm can move to a new neighbourhood. Proximity Removal (PR), Time Removal (TR) and Requirement-Based Removal (RBR) algorithms are special case of the SR heuristic where where θ_1, θ_2 and θ_3 get a value of 1, but others are set to 0.

Waiting-Time Oriented Removal (WTOR) is similar to the WDR, however the cost is the waiting time calculated as $WT_i = sync\_start_i - arrive_i$. Worst Requirement Removal (WRR) algorithm removes nodes with the highest demand from the

solution. Relative-Requirement Removal (RRR1) heuristic is designed for a heterogeneous fleet of vehicles (members). The RRR1 algorithm follows the same logic as the WRR, however considers the available number of members from each type. Therefore, nodes with lowest value of ω defined as $\omega = r_{iq} * Z_i$ have a higher chance of removal, where r_{iq} and $Z_q \in \{-1, 0, 1\}$ are the number of member types q needed to visit node i and the assigned importance rate of member type q, respectively. the parameter Z_q gets larger values when more members of type q are available. The 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 $f_i = \overline{\mathrm{WT_i}} + \overline{\mathrm{DC_i}} - \overline{\psi_\mathrm{i}}$ for removing nodes, where the first two terms are normalised values from the WTOR and WDR algorithms. The HNR removes nodes with the worst $j^* = argmax_{j \in V}\{f_{ji} - f_j^*\}$ where f_j^* is the best position cost of node j before iteration i. Time Windows-Oriented Removal (TWR) algorithm removes nodes with similar time windows using the same approach as the HNR algorithm. The TWR algorithm divides the total time window length to four zones and removes $\Lambda_i = \frac{D * zu_i}{tu}$ number from each zone, where tu, zu_i denote respectively the total unvisited nodes and unvisited nodes in the zone i. Every time one of the WDR, WTR, WTOR, WRR 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 proved to be efficient for classical OPs (Santini (2019)). We select a random cluster and remove D nodes from the chosen cluster that exist in the current solution. The process of random selection of clusters and removal of nodes from the intersection of the cluster and the solution continuous until D nodes are being removed.

The overall description of our algorithm is given in Algorithm 5.2. The ALNS algorithm determines an initial solution (S_0) by the merit based heuristic. Then, probabilities for insertion and removal heuristics are initialised in line 5. The initial temperature needs to be initialised for the SA algorithm and is updated in line

22 through the search process. Using the roulette-wheel mechanism (Lipowski and Lipowska, 2012) as an adaptive layer of the ALNS a destruction and repair heuristic is selected through every iteration in lines 10 and 11. Depending on the resulting improvements by the selected algorithms, we update the scores for the heuristics in line 18. Finally, the ALNS returns the best solution (S^*) found after N iterations in line 23.

Algorithm 5.2: Pseudocode for the ALNS Algorithm

Input: Number of iterations (N), Initial solution (S_0), Number of iterations for each segment (n), Set of destruction heuristics (h_d), set of repair heuristics (h_r)

Output: Best solution (S^*)

```
1  function ALNS algorithm with simulated annealing
2      Generate initial solution S_0 using MB heuristic
3      i ← 1
4      Let S* ← S_i ← S_0
5      Initialise Prob(r_s), Prob(d_s) for each s ∈ h_d ∪ h_r
6      Initialise T_initial
7      while (i ≤ N) do
8          j ← 1
9          while (j ≤ n) do
10             Select a removal heuristic d ∈ h_d → (S_i^-)
11             Select a repair heuristic r ∈ h_r → (S_i^+)
12             if z(S*) ≤ z(S_i^+) then
13                 S* ← S_i^+
14             if z(S_i) ≤ z(S_i^+) then
15                 S_i ← S_i^+
16             if z(S_i) ≥ z(S_i^+) then
17                 Using SA criterion to accept/reject S_i^+
18             Update π_s for the selected heuristics
19             i ← i + 1
20             j ← j + 1
21         Update adaptive weights of heuristics, s ∈ h_d ∪ h_r
22         Update temperature
23     return S*
```

5.4 Computational results

Extensive numerical studies were conducted to evaluate the efficacy of the proposed solution approach. A set of benchmark instances was generated by adding the problem-specific attribute to the well-known existing benchmark sets (see Vansteenwegen et al. (2009)). This involved adding the resource requirement at each vertex by picking 1, 2 or 3 randomly. This number indicates how many members of the team are required to collect the associated reward at each node[1].

In the first study, we investigate the performance of the removal and insertion algorithms. Thereafter, we validate the performance of our ALNS algorithm against the published results for the APP due to the similarity to the COPTW. We test our ALNS algorithm on 100 node instances for two sets of members. In the next study, truncated benchmark sets are designed to solve sufficiently small-size instances by means of both the CPLEX commercial solver and our algorithm. The number of vertices in the small-size 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 computation time and objective value on the other hand. We furthermore show the efficient performance of the proposed heuristic in terms of time and accuracy on the large-size benchmark instances and present our results as benchmarks for future studies. All the above computational work was performed on a single CPU with 16GB 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.

The parameter tuning was performed by following literature (Emeç et al. (2016); Ropke and Pisinger (2006)). We defined parameter values as explained in chapter 4.

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

The parameters were determined using the 100-node APP's benchmark instances: R104, R206, RC104, RC108 and RC206. Parameters and their tuned values are summarised in Table 5.1.

Table 5.1: Parameters used in the proposed algorithm.

Description	**Parameter**	**Value**
Parameters for MB 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

We investigate the performance of the insertion and removal heuristics to remove out the inefficient heuristics. we conduct the sensitivity analysis on the heuristics using the same instances as the parameter tuning: R104, R206, RC104, RC108 and RC206. Numbers in Table 5.2 represent the average results for 5 instances after 10 runs (5instances×10runs). The second column denotes the name of algorithms, the third column illustrates the percentage of total iterations that a corresponding algorithm is chosen. Among the removal algorithms, WTOR, 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 does not mean that removing it will improve the solution since it may help the ALNS to escape the local optimum to find a better solution Emeç et al. (2016). To identify whether exclusion of an algorithm may improve the final solution, we conduct further experiments omitting one algorithm at a time while keeping the others. The fourth, fifth and sixth columns show the change in the objective values after exclusion of the

Table 5.2: Statistics for performance of the removal and insertion algorithms. Deviation in objective value after exclusion of each algorithm are shown in percentage. In the last column, "N" and "Y" represent inclusion and exclusion of a heuristic from the algorithm.

#	Algorithm	Average % usage	Average % dev in worst obj value	Average % dev in avg obj value	Average % dev in best obj value	Exclusion
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	NMB	24.69	-0.18	-0.24	-0.30	N
5	MB	24.60	-0.50	-0.90	-0.53	N
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	RBR	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

corresponding algorithm. A positive value shows an improvement in the objective function's value after leaving out the associated algorithm. For instance, exclusion of the MB heuristic has a negative impact on the worst, average and the best values of the objective function in 10 runs. However, exclusion of the NCMB can improve the ALNS performance. Subsequently, the last column shows if an algorithm is excluded from the ALNS. Finally, we keep algorithms that can add positive contribution in either average or best value of the objective function.

We investigate the effectiveness of our algorithm by comparing our average and best results with the results reported in chapter4. They recently studied a problem named as APP during wildfires where community assets are endangered by the moving fire front. An asset can be protected if sufficient number of vehicles (i.e. members) from different types accomplish tasks within the time window. Our results achieved by using the parameters in Table 5.1 after exclusion of inefficient algorithms. In Table

5.3, two sets of vehicles are defined as in the original paper and average and best solution are used for comparison. Results under the label "APP" referring to asset protection problem's results and ALNS refers to our proposed algorithm.

Table 5.3: A summary 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 in the percentage value of scores collected.

Instances	# Vehicles	APP		ALNS		Δ_{avg}	Δ_{best}
		Avg	Best	Avg	Best		
C100	Set1	60.17	61.84	62.34	63.18	3.48	2.12
	Set2	66.66	68.35	68.46	69.70	2.63	1.95
C200	Set1	59.04	60.72	61.06	61.92	3.31	1.94
	Set2	64.87	66.58	66.73	67.73	2.79	1.70
R100	Set1	61.19	62.50	63.39	64.49	3.47	3.09
	Set2	68.45	69.86	70.28	71.31	2.60	2.03
R200	Set1	63.64	65.30	65.80	66.75	3.28	2.17
	Set2	69.76	71.38	71.66	72.74	2.65	1.87
RC100	Set1	66.77	68.59	68.84	69.98	3.01	1.99
	Set2	73.21	75.17	74.94	75.85	2.31	0.90
RC200	Set1	67.17	69.13	69.35	70.45	3.14	1.87
	Set2	73.12	74.71	74.95	75.79	2.44	1.42

In Table 5.3, Δ_{best} and Δ_{avg} are the percentage deviation from the reported results in the literature. Our proposed algorithm improves results for all instances and enhances the average results for the Δ_{best} and Δ_{avg} by around 2% and 3%, respectively. Table 5.3 shows that our approach performs efficiently and validates our algorithm on the existing benchmarks.

For further evaluation, a collection of small-size benchmark instances were generated and solved by means of both CPLEX and the ALNS algorithm. 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 Table 5.4. The sizes of truncated instances are chosen in a way to investigate the correlation between the increase in problem size and exponential growth in computational effort. Infeasible edges are excluded in MILP formulations to simplify models for the CPLEX implementation. Also, larger problems with more than 12 nodes could not be solved within the time limit of 5 hours.

In Table 5.4 computation times and the optimality gaps are reported. The "$\Delta_{average}$"

and "Δ_{best}" represent the percentage gap between the CPLEX optimal solution and ALNS algorithm average and best results. In Table 5.4, the average gap for $\Delta_{average}$ and Δ_{best} are 0.06% and 0.00% which shows the promising performance of the proposed algorithm. Moreover, it can be seen that the computation time increases significantly with minor changes in the problem size for CPLEX compared with the negligible changes for the ALNS.

Table 5.4: 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	$\Delta_{average}$	Δ_{best}	CPLEX	ALNS	$\Delta_{average}$	Δ_{best}
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	-0.20	0.00
RC100	p=3	356.87	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

Table 5.5: 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	$\Delta_{average}$	Δ_{best}	CPLEX	ALNS	$\Delta_{average}$	Δ_{best}
C200	p=3	140.95	51.79	0.00	0.00	217.15	55.87	0.00	0.00
	p=4	145.81	92.63	0.00	0.00	144.80	104.69	0.00	0.00
R200	p=3	351.03	117.21	0.00	0.00	473.92	128.09	0.00	0.00
	p=4	224.51	132.4	0.00	0.00	286.45	134.96	0.00	0.00
RC200	p=3	509.47	130.13	0.00	0.00	4516.54	147.05	-0.38	0.00
	p=4	313.45	143.61	0.00	0.00	336.61	168.53	-0.04	0.00

Table 5.5 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 5.4. 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 5.5, the average computational time remains around two minutes for all instances, while it takes hours to solve some sets by CPLEX. In Table 5.5, the average deviation of the ALNS algorithm from optimal solutions is 0.00% for Δ_{best} and just 0.03% for $\Delta_{average}$ which is reasonable. Due to the CPLEX computational

time we are unable to solve larger instances in Table 5.5 while for some instances large amounts of nodes are covered.

We performed further experiments on larger instances. Proportional to the problem size more team members are considered in order to cover a substantial percentage of available scores. As the objective function we report our results in the percentage value of scores collected. Since there is no benchmark for our tests, we report the results obtained with the ALNS as benchmarks. In the last two columns of Table 5.6, we report the most effective insertion and removal algorithms for each set of instances.

Table 5.6: Computational results for the large-size sets 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/SR	MB/NCMB/NMB/GT
C200	p=4	72.17	73.25	74.65	WDR/WTR/HNR/RR1	MB/NMB/NCMB/GT
	p=6	85.84	87.00	88.26	WDR/HNR/WTR/WTOR	MB/NMB/NCMB/GT
R100	p=4	32.48	33.48	34.44	WDR/WTR/WRR/RRR1	MB/NCMB/NMB/GT
	p=6	42.26	43.40	44.71	WDR/RRR1/WTR/WTOR	MB/NMB/GT/NCMB
R200	p=4	74.26	75.99	78.32	WDR/WTR/SR/TR	MB/NCMB/NMB/GT
	p=6	87.00	88.57	90.61	WDR/WTR/RRR1/SR	MB/NMB/GT/NCMB
RC100	p=4	30.12	30.78	31.51	HNR/WTR/WTOR/WDR	MB/NMB/GT/NCMB
	p=6	39.96	40.74	41.51	WDR/WTOR/HNR/RRR1	MB/NMB/GT/NCMB
RC200	p=4	66.63	68.37	70.44	WDR/WTR/WTOR/TR	MB/NMB/GT/NCMB
	p=6	81.70	83.61	86.09	WTR/WDR/RRR1/SR	MB/NCMB/NMB/GT

5.5 Summary and discussion

The COPTW is an important class of the orienteering problem that arises naturally in many important applications. 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, in this chapter, we developed a new ALNS algorithm equipped with the MB heuristic as an insertion operator to deal with the various issues that arise from this problem. Together with the MB heuristic 19 heuristics were developed using the problem specific attributes. Then, 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 purposes.

In order to validate the solution approach, we modified our algorithm according to the specification of the existing benchmarks and performed sets of tests on the APP. In the APP any minor improvements in the objective function is significant and important as it means that more assets can be protected. Not only is our proposed algorithm running efficiently but improvements in the results are also achieved.

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 various metrics.

Chapter 6

A two-stage stochastic approach for the APP

Many practical applications of the APP and the COPTW involve uncertainties that need to be taken into account. For example, in the APP during uncontrollable wildfires, incident managers dispatch vehicles for tasks aimed at reducing the hazard to key assets. The deployment plan is complicated by the need for vehicle capabilities to match asset requirements within time-windows determined by the progression of the fire. This is often further complicated by a wind change that is expected but with uncertainty in its timing. This chapter aims to identify the best practice for determining plans for the deployment of resources under various circumstances. To this end, a two-stage stochastic model is developed for solving an APP using the context of the 2009 Black Saturday wildfires. Then, a dynamic rerouting approach is presented to empower the reactive ability of decision makers. Additionally, an ALNS algorithm is implemented to solve the dynamic rerouting APP in a a time efficient manner.

6.1 Problem description and model formulation

The two-stage stochastic approach for the APP encompasses multiple characteristics of the problem, such as a heterogeneous fleet of trucks, multiple scenarios, uncertain time of change and locations, imposed time windows by the fire front, and synchronous service requirements. The problem is formulated in a generic manner to facilitate its application to analogous problems.

6.1.1 Sets, parameters and decision variables

The mixed integer linear programming formulation uses the following notation.

Indices and sets

Q	set of vehicle types
A	set of all arcs
$\delta_q^+(i)$	set of feasible arcs (i,j) that can be traversed from i by vehicle type $q \in Q$
$\delta_q^-(j)$	set of feasible arcs (i,j) that can be traversed to j by vehicle type $q \in Q$
N	set of all assets
Ξ	set of all scenarios, $\xi_c \in \Xi$
F	number of scenarios, $\Xi := \{\xi_1, \xi_2, \ldots, \xi_F\}$

Parameters

a_i	service duration associated with location i
c_i^f	latest time that protection activities may commence at location i in stage f (first stage)
$c_i^s(\xi_c)$	latest time that protection activities may commence at location i in stage s (second stage) in scenario $\xi_c \in \Xi$
$P(\xi_c)$	the probability that scenario $\xi_c \in \Xi$ occurs
M	A sufficiently large number
n	number of assets in the graph representation of the problem
o_i^f	earliest time that protection activities may commence at location i in stage f

$o_i^s(\xi_c)$ earliest time that protection activities may commence at location i in stage s, scenario $\xi_c \in \Xi$

κ_q number of vehicles of type $q \in Q$

r_i^f vector of protection requirement for asset i in stage f, e.g. $r_i =< r_{i,q=1} = 2, r_{i,q=2} = 1, r_{i,q=3} = 0 >$, representing number of required vehicles of each type at node i

$r_i^s(\xi_c)$ vector of protection requirement for asset i in stage s, scenario $\xi_c \in \Xi$

t_{ijq} travel time from location i to location j by vehicle type $q \in Q$

ν_i value of asset i

TO_c time of occurrence for scenario c, $TO_1 \leq TO_2 \leq \cdots \leq TO_c \leq \cdots \leq TO_F$, where TO_1 is the staging time that transition from one stage to another takes place

T_{max} latest time allowed to start a protective task

$0, N+1$ represents the start and final locations (may be the same location)

Variables

w_j 1 if asset j is the last node visited before stage transition occurs, 0 otherwise

First stage decision variables

X_{ijq}^f number of vehicles of type $q \in Q$ travelling from location i to location j at stage f

z_{ijq}^f 1 if arc (i, j) is traversed by vehicle type $q \in Q$, 0 otherwise

S_i^f time at which service commences in location i at stage f

Y_i^f 1 if location i is serviced at stage f, 0 otherwise

Second stage decision variables

$z_{ijq}^s(\xi_c)$ 1 if arc (i, j) is traversed by vehicle type $q \in Q$ at stage s in scenario $\xi_c \in \Xi$, 0 otherwise

$\Gamma_i(\xi_c)$ 1 if node i under scenario ξ_c is visited before the time TO_c

$S_i^s(\xi_c)$ time at which service commences in location i at stage s in scenario $\xi_c \in \Xi$

$X_{ijq}^s(\xi_c)$ number of vehicles of type $q \in Q$ travelling from location i to location j at stage s in scenario ξ_c

$Y_i^s(\xi_c)$ 1 if location i is serviced at stage s in scenario $\xi_c \in \Xi$, 0 otherwise

6.1.2 Mathematical model

Consider a set of n assets with depots 0 and $N+1$. The start and end depots may be the same. Each asset i is associated with a value ν_i and a vector of service requirements $r_i =< r_{i1}, r_{i2}, \ldots, r_{iq} >$. Variable R_i indicates the number of vehicles of each type needed to accomplish a task cooperatively to service an asset. Variables Y_i is binary and equal 1 if an asset is protected and 0 otherwise. The decision variable z_{ijq} takes value 1 if arc (i,j) traversed by vehicle q while variables X_{ijq} and t_{ijq} represent number of vehicles travel along arc (i,j) and the travel time. Superscripts f and s indicate the stage and $P(\xi_c)$ is the probability of scenario ξ_c at the second stage. Every asset should be visited within the associated time window $[o_i, c_i]$ depending on the scenario and the stage. The set of all arcs is defined by A. The set $\delta_q^+(i)$ is the set of all feasible arcs (i,j) for each asset i such that both assets i and j need to be visited by vehicle type $q \in Q$ and can be reached within their time windows. Similarly, $\delta_q^-(j)$ is the set of all feasible incoming arcs to asset j. Parameter TO_1 represents the staging time where a change in the problem conditions occurs. A stage transition may occur at any location on an arc or at an asset where a vehicle is located when the staging time occurs. w_i is assigned the value 1 if i is the last asset on a route to be visited before the commencement of stage two. This enables us to maintain information through the stage transition. The superscripts f and s indicate variables for the first and second stage, respectively.

With the notations and explanations above, the problem is formulated as follows:

$$Max \quad \sum_{i \in N} \nu_i Y_i^f + \sum_{\xi_c \in \Xi} P(\xi_c)(\sum_{i \in N} \nu_i Y_i^s(\xi_c)) \tag{6.1}$$

$$\begin{aligned} s.t.: \sum_{j \in \delta_q^+(0)} X_{0jq}^f + \sum_{j \in \delta_q^+(0)} X_{0jq}^s(\xi_c) = \sum_{i \in \delta_q^-(N+1)} X_{i(N+1)q}^f + \\ \sum_{i \in \delta_q^-(N+1)} X_{i(N+1)q}^s(\xi_c), \qquad \forall q \in Q, \forall \xi_c \in \Xi; \end{aligned} \tag{6.2}$$

$$\sum_{i\in\delta_q^-(j)} X_{ijq}^f + \sum_{i\in\delta_q^-(j)} X_{ijq}^s(\xi_c) = \sum_{k\in\delta_q^+(j)} X_{jkq}^f + \sum_{k\in\delta_q^+(j)} X_{jkq}^s(\xi_c), \qquad \forall q \in Q, \forall j \in N, \forall \xi_c \in \Xi; \tag{6.3}$$

$$\sum_{q\in Q}\sum_{k\in\delta_q^+(j)} X_{jkq}^f - \sum_{q\in Q}\sum_{i\in\delta_q^-(j)} X_{ijq}^f \geq -M \times w_j, \qquad \forall j \in N; \tag{6.4}$$

$$\sum_{q\in Q}\sum_{k\in\delta_q^+(j)} X_{jkq}^f - \sum_{q\in Q}\sum_{i\in\delta_q^-(j)} X_{ijq}^f \leq -w_j, \qquad \forall j \in N; \tag{6.5}$$

$$S_j^f + a_j - TO_1 \leq M(1 - z_{ijq}^f), \qquad \forall q \in Q, \forall (i,j) \in A; \tag{6.6}$$

$$S_j^s(\xi_c) + a_j - TO_1 \geq M(z_{ijq}^s(\xi_c) - 1), \qquad \forall q \in Q, \forall (i,j) \in A, \forall \xi_c \in \Xi; \tag{6.7}$$

$$\sum_{j\in\delta_q^+(k)} X_{0jq}^f + \sum_{j\in\delta_q^+(j)} X_{0jq}^s(\xi_c) \leq \kappa_q, \qquad \forall q \in Q, \forall \xi_c \in \Xi; \tag{6.8}$$

$$\sum_{i\in\delta_q^-(j)} X_{ijq}^f = r_{jq}^f Y_j^f, \qquad \forall j \in N, \forall q \in Q; \tag{6.9}$$

$$\sum_{i\in\delta_q^-(j)} X_{ijq}^s(\xi_c) = r_{jq}^s(\xi_c) Y_j^s(\xi_c), \qquad \forall j \in N, \forall q \in Q, \forall \xi_c \in \Xi; \tag{6.10}$$

$$Y_j^f + Y_j^s(\xi_c) \leq 1, \qquad \forall j \in N, \forall q \in Q, \forall \xi_c \in \Xi; \tag{6.11}$$

$$X_{ijq}^f \leq \kappa_q z_{ijq}^f, \qquad \forall (i,j) \in A, \forall q \in Q; \tag{6.12}$$

$$X^s_{ijq}(\xi_c) \leq \kappa_q z^s_{ijq}(\xi_c), \qquad \forall (i,j) \in A, \forall q \in Q, \xi_c \in \Xi; \tag{6.13}$$

$$S^f_i + t_{ijq} + a_i - S^f_j \leq M(1 - z^f_{ijq}), \qquad \forall (i,j) \in A, \forall q \in Q; \tag{6.14}$$

$$S^s_i(\xi_c) + t_{ijq} + a_i - S^s_j(\xi_c) \leq M(1 - z^s_{ijq}(\xi_c)), \qquad \forall (i,j) \in A, \forall q \in Q, \forall \xi_c \in \Xi; \tag{6.15}$$

$$S^f_i - S^s_i(\xi_c) \leq M(1 - w_i), \qquad \forall (i) \in N, \forall \xi_c \in \Xi; \tag{6.16}$$

$$S^f_i - S^s_i(\xi_c) \geq M(w_i - 1), \qquad \forall (i) \in N, \forall \xi_c \in \Xi; \tag{6.17}$$

$$TO_c \leq S^s_i(\xi_c) + M \times \Gamma_i(\xi_c), \qquad \forall i \in N, \xi_c, \xi_{c'} \in \Xi \setminus \xi_1; \tag{6.18}$$

$$TO_c \geq S^s_i(\xi_c) - M \times (1 - \Gamma_i(\xi_c)), \qquad \forall i \in N, \xi_c, \xi_{c'} \in \Xi \setminus \xi_1. \tag{6.19}$$

$$S^s_i(\xi_c) - S^s_i(\xi_{c'}) \leq M(1 - \Gamma_i(\xi_c)), \qquad \forall i \in N, \xi_c, \xi_{c'} \in \Xi \setminus \xi_1, c < c'; \tag{6.20}$$

$$S^s_i(\xi_c) - S^s_i(\xi_{c'}) \geq M(\Gamma_i(\xi_c) - 1), \qquad \forall i \in N, \xi_c, \xi_{c'} \in \Xi \setminus \xi_1, c < c'; \tag{6.21}$$

$$X^s_{jiq}(\xi_c) - X^s_{jiq}(\xi_{c'}) \leq M(1 - \Gamma_i(\xi_c)), \qquad \forall i, j \in N, q \in Q, \xi_c, \xi_{c'} \in \Xi \setminus \xi_1, c < c'; \tag{6.22}$$

$$X^s_{jiq}(\xi_c) - X^s_{jiq}(\xi_{c'}) \geq M(\Gamma_i(\xi_c) - 1), \qquad \forall i, j \in N, q \in Q, \xi_c, \xi_{c'} \in \Xi \setminus \xi_1, c < c'; \tag{6.23}$$

$$o^f_j \leq S^f_j \leq c^f_j, \qquad \forall j \in N; \tag{6.24}$$

$$o^s_j(\xi_c) \leq S^s_j(\xi_c) \leq c^s_j(\xi_c), \qquad \forall j \in N, \forall \xi_c \in \Xi; \tag{6.25}$$

$$Y^f_i, Y^s_i(\xi_c) \in \{0, 1\}, \qquad \forall i \in N, \forall \xi_c \in \Xi; \tag{6.26}$$

$$z^f_{ijq}, z^s_{ijq}(\xi_c) \in \{0, 1\}, \qquad \forall (i, j) \in A, \forall q \in Q, \forall \xi_c \in \Xi; \tag{6.27}$$

$$w_i \in \{0, 1\}, \qquad \forall i \in N; \tag{6.28}$$

$$\Gamma_i(\xi_c) \in \{0, 1\}, \qquad \forall i \in N, \xi_c, \in \Xi. \tag{6.29}$$

$$X^f_{iiq}, X^s_{ijq}(\xi_c) \in \{0, 1, \ldots, \kappa_q\}, \qquad \forall (i, j) \in A, \forall q \in Q, \forall \xi_c \in \Xi; \tag{6.30}$$

$$S^f_i, S^s_i(\xi_c) \in [0, T_{max}], \qquad \forall i \in N, \forall \xi_c \in \Xi. \tag{6.31}$$

The objective function (6.1) maximises the expected value of serviced assets. Constraints (6.2) require that all vehicles leaving the starting depot must reach the final depot. Constraints (6.3) enforce the conservation of flow at each asset. Constraints (6.4 and 6.5) indicate the staging location. If a vehicle depart from a node and cannot arrive to the destination by the staging time, the departing node will be

denoted as staging location. This assists in assigning the right values to $S_j^s(\xi_c)$ in constraints (6.16 and 6.17) in order to start trips from staging locations in various scenarios at stage two. Constraints (6.6 and 6.7) impose the staging time. If an arc is traversed in stage f, $z_{ijq}^s(\xi_c)$ for all scenarios must be zero. On the other hand, if z_{ijq}^f is zero, one of the $z_{ijq}^s(\xi_c)$ could be one, but not necessarily. Constraints (6.8) indicate that the number of vehicles departing from a depot may not exceed the number of vehicles stationed at the depot. Constraints (6.9 and 6.10) guarantee an asset is serviced only if its service requirements are fulfilled by the right combination of incoming vehicles. Constraints (6.11) ensure each asset will be visited at most once in both stages. Constraints (6.12 and 6.13) make sure that vehicles travelling through an arc never exceed the number available κ_q. Constraints (6.14 and 6.15) ensure that an asset may only be visited if the service requirements of the previous location has been satisfied and there is enough time to reach the next asset. Equations (6.16 and 6.17) guarantee that proper values are assigned to $S_j^s(\xi_c)$ through a transition from one stage to another. Constraints (6.18 and 6.19) identify nodes that are visited in shared time intervals, where decisions in one scenario must match with those in other scenarios, to get the same values of decision variables. Shared time intervals refer to the time periods between two wind changes where any decision made during this time would must hold for the remaining scenarios. Equations (6.20 and 6.23) enforce equality of shared decision variables under different scenarios for the nodes identified by constraints (6.18 and 6.19). Terms (6.24) and (6.25) ensure that the time window constraints are not violated. Constraints (6.26-6.31) impose non-negative, binary and integer restrictions on the variables.

6.1.3 An illustrative example

The time windows during which an asset must be serviced will depend on the progress of a wildfire as it spreads across the landscape. Fire-spread models used by IMTs can generate these time windows. Reviews of wildfire models can be found in Scott and Burgan (2005); Sullivan (2009); Johnston et al. (2008); Bakhshaii and Johnson (2019) while Petrasova et al. (2018) is an example of recent developments.

For our illustrative hypothetical landscape we note that with a constant wind, fire-fronts mostly progress in an elliptical shape (Anderson et al. (1982)). Thus we use the general equation of an ellipse to generate time windows with similar characteristics to those occurring in real landscapes.

The problem we want to investigate is the frequent situation where it is known that a wind change will occur but there is uncertainty over its exact timing. As shown in Figure 6.1, the timing of the wind change is represented by two scenarios, for the purpose of simplicity. In the first scenario the wind change occurs at the staging time, (TO_1), and an hour later in scenario 2. The change in wind direction and its timing has a significant impact on the fire front. This in turn affects which assets will be in the path of the wildfire. There is a significant difference in the number of assets at risk in each scenario while the hazard for some assets is common to both scenarios although with different probabilities associated with each scenario.

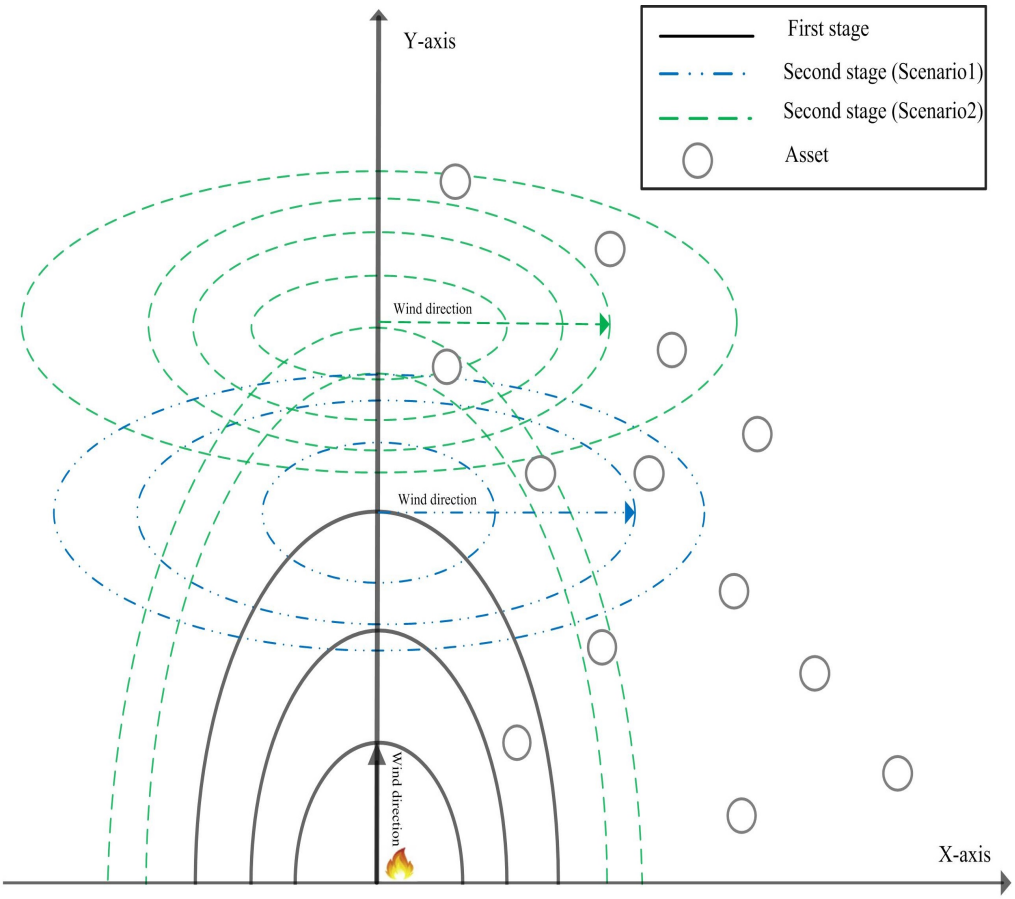

Figure 6.1: Fire front progressing through a landscape. The initial north wind (Y-direction) is expected to change to a westerly but there is uncertainty as to the timing of the change. Two scenarios are shown representing the times at which the wind change may occur. In each scenario a new set of assets are at risk.

A simple demonstration of the two-stage optimisation model for the APP problem is sketched in Figure 6.2. The service requirements of each asset were chosen to highlight some of the key features of the APP problem. Due to resource limitations and time windows not all assets can be serviced. The optimally deployed vehicles maximise the expected value of assets serviced. The assets actually serviced also depend on which scenario is realised.

As shown in Figure 6.2, twenty community assets are at risk. Each vehicle has particular capabilities that need to be matched with the service requirements of each asset. For example, an asset in stage 1 which has requirements of $< 2, 1, 1 >$ must be serviced by two type 1 vehicles and one of the other two vehicle types. The service must start cooperatively and simultaneously within the time window $[o_i^f, c_i^f]$ for a duration of a_i. In the Figure 6.2, the probability of occurrence for each scenario is known and routing in the first stage is planned to gain the highest benefit from the expected value of the objective function in the second stage.

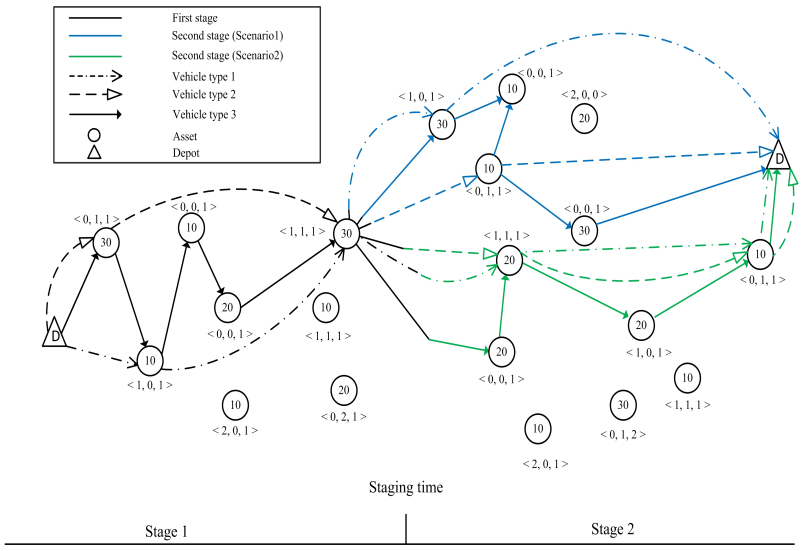

Figure 6.2: Illustrative APP problem with two wind change scenarios. The area impacted in each scenario is shown in different colours. Asset values are shown within the circles with the requirements shown below them. Vehicle types 1, 2 and 3 have capability vectors of $< 1, 0, 0 >$, $< 0, 1, 0 >$ and $< 0, 0, 1 >$. To the extent possible they must be assigned to the assets so that their capabilities match the asset's requirements. The optimal assignment of the three vehicles is shown.

6.2 Dynamic rerouting approach

The dynamic rerouting of vehicles reallocate resources in the event of a change in conditions (e.g. road closure). The initial deployment of vehicles is based on a deterministic MILP (see 4) assuming no changes will occur. If any change in conditions or disruptions occur during operations, new parameters (e.g. time windows) are updated accordingly. Then, the problem is solved to optimality from where the disruption took place. The pseudo-code of the dynamic rerouting asset protection problem is described in Algorithm 6.1.

Algorithm 6.1. returns the expected value of the protected assets using the dynamic rerouting approach. The algorithm defines a scenario that has the highest probability (lines 2-8). Then, constructs a set of nodes including the first stage assets and those belonging to the scenario with the highest probability of occurring. After solving the deterministic MILP, the model determines the set of protected assets. By solving the initial MILP at lines 9 and 10 the algorithm also determines locations where the resources would be at when the disruptions occur. Algorithm 6.1. solves the MILP model again for assets belonging to other scenarios, while resources start their routes from where they were at the staging time. In lines (13-19) the algorithm calculates the expected value of the assets protected by multiplying the total asset value at each scenario by the probability of each scenario. The same benchmark instances are used and solved according to the above procedure. To address this NP-hard problem and solve realistically large instances, we developed an efficient metaheuristic approach to solve the dynamic rerouting APP in operational computation time.

Algorithm 6.1: Pseudocode for the dynamic rerouting approach

Input:
staging time (TO_1), set of at risk assets in stage 1 (N^f), set of at risk assets in stage 2 under scenario # 1 (N_1^s), set of at risk assets in stage 2 under scenario #2 (N_2^s), probability of scenario #1 ($P(1)$), probability of scenario #2 ($P(2)$), collected rewards from assets in set N^f ($\nu^f = 0$), collected rewards from assets in set N_1^s ($\nu_1^s = 0$), collected rewards from assets in set N_2^s ($\nu_2^s = 0$), set of all assets (N)

Output: Expected value of the collected rewards ($E(\nu)$).

Function *Dynamic Rerouting*
  if $(P(1) > P(2))$ **then**
    $N_a \leftarrow N^f \cup N_1^s$
    $N_b \leftarrow N_2^s$
  else
    if $((P(2) > P(1)))$ **then**
      $N_a \leftarrow N^f \cup N_2^s$
      $N_b \leftarrow N_1^s$
  $\Upsilon \leftarrow$Solve the deterministic APP for $assets \in N_a$ and determine the visited assets
  $D^{st} \leftarrow$ Define the set of locations where resources are at time TO_1
  $\Upsilon \leftarrow$ Solve the deterministic asset protection model for $assets \in N_b$ starting from D^{st}
  $E(\nu) \leftarrow 0$
  forall $(i \in N)$ **do**
    if $(i \in \Upsilon \ \& \ i \in N^f)$ **then**
      $E(\nu) \leftarrow E(\nu) + \nu_i$
    if $(i \in \Upsilon \ \& \ i \in N_1^s)$ **then**
      $E(\nu) \leftarrow E(\nu) + P(1) \times \nu_i$
    if $(i \in \Upsilon \ \& \ i \in N_2^s)$ **then**
      $E(\nu) \leftarrow E(\nu) + P(2) \times \nu_i$
  return $E(\nu)$

6.3 Adaptive large neighbourhood search

The asset protection problem is NP-hard and cannot be solved analytically. The ALNS algorithm as discussed in chapters 4 and 5 is implemented to solve the dynamic rerouting APP in operational times.

6.4 Experiments and empirical results

We carried out extensive computational experiments to test our model. Initially, we present a case study using the context of the February 2009, Black Saturday bushfires in Australia. Then, we perform further tests on a large set of generated benchmark instances using realistic parameters [1] and compare them against a dynamic rerouting approach solved by the both CPLEX and ALNS. All computational experiments are implemented on a desktop computer equipped with Intel Core i5 (3.2GHz) and 8.0 GB of RAM, where MILP models are solved by the commercial solver, CPLEX 12.8, coded in Python 3.6; and the algorithm coded in C++ using the Microsoft Visual Studio 2017. Computational times are measured in elapsed time with a time limit of thirty minutes for the CPLEX solver.

6.4.1 Case study - Murrindindi Mill fire Black Saturday

The shire of Murrindindi is $3,889\ km^2$ in extent with a population of 13,732 (2016 census) located in the north-eastern part of Victoria, Australia. About 46% of the total land area of the municipality is forest ($1788\ km^2$), and a large proportion of this land is mountainous (Shahparvari et al. (2016)). On 7 February 2009, a series of bushfires known as Black Saturday raged through the shire (see Figure 6.3). The fire swept the 50 km distance between Saw Mill in Wilhelmina Falls Road and Narbethong in about 90 minutes. Following a wind change the long narrow fire-front become a wide fire-front that burned through a number of townships with tragic consequences (Cruz et al. (2012)). Five of the bushfires on Black Saturday claimed people's lives. The second highest number of deaths resulted from the Murrindidi bushfires (40 people), which reveals the importance of the area under study (Whittaker et al. (2009)).

A set of 25 assets are identified in the area as being impacted (see Appendix one). The Yea country fire authority is defined as the starting point of the operations

[1] All the benchmark instances are available at www.sites.google.com/site/imanrzbh/datasets

(depot). Distance matrices were obtained using the Google Maps API (Google API (2018)). Time windows are set approximately based on the Victorian Bushfires Royal Commission report (see Teague et al. (2010)). The number of vehicles are considered proportional to the problem size in order to cover a significant amount of at risk assets $\{\kappa_1 = 5, \kappa_2 = 3, \kappa_3 = 2\}$. Vehicle velocity ($V = 60\,km.h^{-1}$) and probability of scenarios ($P(1) = 25\%$, $P(2) = 35\%$, $P(3) = 40\%$) are other parameters involved in the problem. The wind change can occur at $TO_1 = 4:00\,pm$ (scenario 1), $TO_2 = 5:30\,pm$ (scenario 2) and $TO_3 = 6:00\,pm$(scenario 3). As a consequence of the wind change a new set of assets will be at risk. Those assets common to both sets may have new time windows for servicing. As illustrated in Figure 6.3, two assets are at risk regardless of the wind change scenarios (A6 and A10), while the risk to other assets depends on the timing of the wind change. As shown in Figure 6.3, the value of the at-risk assets in scenario 2 of stage two is 9% and 12% more than the impacted assets in scenario 1 and 3 of stage two, respectively. This illustrates the impact of wind change and how significant the difference between scenarios may be. However, when we look at the expected value of the at-risk assets, scenario 3 of stage two has the highest value due to the probability of the scenario.

We use CPLEX to solve the case study discussed above using the two-stage stochastic programming model formulated in Section 6.1.2. Results of this case study are reported in Table 6.1. The optimal solution shows that all assets under threat during the first stage can be serviced. A total of ten vehicles finished their protective

Table 6.1: Summary of results for the case study

Stage (scenario)	Number of at risk assets	Probability of occurrence	At risk assets	Value of the at risk assets	Serviced assets	Value of the serviced assets
First stage	5	100%	2, 4, 5, 17, 18	203.00	2, 4, 18	132.00
Second stage (1)	10	25%	3, 6, 9, 10, 15, 19, 21, 22, 23, 24	378.00	3, 6, 9, 10, 15, 22 23, 24	322.00
Second stage (2)	14	35%	1, 3, 6, 7, 8, 10, 11, 12, 14, 15, 20, 21, 23, 25	410.00	3, 6, 7, 10, 11, 12, 15, 20, 21, 23	325.00
Second stage (3)	12	40%	1, 6, 7, 8, 10, 11, 12, 13, 14, 16, 20, 25	366.00	6, 7, 8, 10, 11, 12, 13, 16, 20	299.00

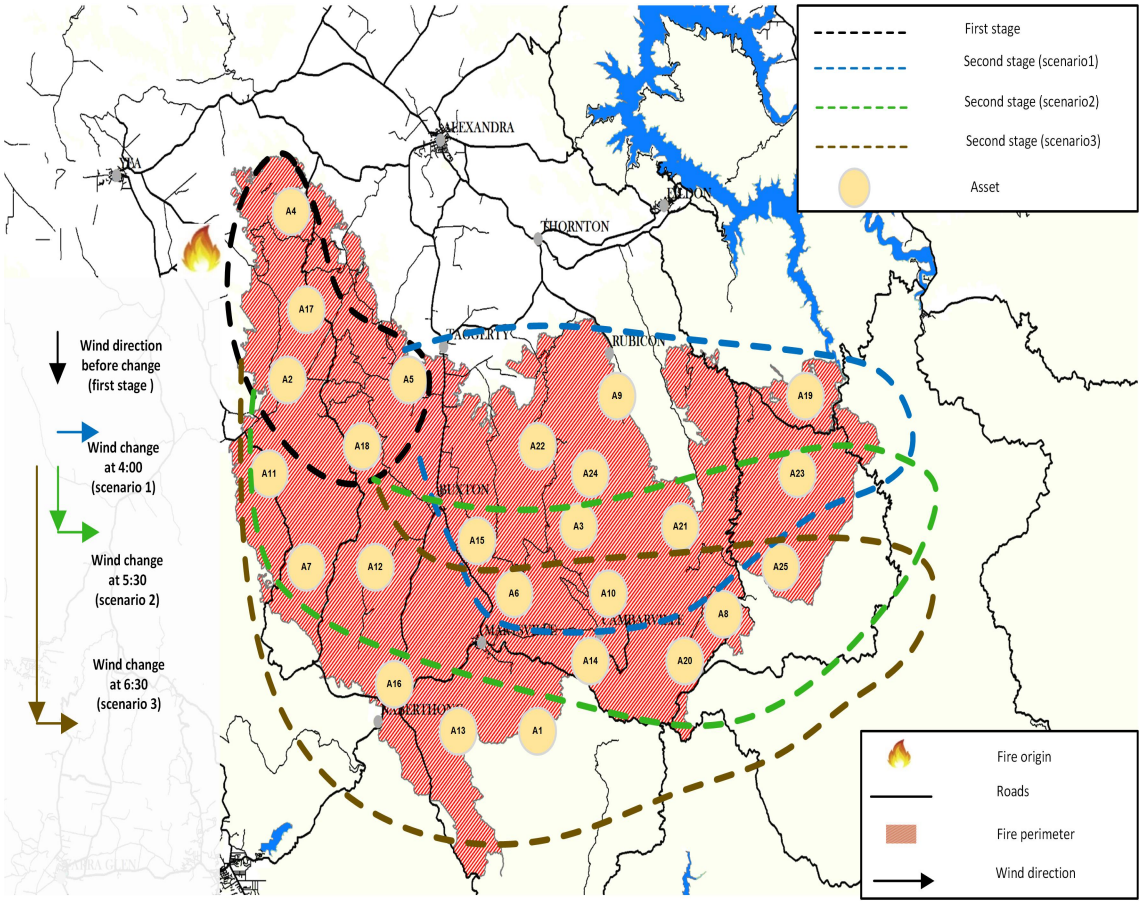

Figure 6.3: Case study region - Murrindindi Shire, Victoria, Australia.

activities at assets $A2$, $A4$ and $A18$ before immediately heading to the at-risk assets in the next stage. It can be seen in Figure 6.3 that some assets, $A11$, $A7$ and $A12$ are impacted after $TO_1 = 4:00\,pm$ but before the time of wind change in scenario 2, $TO_2 = 5:30\,pm$. For such assets the associated decision variables are unchanged in scenarios 2 and 3. Overall, the optimal solution provides a strategy for servicing assets to the value of 65%, 85%, 79% and 82% of the total asset value in stage 1, and scenarios 1, 2 and 3, respectively.

6.4.2 Test instances

To further evaluate the two-stage stochastic model, the dynamic rerouting approach and the proposed ALNS algorithm, a set of benchmark instances are generated by randomly distributing assets over an 80×80 grid. Five sets of problems with asset numbers from 20 to 200 are solved by ALNS while CPLEX is used, where applicable,

for the dynamic rerouting and the two-stage stochastic models. Every instance is solved for two cases representing different sets of vehicles as given in Table 6.4. We use the elliptical fire spread to simulate a realistic scenario and set time windows for each asset. Similar to the events of Black Saturday we consider a wind change. However, two scenarios are possible for the timing of the wind change. In scenario 1 the wind change occurs at the staging time whereas it occurs two hours later in scenario 2. The realisation of which scenario will occur will only be known at the staging time.

For representing a realistic situation where the impact of the wind change is significant, a different set of assets are impacted with each scenario. Therefore, there are four types of assets: (1) assets impacted at the first stage; (2) assets in either scenario 1 or scenario 2 but not both at the second stage; (3) assets that are affected in both scenarios, (4) and assets that are not at risk at all in either scenario. Fire velocities are set in a manner to simulate escaped wildfires. Travel time is calculated with vehicle velocity set at a conservative but realistic (30 $km.h^{-1}$). To investigate the impact of vehicle numbers on the percentage of the assets protected, we performed experiments with two different sets of vehicles $Set_1 = \{\kappa_1 = 3,\ \kappa_2 = 2,\ \kappa_3 = 2\}$ and $Set_2 = \{\kappa_1 = 4,\ \kappa_2 = 3,\ \kappa_3 = 3\}$ for small instances. The key parameters for generating benchmark instances are mostly inspired by real life scenarios, and are listed in Table 6.3.

Table 6.2: Parameters used in the benchmark instances

Parameter	**Explanation**	**Value**
v_{x0}	Fire velocity along x in stage one.	14
v_{y0}	Fire velocity along y in stage one.	16
v_{x1}	Fire velocity along x in stage two, scenario one.	19
v_{y1}	Fire velocity along y in stage two, scenario one.	17
v_{x2}	Fire velocity along x in stage two, scenario two.	21
v_{y2}	Fire velocity along y in stage two, scenario two.	19
$Delay$	Delay between staging time and change of wind, in scenario two.	2
TW_1	Length of time window for assets impacted in stage one.	2
TW_2	Length of time window for assets impacted in stage two.	2
a	Service duration time.	1
$P(1)$	Probability of scenario 1.	0.6
$P(2)$	Probability of scenario 2.	0.4
V	Travel speed of vehicles.	30
TO_1	Staging time.	4.5

To evaluate the performance of different methods, we solve each benchmark in-

stance using dynamic rerouting model, two-stage stochastic model and the ALNS algorithm, where they are applicable. Please note that we initially tune the ALNS parameters following the same mythology as literature (Roozbeh et al., 2018). We perform five runs on tuning instances for ten different values for parameters and chose the one with least deviation from the best solution. The value of the parameters are determined in Table 6.3.

Table 6.3: Parameters used in the proposed algorithm

Description	**Parameter**	**Value**
Parameters for MB heuristic	$(\lambda, \mu, \vartheta)$	(2,1,3)
Improving solution score	σ_2	12
Number of iterations	n_{total}	3000
Number of iterations over each segment	n_{seg}	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
Noise parameter	α_{noise}	0.6
WDR determinism factor	κ	8
Shaw determinism factor	η	12

6.4.3 Experiments on small-sized instances

Both the dynamic rerouting and the two-stage stochastic programming approaches are implemented to solve the benchmark instances by means of CPLEX. The ALNS algorithm is used to solve the same instances for validation purposes. Note that we eliminate infeasible arcs due to the time window constraints, protection requirements and the impact time in a preprocessing step to sufficiently reduce the problem size for all approaches. The result are presented in Table 6.4.

The results for the two-stage stochastic programming model (TSSP), the dynamic rerouting approach (DR) and the adaptive large neighbourhood algorithm (ALNS) are reported as percentages of the total value of assets impacted. The objective value ("asset value protected") is the summation of the actual values of protected assets at stage one and the expected values of them at stage two.

Table 6.4: A summary of results for 50-55-60 assets. Vehicle numbers are defined at two levels: $Set1 = \{\kappa_1 = 3, \kappa_2 = 2, \kappa_3 = 2\}$ and $Set2 = \{\kappa_1 = 4, \kappa_2 = 3, \kappa_3 = 3\}$. Results are reported as percentage of the assets' values protected. Computation times are reported in seconds(sec). Under "Gap(%)" positive numbers imply better performance of the first method. Asterisk is used to define sets that couldn't be solved within the time limit by CPLEX.

#Vehicles	#Assets	TSSP		DR		ALNS			Gap(%)		
		Asset value protected(%)	Time (sec)	Asset value protected(%)	Time (sec)	Asset value protected(%)		Time (sec)	TSSP vs DR	ALNS vs DR	
						Avg	Best			Avg	Best
Set 1	20	64.49	183.08	62.33	24.54	61.94	62.24	5.49	3.36	-0.65	-0.17
	25	58.47	1637.03	56.69	51.59	56.11	56.50	8.05	3.25	-0.88	-0.19
	30*	52.10	1845.94	49.68	810.74	49.19	49.89	11.84	5.00	-0.96	0.43
Set 2	20	77.37	769.89	76.47	140.49	75.49	76.10	5.83	1.17	-1.24	-0.42
	25*	73.54	1578.42	71.29	674.34	69.83	70.69	9.45	3.09	-1.92	-0.74
	30*	67.27	1850.96	64.12	1831.34	64.21	64.38	14.03	4.77	-0.67	0.93

Table 6.4 shows that the TSSP takes advantage of integrating all scenarios and maximises the total expected value of the serviced assets. Therefore, the two-stage stochastic programming performs better for small instances at a cost of higher computation time, as more decision variables are involved. The TSSP results are 3.44% better than those by the DR. As is to be expected for both DR and TSSP methods, it is seen in Table 6.4 that computational time increases with increasing assets. Less obvious is that more time is required to solve instances with a larger number of vehicles (set 2). Dispatching more vehicles increases the complexity of the IMTs task but more assets are serviced. Both DR and TSSP failed to solve all instances to optimality for 30 nodes in set 1, and this drops to 25 nodes for set 2, where more vehicles are operating.

On the other hand, the ALNS has a consistent performance both in terms of speed and accuracy. The average computation time for ALNS, in Table 6.4, is 9.1 seconds compare to 588.84 and 1310.9 for DR and TSSP, respectively. This makes the ALNS suitable for operational purposes and it gives the IMTs a close to optimal solution. The average optimality gap of -1.05% and -0.02% in the last two columns of the table illustrate the efficacy of the ALNS. The algorithm generates better solutions for instances with 30 nodes where CPLEX cannot achieve optimal solutions within the time limit.

Overall, the results indicate that the two-stage stochastic approach can improve the quality of the solution but that it comes at a computational cost. For less

than 30 assets this computational cost is not a concern as solution times are still satisfactory for operational purposes. For cases involving either a larger number of assets or resources the ALNS algorithm would need to be employed despite the reduction in solution quality.

6.4.4 Experiments on large-sized instances

Both from theoretical considerations and from Table 6.4 we expect the TSSP to generate optimal results when prior knowledge of a wind change and its timing is available. Thus, we use the TSSP to test the the quality of the ALNS for larger problems with 100 and 200 nodes. We run CPLEX for twelve hours and compare the best integer and the best upper bound with results by the ALNS. The twelve hours time limit for the CPLEX is not relevant for operational purposes but is set merely for evaluation of the ALNS solutions.

The results for the large-sized instances are presented in Table 6.5. Results are illustrated as percentage of asset values; and number of vehicles are set proportional to the problem size. In Table 6.5 the TSSP can only covers about 28.15% , on

Table 6.5: A summary of results for 100 and 200 nodes. Vehicle numbers are defined in four categories. For 100 nodes: Set1 = (V 1 = 6 , V 2 = 5 , V 3 = 4) and Set2 = (V 1 = 7 , V 2 = 6 , V 3 = 5). For 200 nodes: Set1 = (V 1 = 9 , V 2 = 8 , V 3 = 7) and Set2 = (V 1 = 12 , V 2 = 11 , V 3 = 10)

#Vehicles	#Assets	TSSP		ALNS			Gap(%)	
		Asset value protected(%)		Asset value protected(%)		Time (sec)	ALNS vs DR	
		CPLEX LB	CPLEX UB	Avg	Best		Avg	Best
Set 1	100	38.04	99.90	42.28	43.24	225.75	12.34	14.90
	200	8.86	100	37.05	37.85	609.46	680.65	696.43
Set 2	100	44.98	100	48.03	49.25	334.42	10.07	12.85
	200	20.74	100	46.71	47.39	764.08	795.66	809.44

average, after 43200 seconds. On the other hand, the ALNS collects 44.43% of the total value of assets in 483 seconds. The average gap between ALNS and TSSP gets larger when we increase the problem size by 100 nodes. There is a small difference between the average and the best result by the ALNS which emphasises the robust

performance of the algorithm through each run. Table 6.5 shows that the ALNS can perform better in practice when sudden changes take place as it requires less computational resources. However, a better solution by ALNS does not assure its quality as CPLEX could not achieve reliable upper bounds with which to compare our results. Overall, the ALNS can generate practical solutions within times suitable for operational purposes.

6.5 Summary and discussion

In this chapter a two-stage stochastic programming approach is developed to handle the unusual feature of uncertainty in the timing of changes in conditions. In this study the changed conditions refer to wind direction and velocity. These changes determine new time windows during which assets must be serviced and hence the optimal deployment schedule and routing of vehicles. To the best of my knowledge this is the first two-stage stochastic programming problem dealing with uncertainties in the timing of changes.

The two-stage stochastic programming approach (TSSP) requires forecasts of changes. We also investigated a dynamic rerouting approach to the problem. In an operational setting the deployment model would simply be solved again once the wind actually changed or the forecast uncertainty had been removed. Solving the dynamic rerouting problem (DR) with exact methods using a commercial solver, as expected, yielded results that were not as good as the TSSP but nevertheless were at worst within five per cent of that achieved by the theoretically superior method. The exact method for both approaches, however, was not able to produce results for larger problems in times useful for practical purposes. Thus, we investigated the use of an adaptive large neighbourhood search algorithm (ALNS) to solve the rerouting problem in a more time efficient manner.

In addition to a case study, the proposed approaches were tested with extensive computational experiments. To evaluate the efficacy of our solution scheme, results were compared to the dynamic rerouting approach and the two-stage stochastic approach.

As expected the two-stage stochastic program produced better solutions than the rerouting approach. This difference became more pronounced as the complexity of the problem increased with more assets and vehicles. However, the two-stage stochastic program was unable to solve realistic size problems in operational time. On the other hand, the performance of the ALNS in terms of computational time was superior to the stochastic programming approach and generates high quality solutions.

Fire authorities can utilise the proposed models in their decision making process to incorporate any uncertainties in the problem condition due to the change of fire direction, demand of suppression and protection operations, change of time windows and/or other problem attributes. Also, along with weather change, other disruptions with uncertain time of occurrence can be considered such as network accessibility, and change in travel time. These can be incorporated into the model in a similar way.

Chapter 7

Conclusion

Wildfires are often hazardous to key assets whose loss can disrupt community life for months and be costly to replace. During an uncontrollable wildfire IMT's deploy response vehicles to mitigate the risk of losing crucial assets. Planning, coordinating and managing protective activities using limited resources (i.e., personnel and equipment) in an optimal way is critically important. The optimal deployment of resources during wildfires, known as the asset protection problem, is NP-hard. The literature review conducted in chapter 2 reveals that there is no solution scheme to solve either the APP or the COPTW in times suitable for operational purposes. In the APP any strategy would not be optimal without utilising all available knowledge including the forecast of a wind change even if there is some uncertainty about the timing of that change. There is very little in the literature dealing with uncertainty in the timing of changes in the problem conditions.

This thesis introduces a heuristic solution approach to solve large size COPTW for the first time. The solution approach demonstrated in chapter 3 can efficiently handle synchronisation of team members within predefined time windows. The performance of the proposed solution approach was evaluated by comparing against optimal solutions for small instances. Furthermore, the results of the large instances are compared to another heuristic method which proved the superior performance of the proposed approach.

In chapter 4 a metaheuristic approach was designed to deal with existing complexities of the APP. New problem specific insertion and removal algorithms were developed to implement, dynamically and adaptively, in the body of an adaptive large neighbourhood algorithm. The efficacy of the solution approach was evaluated through extensive computational experiments. Proposing a new approach to evaluate the feasibility of insertion at any point in a specific time favoured the ALNS algorithm both in terms of solution quality and computation effort. The algorithm developed was able to solve realistic sized asset protection problems for the first time in computational times suitable for implementation in practical problems.

In chapter 5 additional algorithms were formulated to improve the ALNS performance introduced in chapter 4. The performance of every insertion and removal heuristic was evaluated and the least efficient algorithms excluded. Doing so provides more time for the efficient heuristics to improve the solution. The enhanced ALNS algorithm achieved better solutions than previously published results which demonstrated the efficacy of the solution approach. Once the performance of the algorithm was validated, the ALNS was used for solving the COPTW and the results evaluated and released as benchmarks for future research.

In chapter 6 a two-stage stochastic programming approach for the APP was introduced. The model incorporates uncertainties in the timing of changes in the problem condition. Such changes could be the result of factors such as changes in wind direction, protection requirements or network accessibility. This is an important problem that arises frequently with wildfires in south-eastern Australia when a change in wind direction is predicted but the timing of the change is uncertain. The change in the wind direction impacts on the time windows during which protection activities need to be accomplished. The performance of the two-stage stochastic programming model, the dynamic rerouting model and the ALNS algorithm for solving dynamic rerouting were evaluated through extensive computational experiments. A case-study of the Black Saturday bushfires was presented and the impact of wind change was investigated.

Although the discussion focused on asset protection and change in wind direction,

other logistic operations can be accommodated by the proposed modelling approach and solution schemes. An example of which could be suppression operations during bushfires. A common situation that arises in wildfire suppression operations is dealing with precedence and synchronisation constraints, similar to the APP. This is due to different tasks and resources having to be allocated so that aerial and ground vehicles can do their job effectively. When wildfires take place, initial resources, e.g. helicopters, are deployed to achieve the earliest possible suppression of the fire. After the initial attack, depending on fire suppression requirements and available resources, other response vehicles such as tankers and pumpers will be deployed to work either cooperatively or independently.

Another application is the home care services. In such services, some operations may require more than one staff member. For example, where heavy lifting and specialist medical expertise are required (Bredström and Rönnqvist (2008)). A closely related problem is a home cleaning service where some homes may require one or more services such as the following: basic cleaning, window cleaning, and washing. Access to each house is within a given time-window.

7.1 Thesis contributions

As demonstrated the proposed solution algorithms and modelling approaches can be used for operational purposes. They can also be implemented for strategic planning decisions to identify home-basing of resources and fleet composition. Solving large scale asset protection problems are now possible by using the solution approaches proposed in this thesis. Strategically, fire authorities can use the proposed models to determine optimal deployment plans for various simulated wildfire scenarios. In this way they can gain better insight about the proportion of community assets that can be protected. Identifying assets that would go unprotected with different combinations of resources can assist decision makers to determine what additional resources are required to cover a majority of important community assets.

Contribution 1 *Providing algorithmic approaches to solve the COPTW*

This is the first time that a large scale COPTW with real-life applications, requiring quick solutions, is solved. A merit-based heuristic was developed and the performance of the algorithm was evaluated. The proposed solution approach can be implemented on regular desktop computers by using free-of-charge existing software technology.

Contribution 2 *Developing an adaptive large neighbourhood algorithm to solve the APP in operational times* Decision-makers are faced with complex problems under severe time-pressure.One such problem is the Asset Protection Problem. For the first time this problem was solved within times suitable for operational purposes. The newly designed heuristics achieved high quality solutions. By implementing a new approach, the feasibility of insertions were evaluated in a time efficient manner that improved the efficacy of the adaptive large neighbourhood algorithm.

Contribution 3 *The development of a comprehensive metaheuristic solution approach for the COPTW.*

Following the success of the ALNS approach introduced in chapter 4 a thorough exploration of additional heuristics was made with a view to making further performance improvements. Inefficient heuristics were identified and excluded and new heuristics designed as explained in chapter 5. The resulting solution procedure produced reliable solutions both in terms of solution quality and computational times in solving the COPTW.

Contribution 4 *A two-stage stochastic programming model to incorporate existing uncertainties in the APP*

A two-stage stochastic programming approach is presented to handle uncertainties in the asset protection during bushfires. This is one of the first studies incorporating uncertainty in the timing of changes. The proposed model considers changes in wind direction that impacts on the availability of networks, protection requirements, and time windows.

7.2 Suggestions for future works

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 can accommodate further constraints to deal with such problems.

Fire suppression operations are crucial at the early stages of wildfires. The models discussed and the solutions approaches presented in this thesis can be extended and implemented for such operations. Some problem attributes should be taken into account for fire suppression operations, such as the travel speed of the different aerial and ground vehicles, different service times at various points and incidents of vehicles breaking down. The characteristics of the suppression operations can be integrated into the models and algorithms formulated in this thesis.

Developing a decision support tool for both defensive tasks and suppression operations by implementing the models and solution approaches developed is a natural next step. Doing so, would facilitate constructive feedback from fire authorities. This is turn would have the benefit of identifying possible modifications to increase the practical utility of the models under operational conditions.

Bibliography

Rahim A Abbaspour and Farhad Samadzadegan. Time-dependent personal tour planning and scheduling in metropolises. *Expert Systems with Applications*, 38 (10):12439–12452, 2011.

Sohaib Afifi, Duc-Cuong Dang, and Aziz Moukrim. Heuristic solutions for the vehicle routing problem with time windows and synchronized visits. *Optimization Letters*, 10(3):511–525, 2016.

Alfred V. Aho, Michael R Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

Deniz Aksen, Onur Kaya, F Sibel Salman, and Özge Tüncel. An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem. *European Journal of Operational Research*, 239(2):413–426, 2014.

Kimberly Amadeo. California wildfires. `https://www.thebalance.com/wildfires-economic-impact-4160764`, January 2019.

DH Anderson, EA Catchpole, NJ De Mestre, and T Parkes. Modelling the spread of grass fires. *The ANZIAM Journal*, 23(4):451–466, 1982.

C. Archetti, M.G. Speranza, and D. Vigo. Vehicle routing problems with profit. vehicle routing: Problems, methods, and applications, second edition. *MOS-SIAM Series on Optimization: 18*, pages 273–298, 2014.

Peter M Attiwill and Mark A Adams. Mega-fires, inquiries and politics in the eucalypt forests of victoria, south-eastern australia. *Forest Ecology and Management*, 294:45–53, 2013.

Nabila Azi, Michel Gendreau, and Jean-Yves Potvin. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, 41:167–173, 2014.

Hossein Badri, SMT Fatemi Ghomi, and Taha-Hossein Hejazi. A two-stage stochastic programming approach for value-based closed-loop supply chain network design. *Transportation Research Part E: Logistics and Transportation Review*, 105: 1–17, 2017.

Atoossa Bakhshaii and Edward A Johnson. A review of a new generation of wildfire–atmosphere modeling. *Canadian Journal of Forest Research*, 49(6):565–574, 2019.

MAF Belo-Filho, P Amorim, and B Almada-Lobo. 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, 2015.

Souleyman Benkraouda, Benabdellah Yagoubi, Mustapha Rebhi, and Ahmed Bouziane. Belonging probability inverse image approach for forest fire detection. *African Journal of Ecology*, 52(3):363–369, 2014.

Nicola Bianchessi, Renata Mansini, and M Grazia Speranza. A branch-and-cut algorithm for the team orienteering problem. *International Transactions in Operational Research*, 25(2):627–635, 2018.

Joan Borràs, Antonio Moreno, and Aida Valls. Intelligent tourism recommender systems: A survey. *Expert Systems with Applications*, 41(16):7370–7389, 2014.

Hermann Bouly, Duc-Cuong Dang, and Aziz Moukrim. A memetic algorithm for the team orienteering problem. *4or*, 8(1):49–70, 2010.

David Bredström and Mikael Rönnqvist. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European journal of operational research*, 191(1):19–31, 2008.

David Canca, Alicia De-Los-Santos, Gilbert Laporte, and Juan A Mesa. An adaptive neighborhood search metaheuristic for the integrated railway rapid transit network

design and line planning problem. *Computers & Operations Research*, 78:1–14, 2017.

Mohamed Cissé, Semih Yalçındağ, Yannick Kergosien, Evren Şahin, Christophe Lenté, and Andrea Matta. Or problems related to home health care: A review of relevant routing and scheduling problems. *Operations Research for Health Care*, 13:1–22, 2017.

GU Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

Victorian Bushfires Royal Commission, Bernard Teague, Ronald McLeod, Susan Mary Pascoe, et al. *2009 Victorian Bushfires Royal Commission: Interim Report 2: Priorities for Building in Bushfire Prone Areas*. Government Printer, South Africa, 2009.

Jean-François Cordeau, Gilbert Laporte, Martin WP Savelsbergh, and Daniele Vigo. Vehicle routing. *Handbooks in operations research and management science*, 14: 367–428, 2007.

MG Cruz, AL Sullivan, JS Gould, NC Sims, AJ Bannister, JJ Hollis, and RJ Hurley. Anatomy of a catastrophic wildfire: the black saturday kilmore east fire in victoria, australia. *Forest Ecology and Management*, 284:269–285, 2012.

Miguel G Cruz, Martin E Alexander, Andrew L Sullivan, James S Gould, and Musa Kilinc. Assessing improvements in models used to operationally predict wildland fire rate of spread. *Environmental Modelling & Software*, 105:54–63, 2018.

Duc-Cuong Dang, Racha El-Hajj, and Aziz Moukrim. A branch-and-cut algorithm for solving the team orienteering problem. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 332–339. Springer, 2013a.

Duc-Cuong Dang, Rym Nesrine Guibadj, and Aziz Moukrim. An effective pso-inspired algorithm for the team orienteering problem. *European Journal of Operational Research*, 229(2):332–344, 2013b.

George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

Ivanoe De Falco, Umberto Scafuri, and Ernesto Tarantino. A multiobjective evolutionary algorithm for personalized tours in street networks. In *European Conference on the Applications of Evolutionary Computation*, pages 115–127. Springer, 2015.

Emrah Demir, Tolga Bektaş, and Gilbert Laporte. An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research*, 223(2):346–359, 2012.

Tamer Doyuran and Bülent Çatay. A robust enhancement to the clarke–wright savings algorithm. *Journal of the Operational Research Society*, 62(1):223–231, 2011.

Michael Drexl. Synchronization in vehicle routing—a survey of vrps with multiple synchronization constraints. *Transportation Science*, 46(3):297–316, 2012.

Daniel Duque, Leonardo Lozano, and Andrés L Medaglia. Solving the orienteering problem with time windows via the pulse framework. *Computers & Operations Research*, 54:168–176, 2015.

Uğur Emeç, Bülent Çatay, and Burcin Bozkaya. An adaptive large neighborhood search for an e-grocery delivery routing problem. *Computers & Operations Research*, 69:109–125, 2016.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

Mauro Falasca and Christopher W Zobel. A two-stage procurement model for humanitarian relief supply chains. *Journal of Humanitarian Logistics and Supply Chain Management*, 1(2):151–169, 2011.

Dominique Feillet, Pierre Dejax, and Michel Gendreau. Traveling salesman problems with profits. *Transportation science*, 39(2):188–205, 2005.

Luca Maria Gambardella, Roberto Montemanni, and Dennis Weyland. Coupling ant colony systems with strong local searches. *European Journal of Operational Research*, 220(3):831–843, 2012.

G Ganewatta and J Handmer. The cost-effectiveness of aerial firefighting in australia. *Centre for Risk and Community Safety, RMIT University, Bushfire CRC*, pages 978–990, 2009.

Margaretha Gansterer and Richard F Hartl. Collaborative vehicle routing: a survey. *European Journal of Operational Research*, 268(1):1–12, 2018.

Ander Garcia, Olatz Arbelaitz, Pieter Vansteenwegen, Wouter Souffriau, and Maria Teresa Linaza. Hybrid approach for the public transportation time dependent orienteering problem with time windows. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 151–158. Springer, 2010.

Damianos Gavalas, Charalampos Konstantopoulos, Konstantinos Mastakas, Grammati Pantziou, and Yiannis Tasoulas. Cluster-based heuristics for the team orienteering problem with time windows. In *International Symposium on Experimental Algorithms*, pages 390–401. Springer, 2013.

Hermann Gehring and Jörg Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In *Proceedings of EUROGEN99*, volume 2, pages 57–64. Citeseer, 1999.

Michel Gendreau, Ola Jabali, and Walter Rei. 50th anniversary invited article—future research directions in stochastic vehicle routing. *Transportation Science*, 50(4):1163–1173, 2016.

Bruce L Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval research logistics*, 34(3):307–318, 1987.

Google API. Google api. https://developers.google.com/maps/., 2018.

Emilia Grass and Kathrin Fischer. Two-stage stochastic programming in disaster management: A literature survey. *Surveys in Operations Research and Management Science*, 21(2):85–100, 2016.

A. Gunawan, H.C. Lau, and P. Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.

Aldy Gunawan, Hoong Chuin Lau, and Kun Lu. An iterated local search algorithm for solving the orienteering problem with time windows. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 61–73. Springer, 2015.

Aldy Gunawan, Hoong Chuin Lau, Pieter Vansteenwegen, and Kun Lu. Well-tuned algorithms for the team orienteering problem with time windows. *Journal of the Operational Research Society*, 68(8):861–876, 2017.

Katharine Haynes, John Handmer, John McAneney, Amalie Tibbits, and Lucinda Coates. Australian bushfire fatalities 1900–2008: exploring trends in relation to the 'prepare, stay and defend or leave early'policy. *environmental science & policy*, 13(3):185–194, 2010.

Maria Camila Hoyos, Ridley S Morales, and Raha Akhavan-Tabatabaei. Or models with stochastic components in disaster operations management: A literature survey. *Computers & Industrial Engineering*, 82:183–197, 2015.

Taylan Ilhan, Seyed MR Iravani, and Mark S Daskin. The orienteering problem with stochastic profits. *Iie Transactions*, 40(4):406–421, 2008.

Paul Johnston, Joel Kelso, and George J Milne. Efficient simulation of wildfire spread on an irregular grid. *International Journal of Wildland Fire*, 17(5):614–627, 2008.

Morteza Keshtkaran, Koorush Ziarati, Andrea Bettinelli, and Daniele Vigo. Enhanced exact solution methods for the team orienteering problem. *International Journal of Production Research*, 54(2):591–601, 2016.

Byung-In Kim, Hong Li, and Andrew L Johnson. An augmented large neighborhood search method for solving the team orienteering problem. *Expert Systems with Applications*, 40(8):3065–3072, 2013.

Byoung Chul Ko, Joon-Young Kwak, and Jae-Yeal Nam. Wildfire smoke detection using temporospatial features and random forest classifiers. *Optical Engineering*, 51(1):017208, 2012.

Vitaliy Krasko and Steffen Rebennack. Two-stage stochastic mixed-integer nonlinear programming model for post-wildfire debris flow hazard management: Mitigation and emergency evacuation. *European Journal of Operational Research*, 263(1): 265–282, 2017.

Nacima Labadie, Renata Mansini, Jan Melechovskỳ, and Roberto Wolfler Calvo. The team orienteering problem with time windows: An lp-based granular variable neighborhood search. *European Journal of Operational Research*, 220(1):15–27, 2012.

Gilbert Laporte. Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416, 2009.

Baoxiang Li, Dmitry Krushinsky, Tom Van Woensel, and Hajo A Reijers. An adaptive large neighborhood search heuristic for the share-a-ride problem. *Computers & Operations Research*, 66:170–180, 2016.

Yun-Chia Liang, Sadan Kulturel-Konak, and Min-Hua Lo. A multiple-level variable neighborhood search approach to the orienteering problem. *Journal of Industrial and Production Engineering*, 30(4):238–247, 2013.

Shih-Wei Lin and F Yu Vincent. A simulated annealing heuristic for the team orienteering problem with time windows. *European Journal of Operational Research*, 217(1):94–107, 2012.

Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196, 2012.

Ran Liu, Yangyi Tao, and Xiaolei Xie. An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and synchronized visits. *Computers & Operations Research*, 101:250–262, 2019.

Yousef Maknoon and Gilbert Laporte. Vehicle routing with cross-dock selection. *Computers & Operations Research*, 77:254–266, 2017.

Yannis Marinakis, Michael Politis, Magdalene Marinaki, and Nikolaos Matsatsinis. A memetic-grasp algorithm for the solution of the orienteering problem. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 105–116. Springer, 2015.

David L Martell. A review of recent forest and wildland fire management decision support systems research. *Current Forestry Reports*, 1(2):128–137, 2015.

Dmytro Matsypura, Oleg A Prokopyev, and Aizat Zahar. Wildfire fuel management: network-based models and optimization of prescribed burning. *European Journal of Operational Research*, 264(2):774–796, 2018.

Yi Mei, Flora D Salim, and Xiaodong Li. Efficient meta-heuristics for the multi-objective time-dependent orienteering problem. *European Journal of Operational Research*, 254(2):443–457, 2016.

James P Minas, John W Hearne, and John W Handmer. A review of operations research methods applicable to wildfire management. *International Journal of Wildland Fire*, 21(3):189–196, 2012.

James P Minas, John W Hearne, and David L Martell. A spatial optimisation model for multi-period landscape level fuel management to mitigate wildfire impacts. *European Journal of Operational Research*, 232(2):412–422, 2014.

Roberto Montemanni and Luca Maria Gambardella. An ant colony system for team orienteering problems with time windows. *Foundation Of Computing And Decision Sciences*, 34(4):287, 2009.

Newsroom. Attica wildfires damages. `http://thegreekobserver.com/greece/article/47751/according-to-eaee-insurance-claims-for-attica-wildfires-damages-total-33-7-million-euros-so-far/`, August 2018.

Linet Özdamar and Mustafa Alp Ertem. Models, solutions and enabling technologies

in humanitarian logistics. *European Journal of Operational Research*, 244(1):55–65, 2015.

Vassilis Papapanagiotou, Roberto Montemanni, and Luca Maria Gambardella. Objective function evaluation methods for the orienteering problem with stochastic travel and service times. *Journal of applied Operational research*, 6(1):16–29, 2014.

A Petrasova, B Harmon, V Petras, P Tabrizian, and H Mitasova. Wildfire spread simulation. In *Tangible Modeling with Open Source GIS*, pages 155–163. Springer, 2018.

David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435, 2007.

Marcus Poggi, Henrique Viana, and Eduardo Uchoa. The team orienteering problem: Formulations and branch-cut and price. In *OASIcs-OpenAccess Series in Informatics*, volume 14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

Jeffrey P Prestemon, David T Butry, Karen L Abt, and Ronda Sutphen. Net benefits of wildfire prevention education efforts. *Forest Science*, 56(2):181–192, 2010.

Ramya Rachmawati, Melih Ozlen, John Hearne, and Karin Reinke. Fuel treatment planning: Fragmenting high fuel load areas while maintaining availability and connectivity of faunal habitat. *Applied Mathematical Modelling*, 54:298–310, 2018.

Ulrike Ritzinger, Jakob Puchinger, and Richard F Hartl. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1):215–231, 2016.

Iman Roozbeh, Melih Ozlen, and John W Hearne. An adaptive large neighbourhood search for asset protection during escaped wildfires. *Computers & Operations Research*, 97:125–134, 2018.

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

Alberto Santini. An adaptive large neighbourhood search algorithm for the orienteering problem. *Expert Systems with Applications*, 2019.

Alberto Santini, Stefan Ropke, and Lars Magnus Hvattum. A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic. *Journal of Heuristics*, 24(5):783–815, 2018.

Joe H Scott and Robert E Burgan. Standard fire behavior fuel models: a comprehensive set for use with rothermel's surface fire spread model. *Gen. Tech. Rep. RMRS-GTR-153. Fort Collins, CO: US Department of Agriculture, Forest Service, Rocky Mountain Research Station. 72 p.*, 153, 2005.

Shahrooz Shahparvari, Prem Chhetri, Babak Abbasi, and Ahmad Abareshi. Enhancing emergency evacuation response of late evacuees: Revisiting the case of australian black saturday bushfire. *Transportation research part E: logistics and transportation review*, 93:148–176, 2016.

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

Wouter Souffriau, Pieter Vansteenwegen, Greet Vanden Berghe, and Dirk Van Oudheusden. The multiconstraint team orienteering problem with multiple time windows. *Transportation Science*, 47(1):53–63, 2013.

Andrew L Sullivan. Wildland surface fire spread modelling, 1990–2007. 3: Simulation and mathematical analogue models. *International Journal of Wildland Fire*, 18 (4):387–403, 2009.

B Teague, R McLeod, and S Pascoe. Final report 2009 victorian bushfires royal commission. *Parliament of Victoria, Melbourne Victoria, Australia*, 2010.

Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

Fabien Tricoire, Martin Romauch, Karl F Doerner, and Richard F Hartl. Heuristics

for the multi-period orienteering problem with multiple time windows. *Computers & Operations Research*, 37(2):351–367, 2010.

Martijn Van Der Merwe, James P Minas, Melih Ozlen, and John W Hearne. A mixed integer programming approach for asset protection during escaped wildfires. *Canadian Journal of Forest Research*, 45(4):444–451, 2015.

Yer Van Hui, Jia Gao, Lawrence Leung, and Stein Wallace. Airfreight forwarder's shipment planning under uncertainty: A two-stage stochastic programming approach. *Transportation Research Part E: Logistics and Transportation Review*, 66:83–102, 2014.

Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated Annealing: Theory and Applications*, pages 7–15. Springer, 1987.

Pieter Vansteenwegen and Dirk Van Oudheusden. The mobile tourist guide: an or opportunity. *OR insight*, 20(3):21–27, 2007.

Pieter Vansteenwegen, Wouter Souffriau, Greet Vanden Berghe, and Dirk Van Oudheusden. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12):3281–3290, 2009.

Pieter Vansteenwegen, Wouter Souffriau, Greet Vanden Berghe, and Dirk Van Oudheusden. The city trip planner: an expert system for tourists. *Expert Systems with Applications*, 38(6):6540–6546, 2011a.

Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011b.

Pradeep Varakantham and Akshat Kumar. Optimization approaches for solving chance constrained stochastic orienteering problems. In *International Conference on Algorithmic DecisionTheory*, pages 387–398. Springer, 2013.

Cédric Verbeeck, Pieter Vansteenwegen, and E-H Aghezzaf. An extension of the arc orienteering problem and its application to cycle trip planning. *Transportation research part E: logistics and transportation review*, 68:64–78, 2014.

Thibaut Vidal, Nelson Maculan, Luiz Satoru Ochi, and Puca Huachi Vaz Penna. Large neighborhoods with implicit customer selection for vehicle routing problems with profits. *Transportation Science*, 50(2):720–734, 2015.

F Yu Vincent, Parida Jewpanya, Shih-Wei Lin, and AAN Perwira Redi. Team orienteering problem with time windows and time-dependent scores. *Computers & Industrial Engineering*, 127:213–224, 2019.

Yu Wei, Michael Bevers, Erin Belval, and Benjamin Bird. A chance-constrained programming model to allocate wildfire initial attack resources for a fire season. *Forest Science*, 61(2):278–288, 2014.

J Whittaker, J McLennan, G Elliott, J Gilbert, J Handmer, K Haynes, and S Cowlishaw. Victorian 2009 bushfire research response: Final report. *Bushfire CRC Post-fire Research Program in Human Behaviour.(Bushfire CRC: Melbourne) Available at http://www. bushfirecrc. com/managed/resource*, 2009.

Mengying Zhang, Jin Qin, Yugang Yu, and Liang Liang. Traveling salesman problems with profits and stochastic customers. *International Transactions in Operational Research*, 25(4):1297–1313, 2018.

Shu Zhang, Jeffrey W Ohlmann, and Barrett W Thomas. A priori orienteering with time windows and stochastic wait times at customers. *European Journal of Operational Research*, 239(1):70–79, 2014.

Appendix One

A.1 Supplementary data for the case study

A weather change is forecast after the initial trip scheduled for the Black Saturday case study (chapter 6). In table A1, assets impacted as a result of the change in the weather condition are identified on the map. The protection requirement vector shows the number of resources of each type needed for accomplishing the protection task. Protection requirement vectors are assigned randomly and time windows are defined approximately based on The Victorian Bushfires Royal Commission report (Whittaker et al. (2009)). Some assets are impacted in multiple scenarios and others in one scenario, while those that are not at risk require no protection and have the impact time of 0.

Table A1: List of assets being impacted in chapter 6. For sensitivity reasons asset names are not mentioned.

#	Asset ID	Protection requirement vector	Time of being impacted at stage 1	Time of being impacted at scenario 1 (stage 2)	Time of being impacted at scenario 2 (stage 2)	Time of being impacted at scenario 3 (stage 2)
1	A1	<0, 2, 1>	-	-	6.5	7
2	A2	<2, 1, 0>	2	-	-	-
3	A3	<1, 2, 1>	-	5.5	8	-
4	A4	<1, 2, 1>	1	-	-	-
5	A5	<2, 0, 1>	3	-	-	-
6	A6	<1, 2, 0>	-	5	6.5	7
7	A7	<1, 0, 0>	-	-	4.4	4.4
8	A8	<1, 0, 2>	-	-	8	8.5
9	A9	<2, 0, 1>	-	9	-	-
10	A10	<2, 1, 0>	-	6.5	7.5	8
11	A11	<2, 1, 0>	-	-	4	4
12	A12	<0, 2, 1>	-	-	4.5	4.5
13	A13	<2, 1, 0>	-	-	-	5.5
14	A14	<0, 2, 1>	-	-	6.5	7
15	A15	<1, 0, 2>	-	4.5	6	-
16	A16	<2, 1, 0>	-	-	-	5.5
17	A17	<0, 2, 1>	1	-	-	-
18	A18	<1, 1, 2>	3	-	-	-
19	A19	<1, 0, 2>	-	11	-	-
20	A20	<1, 0, 2>	-	-	7.5	8
21	A21	<2, 0, 1>	-	9.5	9	0
22	A22	<1, 2, 1>	-	7	-	-
23	A23	<2, 1, 0>	-	10.5	10	-
24	A24	<0, 2, 1>	-	9	-	-
25	A25	<1, 2, 1>	-	-	9	9.5