University of Thessaly

Master Thesis

Investigating the Dynamic Multi-Vehicle Routing Problem under Energy Constraints

Διερεύνηση του δυναμικού προβλήματος δρομολόγησης πολλών οχημάτων υπό περιορισμούς ενέργειας

Author:
Georgios Polychronis

Supervisors:
Spyros Lalis
Nikos Bellas
Christos Antonopoulos

A thesis submitted in fulfilment of the requirements for the degree of Master in the

Department of Electrical and Computer Engineering
University of Thessaly

Volos, July 7, 2020

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

ABSTRACT

The multiple vehicle routing problem (mVRP) concerns the scheduling of multiple vehicles so as to visit some locations of interest. Typical optimization objectives are the minimization of the total travel cost and the minimization of the time it takes to perform the required visits.

We study a dynamic version of mVRP where the travel costs are not a priori known and may vary at runtime. Moreover, we introduce certain energy-related constraints which make the problem more complex. On the one hand, vehicles have only finite energy reserves, which gradually diminish as the vehicles move between different locations. On the other hand, vehicles can gain some energy at specific depot locations. The objective is to visit all locations of interest as fast as possible without any vehicle exhausting its energy.

We propose a metaheuristic solution to build an initial schedule offline, and an online heuristic to dynamically update this schedule according to the real travel costs that are observed during the algorithm's execution. Both offline and online algorithms are based on the Large Neighbourhood Search (LNS) algorithm.

Our evaluation is twofold. On the one hand, we compare our metaheuristic for the initial schedule with a state of the art algorithm, showing that the proposed approach achieves better or in some cases equivalent results, but with significantly lower time complexity. On the other hand, we evaluate the online heuristic for different network topologies and degrees of vehicle autonomy. The results show that the online algorithm achieves significantly better results than the offline algorithm that produces a static schedule based on worst-case cost estimates.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

ΠΕΡΙΛΗΨΗ

Το πρόβλημα δρομολόγησης πολλών οχημάτων αφορά την εύρεση διαδρομών έτσι ώστε ένας στόλος οχημάτων να επισκεφτεί ένα σύνολο συγκεκριμένων περιοχών / θέσεων ενδιαφέροντος. Τυπικοί στόχοι για τον σχεδιασμό τέτοιων διαδρομών είναι η ελαχιστοποίηση του συνολικού κόστους μετακίνησης και η ελαχιστοποίηση του χρόνου επίσκεψης των περιοχών.

Στην παρούσα εργασία, μελετάμε μία δυναμική εκδοχή αυτού του προβλήματος, όπου τα κόστη μετακίνησης δεν είναι γνωστά εκ των προτέρων. Επιπλέον, τα οχήματα έχουν πεπερασμένη ενέργεια, η οποία μειώνεται όσο αυτά κινούνται, ενώ μπορούν να επαναχτήσουν ενέργεια σε συγκεκριμένους σταθμούς-κόμβους. Σε αυτή την περίπτωση, ο στόχος είναι να επισκεφτούν ένα σύνολο περιοχών όσο το δυνατόν πιο γρήγορα χωρίς κανένα όχημα να εξαντλήσει πλήρως την ενέργεια του .

Προτείνουμε ένα μετά-ευρετικό (metaheuristic) αλγόριθμο για να σχεδιάσουμε τις αρχικές διαδρομές των οχημάτων, και έναν ευρετικό (heuristic) αλγόριθμο που λειτουργεί σε πραγματικό χρόνο έτσι ώστε να ανανεώνει τις διαδρομές λαμβάνοντας υπόψη το πραγματικό κόστος της μετακίνησης που διαπιστώνεται την ώρα της εκτέλεσης . Οι αλγόριθμοι μας, βασίζονται στον αλγόριθμο Large Neighbourhood Search (LNS).

Η αξιολόγησή μας αποτελείται από δύο μέρη. Πρώτα συγκρίνουμε τον μετά-ευρετικό αλγόριθμό μας με ένα διεθνώς αναγνωρισμένο αλγόριθμο αιχμής, και παρατηρούμε ότι έχουμε καλύτερα ή και ίδια αποτελέσματα με μικρότερη χρονική πολυπλοκότητα. Στην συνέχεια αξιολογούμε τον αλγόριθμο πραγματικού χρόνου για διάφορες τοπολογίες και διαφορετικούς βαθμούς αυτονομίας των οχημάτων. Τα αποτελέσματα δείχνουν ότι ο αλγόριθμός μας πετυχαίνει σημαντικά καλύτερα αποτελέσματα από έναν στατικό αλγόριθμο που δημιουργεί μία φορά τις διαδρομές με βάση τις χειρότερες δυνατές τιμές των κοστών μετακίνησης.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

ACKNOWLEDGEMENTS

I would like to thank my supervisor Spyros Lalis, for all his valuable help, advices and support. I would also want to thank my family and friends for being supportive in this important journey.

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.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

CONTENTS

Abstract 2
Περίληψη 3
Acknowledgments 4
Contents 5
List of Figures 6
List of Tables 7
1 INTRODUCTION 8
2 RELATED WORK 10
2.1 Min-max multiple TSP 10
2.2 Dynamic capacitated multiple VRP 12
3 PROBLEM FORMULATION 14
3.1 Terrain 14
3.2 Energy capacity, costs and gains 14
3.3 Path feasibility 15
3.4 Path completion time and schedule makespan 15
3.5 Problem statement 15
4 OFFLINE ALGORITHM 17
4.1 TS-LNS algorithm 17
4.2 Large neighbourhood search 17
4.3 Node insertion 18
4.4 Node removal (path destruction) 20
4.5 Complexity 23
4.6 Evaluation of offline algorithm 24
4.6.1 Setup/configuration 24
4.6.2 TS-LNS-g vs. IWO for Euclidean problems . . . 25
4.6.3 TS-LNS-g vs.TS-LNS-e for Euclidean problems . 26
4.6.4 TS-LNS-g/e vs. IWO for general problems . . . 27
4.7 Adaptations for capacitaty constraints 29
5 ONLINE ALGORITHM 30
5.1 Representation of schedules and state information . . . 30
5.2 Cost estimation 30
5.3 Main loop 31
5.4 Reschedule heuristic 31
5.5 Evaluation of online algorithm 32
5.5.1 Experimental setup 32
5.5.2 Configurations of the online algorithm and reference 33
5.5.3 Results 34
6 CONCLUSIONS 38
Bibliography 39

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

LIST OF FIGURES

Figure 4-1 Node insertion sequence. Starting with the orange and blue paths, all possible insertion points are checked (only the best option for each path is shown). The node is finally inserted in the blue path, because the best node insertion in the orange path would increase its total cost to 14, whereas the cost of the blue path increases to 12 thus the cost of the worst / most expensive path remains 12. 21

Figure 4-2 Sequence of random destruction with a following repair for a solution with two paths. Destruction is performed by removing a total of six nodes. The nodes are all chosen randomly, and then they are reinserted in the solution using the node insertion method (insertion details not shown). 21

Figure 4-3 Sequence of proximity-based destruction with a following repair for a solution with two paths. Destruction is performed based on two randomly selected seed nodes, which happen to belong to different paths. For each seed, another two nodes are removed, chosen based on their proximity to the respective seed node. Note that the nodes are chosen based on their physical proximity to the seed, and may be part of a different path. In total, six nodes are removed, and are reinserted in the the solution using the node insertion method (insertion details not shown). 21

Figure 4-4 Speed-up of TS-LNS-g vs. IWO. 26

Figure 4-5 Speed-up of TS-LNS-e vs. TS-LNS-g. 28

Figure 5-1 Makespan of the online algorithm vs. the offline algorithm. 35

Figure 5-2 Rescheduling and schedule update frequency of the online algorithm (average over all search intensity configurations). 36

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

LIST OF TABLES

Table 4-1 Results of IWO. 25
Table 4-2 Results of TS-LNS-g. 25
Table 4-3 Results of TS-LNS-e. 27
Table 4-4 Solution cost of the algorithms for general graphs. 28

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

1

INTRODUCTION

Unmanned vehicles (UVs) are becoming more popular and have the potential to revolutionize several civilian application domains, such as agriculture, transport, surveillance. A common mission pattern is to use one or more UVs to visit certain points of interest in order to perform certain sensing/actuation tasks. When planning such missions, one must take into account the distances that need to be covered by the UVs in order to reach the different points of interest as well as the time and energy that is spent in travel.

This problem corresponds to the multiple vehicle routing problem (mVRP), which has been studied in many variants and for different optimization objectives. A particular challenging variant is the dynamic mVRP, where the system exhibits certain dynamics. For instance, new locations of interest may appear while the planned trip is still in progress, or the cost of travel for the locations to be visited may differ from what was initially assumed during planning.

In this work, we focus on a different variant of the dynamic mVRP. As in other problem versions that have been proposed in the literature, we assume that the travel cost can vary at runtime. Also, we let the travel cost correspond to the energy that is spent by a vehicle in order to move between two locations, and introduce special depot nodes that can be used to restore the energy reserves of the vehicles. However, we introduce a hard constraint regarding the energy reserves of the vehicles. As a result, when the vehicle exhausts its reserves it stops operating and can no longer contribute to the task of visiting nodes of interest.

This formulation closely models the problem of managing a fleet of UVs that use batteries or fuel to power their motors, and need to recharge/change batteries or refuel when running out of energy to be able to continue their mission. When a UV runs our of energy, it stops its mission and typically enters a safe mode, which may involve performing an emergency landing/parking action. When the UV enters the safe mode, it becomes unavalable and cannot participate in the mission at hand. To the best of our knowledge, this problem has not yet been investigated in the literature.

The problem can be decomposed into two distinct parts: the offline and the online part. In the offline part, the initial routes to be followed by the UVs have to be constructed. Then, in the online part, the system dynamics must be monitored in real time in order to adjust the routes

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

accordingly. More precisely, the routes are adjusted according to the real travel costs that are observed online. This thesis addresses both parts of the problem.

The main contributions of this thesis are: (i) We describe the above VRP problem in a formal way. (ii) We propose a metaheuristic approach to build the initial schedule based on worst-case cost estimates for the travel costs. (iii) We compare our offline approach with a state of the art algorithm, showing that it achieves equally or better results but is much faster in terms of computation time. (iv) We propose an online heuristic that adjusts the initial conservative schedule according to the dynamic outcomes of the system. (v) We evaluate the online algorithm for different topologies, degrees of uncertainty and vehicle autonomy.

The rest of the thesis is structured as follows. Chapter 2 discusses related work. Chapter 3 gives the problem formulation. Chapter 4 describes the proposed offline algorithm along with the comparison with the state of the art algorithm. Chapter 5 describes the proposed online algorithm with the respective evaluation. Finally, Chapter 6 concludes the thesis and points to some directions for further research.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

2

RELATED WORK

2.1 MIN-MAX MULTIPLE TSP

The multiple travelling salesman problem (mTSP) has been studied extensively and there is a wide literature on different solutions for it. Indicative surveys can be found in [1], [2], [3]. In this thesis, we focus on the single depot min-max variant of the problem, where all salesmen start from and have to return back to the same city, and where the objective is to minimize the longest / most costly path. Next, we give an overview of the various algorithms that have been proposed for this problem.

A popular method for tackling the mTSP are genetic algorithms. A genetic algorithm is basically a metaheuristic that is inspired by the principle of natural selection. Genetic algorithms start with an initial population, where each individual (chromosomes) represent a different solution to the problem. New solutions can be created either as the result of crossover operations between two different solutions or by performing mutation operations on an individual solution. Given that the population is not allowed to exceed a maximum number of solutions, only the fittest ones are typically selected to remain in the population while the rest are dropped. In [4], the two-part chromosome representation is proposed for the mTSP, where a solution is encoded using a n-length part that is the order of cities together with a m-length part that corresponds to the assignment of cities to the different salesmen. This encoding reduces the search space of the problem compared to other representations. This representation is also used by [5] to devise a new operator that generates a new solution by removing and reinserting genes (cities) at each salesman separately while modifying the second part of the representation in a random way. This approach improves the search component of the algorithm. The group-based genetic algorithmic principle was first introduced in [6] in combination with a two-part chromosome, where the first part encodes a solution of the problem and the second part includes the groups of the main part. The mutation and crossover operators are applied in the second part. Based on this approach, [7] proposed a grouping generic algorithm for the mTSP with a suitably adapted solution structure. Finally, [8] propose a grouping genetic algorithm with a different solution representation, where the solutions are represented as m different routes without any ordering or

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

mapping to a specific salesman. This makes it possible to reduce the redundant individuals in a population.

Several researchers have proposed nature-inspired methods. In [9], the authors propose two metaheuristic solutions for the min-max mTSP, an artificial bee colony algorithm and an invasive weed optimization algorithm (IWO). The former is an optimization technique that simulates the foraging behaviour of honey bees. On the other hand, IWO is a technique inspired by the weed colonization and distribution in the ecosystem. The IWO algorithm starts from an initial population of weeds, each representing a solution. Based on the fitness of the weeds they produce a number of seeds, which in their turn join the previous population. However, the number of weeds in the population must remain lower than an upper bound, so there is strong similarity to the genetic algorithms where the fittest individuals stay in the population. [10] and [11] approach the problem using an ant colony optimization algorithm. This is a probabilistic technique, simulating an ant colony and the pheromone used by ants to communicate with each other in order to find good paths toward a food source. Simulated ants move to a customer/city/node randomly, but with a higher chance to pick nodes with high pheromone trails. An approach based on neural networks is proposed in [12]. In [13], the authors hybridize the neural network approach with different metaheuristic techniques such as evolutionary algorithms and ant colony systems.

In [14] a memetic algorithm is proposed, based on variable neighbourhood descend. A memetic algorithm is a hybridization of a genetic algorithm with a local search procedure. Variable neighbourhood descend is a local search where multiple neighbourhoods of a solution are checked until a local minimum is reached. Each neighbourhood corresponds to a different mutation operator. [15] propose a general variable neighbourhood search. The general variable neighbourhood search is a metaheuristic that starts from an initial feasible solution (which at first is also the current solution), improves the current solution with a local search procedure (it usually uses multiple operators) and escapes local minimums with a shaking function.

[16] and [17] propose a tabu search, which is a metaheuristic technique to escape local minimums. The solution moves to the best neighbour solution and avoids cycling by keeping a list of forbidden moves, the so called tabu list. The authors in [16], also propose two exact algorithms to tackle the min-max problem. [18] propose a task allocation strategy to solve the mTSP. They present an algorithm that first partitions the graph to m subgraphs, and then solve the 1-TSP for each subgraph.

In [19], a comparison is made between the min-sum and min-max mTSP. It is shown that the length of the longest tour in the min-sum problem is at most m times longer than the length of the longest tour in the min-max problem with m vehicles, whereas the total cost is at most m times higher in the min-max than in the min-sum problem.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

The fact that min-sum mTSP solutions can be highly suboptimal for the min-max mTSP justifies the design of heuristic and metaheuristic algorithms specifically for the latter. But one should also keep in mind that a min-max solution may result to higher aggregated cost compared to a min-sum solution.

Finally, there are approaches for finding exact solutions to the mTSP, such as [16]. However, given the complexity of the problem, these are not practically applicable when the number of cities is large and there are many alternative paths that can be followed by the salesmen to visit them.

2.2 DYNAMIC CAPACITATED MULTIPLE VRP

The vehicle routing problem (VRP) and its multiple-vehicle variant (mVRP) have been studied for different dynamic aspects. Indicative surveys can be found in [20–22]. Most studies focus on the dynamic arrival of new targets, also commonly referred to as customers or customer requests. Some works consider dynamic travel times while others address the problem of known customers with uncertain service needs, which are revealed to the vehicles and the planning system when a customer is actually visited. Below, we provide an overview of indicative approaches that include the aspect of dynamic travel costs/times and are thus closer to the problem we tackle in this thesis.

In [23], the authors address the problem where new customer requests arrive online and there is uncertainty regarding the travel times. Their online approach re-constructs the routes of the vehicles by using the insertion heuristic, every time a new demand arrives or when the travel times change. They further improve the solution with the Or-opt algorithm [24], where a segment of a route (a number of consecutive customers) is moved to a different position.

A genetic algorithm is proposed for the same problem in [25], where the routes of the vehicles are periodically adjusted taking into account the new information that becomes available regarding changes in the travel times and new customer requests. Another genetic algorithm is presented in [26], which is used to produce the initial schedule as well as to perform any adaptations at a later point in time. In this case, rescheduling occurs when the estimated travel times for an edge is updated (once a vehicle reaches a customer) with the latest information from a dynamic traffic simulator.

In the approach described in [27], every vehicle is allowed to visit the next customer with some tolerance for delays beyond the expected arrival times. If this tolerance is exceeded, the customer is removed from the current route and is reinserted in the best position in the route of another vehicle. In a continuation of this work, the customer can also be added back to the route from where it was removed [28]. In both cases, the paths after the insertions are further improved with the

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

CROSS exchange algorithm, which is proposed in [29] for exchanging entire segments (consecutive customers) between routes. As a further extension, [30] considers a continuous tracking of the vehicle, where unexpected delays are detected earlier, before the tolerance is actually exhausted.

The authors of [31] also deal with the problem of newly arriving requests, in combination with varying travel speeds. In this case, the speed of the vehicle is known as soon as it starts its trip towards the next customer. The routes are adapted using four metaheuristics. The first two approaches are based on a dynamic Variable Neighbourhood Search (VNS) algorithm, based on static and stochastic information, respectively. Two additional metaheuristics are presented, using a Multiple Scenario and a Multiple Plan approach, where multiple solutions are pursued/maintained in parallel, based on static and stochastic costs, respectively. Both approaches use as a search procedure the VNS algorithm.

A wider range of dynamic events is considered in [32], each one possibly necessitating an adaptation of the scheduled routes. More specifically, these events are: late arrival of a vehicle at a customer late; timely arrival of the vehicle at a customer that is no longer valid; cancellation of a customer request; generation of a new customer request; break-down of a vehicle; vehicles being stuck in a traffic jam. Two operators are used to optimize the schedule. In the first case, individual customers are removed from a route and it is attempted to insert them in another route. In the second case, customers are exchanged in a pairwise fashion between routes. The customer insertions are done greedily. The authors also use a secondary objective function to drive the local search so that it focuses on more promising neighbourhoods.

The main difference of our work is that we place a hard constraint on the energy capacity of the vehicles. As a consequence, any dynamic changes in the travel costs have a direct impact not only on the optimality but also on the feasibility of the schedule. Also, any adaptations that are made to optimize the routes of the vehicles, must take this constraint into account in order to produce schedules that are actually feasible.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

3

PROBLEM FORMULATION

3.1 TERRAIN

We model the terrain where the mission takes place as a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is a set of nodes and $\mathcal{E}$ is a set of edges. Each node $n_i \in \mathcal{N}$ represents a target location that has to be visited by a vehicle. An edge $e_{i,j} \in \mathcal{E}$ represents the ability to move directly from node n_i to node n_j.

3.2 ENERGY CAPACITY, COSTS AND GAINS

The vehicles used to visit the nodes have a finite energy storage capacity B. This can be thought of as the capacity of a fuel tank or the capacity of a battery, depending on whether the vehicles are equipped with internal combustion engines or electrical motors. Let $b \leq B$ denote the current energy budget (reserves) of the vehicle during travel.

When the vehicle moves between two nodes, its motors consume some of the available energy budget. Let $c_{i,j}$ denote the cost for moving from n_i to n_j over $e_{i,j}$, also referred to as edge cost. Notably, the edge costs can be defined based on the Euclidean distance between the locations of the nodes (Euclidean problem), or they may not be directly related to the node's location (general problem). The latter makes it possible to flexibly model additional factors, like the quality, wideness, curviness, steepness of a road, which can have significant impact in travel time/cost. Edge costs can be symmetrical $c_{i,j} = c_{j,i}, \forall n_i, n_j \in \mathcal{N}$, or asymmetrical $\exists n_i, n_j \in \mathcal{N} : c_{i,j} \neq c_{j,i}$.

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 $brem = b - c_{i,j}$. If $brem \leq 0$, the vehicle exhausts its energy and becomes non-operational before reaching n_j.

The vehicle may increase its energy budget by gaining some energy at so-called depot nodes. One can think of a depot as a refuelling or recharging station. Let g_i be the energy that can be gained at n_i. If n_i is not a depot, $g_i = 0$.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

3.3 PATH FEASIBILITY

A path p is encoded as a sequence of nodes, where $p[k], 1 \leq k \leq |p|$ is the k^{th} node in p and $|p|$ is the number of nodes in the path. Equivalently, let $e_{i,j} \in p$ if $p[k] = n_i$ and $p[k+1] = n_j$ for $1 \leq k \leq |p| - 1$. Note that if p is non-empty then $|p| \geq 2$. Also, let $p[k1 : k2]$ denote the part of the p that starts from node $p[k1]$ and ends at node $p[k2]$.

Let $remb(b, p)$ denote the remaining budget of the vehicle if it starts with an initial budget b and travels along path p. This can be expressed as follows:

$$remb(b,p) = \begin{cases} min(B, b + g_{p[1]}) - c_{p[1],p[2]}, & |p| = 2 \\ remb(remb(b, p[1:2]), p[2:|p|]), & |p| > 2 \end{cases} \quad (1)$$

Namely, if p consists of a single hop, the remaining budget is equal to the initial budget plus the energy gain (if any) at the start node $p[1]$ less the edge cost for moving from $p[1]$ to $p[2]$. Note that the gain at the destination node $p[2]$ (if any) is not taken into account as this cannot be used to perform the hop in question. If p includes more than one hops, the remaining budget at the end of p is equal to 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. In this case too, the remaining budget at the end of the path does not include the gain at the destination node.

Based on the above, we say that a path p is feasible for an initial budget b, if $rem(b, p[1 : k]) > 0, 1 < k \leq |p|$. In other words, p is feasible if the vehicle will not exhaust its energy budget at any point along p. Also, let $nodes(p)$ denote the set of nodes that are part of p.

3.4 PATH COMPLETION TIME AND SCHEDULE MAKESPAN

The edge cost $c_{i,j}$ represents the energy spent by a vehicle in order to move from n_i to n_j. In addition, the edge cost is a proxy for the time that is required to perform this movement. Let $time()$ be a function that transforms edge costs to time. Then, the amount of time that is needed to complete path p is $d(p) = \sum_{e_{i,j} \in p} time(c_{i,j})$.

Let a schedule $s[]$ consist of M paths $s[m], 1 \leq m \leq M$. Assuming that each of these paths can be pursued in parallel, the makespan of s is $max_{m=1}^{M} d(s[m])$. In other words, it is equal to the completion time of the most costly path.

3.5 PROBLEM STATEMENT

Let there be M vehicles that can travel independently. The objective is to find a *feasible* schedule such that the vehicles will visit all the nodes of interest with the *smallest* possible cost/delay. More formally, $s[]$

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

should be chosen so that $\cup_{m=1}^{M} nodes(s[m]) = \mathcal{N}$ and $max_{m=1}^{M} d(s[m])$ is minimized.

In this work, we focus on a dynamic version of the problem. Namely, the edge/travel costs are unknown to the fleet scheduler and may vary in time. More specifically, we model the edge cost $c_{i,j}$ as a random variable over the range $[c_{i,j}^{min}..c_{i,j}^{max}]$ with an expected/mean value of $c_{i,j}^{mean}$.

We assume that depot nodes are known in advance and have infinite energy supply so that they can always refuel/recharge any vehicle to the maximum capacity B. In addition, we let all vehicles start from a depot node, and require that they also return back to a depot node. The makespan applies to the fully successful completion of the mission: all nodes have been visited and all vehicles have safely returned to a depot. Last but not least, we assume that there are no dead-ends, that is, the maximum edge costs $c_{i,j}^{max}$ and maximum capacity B are so that each node can be visited by a vehicle that starts from a depot and then has enough energy to return back to a depot.

The offline algorithm solves the classic mVRP for static/fixed costs. For the online problem, we modify the offline algorithm to take into account the vehicle's capacity constrains and the need to return to the depot node in order to refuel. In this case, the static/fixed edge costs used in the algorithm are equal to the worst case costs $c_{i,j}^{max}$. This guarantees that the produced schedule will be feasible irrespectively of the system dynamics. The online algorithm starts with the offline schedule and adjusts it during mission time as a function of dynamic changes in travel costs. The online algorithm also conservatively assumes worst case edge costs $c_{i,j}^{max}$. As a result, the schedule is optimized in terms of cost/time without violating feasibility.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

4

OFFLINE ALGORITHM

The proposed algorithm is a metaheuristic based on the principle of a tournament selection (TS) heuristic combined with a large neighbourhood search (LNS) method, originally proposed by [33]. Each solution is a schedule $s[]$ that includes a separate path $s[m]$ for each salesman $1 \leq m \leq M$. The representation of the individual paths is similar to [8], [9] and [14]. As a fitness function for a given solution, we use the inverse of the cost of the most expensive path. When comparing between two solutions, we prefer the one for which the fitness function returns the larger value.

In the sequel, we present the algorithm in a top-down fashion. We start with the main logic and then proceed to discuss the different functional components of the algorithm in more detail.

4.1 TS-LNS ALGORITHM

The starting point of the algorithm is the TS-LNS() function, described in Algorithm 1. It builds an initial population consisting of $MaxPopSize$ random solutions, and subsequently evolves this population in an iterative fashion.

In each iteration, the fittest solutions from the previous population are kept, decreasing the size of the population by a factor f. For each solution in the remaining population, a large neighbourhood search (LNS) is performed.

The iterations are repeated until the size of the population drops to/below a pre-specified threshold $MinPopSize$. The fittest of the remaining solutions is returned as the end result.

4.2 LARGE NEIGHBOURHOOD SEARCH

The large neighbourhood search procedure is described as a separate function LNS() in Algorithm 2. It takes a solution as input and returns as a result another solution, which is produced by trying out a number of so-called mutations. The number of mutations to be performed is a parameter, provided by the top-level TS-LNS() function.

Each mutation generates a new solution based on the best solution found up to that point, first by destroying it and then by repairing it.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

Algorithm 1 TS-LNS algorithm for M salesmen (option *rmvopt* sets the node removal method)

```
function TS-LNS(𝒩, M, rmvopt)
    nullsol ← ∅
    pop ← {}
    repeat M times
        nullsol = nullsol + {[n0, n0]}
    end repeat
    repeat MaxPopSize times
        pop ← pop + INSERT(nullsol, 𝒩)
    end repeat
    sort(pop)
    mut ← initLNSMutations()
    repeat
        pop ← getFittest(pop,size(pop) / f)
        for each sol ∈ pop do
            sol ← LNS(sol, mut, rmvopt)
        end for
        sort(pop)
        mut ← adjustLNSMutations(mut)
    until size(pop) ≤ MinPopSize
    return getFittest(pop, 1)
end function
```

The destruction operation involves the removal of some nodes from their assigned paths, and the repair operation reinserts those nodes to some (other) paths. If the new solution is fitter than the one that was used as a basis for the mutation, it is adopted as the best solution, which, in turn, will be used as a basis for the remaining mutations.

The node insertion and removal methods used to implement the mutations are discussed separately. Note that LNS() is designed to work using two different node removal methods. The selection is done via the *rvmopt* parameter, which is set by the user when invoking the top-level TS-LNS() function. In any case, the number of nodes to be removed and then reinserted in every mutation is decided randomly. However, the interval for this random pick is defined as a function of either $|\mathcal{N}|$ or $\sqrt{|\mathcal{N}|}$, depending on the node removal method used.

4.3 NODE INSERTION

The node insertion logic is given as a separate function INSERT() in Algorithm 3. This seems to be similar to the approach used in [9], however the authors only give a very informal (verbal) description for it.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

Algorithm 2 LNS method (option *rmvopt* sets the node removal method)

```
function LNS(sol, nofmutations, rmvopt)
    if rmvopt = RAND then
        lower, upper ← α * |𝒩|, β * |𝒩|
    else if rmvopt = PROXIMITY then
        lower, upper ← α * √|𝒩|, β * √|𝒩|
    end if
    best ← sol
    repeat nofmutations times
        rmv ← random(lower, upper)
        if rmvopt = RAND then
            tmp, free ← RmvR(best, rmv)
        else if rmvopt = PROXIMITY then
            seeds ← random(1, upper/10)
            tmp, free ← RmvP(best, rmv, seeds)
        end if
        new ← Insert(tmp, free)
        if fitness(new) > fitness(best) then
            best ← new
        end if
    end repeat
    return best
end function
```

Briefly, a node is picked randomly from the set of nodes to be incorporated in the solution, and an exhaustive search is performed to find the best path and the best position within that path for the node in question. The objective is for the insertion to minimize the cost of the worst (most costly) path in the solution. The current solution is updated accordingly. The process is repeated until all nodes have been added, and the resulting solution is returned.

The paths are considered in such an order so that the worst path with the largest cost will be checked last. This way, the worst path will be checked only if the node's insertion at any other path makes that path even more costly than the currently worst path. Also, if several insertion options result to the same worst-case cost, as a tie-break we pick the one that minimizes the cost increase for the path where the node is added. These optimizations are not shown in Algorithm 3, for brevity.

Figure 4-1 gives an indicative example for the insertion of a node in a solution that includes two paths (for two salesmen). In this case, the node is added to the blue path (on the right) because this does not increase the cost of the worst (most costly) orange path (on the left).

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

Also, the node is added in the blue path in a position that minimizes the cost increase.

The INSERT() function is invoked in two places. On the one hand, it is used in the top-level TS-LNS() function (Algorithm 1) to construct the initial population. In this case, different random solutions are generated by inserting each time the full set of nodes to an empty solution, thereby building a solution from scratch. On the other hand, INSERT() function is used in the LNS() function (Algorithm 2) in order to repair a solution, by adding-back the nodes that have been previously removed from it in the destruction process.

Algorithm 3 Node insertion method

```
function INSERT(sol, nodes)
    cur ← sol
    while nodes ≠ ∅ do
        minwcost ← ∞                          ▷ min cost of worst path
        n_j ←rmvNodeRandom(nodes)
        for each p ∈ cur (increasing cost order) do
            for each n_i ∈ p do
                p' ← addNode(p, n_i, n_j)
                wc ← worstCost(cur − p + p')
                if wc < minwcost then
                    bestp, bestp' ← p, p'
                    minwcost ← wc
                end if
            end for
        end for
        cur ← cur − bestp + bestp'
    end while
    return cur
end function
```

4.4 NODE REMOVAL (PATH DESTRUCTION)

For the destruction of a given solution, we support two different node removal methods, which are described in Algorithm 4.

The first method, shown in function RMVR(), removes a number of nodes from the given solution in a random way. Figure 4-2 gives an example of such random node removal, followed by the node insertion. This method is suitable for the general form of the problem, where edge costs do not necessarily reflect the Euclidean distance between the nodes.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

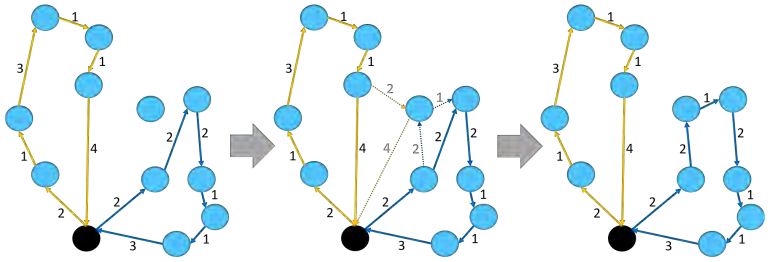


Figure 4-1: Node insertion sequence. Starting with the orange and blue paths, all possible insertion points are checked (only the best option for each path is shown). The node is finally inserted in the blue path, because the best node insertion in the orange path would increase its total cost to 14, whereas the cost of the blue path increases to 12 thus the cost of the worst / most expensive path remains 12.

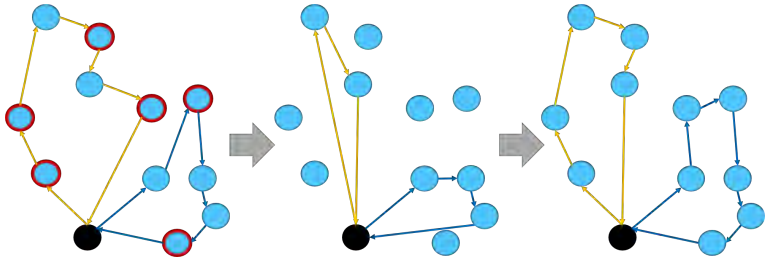


Figure 4-2: Sequence of random destruction with a following repair for a solution with two paths. Destruction is performed by removing a total of six nodes. The nodes are all chosen randomly, and then they are reinserted in the solution using the node insertion method (insertion details not shown).

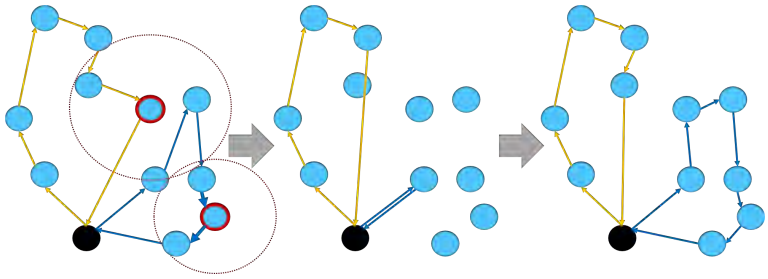


Figure 4-3: Sequence of proximity-based destruction with a following repair for a solution with two paths. Destruction is performed based on two randomly selected seed nodes, which happen to belong to different paths. For each seed, another two nodes are removed, chosen based on their proximity to the respective seed node. Note that the nodes are chosen based on their physical proximity to the seed, and may be part of a different path. In total, six nodes are removed, and are reinserted in the the solution using the node insertion method (insertion details not shown).

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

Algorithm 4 Node removal methods

```
function RmvR(sol, nofnodes)
    cur ← sol
    nodes ← pickRandom(𝒩, nofnodes)
    free ← ∅
    for each n ∈ nodes do
        p ← pathOf(cur, n)
        p ← rmvNode(p, n)
        free ← free + n
    end for
    return cur, free
end function

function RmvP(sol, nofnodes, nofseeds)
    cur ← sol
    nofnodes′ ← nofnodes/nofseeds
    seeds ← pickRandom(𝒩, nofseeds)
    free ← ∅
    for each s ∈ seeds do
        repeat nofnodes′ times
            n ← nearestNode(s)              ▷ incl. s itself
            p ← pathOf(cur, n)
            p ← rmvNode(p, n)
            free ← free + n
        end repeat
    end for
    return cur, free
end function
```

The second method, in function RmvP(), removes nodes in a more targeted way, assuming that the edge costs reflect the Euclidean distance between nodes. The rationale is to remove nodes that are in the proximity of so-called seed nodes (the number of seeds is an additional parameter of this method). The seed nodes are picked randomly, but the rest of the nodes to be removed are picked with reference to the seed nodes. More specifically, for each seed the method removes the nodes that are closer to it, based on the costs of the edges that connect the seed to other nodes. As a form of balancing, the total number of nodes to be removed is evenly distributed among the seed nodes. We refer to this method as the proximity-based method, as opposed to the fully random node removal method. Figure 4-3 gives an example of the proximity-based node removal, followed by node insertion.

The node removal functions are invoked from the LNS() function, for a randomly chosen number of nodes (Algorithm 2). When using the proximity-based node removal method, the number of seed nodes is also chosen in random but from a smaller interval so that the number if

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

seeds is guaranteed to be smaller than the total number of nodes to be removed. Note that RMVR() is always invoked for a number of nodes that is in the order of $|\mathcal{N}|$, whereas RMVP() is invoked for a number of nodes in the order of $\sqrt{|\mathcal{N}|}$. The rationale for removing (and then reinserting) fewer nodes when using the proximity-based method is that since in this case removal is more targeted, around relatively few seed nodes, removing a large number of nodes in the same neighbourhood would lead to an overly aggressive destruction of the current solution, which, in turn, may actually reduce the chances of finding a better assignment of those nodes to different paths.

4.5 COMPLEXITY

We discuss the complexity of the algorithm in a bottom-up fashion, starting from the node insertion and removal functions, then for the large neighbourhood search and finally for the entire algorithm. For convenience, we let $N = |\mathcal{N}|$.

The INSERT() function checks for every node to be added every possible insertion point in the paths of the current solution. Given that an exhaustive search is performed for each node, this procedure has complexity of $O(k \times N)$, where k is the number of nodes that need to be added.

The random node removal function RMVR() has $O(k)$ complexity, where k is the number of nodes to remove. The same holds for the proximity-based removal function RMVP(), where k is the total number of nodes to be removed (the seeds plus the nodes in proximity).

In each mutation that is performed within LNS(), the number k of nodes to be removed from and then reinserted into the solution is chosen randomly. However, recall that when using RMVR() then k is in the order of N, but when using RMVP() then k is in the order of $\sqrt{N}$ (see Algorithm 2). As a consequence, in the first case, the combined complexity of every mutation is $O(N) + O(N \times N)$ which translates to $O(N^2)$, whereas in the second case the complexity is $O(\sqrt{N}) + O(N \times \sqrt{N})$ or equivalently $O(N \times \sqrt{N})$.

Finally, we focus on the top-level TS-LNS() function (Algorithm 1). Note that the number of LNS() invocations decreases in each iteration as the size of the population becomes smaller, but the number of mutations that are performed in each invocation of LNS() is also adjusted. Assuming an average of K total LNS mutations in each top-level iteration, and a total number of I iterations, the overall complexity of the algorithm is $O(I \times K \times N^2)$ when using the random node removal method and $O(I \times K \times \sqrt{N} \times N)$ when using the proximity-based method. Note that, in turn, I depends on the size of the initial population $MaxPopSize$, the lower threshold for the population size $MinPopSize$ and the rate f at which the size of population is decreased in each iteration.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

4.6 EVALUATION OF OFFLINE ALGORITHM

We compare the proposed TS-LNS algorithm with a state of the art algorithm, the IWO algorithm proposed in [9]. A recent comparison that is presented in [14] shows that IWO is dominant in various benchmark problems. Next, we describe the experimental setup and configurations of the two algorithms, and then we discuss the results obtained through experiments on both Euclidean and general problems/graphs.

4.6.1 *Setup/configuration*

We implement the IWO algorithm and the TS-LNS algorithm in Python 3.5.2, and run them on a Ubuntu 16.04 distribution in a VM using Vmware on top of Windows 10. The machine we use to run the experiments has an Intel i7-8550u CPU at 1.8GHz-4.0GHz and 8GB of RAM. The CPU has 4 physical cores with hyperthreading support for a total of 8 threads. The VM is configured to have 6 virtual cores (mapped to 6 threads) and 4GB of RAM.

We configure the IWO algorithm to perform 100 top-level iterations. Each such iteration leads to 300 node removal/insertion operations, yielding a total of 30000 operations.

The TS-LNS algorithm is configured to run for $MaxPopSize = 100$ and $MinPopSize = 6$. The rate of population reduction is set to $f = 2$, so in every iteration we keep only half of the population, the fittest 50% of the solutions. In this configuration, TS-LNS performs four top-level iterations.

Regarding the number of LNS mutations that are performed on each solution of the current population, we initially start with 200 LNS mutations, increasing this number by 200 in each iteration. The rationale is for the search effort to be smaller when the number of solutions is large, and increase as the number of solutions gets smaller. More specifically, 200 mutations are performed for each of the fittest 50 random solutions in the first iteration, 400 LNS mutations are performed for each of the fittest 25 solutions in the second iteration, and 600 LNS mutations are performed for each of the fittest 12 solutions in the third iteration. For the remaining 6 solutions, as an exception, only 467 LNS mutations are performed in order to have a total of 30002 node removal/insertion operations, on par with the IWO algorithm.

We refer to TS-LNS with the random node removal method as TS-LNS-g given that this configuration is more suitable for the general form of mTSP. In this case, we set $\alpha = 0.2$ and $\beta = 0.4$, so that the interval that is used to randomly pick the number of nodes to be removed is $[0.2 * N..0.4 * N]$. TS-LNS with the proximity-based node removal method is referred to as TS-LNS-e as this is more suitable for the Euclidean form of mTSP. When using this configuration, we set $\alpha = 1.0$ and $\beta = 4.0$, so the respective interval is $[\sqrt{N}..4 * \sqrt{N}]$.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

Table 4-1: Results of IWO.

Benchmark	M	Cost Avg	Cost StD	Exec (s)
*eil*51	3	159.56	0	16.58
	5	118.13	0	18.71
	10	112.08	0	22.86
*kroB*100	3	8503.41	22.73	56.69
	5	7008.01	15.06	63.61
	10	6700.04	0	75.57
*ch*150	3	2455.21	20.66	124.20
	5	1768.66	8.86	135.26
	10	1554.64	0	162.98
*lin*318	3	17138.27	161.83	593.22
	5	12379.09	68.36	651.16
	10	9816.99	20.62	751.88

Table 4-2: Results of TS-LNS-g.

Benchmark	M	Cost Avg	Cost StD	Exec (s)
*eil*51	3	159.56	0	13.39
	5	118.13	0	15.03
	10	112.08	0	18.13
*kroB*100	3	8497.79	19.69	43.98
	5	6982.58	17.05	48.02
	10	6700.04	0	59.40
*ch*150	3	2446.41	15.03	97.41
	5	1764.80	8.68	107.05
	10	1554.64	0	128.31
*lin*318	3	16556.07	109.40	451.50
	5	11701.93	53.67	486.72
	10	9731.16	0	581.72

4.6.2 *TS-LNS-g vs. IWO for Euclidean problems*

In a first set of experiments, we compare TS-LNS-g against IWO. As input graphs, we use the *eil*51, *kroB*100, *ch*150 and *lin*318 benchmarks from the TSPLIB suite [34]. These correspond to Euclidean problems for graphs with $51, 100, 150$ and 318 nodes, respectively. For each benchmark, we run the algorithms for a team of 3, 5 and 10 salesmen. The results for IWO are given in Table 4-1 and for TS-LNS-g in Table 4-2. We report the averages over 20 runs.

As fas as the quality of the solutions is concerned, TS-LNS-g produces the same solutions as IWO for the small problem with 51 nodes, and equal or better solutions for the larger problems. More specifically, the solutions of TS-LNS-g are on average marginally better, by 0.1% and 0.2%, than those of IWO for 100 and 150 nodes, respectively. For the problem with 318 nodes, the solution of TS-LNS-g is on average 3.2% better than IWO. The standard deviation is small in all cases, with TS-LNS-g having an even smaller deviation than IWO.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

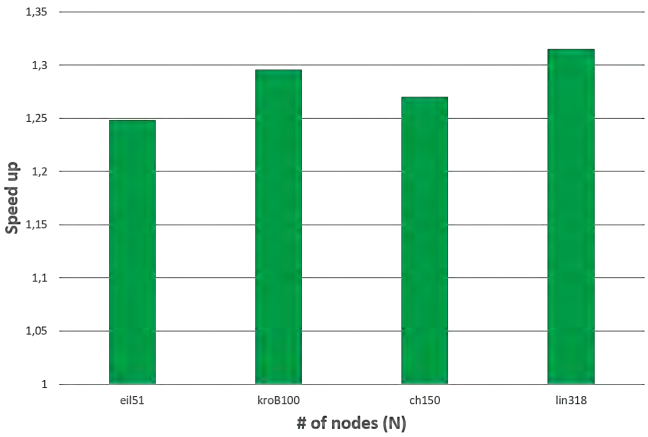


Figure 4-4: Speed-up of TS-LNS-g vs. IWO.

We note that both algorithms manage to find the optimal solution in the problems with 51, 100 and 150 nodes with 10 salesmen. In all these cases, the cost of the solution is indeed equal to twice the cost of the edge that connects the depot node and the node that is farthest away from it (it is impossible for the worst route to have a lower cost). Moreover, TS-LNS-g also finds optimal solution for the problem with 318 nodes and 10 salesmen.

At the same time, TS-LNS-g is considerably faster than IWO, as shown in Figure 4-4. The average speed-up is equal to $1.25x$, $1.30x$, $1.27x$ and $1.31x$ for the benchmarks with 51, 100, 150 and 318 nodes, respectively, at an overall average of $1.28x$. Note that both algorithms perform the same number of mutations, with each mutation (node removal and reinsertion operation) having $O(N^2)$ complexity. However, the mutations of TS-LNS-g involve fewer nodes, on average $0.3 \times N$ vs. $0.5 \times N$ in IWO, leading to a smaller total number of node removal/reinsertions. This reduction in the search space does not seem to have any impact on the solution quality of TS-LNS-g.

4.6.3 *TS-LNS-g vs.TS-LNS-e for Euclidean problems*

In a second series of experiments, we run TS-LNS-e for the same same set of benchmarks as above. Recall that TS-LNS-e is designed to work well specifically for Euclidean problems. Table 4-3 shows the results. Again, the averages over 20 runs are reported.

We observe that TS-LNS-e produces the same results as TS-LNS-g and IWO for the problems with 51 nodes, and finds better solutions for all larger problems. Namely, for the problems with 100, 150 and 318 nodes, the solutions of TS-LNS-e are on average 0.1%, 0.7% and 1.5% better than TS-LNS-g, and roughly 0.3%, 0.9% and 4.6% than the solutions found by IWO.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

Table 4-3: Results of TS-LNS-e.

Benchmark	M	Cost Avg	Cost StD	Exec (s)
$eil51$	3	159.56	0	13.98
	5	118.13	0	15.87
	10	112.08	0	19.41
$kroB100$	3	8482.50	5.88	34.43
	5	6965.85	17.56	38.81
	10	6700.04	0	46.98
$ch150$	3	2416.55	13.47	65.09
	5	1747.36	5.66	72.32
	10	1554.64	0	86.62
$lin318$	3	16113.78	46.12	215.62
	5	11500.98	43.78	236.42
	10	9731.16	0	281.31

The standard deviation of TS-LNS-e is less or equal to that of TS-LNS-g in most of the problems. As an exception, for 100 nodes and 5 salesmen the deviation of TS-LNS-e is slightly larger than TS-LNS-g for the same problem, but it is also higher than that of TS-LNS-e itself for the problem with 100 nodes and 3 salesmen. This could be an indication that it might be beneficial to take into account the number of salesmen when deciding the number of nodes to be removed/reinserted in each mutation.

Importantly, TS-LNS-e is also much faster than TS-LNS-g for bigger problem sizes, as shown in Figure 4-5. The average speed-up is $1.26x$, $1.49x$ and $2.07x$, for the benchmarks with 100, 150 and 318 nodes, respectively. This performance is even more impressive if compared with IWO, yielding a speed-up of $1.64x$, $1.89x$ and $2.71x$, for these benchmarks. This significant improvement is due to the lower $O(N \times \sqrt{N})$ complexity of TS-LNS-e compared to $O(N^2)$ for TS-LNS-g and IWO.

Note, however, that TS-LNS-e is somewhat slower than TS-LNS-g for the smallest problem with 51 nodes. The reason is that, in this particular case, the interval $[\sqrt{N}..4 * \sqrt{N}]$ used in TS-LNS-e to decide the number of nodes to be removed/reinserted in each mutation, becomes $[7..29]$ and has a much larger upper bound than the interval $[0.2 * N..0.4 * N]$ used in TS-LNS-g, which is $[10..20]$. As a result, in each mutation, TS-LNS-e removes/reinserts on average a larger number of nodes than TS-LNS-g.

4.6.4 *TS-LNS-g/e vs. IWO for general problems*

In a last series of experiments, we evaluate the algorithms using as input more general graphs. For this purpose we use three different benchmarks of the TSPLIB suite [34], $kro124p$, $gr120$ and $ftv170$ with 100, 120 and 171 nodes, respectively. In all cases, the edge costs are

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

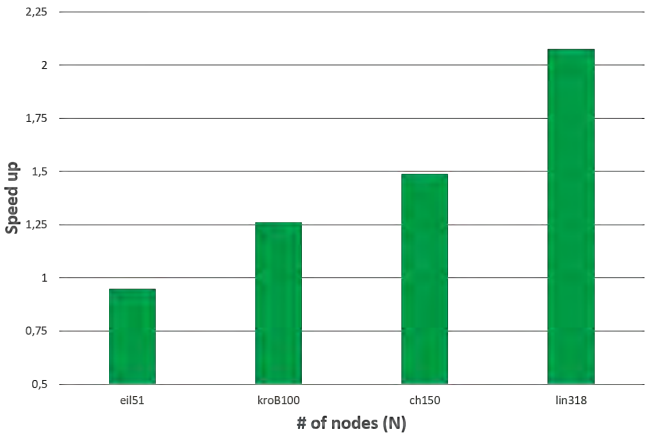


Figure 4-5: Speed-up of TS-LNS-e vs. TS-LNS-g.

Table 4-4: Solution cost of the algorithms for general graphs.

Benchmark	M	IWO	TS-LNS-g	TS-LNS-e
$kro124p$	3	13470.05	13539.90	13313.2
	5	9137.3	9157.15	8990.55
	10	6419.4	6343.45	6322.45
$gr120$	3	2614.15	2604.95	2580.50
	5	1834.25	1823.0	1812.30
	10	1558.0	1554.40	1555.35
$ftv170$	3	1026.75	1026.1	986.65
	5	688.85	673.63	654.15
	10	447.70	432.37	427.53

non-Euclidean. In $gr120$ the edge/travel costs are symmetrical, whereas in $kro124p$ and $ftv170$ they are asymmetrical. In order for TS-LNS-e to work on graphs with asymmetrical costs, the nodes to be removed around a seed are chosen based on the two-way trip cost between them.

Table 4-4 reports the cost of the solutions that are generated by each algorithm, averaged over 20 runs. As in the previous experiments, the standard deviation is small. Also, the previous observations regarding the execution speed of the algorithm still hold. This information is not reported here, for brevity.

It can be seen that both TS-LNS variants once again produce solutions that are close and most of the times even better than those of IWO, yielding an improvement of up to 3.4% for TS-LNS-g or even 5.0% for TS-LNS-e. In fact, TS-LNS-e always generates better solutions than IWO. We attribute this (somewhat surprisingly) good performance to the fact that these general benchmark graphs are non-random and specifically in the case of gr120 and ftv170 they are based on real-world routes and travel costs. So, even though the costs are not a direct function of the straight-line Euclidean distances, the most significant costs can still have a strong affinity to them.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

4.7 ADAPTATIONS FOR CAPACITATY CONSTRAINTS

The TN-LS algorithm can be adapted in a straightforward way for vehicles that have capacity/energy constraints and need to reload/refuel at a depot node. More specifically, the feasibility of a node insertion must be checked before adding the node to a path. The insertion is considered as a candidate only if it does not violate the capacity constraint of the corresponding vehicle. If a node cannot be inserted in any of the M paths, a new path is created for visiting that node, and this is assigned to the vehicle with the smaller makespan. As a result, each vehicle m may have more than one paths assigned to it, and each entry $s[m]$ in the schedule contains a list of paths(as opposedto just one path). A new path is always added at the end of the vehicle's path list.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

5

ONLINE ALGORITHM

5.1 REPRESENTATION OF SCHEDULES AND STATE INFORMATION

We represent the schedule and state of a given vehicle as a tuple $(paths[], remb_{est}[], remb_{upd})$, where $paths[]$ is the list of paths that have been assigned to it. Each path starts from a depot node and end at a depot node, thus the vehicles always start a new path with the maximum energy budget B. Note that the end depot node of $paths[i]$ is the start depot node of $paths[i+1]$. The estimated remaining budget after having completed $paths[i]$ is stored in $remb_{est}[i] = remb(B, paths[i])$.

The path that is currently followed by a vehicle is always the first path in the list $paths[1]$. When this path is completed, it is removed from the list, and the vehicle begins to follow the next (first) path in the list. Each time the vehicle performs the next hop along the current path, it experiences the actual cost for this movement. This can be different than what was estimated when that path was planned. To keep track of this deviation, the updated estimate for the remaining budget at the end of the current path is stored in $remb_{upd}$.

The complete schedule and state information $s[]$ is as a list of M tuples, where $s[m], 1 \leq m \leq M$ corresponds to the m^{th} vehicle that is used to visit the nodes of interest.

5.2 COST ESTIMATION

There are different ways to estimate the remaining budget when a path is planned or it is updated. One policy could be to use the average / expected edge costs. Another policy would be to use the maximum possible (worst-case) costs.

Optimistic approaches may lead to better schedules with a smaller makespan, but they also introduce the risk of vehicles exhausting their energy budget and becoming non-operational. This, in turn, can have a very negative impact to the mission, and lead to far worse results. Conservative approaches may generate less optimal schedules but reduce the probability of some vehicle becoming non-operational.

In this work, we explore the most conservative approach. Namely, we estimate the remaining budgets based on the maximum possible (worst-

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

case) edge costs. This way it is guaranteed that the generated schedules are *always* feasible ($s[m].remb_{est}[i] > 0, \forall m, i$). In turn, this guarantees that all nodes will be visited and all vehicles will manage to return to a depot. Moreover, this means that any cost deviations concerning the current path of a vehicle will result in greater energy reserves than the ones estimated when that path was planned ($s[m].remb_{upd} > s[m].remb_{est}[1]$).

5.3 MAIN LOOP

The algorithm starts from an initial (feasible) schedule $s[]$. Then, the vehicles are instructed to commence the mission, and start following the paths that have been assigned to them in the order in which they appear in their path lists.

The high-level logic of the algorithm is shown in the form of pseudocode in Algorithm 5. Each iteration corresponds to the attempt of one or more vehicles to perform the next hop in their current path. If the actual cost for this movement is different than what was estimated when the path was planned, the state of the vehicle is updated, adjusting $s[m].remb_{upd}$ accordingly.

If this wasn't the last hop along the current path, it is checked whether there is a large relative deviation between the updated remaining budget for that path $s[m].rem_{upd}$ and the corresponding estimate $s[m].rem_{est}[1]$ that was made when the path was planned. If such a deviation exceeds a threshold, a flag is set. The threshold for this rescheduling is configurable.

The flag is checked after the state of all vehicles has been updated. If it is set, this triggers a rescheduling attempt. The heuristic for this is abstracted via function LNS() and is discussed in more detail in the sequel. The rescheduling may change parts of or even the entire schedule $s[]$, adjusting the paths and state of the vehicles accordingly.

The algorithm ends when all vehicles have completed the paths that were assigned to them. Recall that paths are planned in a conservative way, thus the schedules generated are always feasible and no vehicle will run out of energy during travel.

5.4 RESCHEDULE HEURISTIC

The reschedule optimization technique follows the principle of Large Neighbourhood Search (LNS), which was originally proposed in [33]. Its main advantage is that it can explore larger neighbourhoods than local search algorithms. Additionally, with the right removal/reinsertion functions, the search can check promising neighbourhoods in a fast way.

In our case, we use LNS to find a new schedule by performing a number of mutations to the current schedule. The heuristic works along the lines of the LNS() function discussed in the previous chapter (Algorithm 2).

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

Algorithm 5 Online scheduling algorithm.

```
function SCHEDULER(s, threshold, nofmutations)
    while ∃m : s[m].paths[1] ≠ ∅ do
        reschedule ← false
        for each m performed k^th hop along its path do
            i, j ← s[m].paths[1][k], s[m].paths[1][k + 1]
            s[m].rem_upd ← s[m].rem_upd + (c^max_{i,j} − c_{i,j})
            if k ≠ |s[m].paths[1]| − 1 then
                dev ← s[m].remb_upd − s[m].rem_est[1]
                if dev/s[m].rem_est[1] > threshold then
                    reschedule ← true
                end if
            else
                s[m].paths[].pop()                    ▷ remove first path
            end if
        end for
        if reschedule then
            s ← LNS(s, nofmutations)
        end if
    end while
end function
```

As a node removal function, we use RMVP() (Algorithm 4). Node insertion is done as usual, via function INSERT() (Algorithm 3, with the adaptations described in Section 4.7 in order to deal with the capacity constaints. After each node removal/insertion, the internal schedule information is updated accordingly.

5.5 EVALUATION OF ONLINE ALGORITHM

5.5.1 *Experimental setup*

We evaluate the proposed algorithm using simulations. We conduct our experiments for a 11×11 grid of 121 nodes. The nodes represent the geographical locations of an area that has to be scanned by a team of vehicles exhaustively, by visiting all nodes. We assume that the vehicles can freely move from any node n_i to any other node n_j in a straight line. This is typically the case for aerial unmanned vehicles (UAVs) that scan a large area from a relatively high altitude that keeps them safely above trees and power lines. Thus, there is an edge between every two nodes.

The cost of each edge $e_{i,j}$ represents the cost for moving from n_i to n_j. This cost is *not known* with certainty, but randomly varies following a uniform distribution $[c^{min}_{i,j}..c^{max}_{i,j}]$. The actual cost is discovered only at runtime, when a vehicle attempts to perform that movement. The

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

upper and lower bounds of the cost distribution are a function of the Euclidean distance between the nodes' positions. More specifically, $c_{i,j}^{min} = \alpha \times dist(n_i, n_j)$ and $c_{i,j}^{max} = \beta \times dist(n_i, n_j)$. We perform experiments for *small uncertainty* where $\alpha = 0.5$ and $\beta = 1$, and *large uncertainty* where $\alpha = 0.25$ and $\beta = 1$. Without loss of generality, we let the amount of time that is needed for a vehicle to cover a distance be a linear function of the respective edge cost.

To capture the uncertainty of the travel cost, we create 50 different scenarios. In each scenario, the cost of every edge is randomly chosen based on the respective random distribution. The actual costs are discovered only when the vehicles cross the respective edges. Given that each node has to be visited once and that the graph is fully connected, every edge is traversed at most once, thus it suffices to have only one randomly chosen cost value for each edge.

In our experiments we have a single depot node, and investigate two different scenarios regarding its location. In the *peripheral depot* scenario, the depot node is located at one of the corners of the grid. In the *center depot* scenario, the depot node is located at the center of the grid.

We use a fleet of $M = 3$ vehicles, which all have the same energy capacity. The capacity of the vehicles is set to the worst-case round-trip cost, between the depot node and the node that is farthest away from it. This ensures that it is indeed possible to visit all nodes, even if the edge costs turn out to be the maximum possible.

5.5.2 *Configurations of the online algorithm and reference*

We test the online algorithm for various configurations. On the one hand, we experiment with four rescheduling thresholds 0.2, 0.1, 0.05 and 0.0, referred to as *conservative*, *moderate*, *aggressive* and *always* configurations, respectively. Recall that lower thresholds lead to more frequent/earlier rescheduling. Note that in the *always* configuration the algorithm reschedules whenever a deviation is experienced, irrespectively of how significant this is relative to the path cost estimates. On the other hand, we vary the number of iterations performed in the LNS component of the algorithm, from $25, 50$ to 100 iterations. We refer to this as *low*, *medium* and *high* search intensity, respectively.

As a reference, we use static schedules that are produced by the offline algorithm that was presented in Chapter 4, which provides good results for the min-max mTSP, with the adaptations described in Section 4.7 to tackle the vehicle capacity and path feasibility constraints. The offline schedule is generated based on the worst-case cost of each edge, and thus is guaranteed to be feasible (no vehicle will ever run out of energy) independently of the actual costs that will be experienced by the vehicles during the mission. The offline schedules are also used as the initial schedules for the online algorithm.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

5.5.3 *Results*

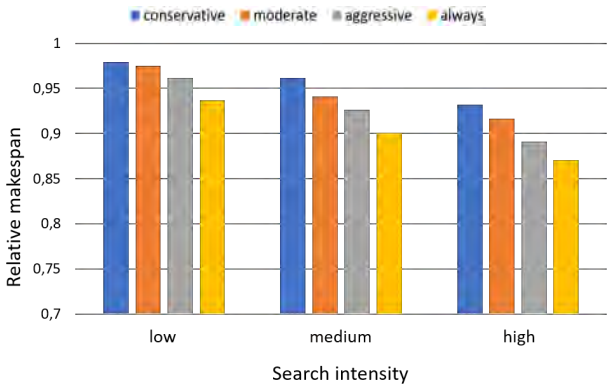
We run each configuration of the online algorithm 5 times for each of the 50 edge cost scenarios (for each depot and uncertainty scenario). Figure 5-1 shows the average makespan of the different configurations as a percentage of the makespan of the offline algorithm. At the top, Figure 5-1a and Figure 5-1b show the results for peripheral depot scenario, while Figure 5-1c and Figure 5-1d at the bottom show the central node scenario. The figures on the left show the results for the small uncertainty scenario, and the ones on the right for the large uncertainty scenario. For the two "extremes" of our experimental study, the improvement varies from 2% for the most conservative rescheduling and low search intensity configuration in the peripheral depot node scenario with small uncertainty, up to nearly 20% for the most aggressive rescheduling and highest search intensity configuration in the central depot node scenario with large uncertainty.

Increasingly better results are achieved when the online algorithm reschedules more aggressively (more often). This is because the paths are planned based on worst-case cost estimates, so any deviations lead to higher reserves in the vehicles. The sooner one replans during the mission, the better are the chances of a significant optimization in the paths of the vehicles, as there is still a larger number of unvisited nodes. More conservative (less frequent) rescheduling misses such opportunities. Even though the accumulated reserves are larger in this case, the rescheduling takes place after having visited several nodes, so there is actually less room for major optimizations. Also, as expected, better results are achieved for the more intensive search configurations.

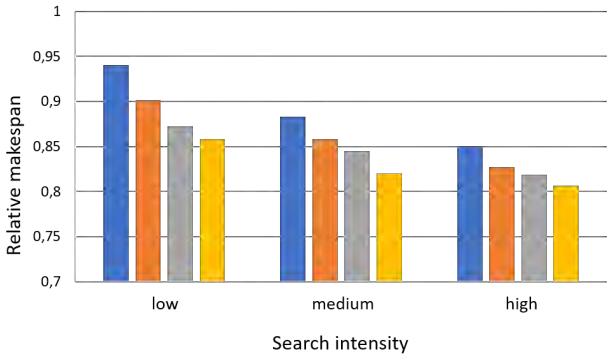
The improvements are more significant when the uncertainty about the edge costs is larger. Given that path planning is based on worst-case estimates, any scheduling decisions become increasingly sub-optimal under larger uncertainty, and this can only be repaired by rescheduling more often.

Another observation is that the improvements are more significant in the central depot scenario for the conservative, moderate and aggressive reschedule policies. The reason is that the paths are shorter so the respective worst-case estimates are smaller than in the peripheral scenario. As a consequence, the same absolute cost deviations meet the respective thresholds more often, leading to more frequent rescheduling attempts. Notably, an equally good improvement is achieved in both depot scenarios with the always reschedule policy. In this case, a rescheduling attempt is done at the slightest cost deviation (even if minor compared to the estimated path costs), thus rescheduling is performed equally frequently in both scenarios.

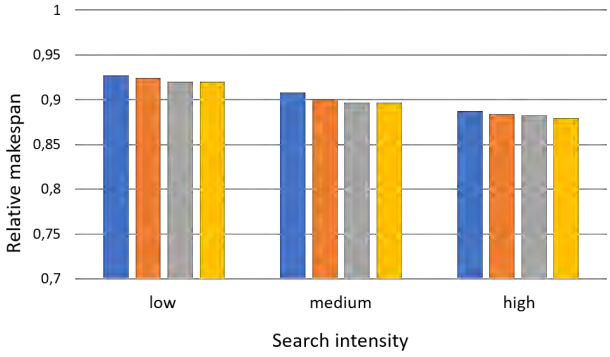
Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159



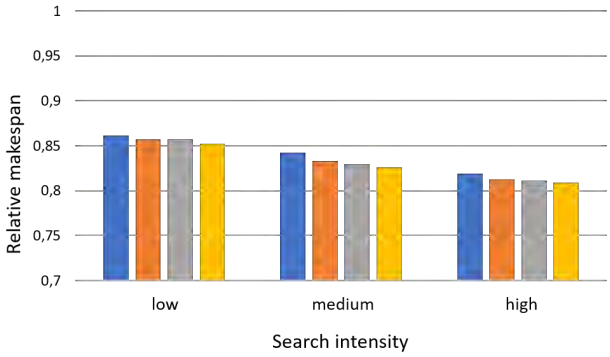
(a) Peripheral depot, small uncertainty



(b) Peripheral depot, large uncertainty



(c) Central depot, small uncertainty

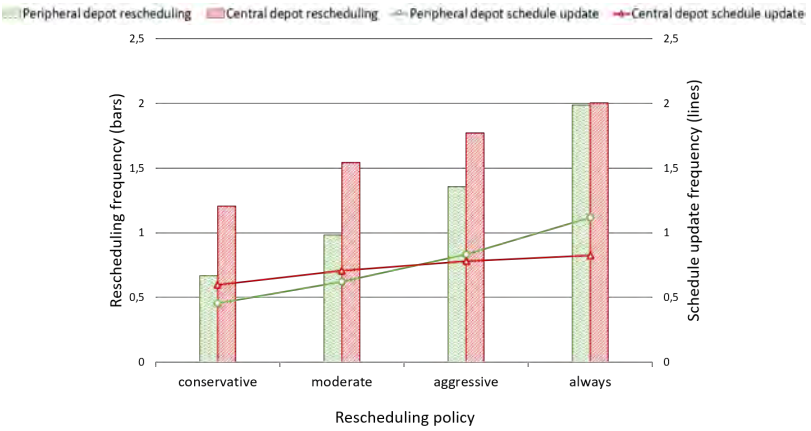


(d) Central depot, large uncertainty

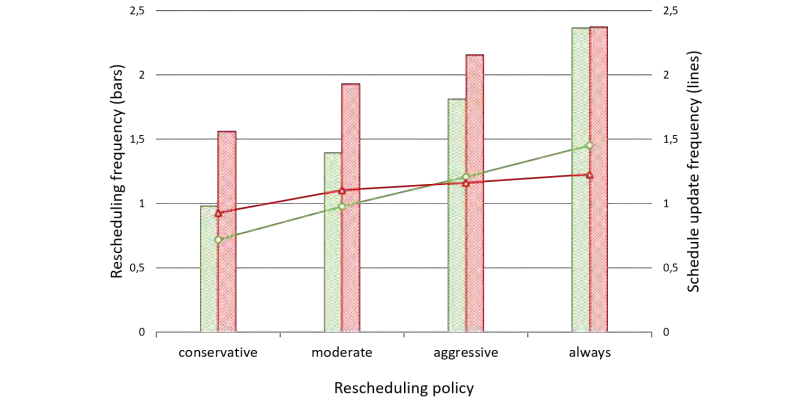
Figure 5-1: Makespan of the online algorithm vs. the offline algorithm.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

Figure 5-2 shows the frequency of rescheduling and actual schedule updates, averaged over all search intensity configurations. The rescheduling frequency is calculated as the total number of rescheduling attempts that are performed during the entire mission divided by the makespan of the mission, whereas the schedule update frequency is calculated as the number of successful rescheduling attempts divided by the makespan. In practical terms, this corresponds to the average number of rescheduling attempts / schedule updates performed while a vehicle is travelling between two nodes; values can be larger than 1 as there are many vehicles that travel concurrently over different distances.



(a) Small uncertainty



(b) Large uncertainty

Figure 5-2: Rescheduling and schedule update frequency of the online algorithm (average over all search intensity configurations).

Naturally, the rescheduling frequency (bars) increases with more aggressive rescheduling and for larger uncertainty. As discussed above, the conservative, moderate and aggressive policies (non-zero thresholds) indeed lead to more frequent rescheduling in the central than in the peripheral depot scenarios, while the always reschedule policy (zero threshold) leads to practically the same rescheduling frequency.

The more rescheduling attempts are made, the more likely it is that at least some of them will succeed, thereby leading to a higher sched-

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

ule update frequency (lines). Note that conservative and moderate rescheduling leads to a higher update frequency in the central depot than in the peripheral depot scenarios, whereas the situation is reversed in the aggressive and always rescheduling policies. Rescheduling attempts are in general more likely to succeed in the peripheral depot scenarios because paths are longer hence the worst-case estimates are also more likely to be overly pessimistic. Therefore, as the gap of the rescheduling frequency between the two scenarios shrinks, the schedule update frequency becomes higher in the peripheral depot scenario. Nevertheless, the impact of each individual update is less significant in the peripheral depot scenario, which is the reason why practically the same improvement is achieved in both cases with the always reschedule policy.

While the proposed heuristic clearly outperforms the static scheduling, it is far from optimal. An oracle with a priori knowledge of the costs that will be experienced during the mission, can further reduce the makespan by 25% up to 45% depending on the scenario. So, there seems to be plenty of room for optimizations, provided one is willing to risk that some vehicles may not be able to safely return to a depot.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

6

CONCLUSIONS

We have presented an online algorithm for tackling the mVRP for uncertain travel costs and vehicles with capacity constraints, based on an LNS component for schedule updates. Starting from a conservative offline schedule, cost deviations that occur at runtime are exploited to let vehicles visit a larger number of nodes before returning to a depot for refuelling/recharging. Our experiments show that the online approach can reduce the makespan significantly vs. an offline schedule, especially when there is large uncertainty regarding the travel costs. Additionally, we have proposed our own metaheuristic solution in order to build the offline schedule, which is faster than a well-known state-of-the-art method while producing solutions of equal or even better quality.

It would be interesting to perform an evaluation of the proposed algorithms using graph topologies and cost estimates taken from real missions that employ UVs (drones). One could also explore more sophisticated heuristics for the node removal/insertion functions. Last but not least, it would be interesting to consider more optimistic variants of the proposed algorithms, possibly in conjunction with methods for learning based on past travel experience.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

BIBLIOGRAPHY

[1] S. Anbuudayasankar, K Ganesh, and S. Mohapatra. "Survey of methodologies for tsp and vrp". In: *Models for Practical Routing Problems in Logistics*. Springer, 2014, pp. 11–42.

[2] G. Gutin and A. P. Punnen. *The traveling salesman problem and its variations*. Vol. 12. Springer Science & Business Media, 2006.

[3] D. Davendra. *Traveling Salesman Problem: Theory and Applications*. BoD–Books on Demand, 2010.

[4] A. E. Carter and C. T. Ragsdale. "A new approach to solving the multiple traveling salesperson problem using genetic algorithms". In: *European journal of operational research* 175.1 (2006), pp. 246–257.

[5] S. Yuan, B. Skinner, S. Huang, and D. Liu. "A new crossover approach for solving the multiple travelling salesmen problem using genetic algorithms". In: *European Journal of Operational Research* 228.1 (2013), pp. 72–82.

[6] E. Falkenauer. "The grouping genetic algorithms-widening the scope of the GAs". In: *Belgian Journal of Operations Research, Statistics and Computer Science* 33.1 (1992), p. 2.

[7] E. C. Brown, C. T. Ragsdale, and A. E. Carter. "A grouping genetic algorithm for the multiple traveling salesperson problem". In: *International Journal of Information Technology & Decision Making* 6.02 (2007), pp. 333–347.

[8] A. Singh and A. S. Baghel. "A new grouping genetic algorithm approach to the multiple traveling salesperson problem". In: *Soft Computing* 13.1 (2009), pp. 95–101.

[9] P. Venkatesh and A. Singh. "Two metaheuristic approaches for the multiple traveling salesperson problem". In: *Applied Soft Computing* 26 (2015), pp. 74–89.

[10] W. Liu, S. Li, F. Zhao, and A. Zheng. "An ant colony optimization algorithm for the multiple traveling salesmen problem". In: *2009 4th IEEE conference on industrial electronics and applications*. IEEE. 2009, pp. 1533–1537.

[11] I. Vallivaara. "A team ant colony optimization algorithm for the multiple travelling salesmen problem with minmax objective". In: *Proceedings of the 27th IASTED International Conference on Modelling, Identification and Control*. ACTA Press. 2008, pp. 387–392.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

[12] S. Somhom, A. Modares, and T. Enkawa. “Competition-based neural network for the multiple travelling salesmen problem with minmax objective”. In: *Computers & Operations Research* 26.4 (1999), pp. 395–407.

[13] V.-I. Lupoaie, I.-A. Chili, M. E. Breaban, and M. Raschip. “SOM-Guided Evolutionary Search for Solving MinMax Multiple-TSP”. In: *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2019, pp. 73–80.

[14] Y. Wang, Y. Chen, and Y. Lin. “Memetic algorithm based on sequential variable neighborhood descent for the minmax multiple traveling salesman problem”. In: *Computers & Industrial Engineering* 106 (2017), pp. 105–122.

[15] B. Soylu. “A general variable neighborhood search heuristic for multiple traveling salesmen problem”. In: *Computers & Industrial Engineering* 90 (2015), pp. 390–401.

[16] P. M. França, M. Gendreau, G. Laporte, and F. M. Müller. “The m-traveling salesman problem with minmax objective”. In: *Transportation Science* 29.3 (1995), pp. 267–275.

[17] B. L. Golden, G. Laporte, and É. D. Taillard. “An adaptive memory heuristic for a class of vehicle routing problems with minmax objective”. In: *Computers & Operations Research* 24.5 (1997), pp. 445–452.

[18] I. Vandermeulen, R. Groß, and A. Kolling. “Balanced task allocation by partitioning the multiple traveling salesperson problem”. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2019, pp. 1479–1487.

[19] L. Bertazzi, B. Golden, and X. Wang. “Min–max vs. min–sum vehicle routing: A worst-case analysis”. In: *European Journal of Operational Research* 240.2 (2015), pp. 372–381.

[20] U. Ritzinger, J. Puchinger, and R. F. Hartl. “A survey on dynamic and stochastic vehicle routing problems”. In: *International Journal of Production Research* 54.1 (2016), pp. 215–231.

[21] H. N. Psaraftis, M. Wen, and C. A. Kontovas. “Dynamic vehicle routing problems: Three decades and counting”. In: *Networks* 67.1 (2016), pp. 3–31.

[22] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia. “A review of dynamic vehicle routing problems”. In: *European Journal of Operational Research* 225.1 (2013), pp. 1–11.

[23] H.-K. Chen, C.-F. Hsueh, and M.-S. Chang. “The real-time time-dependent vehicle routing problem”. In: *Transportation Research Part E: Logistics and Transportation Review* 42.5 (2006), pp. 383–408.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159

[24] I Or. "Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking". In: *PhD thesis (Department of Industrial Engineering and Management Science, Northwestern University)* (1976).

[25] A. Haghani and S. Jung. "A dynamic vehicle routing problem with time-dependent travel times". In: *Computers & operations research* 32.11 (2005), pp. 2959–2986.

[26] E. Taniguchi and H. Shimamoto. "Intelligent transportation system based dynamic vehicle routing and scheduling with variable travel times". In: *Transportation Research Part C: Emerging Technologies* 12.3-4 (2004), pp. 235–250.

[27] J.-Y. Potvin, Y. Xu, and I. Benyahia. "Vehicle routing and scheduling with dynamic travel times". In: *Computers & Operations Research* 33.4 (2006), pp. 1129–1137.

[28] S. Lorini, J.-Y. Potvin, and N. Zufferey. "Online vehicle routing and scheduling with dynamic travel times". In: *Computers & Operations Research* 38.7 (2011), pp. 1086–1090.

[29] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. "A tabu search heuristic for the vehicle routing problem with soft time windows". In: *Transportation science* 31.2 (1997), pp. 170–186.

[30] J. Respen, N. Zufferey, and J.-Y. Potvin. "Online vehicle routing and scheduling with continuous vehicle tracking". In: 2014.

[31] M. Schilde, K. F. Doerner, and R. F. Hartl. "Integrating stochastic time-dependent travel speed in solution methods for the dynamic dial-a-ride problem". In: *European journal of operational research* 238.1 (2014), pp. 18–30.

[32] Z. Xiang, C. Chu, and H. Chen. "The study of a dynamic dial-a-ride problem under time-dependent and stochastic environments". In: *European Journal of Operational Research* 185.2 (2008), pp. 534–551.

[33] P. Shaw. "Using constraint programming and local search methods to solve vehicle routing problems". In: *International conference on principles and practice of constraint programming*. Springer. 1998, pp. 417–431.

[34] G. Reinelt. "TSPLIB—A traveling salesman problem library". In: *ORSA journal on computing* 3.4 (1991), pp. 376–384.

Institutional Repository - Library & Information Centre - University of Thessaly
26/09/2024 00:50:32 EEST - 52.14.224.159