Dynamic Vehicle Routing under Uncertain Travel Costs and Refueling Opportunities

Giorgos Polychronis and Spyros Lalis
Department of Electrical and Computer Engineering, University of Thessaly, Volos, Greece

Keywords: Traveling Salesman Problem, Vehicle Routing Problem, Uncertainty, Dynamic Route Planning.

Abstract: We study the vehicle routing problem for a system where there is some uncertainty regarding both the cost of travel and the refueling opportunities. Travel cost stands for the energy spent by the vehicle to move between locations. Refueling opportunities are offered at known locations where the vehicle can harvest or re-gain some of the lost energy. The objective is to visit a set of predefined locations without exhausting the energy of the vehicle. We describe the problem in a formal way, and propose a heuristic algorithm for taking routing decisions at runtime. We evaluate the algorithm for a grid topology as a function of the number of locations to be visited and the autonomy degree of the vehicle, showing that the proposed algorithm achieves good results as long as the energy margins are not very tight.

1 INTRODUCTION

Unmanned Vehicles (UVs) will play a major role in next-generation applications. In particular, Unmanned Aerial Vehicles (UAVs) are already being used in the domain of agriculture and surveillance. Other types of UVs, such as Unmanned Ground Vehicles (UGVs) or Unmanned Underwater Vehicles (UUVs), though more exotic, are becoming more mature and affordable, and will most likely be used in several applications in the future.

A typical scenario is for UVs to visit specific locations or entire areas in order to take measurements or to detect objects / phenomena of interest. For example, in agriculture, UVs scan a crop field to detect problematic spots that are infected with pests. Similarly, in search and rescue missions, UVs scan a target area to find missing persons. Yet another example is for a UV to patrol an area by visiting several predefined locations. In all these cases, the UV should perform the mission as efficiently as possible.

This problem is known as the Vehicle Routing Problem (VRP), which in turn is an extension of the Traveling Salesman Problem (TSP). In a nutshell, the VRP consists in finding a travel plan that can be followed by a vehicle in order to visit a set of target locations. The problem has been studied in many variants and different constraints regarding the specific paths that can be followed by the vehicle, the locations to be visited or the time window where certain locations have to be visited. Also, in several formulations, the vehicle is assumed to have finite fuel/energy reserves or serving capacity, which can be replenished by visiting special so-called depot nodes.

In this paper, we investigate a variant of the VRP where there is *uncertainty* regarding the cost of travel and the energy harvesting / refueling opportunities that can be exploited by the vehicle. Algorithms that compute the travel plan based on static data are not suitable in this case. This is because these plans may turn out to be infeasible for the situation that the vehicle faces during the mission — its energy reserves will be depleted before completing the mission. Instead, a more dynamic approach is required, so that routing decisions can be taken and adjusted at runtime, based on the situation at hand.

The main contributions of the paper are: (i) it provides a formal description for a new variant of the vehicle routing problem; (ii) it proposes a heuristic algorithm to tackle the problem; (iii) it evaluates the proposed algorithm, showing that it performs well compared to other static solutions of the problem.

The rest of the paper is structured as follows. Section 2 gives an overview of related work. Section 3 describes the problem we study in a formal way. Section 4 presents the algorithm for solving the problem. Section 5 evaluates the algorithm for different scenarios. Finally, Section 6 concludes the paper.

Polychronis, G. and Lalis, S.
Dynamic Vehicle Routing under Uncertain Travel Costs and Refueling Opportunities.
DOI: 10.5220/0007673900520063
In *Proceedings of the 5th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2019)*, pages 52-63
ISBN: 978-989-758-374-2

2 RELATED WORK

Different variants of the VRP and the TSP have been investigated. We give a brief overview, and compare with the problem we address in our work.

2.1 Vehicle Routing with Fuel/Energy Constraints and Periodic Visits

In the so-called *periodic vehicle routing problem* and *multi-depot periodic vehicle routing problem* (Angelelli and Speranza, 2002; Cordeau et al., 1997; Gaudioso and Paletta, 1992; Vidal et al., 2012; Alonso et al., 2008; Rahimi-Vahed et al., 2013; Escobar et al., 2014), the objective is to periodically visit a set of destinations (target nodes) using one or more vehicles. Vehicles have limited energy reserves and may recharge/refuel at special depot stations. There are more variations where the targets have different priorities, vehicles have different capabilities/capacities, and the targets must be visited within a time interval. We briefly discuss such work below.

Work in (Las Fargeas et al., 2012) investigates the problems of *persistent visitation with fuel constraints*. In this case, the vehicle perpetually visits the target nodes (customer nodes), while each customer must be visited at a specific rate. The fuel of the vehicle is limited, so it is necessary to visit nodes with refueling properties. The problem aims at satisfying the rate of visit for each destination, as well as at minimizing the total cost of fuel consumption.

The problem of *continuous monitoring* is studied in (Mersheeva, 2015), where the mission is accomplished by several heterogeneous vehicles of different types with limited resources. The objective is to visit periodically a set of target nodes with different priorities in a given time interval. The rate at which each target node needs to be visited by any of the vehicles is proportional to its priority. There is also a set of depot nodes where vehicles can change their batteries, however each type of vehicle requires a different type of battery; only vehicles of the same type can use the same type of battery. Also, the number of the different types of batteries available at each depot is limited.

All the above formulations correspond to VRPs with a fleet of one or more vehicles that have energy/fuel limitations. Each edge of the graph is associated with a cost, and the energy of the vehicles decreases accordingly based on that cost. There are also one or more so-called depot nodes, where the vehicles can refuel, fully or partially. The main goal is to find a path that minimizes the total cost of the mission, which is the sum of the costs of the edges chosen to be used in the path of the vehicles. Another objective is to satisfy the visit rates of the target nodes.

The main difference between the above and our work is that in our case the cost of travel between nodes as well as the amount of energy/fuel that can be gained at depot nodes, is not known a priori with full certainty.

2.2 Vehicle Routing with Stochastic Elements

The objective of *vehicle routing problem with stochastic demands* is for a fleet of one or more vehicles, which have a finite serving capacity and may also have fuel constraints, to visit and serve a set of known target nodes (customers). However, the exact demand of each customer is not known beforehand. As a result, when a vehicle visits a customer, it may find out that the demand exceeds its remaining capacity. The failure to properly service the customer can be ignored, at a penalty, and the vehicle may proceed to the next customer. Alternatively, the vehicle may go to a depot node in order to acquire the missing resources to restore its servicing capacity, and then return to service the customer. This problem is researched in many works (Bertsimas, 1992; Secomandi, 2001; Laporte et al., 2002; Marinakis et al., 2013; Rei et al., 2010; Juan et al., 2011; Marinaki and Marinakis, 2016). Moreover, in (Erera et al., 2010) and (Mendoza et al., 2016) the problem is investigated for the more general case where the goal is to minimize the travel time with the additional constraint of keeping the travel time below a given upper bound.

In the *vehicle routing problem with stochastic travel times*, a fleet of one or more vehicles, which may have capacity and/or fuel constraints, visit a set of known targets, but in this case, the travel time is a random variable. Also, each target has a time window during which it is available for visits by a vehicle. If the target is not visited/served within that time window, there is a penalty. The objective is to route the vehicles so as to minimize the total penalty. Algorithms tackling this problem are described in (Taş et al., 2013; Laporte et al., 1992; Van Woensel et al., 2003; Kenyon and Morton, 2003; Ehmke et al., 2015; Miranda and Conceição, 2016; Taş et al., 2014).

The *vehicle routing problem with stochastic customers* consists in using a fleet of vehicles, which may have capacity and/or fuel constraints, to visit a set of target customers. However, these targets require a visit only with some probability, thus a vehicle may visit a node that does not need to be serviced. Indicative work can be found in (Gendreau et al., 1996; Gendreau et al., 1995; Bent and Van Hentenryck, 2004).

Our work is more similar to the VRP with stochastic travel times. However, the main constraint there, is the travel time, whereas in our work the main constraint is the energy of the vehicle, which can also increase its energy reserves by visiting certain nodes. It is also important to note that in our work the mission ends when the vehicle exhausts its energy. This is a major difference compared to the VRP with stochastic demands, in which case the vehicle can always go to a depot node in order to restore its capacity. As a consequence, in our work, it is not always possible to visit all nodes, in contrast to most other problem formulations where this is always feasible and the problem consists in minimizing the travel cost and/or time violation penalty. In this sense, our work is closer to (Erera et al., 2010) and (Mendoza et al., 2016), which have a similar constraint for the travel cost. But in our case the stochasticity concerns the travel cost and energy gains, not the customer demands, and the objective is to maximize the number of target nodes that are visited by the vehicle.

2.3 Energy-efficient Path Planning

Several algorithms have been designed to compute an energy-efficient route between two nodes, based on an abstract graph where the weights at edges represent the energy consumption or the energy restoration. The difference to a typical shortest-path algorithm is that the computed routes minimize the sum of edge costs, not the number of edges. For instance, (Sachenbacher et al., 2011) deals with the problem of energy-efficiency as a special case of the constrained shortest path problem, while in (Artmeier et al., 2010) the problem of energy-efficient path planning is viewed as a cost minimization problem. In both cases, the ability of the vehicle to gain energy / recharge is modeled by introducing edges with negative weights.

Our work is also related to the problem of finding a cost-effective path between two nodes. However, the minimization of the travel cost per se is not as central as in the above algorithms, because it can be counter-balanced by gaining energy at depot nodes. What is ultimately important is for the vehicle to take a route that turns out to be feasible, while visiting as many target nodes as possible. To this end, the proposed heuristic employs a suitably adapted version of the Bellman-Ford algorithm (Bellman, 1958; Ford Jr, 1956), which finds the path between two nodes that maximizes the energy reserves of the vehicle (as opposed to minimizing travel cost). Also, our version can exploit beneficial cycles —not allowed in the original algorithm as the travel cost could then be reduced infinitely.

3 MODEL AND PROBLEM FORMULATION

This section presents the system model, and defines the problem we study in a more formal way.

3.1 Terrain and Travel Paths

The terrain of travel is modeled as an incomplete directed graph $(\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges between nodes. Each node $n_i \in \mathcal{N}$ represents a distinct location/position in the terrain. Each edge $e_{i,j} \in \mathcal{E}$ captures the fact that the vehicle can move directly from node n_i to node n_j without having to go via one or more intermediate nodes.

Edges are directed, and there can be at most one edge between two nodes per direction. Note that the fact that it is possible to move directly from n_i to n_j does not mean that this is possible in the reverse direction. In other words, $e_{i,j} \in \mathcal{E} \not\Rightarrow e_{j,i} \in \mathcal{E}$.

The vehicle can visit nodes by traveling across the edges of the graph. The travel path is encoded as the sequence of nodes that are visited by the vehicle, including the start and end node of the path. Let $p_{k_1,k_2,k_3,..,k_{m-1},k_m}$ denote the path that starts from node n_{k_1}, goes through n_{k_2}, n_{k_3} etc, and ends at node n_{k_m}, also corresponding to the list of edges $(e_{k_1,k_2}, e_{k_2,k_3}, ..., e_{k_{m-1},k_m})$. Let $occ(p, e_{i,j})$ be the number of times edge $e_{i,j}$ occurs in p, and $occ(p, n_i)$ be the number of times node n_i occurs as the starting point of an edge in p. Also, let $p + e_{i,j}$ denote the path that results by appending edge $e_{i,j}$ to p. Finally, let $len(p)$ denote the number of edges (hops) in path p, and let $nodes(p)$ denote the set of nodes in p.

The set $\mathcal{V} \subseteq \mathcal{N}$ includes the nodes that correspond to the target locations that have to be visited by the vehicle. We are interested in algorithms that guide the movement of the vehicle so that it ideally visits all nodes in $\mathcal{V}$. Note that the vehicle may start its journey from node $n_s \notin \mathcal{V}$, and may end its journey at node $n_d \notin \mathcal{V}$. Also, the path that will be followed by the vehicle may include nodes that do not belong in $\mathcal{V}$.

3.2 Energy Reserves, Cost and Gains

We assume that the vehicle has limited energy storage capacity B_{max}. This can be thought of as the capacity of a fuel tank or the capacity of a battery, depending on whether the vehicle is equipped with an internal combustion engine or an electrical motor. Let b denote the current energy budget (reserves) of the vehicle. Obviously, $b \leq B_{max}$ holds at any point in time.

The movement of the vehicle comes at a cost. Let $c_{i,j}$ denote the cost incurred when the vehicle travels from n_i to n_j over $e_{i,j}$, also referred to as edge cost. This represents the amount of energy spent to power the motors of the vehicle in order to perform this movement. If the vehicle has an energy budget b and moves from n_i to n_j over $e_{i,j}$, the remaining energy budget will be $b' = b - c_{i,j}$. If $b' \leq 0$, the vehicle will exhaust its energy and it will stop (abort the mission) before reaching n_j. The edge cost is *not known* in advance with certainty. However, we assume that the edge cost $c_{i,j}$ follows a *known* random distribution over the range $[c_{i,j}^{min}..c_{i,j}^{max}]$ with an expected/mean value of $c_{i,j}^{mean}$.

The vehicle may increase its energy budget, by gaining some energy at so-called *depot* nodes. One can think of depot nodes as refueling or recharging stations. Similarly to the edge costs, the amount of energy that can be gained at a depot node n_i is a random variable g_i, which follows a *known* random distribution over the range $[g_i^{min}..g_i^{max}]$ with an expected/mean value of g_i^{mean}. Without loss of generality, we assume that it is known in advance whether a node can potentially provide some energy gain. Let $\mathcal{D} = \{n_i | g_i^{min} > 0\}$ be the set of all depot nodes, i.e., all nodes with a non-zero probability of energy gain. In the general case, it is possible for a node that has to be visited by the vehicle, to be a depot node, $\mathcal{V} \cap \mathcal{D} \neq \emptyset$.

3.3 Path Feasibility

In order for a planned path to be *feasible*, the budget of the vehicle must be sufficient to cover the cost for crossing each edge along that path. In the following, we capture this constraint in a more formal way.

Let $c_{i,j}^{x}$ denote the actual cost of edge $e_{i,j}$ when the vehicle crosses that edge for the xth time. Similarly, g_i^x is the gain of node n_i when the vehicle visits that node for the xth time.

First, we capture the budget that remains available when starting with an initial budget b and performing a single hop from n_i to n_j over edge $e_{i,j}$:

$$rem(b,h,p_{i,j}) = min(B_{max}, b + g_i^{occ(h,n_i)+1}) - c_{i,j}^{occ(h,e_{i,j})+1} \tag{1}$$

where h (history) is the path that has already been followed up to this point. In words, the gain of n_i is added to the budget b (up to the maximum energy storage capacity B_{max}) and then the edge cost is subtracted in order for the vehicle to cross the edge that leads from n_i to n_j. Note the usage of $occ()$ in order to take into account previous occurrences of node n_i and edge $e_{i,j}$ in the path that was followed so far. The equation does not take into account the gain at the destination node n_j (if any), as this cannot be exploited in order to cross the edge $e_{i,j}$.

We can then define the remaining budget for the general case of a multi-hop path p from node n_i to node n_j, as follows:

$$rem(b,h,p_{i,k_1,k_2,...,k_m,j}) = rem(rem(b,h,p_{i,k_1}), h + e_{i,k_1}, p_{k_2,...,k_m,j}) \tag{2}$$

In words, the budget that remains after taking a multi-hop path equals the remaining budget for the path without the first hop, starting with a budget that is equal to the remaining budget after taking the first hop. As this is the case for the 1-hop path, the remaining budget of a multi-hop path does not include the gain at the destination node.

Finally, we can define the feasibility of a planned path p, assuming the vehicle has already traveled along path h and has a current remaining budget b, as follows. We say that $p_{k_1,k_2,...,k_m}$ is feasible if for all prefix paths $p_{k_1,k_2,...,k_x}, 1 < x \leq m$ (including the full path itself) it holds that $rem(b,h,p_{k_1,k_2,...,k_x}) > 0$. In words, p is feasible if the vehicle will not exhaust its budget at any point along p.

3.4 Performance Metric

We wish to devise an algorithm that will guide the vehicle so that it manages to visit all nodes in $\mathcal{V}$ without exhausting its budget while en route. In other words, starting with h equal to *null* and an initial budget b, we need to find a feasible path $p_{k_1,k_2,,..,k_m}$ so that $\mathcal{V} \subseteq nodes(p)$.

As a general performance metric, we use the *coverage* of a path p, which is equal to the ratio between the number of the nodes of interest that are part of the path that was followed by the vehicle and the total number of nodes of interest that the vehicle should visit: $cov(p,\mathcal{V}) = \frac{|nodes(p) \cap \mathcal{V}|}{|\mathcal{V}|}$. Obviously, the ideal case is to find a path p so that $cov(p,\mathcal{V}) = 1$.

4 MaxBudget ALGORITHM

To solve the problem, we propose a heuristic algorithm, called *MaxBudget*. The input to the algorithm is the graph $(\mathcal{N},\mathcal{E})$, the set $\mathcal{V}$ of the nodes to visit, the node n_s from where the patrol starts, and the initial energy budget b of the vehicle (without the gains of the start node).

4.1 High-level Algorithmic Skeleton

The algorithm works in two phases. In a first step, a path is planned from the current (source) node to some node of interest. In a second step, the vehicle actually tries to follow that path. After each hop, an assessment of the situation is made, which is then compared to the assumptions of the path planning step. If things go more or less according to plan, the vehicle continues its journey according to plan. Else, it stops following the planned path, and a fresh path planning step is performed, based on the situation at hand. When a node of interest is reached, this is removed from the set of nodes to visit, and the algorithm continues for the rest of the nodes, by planning and then following a path to another node of interest.

In the planning phase, the paths are chosen based on *estimated* values for the edge costs and node gains. Two different estimation modes are employed. In the *normal* mode, the estimates correspond to the expected/mean values ($c_{i,j}^{mean}$ and g_i^{mean}) of the respective random distributions. In the *optimistic* mode, the estimates used for the edge costs are equal to the minimum values ($c_{i,j}^{min}$) of the respective distributions, while the estimates for the node gains are the maximum (g_i^{max}) values of the respective distributions. In the planning step, first, an attempt is made to find a feasible path using the normal mode. If this does not succeed, the planning step is repeated, for a second time, in the optimistic mode. If this fails too, the vehicle has reached a dead end —no feasible path exists from the current location to any of the remaining nodes of interest. The algorithm terminates when all target nodes have been visited or a dead end is reached, and returns the path that was followed and the remaining target nodes.

The high-level algorithmic skeleton is shown in Algorithm 1. Note that the path planning step and the assessment of the situation that drives the transition from the normal to the optimistic path planning mode, are captured via separate functions.

The path planning step is captured via function $Plan(n_s, b, \mathcal{V}, mode)$. It takes as parameters the starting node n_s, the available budget b, the set of nodes $\mathcal{V}$ to be visited, and the estimation mode *mode*. It returns a suggested path to follow. The path p is a *complex* data structure, where the hops are recorded in an array $p.hops[k], 0 \leq k \leq len(p)$, $p.hops[k].n$ is the node at the kth hop of the path, and $p.hops[k].b$ is the *estimated* available budget at this point of the path. For convenience, we also use $p.b$ to store the available budget at the end of the path $p.hops[len(p)].b$. Note that $p.hops[0].n = n_s$ and $p.hops[0].b = b$, as per the input parameters of the function.

Algorithm 1: High-level algorithmic skeleton.

```
function SKELETON($\mathcal{V}, n_s, budget$)
    $n_{cur} \leftarrow n_s$                          ▷ current node
    $b \leftarrow budget$                             ▷ remaining budget
    $fp \leftarrow n_{cur}$                           ▷ full path of the vehicle
    $\mathcal{V}' \leftarrow \mathcal{V} - n_{cur}$   ▷ remaining nodes to visit
    $p \leftarrow null$                               ▷ planned path to next target node
    while $\mathcal{V}' \neq \emptyset$ do
        if $p = null$ then
            $mode \leftarrow NORM$
            $p \leftarrow Plan(n_{cur}, b, \mathcal{V}', mode)$
            if $p = null$ then
                $mode \leftarrow OPT$
                $p \leftarrow Plan(n_{cur}, b, \mathcal{V}', mode)$
                if $p = null$ then                    ▷ dead end
                    return $(fp, \mathcal{V}')$
                end if
            end if
            $k \leftarrow 1$                          ▷ init hop counter
        end if
        $b \leftarrow \min(b + g_{cur}, B)$           ▷ enjoy node gain
        $n_{nxt} \leftarrow p.hops[k].n$              ▷ take next hop
        $b \leftarrow b - c_{cur,nxt}$                ▷ pay travel cost
        if $b < 0$ then                               ▷ dead end
            return $(fp, \mathcal{V}')$
        end if
        $n_{cur} \leftarrow n_{nxt}$
        $fp \leftarrow fp + n_{cur}$
        $\mathcal{V}' \leftarrow \mathcal{V}' - n_{cur}$
        if $k < len(p) \land Check(p, k, b, mode)$ then
            $k \leftarrow k + 1$                      ▷ proceed with the next hop
        else
            $p \leftarrow null$                       ▷ plan new path
        end if
    end while
    return $(fp, \mathcal{V}')$
end function
```

Function $Check(p, k, b, mode)$ is used to check if the current situation (at the kth hop of p, with available budget b) is along the lines of what was assumed in the planning step for the *mode* used. If things go as planned, the current plan is followed, else a new plan is devised by invoking $Plan()$. A new plan is also needed when the current path is successfully followed to its end, and there are still some nodes of interest that have not yet been visited.

4.2 Path Planning Algorithm

The path planning heuristic we propose favors paths that lead to nodes of interest while *maximizing the remaining budget*. The intuition is that the budget that remains available after visiting a node of interest can be further exploited in the next steps of the trip so that it is possible to pursue additional paths and to visit more nodes of interest. The high-level pseudocode for this heuristic is given in Algorithm 2.

Algorithm 2: Path planning heuristic.

```
function PLAN($n_s, b, \mathcal{V}, mode$)
    $n_d \leftarrow null$                  ▷ preferred destination node
    $maxb \leftarrow -\infty$              ▷ remaining budget for $n_d$
    $p[] \leftarrow MaxBudgetPaths(n_s, b, mode)$
    for each $n_i \in \mathcal{V}$ do
        if $p[n_i] \neq (null, -\infty)$ then
            if $p[n_i].b > maxb$ then
                $maxb, n_d \leftarrow p[n_i].b, n_i$
            end if
        end if
    end for
    if $n_d = null$ then
        return $null$
    else
        return $p[n_d]$
    end if
end function

function MAXBUDGETPATHS($n_s, b, mode$)
    $p[|\mathcal{N}|]$        ▷ path and budget from $n_s$ to every $n_i$
    for each $n_i \in \mathcal{N}$ do
        $p[n_i], p[n_i].b \leftarrow (null, -\infty), -\infty$
    end for
    $p[n_s], p[n_s].b \leftarrow (n_s, b), b$
    repeat
        $update \leftarrow false$
        for each $n_i, n_j | e_{ij} \in \mathcal{E}$ do
            $b2 \leftarrow \mathbf{min}(p[n_i].b + g_i^{mode}, B_{max})$
            $b2 \leftarrow b2 - c_{ij}^{mode}$              ▷ try edge
            if $(b2 > 0) \wedge (b2 > p[n_j].b)$ then
                $p[n_j], p[n_j].b \leftarrow p[n_i] + (n_j, b2), b2$
                $update \leftarrow true$
            end if
        end for
    until $update = false$
    return $p[]$
end function

function CHECK($p, k, b, mode$)
    if $mode =$ NORM then
        return $\frac{|p.hops[k].b - b|}{p.hops[k].b} \leq Threshold_{norm}$
    else
        return $\frac{|p.hops[k].b - b|}{p.hops[k].b} \geq Threshold_{opt}$
    end if
end function
```

Function *Plan*() finds a feasible path that maximizes the available budget for *every* node of interest, and then picks and returns the path that leads to the maximum budget. Ties are broken by giving preference to shorter paths (for brevity, this is not shown in the pseudocode).

The main path finding logic is structured as a separate function, *MaxBudgetPaths*(). This follows the principle of the Bellman-Ford (BF) algorithm (Bellman, 1958; Ford Jr, 1956), with some extensions that we have introduced for the purposes of our heuristic. Initially, the path from n_s to every node n_i is set to *null* and the remaining budget for that path is set to $-\infty$. Then it repeats an update procedure. A path to a node n_i is updated, when a new path from n_s to this node is found and the remaining budget, when following this new path, is greater than the current one. This procedure is repeated until no more paths are updated. The adaptations to the original BF algorithm are discussed in more detail in the next subsection.

Function *Check*() compares the available budget at any given hop of the path with the estimated available budget for that hop as this was calculated by the *MaxBudgetPaths*() function. In the normal mode, the function returns *true* if this difference is less or equal to a threshold. The rationale is that there is no reason to change the plan as long as things go according to it. In contract, in the optimistic mode, *Check*() returns *true* if the difference is greater or equal to a threshold. The rationale is that as long as the real situation does not come close to the optimistic estimates, it does not make any sense to devise a new plan in the normal, more conservative mode.

4.3 Adaptations to the Bellman-Ford Algorithm

As mentioned above, function *MaxBudgetPaths*() is based on the Bellmann-Ford (BF) algorithm (Bellman, 1958; Ford Jr, 1956). BF finds the shortest paths from a given source node to all the other nodes in a graph. We adapt BF to find the *most beneficial* paths between a given source node and all other nodes in the graph —the paths that maximize the remaining budget. The extensions are explained below.

In BF, so-called negative cycles whose edges sum to a negative value, are considered invalid. If such a path is found, BF, exits with error. While in our case negative edges do not exist (recall that $c_{i,j} \geq 0$), energy gains at nodes can have the same effect because they can increase the budget. This can lead to so-called *beneficial* cycles, where the budget after performing the cycle is greater than the budget before performing the cycle. Unlike BF, in our case, beneficial cycles are allowed. In fact, letting the vehicle perform such cycles may be *necessary* in order to find

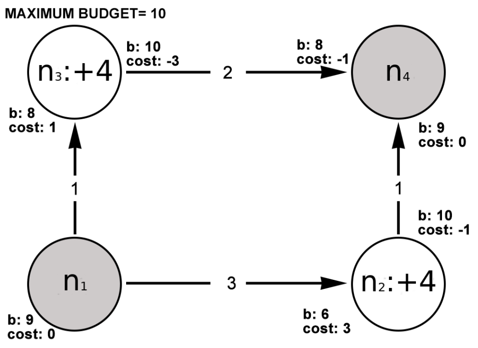


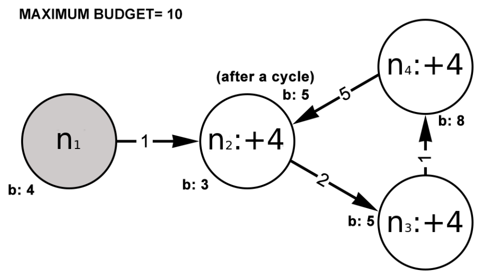
Figure 1: For start node n_1, destination node n_4 and initial budget 9, the path that minimize the cost (via n_3) is not the same as the path that maximizes the budget (via n_2). The costs are shown on the edges, the gains are shown inside each node. The maximum budget is 10.

a solution to the problem. Therefore, instead of exiting, we change the algorithm to accept such cycles and continue as usual. Note that the same cycle can be beneficial or not depending on the current budget.

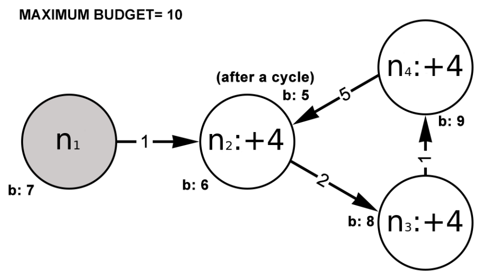
The usual operation of BF is to iteratively check and update the minimum distance for reaching a node. In our case, we check and update the best budget that can be achieved by reaching a node (provided there is sufficient budget to reach that node). It is important to note that minimizing the edge costs is not the same as maximizing the budget, see Figure 1. Also, due to the maximum budget constraint, the remaining budget along a path cannot be calculated simply by subtracting the sum of edge costs from the sum of the node gains.

The original BF associates each node with a predecessor that corresponds to the previous hop of the path with the smallest distance so far for that node. A path can be determined simply by back-tracking the predecessors from a destination node until the source node. In our case, we record the entire path. This is needed because, in the presence of cycles, the knowledge of the predecessor node is not sufficient to reconstruct the path that needs to be followed.

Finally, BF re-computes the distances and node predecessors $|\mathcal{N}|-1$ times, where $|\mathcal{N}|$ is the number of nodes in the graph. It is guaranteed that this suffices to find the shortest paths from the destination node to all nodes. This is not sufficient in our case, where beneficial cycles may exist and thus paths can be longer than $|\mathcal{N}|-1$ hops. For this reason, we run the algorithm as long as some updates still take place for some nodes. The algorithm always terminates despite the possibility of having beneficial cycles. Due to the budget constraint, the same cycle cannot remain beneficial for ever, see Figure 2. Therefore a cycle can be performed only a finite number of times. Note that



(a) Cycle is beneficial.



(b) Cycle is not beneficial.

Figure 2: The same cycle (n_2, n_3, n_4, n_2) can be beneficial (a) or non-beneficial (b), depending on the current budget. The costs are shown on the edges, the gains are shown inside each node. In both cases, the maximum budget is 10.

if there are no beneficial cycles, the algorithm works like the original version and finds the optimal path.

4.4 Complexity

The time complexity of *MaxBudgetPaths*() depends on the number of the nodes and the number of the edges in the graph, but also on the number of beneficial cycles and the number of the nodes involved in them. More concretely, assuming K beneficial cycles (including any iterations of the same cycle) and an average of M nodes in each cycle, the time complexity is $O((|\mathcal{N}|+K\times M)\times|\mathcal{E}|)$, where $|\mathcal{N}|$ and $|\mathcal{E}|$ is the total number of nodes and the total number of edges in the graph, respectively. In case no beneficial cycles exist, the time complexity is the same as the Bellman-Ford algorithm, $O(|\mathcal{N}|\times|\mathcal{E}|)$. So, the overall complexity of the algorithm (the *Skeleton*() function) is $O(((|\mathcal{N}|+K\times M)\times|\mathcal{E}|)\times R)$, where R is the number of replannings that are made during the trip of the vehicle.

5 EVALUATION

We have evaluated the proposed algorithm (we will refer to it as MaxBudget) for different scenarios. This section describes the system configuration and the scenarios that are investigated, and presents results from indicative experiments.

5.1 Reference Algorithms

To put the performance of the proposed algorithm into perspective, we use several algorithms as a reference. These are briefly described in the sequel.

As a first reference, we use an *oracle* algorithm that has a priori knowledge of the actual edge costs and node gains that will apply each time the vehicle were to cross an edge or visit a depot node. The algorithm works in three steps. In the first step, it finds the minimum cost path from each $n_i \in \mathcal{V}$ to each depot node $n_d \in \mathcal{D}$ as well as the minimum cost path from each depot node $n_d \in \mathcal{D}$ to each $n_i \in \mathcal{V}$, based on the actual (known) edge costs. Then, for each depot node n_d, it constructs "out" and "in" node clusters. Node $n_v \in \mathcal{V}$ belongs to the out cluster of n_d if it can be reached from n_d with the smallest cost compared to any other depot node. Node $n_v \in \mathcal{V}$ belongs to the in cluster of n_d if the cost for reaching n_d is the smallest among all other depot nodes. In the second step, the algorithm visits as many nodes of interest it can based on the current budget, before returning to a depot node to gain some energy (this is done based on the adapted Bellman-Ford code discussed earlier). This is repeated, until it is not possible to visit some of the remaining nodes of interest and then return to some depot node. Finally, in a third step, the algorithm tries to visit as many nodes as possible with the remaining budget (without returning to a depot node). This is done by running the Held-Karp algorithm (Held and Karp, 1962), which is an exact solution to the TSP. Initially, the algorithm is run for all remaining nodes of interest, let m. If no solution is found for m, the algorithm is run for $m-1$, and if no solution is found, then it is run for $m-2$ etc. If a solution is found for $m \geq 2$, the path returned by the Held-Karp algorithm is adopted. For $m = 1$, the algorithm simply picks the cheapest path to any node of interest, and marks the node as visited if the budget suffices to reach that node.

We also consider a simpler variant of the above algorithm, which uses static/fixed edge costs and depot node gains, equal to the mean of the random distributions, c^{mean} and g^{mean}. We refer to this as *TS-static* since it essentially solves the standard TSP, while taking into account the energy budget of the vehicle.

As yet another reference, we use an ant colony optimization (ACO) technique (Jones, 2005). This is a probabilistic approach for solving path finding problems, and has been used expensively in previous works that study vehicle routing problems and traveling salesman problems. More specifically, a number n of ants are generated for m generations and each ant has a specific budget. Each ant follows at first a randomly chosen route. In our case the ants leave the pheromone trail of their path if they visit all of the nodes of interest or their budget is exhausted but they have achieved the best coverage. The pheromone trail increases the probability of an ant of a future generation to choose that specific path. When an ant stops traveling, either because it has exhausted its budget or because it has visited all of the nodes of interest, it submits its coverage. The algorithm returns the path that achieves the maximum coverage. Note that ACO basically solves the same problem as TS-static, using an approximation heuristic. Several other algorithms have been proposed for different variants of the VRP, which could also be adapted to tackle the problem we study here in order to serve as additional references to assess the performance of the proposed algorithm. Such a comprehensive comparison is beyond the scope of this paper.

Finally, we experiment with a simpler version of MaxBudget, which performs the path planning step in normal mode only. As a consequence, the estimates for the edge costs and depot node gains are always the mean values of the respective distributions. If no feasible path is found, the algorithm terminates, instead of switching to the optimistic planning mode.

5.2 Setup and Key Parameters

The topology of the graph used in our experiments is a 10×10 grid, for a total of 100 nodes. Each node is connected to its horizontal and vertical neighbors via edges in both directions. The diameter d of the graph is 18 hops. The grid topology is quite representative for a terrain where the vehicle can move with a lot of freedom practically in any direction.

We set the depot nodes to 2% of the total number of nodes (there are 2 depot nodes). The depot nodes are chosen randomly, and remain the same throughout all experiments, at the positions $(2,5)$ and $(8,6)$ in the grid. In all experiments, the vehicle starts its journey from the node at the bottom-left corner of the grid (at the position $(0,0)$. Figure 3 illustrates the setup.

We perform experiments for different sets of nodes $\mathcal{V}_x \subset \mathcal{N}$ that need to be visited by the vehicle, where $x = \frac{|\mathcal{V}_x|}{|\mathcal{N}|}$. We consider four different cases: $x = 5\%, 10\%, 20\%$ and 30%. For each $\mathcal{V}_x$ case, we test the algorithms on three concrete sets, $\mathcal{V}_{x_k}, 1 \leq k \leq 3$. These are constructed in a random way, however we make sure that $\mathcal{V}_{5_k} \subset \mathcal{V}_{10_k} \subset \mathcal{V}_{20_k} \subset \mathcal{V}_{30_k}$ in order to have continuity across the different experiments.

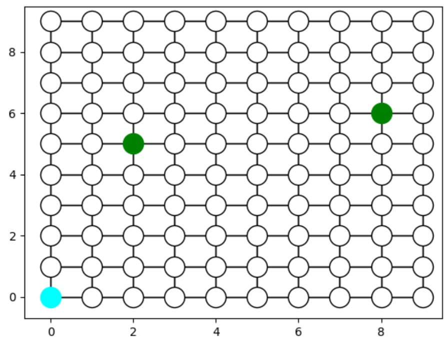


Figure 3: Graph topology used in the experiments. The two depot nodes are green. The start node is light blue.

With no loss of generality, we set the maximum budget limit $B_{max} = 1000$. We also set the initial budget of the vehicle $b = B_{max} = 1000$.

For the gain of depot nodes, we adopt a symmetrical double-truncated normal distribution with $g^{mean} = \frac{3}{4} \times B_{max}$. The rationale behind this choice is that when the vehicle reaches a depot node with marginally exhausted budget, on average we want it to be able to restore its budget to a significant degree (75% of B_{max}). The lower and upper bounds are set to $g^{min} = g^{mean} - \frac{g^{mean}}{3}$ and $g^{max} = g^{mean} + \frac{g^{mean}}{3}$, respectively. In all experiments, we use the same random distribution for all depot nodes..

The edge costs also follow such a random distribution with $c^{min} = c^{mean} - \frac{c^{mean}}{2}$ and $c^{max} = c^{mean} + \frac{c^{mean}}{2}$. In each experiment all edges follow the same cost distribution, but this varies across experiments. More specifically, we let $c^{mean} = \frac{B_{max}}{a}$, where a is the average number of hops that can be performed by the vehicle with a maximum initial budget, to which we refer as the *autonomy* of the vehicle. We investigate different degrees of autonomy, set as a function of the graph diameter: *high* autonomy d, *medium-high* $\frac{5}{6} \times d$, *medium-low* $\frac{2}{3} \times d$ and *low* $\frac{1}{2} \times d$. Higher autonomy enables the vehicle to travel further and visit a larger number of nodes before visiting a depot node to regain some energy.

For each of the above autonomy degrees, we produce 100 different edge cost and node gain scenarios. The values for the cost of each edge and the gain of each depot node are produced offline, and are stored in a file from where they are retrieved at runtime. Note that the cost of an edge changes each time the edge is crossed; in the scenario file, 50 different values are stored for each edge, which are retrieved in a round-robin fashion each time the vehicle crosses that particular edge. The same applies to the gain for the depot nodes. The edge costs and node gains of a scenario are a priori known only to the oracle algorithm.

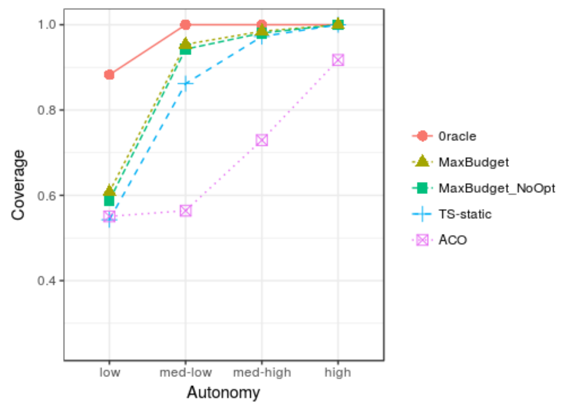
5.3 Results

For each configuration (combination of $\mathcal{V}_x$ and a) we perform different 300 runs (100 scenarios for each $\mathcal{V}_{x_1}, \mathcal{V}_{x_2}, \mathcal{V}_{x_3}$). We report the average *coverage* for each algorithm. In order for the ant-colony optimization method (*ACO*) to produce good results, in each run we use 10 ants and 200 generations of ants. For the thresholds used in the *Check*() function of the MaxBudget algorithm, we set $Threshold_{norm} = 10\%$ and $Threshold_{opt} = 4\%$. These values have been established via separate experiments, not reported here for brevity. Figure 4 shows the results.

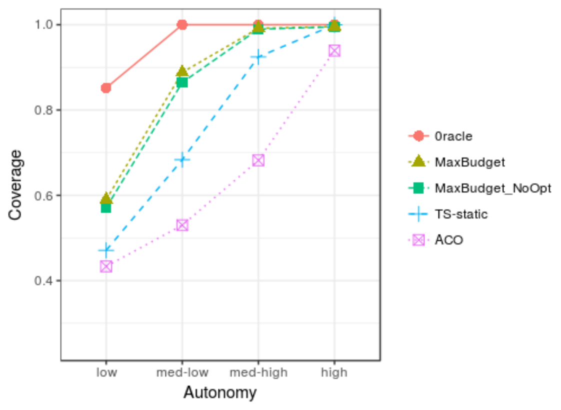
It can be seen that MaxBudget consistently outperforms TS-static. The difference is larger for mid-range autonomy degrees, in particular for medium-low where the overall average improvement is roughly 35% and 44% over $\mathcal{V}_{10}$, $\mathcal{V}_{20}$, $\mathcal{V}_{30}$, and becomes smaller for high but also for low autonomy. This can be explained as follows. On the one hand, when the autonomy is high, non-optimal decisions are more tolerable because the vehicle can perform a larger number of hops without refueling at a depot node. On the other hand, when the autonomy is low, it becomes much harder to find feasible paths, so the planning decisions of MaxBudget are not good enough to achieve a high coverage. For the mid-range autonomy degrees, MaxBudget achieves increasingly better coverage than TS-static as the number of target nodes increases. This is because MaxBudget plans "ahead", not only to visit the next node of interest, but also to keep the available energy budget of the vehicle as high as possible when the target is reached, so that this can be exploited for the next visit. Note that ACO always performs worse than TS-static. This is expected as ACO is an approximation of TS-static.

MaxBudget performs close to the oracle for high and medium-high autonomy. But it cannot match the oracle for lower degrees of autonomy, where the coverage of MaxBudget drops on average to about 87% and 67% to that of the oracle, for medium-low and low autonomy, respectively. Notably, even the oracle cannot achieve full coverage when the autonomy is low, in which case it is impossible to visit all nodes of interest. This also explains why the performance gap between MaxBudget and TS-static shrinks abruptly in all low autonomy scenarios, as discussed above.

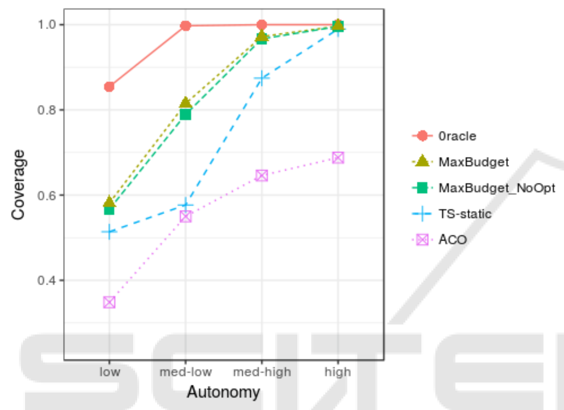
The simpler version of MaxBudget performs very close to the full-fledged version of the algorithm. The fallback to optimistic planning only brings a small benefit for lower degrees of autonomy, 2.5% on average for medium-low, and 3.1% for low. In these cases, optimism occasionally pays-off, allowing the vehicle to explore paths that would otherwise be re-



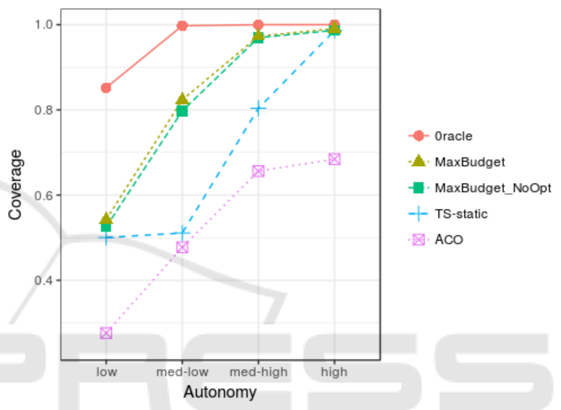
(a) 5 target nodes ($\mathcal{V}_5$)



(b) 10 target nodes ($\mathcal{V}_{10}$)



(c) 20 target nodes ($\mathcal{V}_{20}$)



(d) 30 target nodes ($\mathcal{V}_{30}$)

Figure 4: Coverage as a function of the number of nodes to visit and the autonomy of the vehicle.

jected, leading to a few more target node visits. When the autonomy is relatively high, optimistic planning does not seem to have any impact.

6 CONCLUSION

We have formulated a variant of the vehicle routing problem, where both the travel cost and the energy gain opportunities are stochastic. This is different than other VRP variants that have been studied in the literature. Also, we have proposed a heuristic algorithm that can be used to guide an autonomous vehicle in order to visit the nodes of interest; of course, the same algorithm can be used as a guidance tool for a human operator. The algorithm is designed for a general system model, and can be applied in different application scenarios. We have compared our algorithm with other algorithms, showing that it achieves good results, especially for system configurations where it is indeed feasible to visit all nodes of interest.

In the future, we wish to investigate different variants of the proposed algorithm in order to improve the coverage but also to reduce the runtime complexity (time wise and memory wise). The latter is important in case it is desirable to run the algorithm directly on the UV, which will typically have an embedded computing platform with limited memory and processing capacity. Furthermore, we wish to experiment with different graph topologies in combination with more informed values for the edge costs, the node gains and the budget constraint, based on concrete application scenarios. It is also interesting to consider how different existing algorithms could be adapted to tackle the variant of the VRP we study in this paper, and to see how well they perform compared to our algorithm.

ACKNOWLEDGMENTS

This research has been co–financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH -

CREATE - INNOVATE, project PV-Auto-Scout, code T1EDK-02435.

REFERENCES

Alonso, F., Alvarez, M. J., and Beasley, J. E. (2008). A tabu search algorithm for the periodic vehicle routing problem with multiple vehicle trips and accessibility restrictions. *Journal of the Operational Research Society*, 59(7):963–976.

Angelelli, E. and Speranza, M. G. (2002). The periodic vehicle routing problem with intermediate facilities. *European journal of Operational research*, 137(2):233–247.

Artmeier, A., Haselmayr, J., Leucker, M., and Sachenbacher, M. (2010). The optimal routing problem in the context of battery-powered electric vehicles. In *CPAIOR Workshop on Constraint Reasoning and Optimization for Computational Sustainability (CROCS)*.

Bellman, R. (1958). On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90.

Bent, R. W. and Van Hentenryck, P. (2004). Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, 52(6):977–987.

Bertsimas, D. J. (1992). A vehicle routing problem with stochastic demand. *Operations Research*, 40(3):574–585.

Cordeau, J.-F., Gendreau, M., and Laporte, G. (1997). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks: An International Journal*, 30(2):105–119.

Ehmke, J. F., Campbell, A. M., and Urban, T. L. (2015). Ensuring service levels in routing problems with time windows and stochastic travel times. *European Journal of Operational Research*, 240(2):539–550.

Erera, A. L., Morales, J. C., and Savelsbergh, M. (2010). The vehicle routing problem with stochastic demand and duration constraints. *Transportation Science*, 44(4):474–492.

Escobar, J. W., Linfati, R., Toth, P., and Baldoquin, M. G. (2014). A hybrid granular tabu search algorithm for the multi-depot vehicle routing problem. *Journal of heuristics*, 20(5):483–509.

Ford Jr, L. R. (1956). Network flow theory. Technical report, Rand Corp Santa Monica Ca.

Gaudioso, M. and Paletta, G. (1992). A heuristic for the periodic vehicle routing problem. *Transportation Science*, 26(2):86–92.

Gendreau, M., Laporte, G., and Séguin, R. (1995). An exact algorithm for the vehicle routing problem with stochastic demands and customers. *Transportation science*, 29(2):143–155.

Gendreau, M., Laporte, G., and Séguin, R. (1996). A tabu search heuristic for the vehicle routing problem with stochastic demands and customers. *Operations Research*, 44(3):469–477.

Held, M. and Karp, R. M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210.

Jones, K. O. (2005). Ant colony optimization, by marco dorgio and thomas stützle, a bradford book, the mit press, 2004, xiii+ 305 pp. with index, isbn: 0-262-04219-3, 475 references at the end.(hardback£ 25.95)-. *Robotica*, 23(6):815–815.

Juan, A., Faulin, J., Grasman, S., Riera, D., Marull, J., and Mendez, C. (2011). Using safety stocks and simulation to solve the vehicle routing problem with stochastic demands. *Transportation Research Part C: Emerging Technologies*, 19(5):751–765.

Kenyon, A. S. and Morton, D. P. (2003). Stochastic vehicle routing with random travel times. *Transportation Science*, 37(1):69–82.

Laporte, G., Louveaux, F., and Mercure, H. (1992). The vehicle routing problem with stochastic travel times. *Transportation science*, 26(3):161–170.

Laporte, G., Louveaux, F. V., and Van Hamme, L. (2002). An integer l-shaped algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research*, 50(3):415–423.

Las Fargeas, J., Hyun, B., Kabamba, P., and Girard, A. (2012). Persistent visitation with fuel constraints. *Procedia-Social and Behavioral Sciences*, 54:1037–1046.

Marinaki, M. and Marinakis, Y. (2016). A glowworm swarm optimization algorithm for the vehicle routing problem with stochastic demands. *Expert Systems with Applications*, 46:145–163.

Marinakis, Y., Iordanidou, G.-R., and Marinaki, M. (2013). Particle swarm optimization for the vehicle routing problem with stochastic demands. *Applied Soft Computing*, 13(4):1693–1704.

Mendoza, J. E., Rousseau, L.-M., and Villegas, J. G. (2016). A hybrid metaheuristic for the vehicle routing problem with stochastic demand and duration constraints. *Journal of Heuristics*, 22(4):539–566.

Mersheeva, V. (2015). *UAV Routing Problem for Area Monitoring in a Disaster Situation*. PhD thesis.

Miranda, D. M. and Conceição, S. V. (2016). The vehicle routing problem with hard time windows and stochastic travel and service time. *Expert Systems with Applications*, 64:104–116.

Rahimi-Vahed, A., Crainic, T. G., Gendreau, M., and Rei, W. (2013). A path relinking algorithm for a multi-depot periodic vehicle routing problem. *Journal of heuristics*, 19(3):497–524.

Rei, W., Gendreau, M., and Soriano, P. (2010). A hybrid monte carlo local branching algorithm for the single vehicle routing problem with stochastic demands. *Transportation Science*, 44(1):136–146.

Sachenbacher, M., Leucker, M., Artmeier, A., and Haselmayr, J. (2011). Efficient energy-optimal routing for electric vehicles. In *AAAI*, pages 1402–1407.

Secomandi, N. (2001). A rollout policy for the vehicle routing problem with stochastic demands. *Operations Research*, 49(5):796–802.

Taş, D., Dellaert, N., Van Woensel, T., and De Kok, T. (2013). Vehicle routing problem with stochastic travel times including soft time windows and service costs. *Computers & Operations Research*, 40(1):214–224.

Taş, D., Gendreau, M., Dellaert, N., Van Woensel, T., and De Kok, A. (2014). Vehicle routing with soft time windows and stochastic travel times: A column generation and branch-and-price solution approach. *European Journal of Operational Research*, 236(3):789–799.

Van Woensel, T., Kerbache, L., Peremans, H., and Vandaele, N. (2003). A vehicle routing problem with stochastic travel times. In *Fourth Aegean International Conference on Analysis of Manufacturing Systems location, Samos, Greece*.

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., and Rei, W. (2012). A hybrid genetic algorithm for multi-depot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624.