Median Routing Problems:

Integrated optimisation of network maintenance and emergency response

Dylan Huizing

Promotores:	Prof.dr. R.D. van der Mei Prof.dr. G. Schäfer
Co-promotor:	Prof.dr. S. Bhulai
Other members:	Prof.dr. G.M. Koole (chair) Dr.ir. R. Sitters Dr. F. Phillipson Dr. P.L. van den Berg Dr.ir. T. van Essen

This research has been carried out in the Networks and Optimization group at the Centrum Wiskunde & Informatica in Amsterdam, in collaboration with the Vrije Universiteit Amsterdam, and co-funded by ProRail Incidentenbestrijding in Utrecht.

Printed and bound by [–]

[ISBN –]

VRIJE UNIVERSITEIT

Median Routing Problems:

Integrated optimisation of network maintenance and emergency response

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor
aan de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus ad interim
prof.dr. C.M. van Praag,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculty of Science
op [– 2022 om – uur]
in [de aula van de universiteit,]
[De Boelelaan 1105]

door

Dylan Huizing

geboren te Zoetermeer

Promotores:	Prof.dr. R.D. van der Mei
	Prof.dr. G. Schäfer
Co-promotor:	Prof.dr. S. Bhulai

Contents

1 Introduction **1**
- 1.1 About this thesis . . . 2
- 1.2 Railway emergency response in the Netherlands . . . 3
- 1.3 Overview and publications . . . 7

2 The Median Routing Problem **11**
- 2.1 Introduction . . . 11
- 2.2 Related literature . . . 13
- 2.3 Problem definition . . . 16
  - 2.3.1 Example instance . . . 18
  - 2.3.2 Discussion of modelling choices . . . 20
  - 2.3.3 Complexity . . . 21
- 2.4 Methods . . . 22
  - 2.4.1 Mixed Integer Linear Programming . . . 22
  - 2.4.2 `WAIT-AT-MEDIANS`-heuristic . . . 24
  - 2.4.3 `MEDIATE-DIVIDE-SEQUENCE-AGREE`-heuristic . . . 26
  - 2.4.4 Partial versions of `MDSA` . . . 30
- 2.5 Experimental setup . . . 31
  - 2.5.1 Used instances . . . 31
  - 2.5.2 Metrics and methods for solution structure . . . 33
  - 2.5.3 Hardware specifications . . . 33
- 2.6 Results . . . 34
- 2.7 Conclusions . . . 40

3 The Travelling k-Median Problem **43**
- 3.1 Introduction . . . 43
- 3.2 Related literature . . . 45
- 3.3 Problem definition . . . 46
- 3.4 Hardness and bounds . . . 47
- 3.5 Rounding continuous graph movement . . . 50
- 3.6 Topologies with constant-factor guarantees . . . 53
  - 3.6.1 Path case . . . 53
  - 3.6.2 Cycle case . . . 55

3.7 Approximation algorithms for general graphs . . . 56
3.8 Conclusions . . . 59

4 Representative Distance-preserving Graph Sparsifiers 61
4.1 Introduction . . . 61
4.2 Related literature . . . 63
4.3 Problem definition . . . 65
4.4 Four ways to compute representative subgraphs . . . 66
4.4.1 The `Smallest` RDGS algorithm . . . 68
4.4.2 The `Realigned Smallest` RDGS algorithm . . . 70
4.4.3 The `Greedy On-Ramps` RDGS algorithm . . . 71
4.4.4 The `Greedy Centrality` RDGS algorithm . . . 71
4.5 Mixed Integer Jester Games . . . 74
4.5.1 α-iteration . . . 75
4.5.2 LP-rounding algorithms for usurpers . . . 78
4.6 Examples of sparsification strategies . . . 78
4.6.1 Facility Location Problems . . . 78
4.6.2 Classical routing problems . . . 79
4.6.3 Orienteering Problems . . . 80
4.6.4 The Two-Stage Stochastic Steiner Tree Problem . . . 80
4.7 Bounding MRP performance loss as a Mixed Integer Jester Game . 81
4.8 Experiments . . . 85
4.9 Results . . . 86
4.10 Conclusions . . . 92

5 The Enriched Median Routing Problem 95
5.1 Introduction . . . 95
5.2 Related literature . . . 96
5.3 Enriched Median Routing Problem . . . 98
5.3.1 Problem description . . . 98
5.3.2 Mixed Integer Linear Program formulation . . . 102
5.3.3 An `MDSA`-inspired heuristic . . . 103
5.4 Current Practice model . . . 108
5.5 Performance metrics, planning scenarios and use case description . 113
5.5.1 Performance metrics . . . 113
5.5.2 Planning scenarios . . . 113
5.5.3 Use case description . . . 115
5.5.4 Implementation details . . . 116
5.6 Results . . . 116
5.7 Conclusions . . . 120

6 Real-life Implementation and Next Steps 121

Summary 125

References **128**

A Subroutines of Chapter 2 **143**
A.1 MILP for routing subroutine of `WAM` 143
A.2 Solving path-TSP 144
A.3 Cheapest space-time path for sequence and node costs 145
A.4 Instance generator 145

Chapter 1

Introduction

In 1954, George Dantzig and his colleagues computed the shortest tour over 49 American cities [34]. In 1970, Constantine Toregas and his peers computed at which of 30 locations in New York State to build emergency facilities to provide optimal emergency response service throughout the state [147]. Ever since those days, efficient routes and minimal emergency response times have both been fundamental motivators of the field of Operations Research. The *Travelling Salesman Problem* and *Facility Location Problems* still captivate scientists today: they are problems that are easy to understand, yet notoriously difficult to solve.

Today, determining efficient tours over service points for preventive maintenance and other plannable activities has become increasingly relevant and well studied [22, 125, 155]. Emergency response has also become more efficient, partly due to using computers to find optimal locations from which to await emergencies [159]. The growth in both topics has been accompanied with decades of steady literature and ever-improving algorithms [46, 108]. Evidently, how to minimise emergency response times and how to efficiently route over non-urgent jobs have both become increasingly well understood.

Although these two fields of application have advanced so far, it is surprising to see that their overlap is still in its infancy. An overlap certainly exists: many emergency response organisations are also tasked with scheduling known jobs of lesser urgency. Police officers may investigate suspicious sightings [65], ambulances may perform scheduled patient transportation [150] and repairmen may perform preventive maintenance [64]. When deciding where to stand still and await emergencies with minimal response time, the literature is abundant [45, 98, 101, 111]. When deciding how to optimally schedule and route over tasks known upfront, the literature is overwhelming [4, 18, 21, 41, 92, 107]. However, when deciding how to do both things at the same time, i.e. how to perform scheduled tasks while remaining prepared for emergencies, the literature suddenly becomes much more sparse.

The overlap between emergency response and plannable tasks is a promising one to explore. If emergency responders were to spend their idle time performing

preventive activities, then this would give them a much more productive way of filling their shift than simply waiting for emergencies. Ideally, it would prevent some emergencies and their corresponding tragedies from even occurring in the first place. Of course, when assigned with plannable tasks, the agents would have to retain a good spread over the region, so as to maintain a good emergency response time. But if planned well, combining the emergency fleet with the preventive fleet would be beneficial for the goals of both fleets: more preventive work would get done *and* more agents would be available for emergency response, allowing a finer weave of agents over the region, thus leading to potential decreases in emergency response time.

However, in many current practices, the responders tend to sit at their stations and wait. It appears that this frustrating state is caused by the following. For one, it is non-trivial how to form a planning in which both preventive jobs and emergencies are handled efficiently. But the true difficulty arises as soon as an emergency occurs: this event inevitably influences the planning, meaning an adjusted plan must be formed and communicated rapidly and remotely, while under stress from an emergency. It seems that without the proper support infrastructure, this task is too difficult to manage. Reasoning on that preventive jobs are only 'secondary' to emergency response in the first place, emergency organisations will often conclude it is more manageable to keep the emergency fleet and the plannable fleet separate [77]. Productivity and responsiveness are lost, simply because emergency response organisations do not have the means to rapidly re-coordinate the planning of a joint fleet.

Motivated by this, this thesis will propose methods to assist emergency response organisations computationally. The field of Operations Research offers abundant knowledge on how to efficiently route, as well as on how to efficiently prepare for emergencies. Therefore, we will propose models and algorithms which combine and integrate the knowledge from both these fields. These explorations should culminate in allowing emergency response organisations to both prevent more emergencies and further reduce their emergency response times, to the benefit of all.

1.1 About this thesis

This thesis is the result of four years of collaboration between the Centrum Wiskunde & Informatica [24] (the Dutch centre for mathematics and computer science) and ProRail Incidentenbestrijding [120] (the emergency response organisation of the Dutch railway infrastructure manager). The collaboration was carried out in the form of a PhD project. While the collaboration was initially proposed to involve the improving of emergency forecasting, it was quickly recognised that the emergency response organisation had a desire to plan their non-urgent tasks more systematically and that Operations Research techniques could offer a solution. In parallel to developing and publishing mathematical models and algorithms, the project also involved implementing a rapid solver

for the emergency response organisation, culminating in an application that the emergency response organisation has taken into operational use [119].

1.2 Railway emergency response in the Netherlands

To give context to the mathematical models developed in this thesis, we illustrate the operational dynamics of the case study organisation.

ProRail BV is the sole manager of the Dutch railway infrastructure. Multiple companies operate trains on the Dutch railway system, with Nederlandse Spoorwegen being the predominant transporter of passengers, but the actual physical rail is owned and managed by ProRail. That the railway infrastructure is managed by a different party than the trains are, is not at all uncommon in the European Union: in fact, it is mandated by the Single European Railway Directive 2012 [145].

One of the responsibilities of ProRail is to physically ensure that the railway can be used. That is, if an incident occurs that makes a piece of the railway inoperable, then it is the responsibility of ProRail to resolve the incident and bring the railway back into its operational state as quickly as possible [122]. To this end, ProRail created the specialised department "ProRail Incidentenbestrijding" to manage railway emergencies. This department not only dispatches emergency responders when incidents occur, but also oversees the preparation for emergencies, by deciding on equipment and training among other things. ProRail Incidentenbestrijding is the case study organisation of this thesis, as well as a co-funder of the research.

ProRail Incidentenbestrijding has four major regional headquarters in the Netherlands, namely in the cities of Utrecht, Rotterdam, Eindhoven and Zwolle. See also Figure 1.1a. Roughly speaking, the country is divided into four emergency response regions, corresponding to these four regional headquarters. At the ProRail headquarters, there is also a support staff that facilitates operations on the national level, by managing things such as legislation and hardware acquisition. Moreover, ProRail has many traffic control centres in the country that can also be used as a minor base of operations for emergency responders. The emergency responders are employed directly by ProRail Incidentenbestrijding and work on a shift basis, with the early shift overlapping the late shift by an hour. In the two smallest regions, it is fairly customary that the emergency responders start and end their shifts at the regional headquarters. In the two largest regions, agents who live in the more remote corners of the region may also sometimes start and end their shift at a more nearby ProRail location. Seeing how these remote corners also need an emergency responder nearby, it is often preferable that agents who live there will stay in those areas, rather than driving almost two hours to the regional headquarters and back.

Emergency responders are typically active in either a car, a van with minor

(a) The four regional headquarters.

(b) An emergency response car.

(c) An emergency response van.

(d) An emergency response truck.

Figure 1.1: Photographs and visualisations describing the case study organisation, ProRail Incidentenbestrijding.

equipment, or a truck with extensive equipment. See also Figures 1.1b, 1.1c and 1.1d. All these vehicles are owned by ProRail and equipped with a GPS module. Whenever a railway emergency occurs, the "Slim Alarmeren" system [119] uses those GPS coordinates to determine the nearest active responders, regardless of what emergency response region they belong to. Depending on the type and severity of the incident, a variable number of those agents are deployed. For the heaviest of incidents, the emergency response truck is needed on site. At all times, there is typically exactly one agent per region who has 'truck duty' for that region: throughout the shift, this agent needs to stay nearby the truck, which is typically parked at the regional headquarters. In the Rotterdam region, there is also always one agent with 'harbour duty', who stays in the Rotterdam harbour area to respond to railway incidents there. Finally, the Utrecht area contains the largest airport of the Netherlands, and there is always someone on 'airport duty' specifically to respond to emergencies that would make it difficult to reach the airport by train.

The emergency responders, aside from having access to different types of vehicles, fall roughly into three categories. Most agents are active as 'shift member', and they are typically the ones performing the physical activities needed to resolve an incident, such as clearing obstacles and evacuating train passengers. One agent per region is active as 'shift leader', who coordinates the work of the 'shift members'. For most incidents, there is also one 'general leader' deployed to manage communication with the other involved parties, such as the train operator, fire department and local authorities. In some cases, these other parties are strictly necessary to resolve the incident: for instance, if there is a fatality connected to the incident, the scene may not be altered until local police and a coroner have investigated it.

There are no formal KPIs on how quickly a railway emergency should be 'responded to', partly because the time needed to resolve the incident can depend on these other parties. As a rule of thumb, ProRail desires that the first emergency responder arrives at the site of the incident within 30 minutes in the two smaller regions, or within 60 minutes in the two larger regions. A single agent may not always be able to start resolving the incident on their own, but at the very least, this agent can start assessing more closely which actions and parties will be needed and start up the chain of communication. Most importantly, every minute that a piece of railway is disturbed can cause significant economic damage, and the case study organisation wants agents to arrive at the scene as quickly as possible.

The case study organisation is unlike classical emergency response organisations, like fire departments and ambulance service providers, in the following way: if there are no active emergencies, the responders can perform 'preventive activities' in the region, instead of waiting for the next emergency at a response station. For instance, they can travel along a piece of railway to check for suspicious activities or defects. Surprisingly, citizens walking over or near the rail account for an estimated 36% of train delays [121], meaning much can be gained

from intercepting them beforehand. Alternatively, the agents can patrol at road-railway intersections and fine car drivers who cross when they are not allowed. This latter activity requires at least two agents to be present, and at least one of them must have a 'Buitengewoon Opsporend Ambtenaar' authorisation that allows them to write out fines. Some preventive tasks, like weekly inspections at railway freight stations, are contracted from outside the department and have deadlines that should be met. However, in general, the preventive activities are an effective way to spend idle time but they do not take priority over rapid emergency response.

Perhaps for this reason, the planning of preventive activities has not received as much support in the case study organisation. The different regions tend to handle preventive activities in different, often decentral ways. In one of the regions, the 'shift leader' of a shift will decide what activities the shift members will do and where. Which tasks are considered 'important' is up to the discretion and preference of that particular shift leader. The infrastructure supporting these decisions is often not more than a brief text document recording what was done last week, and a printed table of which freight stations have been inspected per week, combined with requests sent over email. In another region, the shift-by-shift decision making is mitigated somewhat by a weekly strategy meeting among shift leaders in which a rough planning is decided for that week. A complicating matter, however, is that it is entirely unclear when and where emergencies will occur throughout that week. In yet another region, the activities are suggested by the shift members themselves and accorded by the shift leader, trusting that the shift members know 'their' piece of the railway network best. As we will see in Chapter 5, this may occasionally result in shift members encountering each other patrolling the same piece of railway, because they unknowingly decided to perform the same preventive activity. In most regions, the 'general leaders' and 'shift leaders' are exempt from performing preventive activities, but not from emergency response. In rare cases, agents may be exempt from emergency response, but not from preventive activities. In all regions, it is well understood that the shift leader should strive to maintain a reasonable spread of the shift members during their preventive activities, so as to keep response times low.

The following conclusion can be drawn concerning the case study organisation, and with that, concerning railway emergency response in the Netherlands in general. Despite the lack of concrete KPIs, the organisation of emergency response itself is quite advanced. There are several response stations spread over the country, with extra stations dedicated to key infrastructural points, namely the largest airport and the Rotterdam harbour area. Emergency responders are organised into three categories, and independently, their vehicles fall into three size categories that allow for responding effectively to emergencies on different scales. On the level of support software, GPS coordinates are used to automatically find and notify the agents nearest to an incident, regardless of which response region they are part of. However, in contrast to emergency response, the planning of preventive activities is still largely decentral, and is likely to benefit

from dedicated decision support software. The challenge lies in computing, and quickly updating, a planning of these preventive activities that keeps the agents well spread in case of emergencies.

1.3 Overview and publications

The remaining chapters of this thesis will investigate the problem of computing a planning that efficiently handles both emergency response and plannable tasks, from different perspectives. First, in Chapter 2, the **Median Routing Problem** is introduced: this model for joint planning will be at the heart of this thesis. Several algorithms with which to address this problem will be proposed, of which one exact and the rest rapid heuristics. Most notably, in the so-called `MDSA`-algorithm, the problem is split into steps that resemble the well-known k-Median Problem and Multiple Travelling Salesman Problem. These problem instances can still be solved quickly because they act on heavily compressed solution spaces. It is concluded that, indeed, `MDSA` is both a fast and an empirically accurate heuristic for both randomly generated and use-case inspired benchmark instances. This chapter is based on the following publication [70]:

- D. Huizing, G. Schäfer, R. D. van der Mei, and S. Bhulai. The Median Routing Problem for simultaneous planning of emergency response and non-emergency jobs. *European Journal of Operations Research*, 285(2):712–727, 2020.

As it turns out, the Median Routing Problem is not only NP-hard, but it is even NP-complete to decide whether a feasible solution exists. As such, no polynomial-time approximation algorithms exist for this new problem, unless P=NP. Briefly put, this theoretical difficulty is due to having both a hard constraint that all jobs should be processed and a hard limit on how much time the agents have to do so and it is NP-complete to check whether there exists a route that is efficient enough to abide by this time limit. Concluding that no approximation algorithms exist for the Median Routing Problem, we investigate in Chapter 3 whether they do exist for a special case of the Median Routing Problem, namely when the jobs are removed. It may seem contradictory to first introduce jobs and then take them away again, but the studied **Travelling k-Median Problem** reveals that it is already non-trivial to add a notion of movement to the k-Median Problem. However, constant-factor approximation algorithms are derived for special cases. They are found using a novel notion of solving the problem on a 'continuous version' of the graph and rounding this solution back with bounded loss. Presumably, this technique may be interesting for other graph-based problems as well. This chapter is based on the following publication [69]:

- D. Huizing and G. Schäfer. The traveling k-median problem: approximating optimal network coverage. In *International Workshop on Approxima-*

tion and Online Algorithms. Springer (to appear).

Moving back towards application, it should be noted that the Median Routing Problem is only a simplification of the operational practice. In studying the Median Routing Problem, one can study the fundamental difficulty of combining emergency response and plannable tasks. However, many practical side constraints such as time windows are not a part of the basic model. Most notably, in practice, some jobs may require more than one person to complete. Unfortunately, this latter feature is not compatible with the successful `MDSA`-algorithm. However, understanding that `MDSA` was effective because it solved decomposed problems on compressed space, a new heuristic can be developed. A crucial part in keeping this method fast is performing a strong compression of the solution space. In Chapter 4, the essential subroutine is developed with which to perform this compression. That is, several methods are discussed with which to take an input graph and remove nodes from it that likely do not affect the optimal solution value by much. The graph cannot be sparsified too heavily however: if the graph distances increase because essential connections are removed, this may suddenly mean that feasible solutions to the Median Routing Problem no longer exist on the compressed graph.

Motivated by this, **Representative Distance-preserving Graph Sparsifiers** are introduced in this chapter, and several methods by which to compute them are proposed. To keep the discussion clear, these sparsifiers are introduced in the context of the basic Median Routing Problem, not yet keeping the complicated operational side constraints into account. Not only will it be seen that this sparsification method is indeed effective for the Median Routing Problem, in that the previous benchmark instances can be solved much more rapidly with only marginal loss of optimal solution value, but the *Mixed Integer Jester Game* framework will be introduced to *prove* an upper bound on how much solution quality can be lost through sparsification. This chapter is based on the following publication [71]:

- D. Huizing, G. Schäfer, R. D. van der Mei, and S. Bhulai. Distance-preserving graph sparsification with bounded loss for network problems, 2021. Submitted.

Using this sparsification subroutine, an `MDSA`-based heuristic can finally be derived for the **Enriched Median Routing Problem**, which includes fifteen additional features that arise from practice. These features include, among others, planned travel time and makespan and job assignment penalties as additional terms in the objective function, non-contiguous time windows for jobs, jobs requiring a mix of agent specialisations, variable start and end locations of agents and the presence of subregions in which certain agents must remain. In Chapter 5, the Enriched Median Routing Problem is introduced more formally and the `MDSA`-based heuristic is described. With all of these operational features included, a more fair comparison can be made to the current operational practice of the

case study emergency response organisation. While it is possible to compute a planning for the case study organisation, it is difficult to measure how well this plan is followed in practice. Therefore, a current practice model is developed, so as to compare performance metrics in the current plans against the plans proposed by the Enriched Median Routing Problem solutions. From this comparison it is concluded that, indeed, the methods in this thesis can lead to a simultaneous improvement in *both* the emergency response time and the amount of plannable work performed during idle time. This chapter is based on the following publication [72]:

- D. Huizing, R. D. van der Mei, G. Schäfer, and S. Bhulai. The Enriched Median Routing Problem and its usefulness in practice, 2021. Submitted.

Finally, in Chapter 6, other findings and related projects are mentioned briefly and further outlooks are proposed.

Chapter 2

The Median Routing Problem

2.1 Introduction

Optimal positioning, in applications such as firefighting and ambulance management, is an important part of emergency logistics research [23]. In classical models and settings, emergency responders are expected to remain at a base station until an emergency occurs, and an optimal or near-optimal set of waiting positions is determined [19, 111, 118].

However, in some real-world applications, including those discussed in Chapter 1, it may be interesting to assign incident-preventing activities or other scheduled activities to responders when there are no active emergencies to resolve. To recall those examples, one could want to assign routine railway inspections to idle railway emergency responders, or to combine scheduled ambulance transport with emergency response, or to plan routine patrols of the police force such that good emergency coverage is guaranteed. This would allow the emergency responders to spend their idle time much more effectively, namely by proactively preventing emergencies, rather than by waiting at a base station.

Combining an 'emergency response fleet' with a 'maintenance fleet' is challenging, but worthwhile [86]: this would yield more manpower for the scheduled work, as well as a larger pool of emergency responders. When an emergency happens, however, the nearest agent should abort whatever task he or she is doing and hurry towards the emergency. It would be undesirable if an emergency occurs, but due to poor planning, all responders are performing a task at some far-off location.

This gives rise to an interesting Operations Research challenge: how can we schedule these preventive tasks in the network, such that we can guarantee a good spread of emergency responders over the day and minimise the average response time to potential emergencies? This is the central question of this thesis, in its purest form. Answering this question should provide a worthwhile contribution to the field of emergency logistics.

If one were to only care about minimising response time to emergencies, one

could find an optimal distribution of agents over the network by solving a *k-Median Problem* (K-MED) [35]. In K-MED, we have a finite set of nodes, each with some non-negative weight, and a symmetric distance matrix between these nodes. The goal is to select exactly k of these nodes which will act as 'facilities'. Each other node is then connected to its nearest facility, and contributes a cost equal to the distance to that facility times the node's weight. K-MED is the problem of finding the k nodes for which this total weighted distance is minimal. In this context, it would give the emergency responders optimal locations from which to anticipate emergencies.

If one were to only care about processing many preventive tasks in the network in the span of one work shift, one could find the fastest routing of agents over jobs by solving a *Distance-Constrained Vehicle Routing Problem* (DVRP) [94]. DVRP is almost identical to the classical *Vehicle Routing Problem* (VRP), except route lengths may not exceed some parameter ω. More specifically, suppose we have again a finite set of nodes with a symmetric distance matrix. One node is called the 'depot'; the others are called the 'customers'. Each customer has a non-negative amount of processing time. Given a fixed number of vehicles k, a feasible solution to DVRP consists of exactly k tours that start and end at the depot, such that every client is visited exactly once, and such that every tour has a length that does not exceed ω. DVRP is the problem of finding the k feasible tours with the smallest total distance travelled. In this context, it would give the fastest routing of the agents over the preventive tasks such that everyone is back by the end of the shift.

The optimal solutions to these two problems would by definition be conflicting: the former would have the agents standing still, while the latter would have them move around with no explicit regard to emergency response times. In this context, however, one cares about *both* minimal response time and efficient task processing: one seeks to route agents over all given jobs, as in DVRP, but with the objective to minimise emergency response time, as in K-MED. This problem is therefore also interesting in that it lies on an unexpected boundary between two well-studied Operations Research problems, namely K-MED and DVRP.

Despite these motivations from academia and industry, the literature review in Section 2.2 suggests that this combined planning problem has received little attention. Therefore, this chapter proposes the *Median Routing Problem* (MRP) as a mathematical model for scheduling preventive tasks while minimising emergency response time, and proposes methods to find optimal or near-optimal solutions to this problem quickly.

The contribution of this chapter is twofold. First, we propose a mathematical model for this planning problem. This model allows for discretisation of continuous response time, is suited to deal with online re-planning when emergencies occur, and can be solved with Mixed Integer Linear Programming. Second, we propose a heuristic for this model, that for real-life benchmark instances needs only 4.5 seconds to find solutions that are on average 3.4% away from optimal. This heuristic has an unusual approach, in that it decomposes the decisions into

several NP-hard subproblems, but these NP-hard decisions are so much compressed that they can be solved within seconds in the benchmark instances. We propose an explanation for the success of this heuristic by comparing it with related heuristics under variation of instance parameters.

The remainder of this chapter is structured as follows. In Section 2.2, we review literature concerning related problems. In Section 2.3, we give a rigorous problem definition, including its complexity. In Section 2.4, we describe a solution method and several heuristics. In Section 2.5, we detail the experimental setup in which these methods are compared. In Section 2.6, we present the results of computational experiments with some observations. In Section 2.7, we present our conclusions.

2.2 Related literature

Vast literature exists on solving and approximating K-MED, going back at least as early as the work of ReVelle and Swain [126]. The metric problem was first shown to be NP-hard by Kariv and Hakimi [84]. Daskin and Maass [35] have provided a recent overview of solution methods, construction and improvement algorithms and metaheuristics for K-MED. Among these, they provide a Mixed Integer Linear Program. They remark that the variables describing median selection must be binary, but that the variables describing the assignment of nodes to medians may be left continuous. We will exploit a similar result in Section 2.4.1. If the distance matrix satisfies the triangle inequality, K-MED can be approximated to within a factor of $6\frac{2}{3}$ due to an LP-rounding result by Charikar et al. [25]. This approximation factor has since been improved by Arya et al. [6] using a local search method with swaps. They approximate K-MED to a factor $3+2/k$, where k is the number of swaps allowed to be made simultaneously. The more general variant of the problem where the connection cost is non-negative and symmetric (but does not necessarily satisfy the triangle inequality) was first shown to be APX-hard by Lin and Vitter [99]. On the positive side, the problem is polynomial-time solvable on trees [84, 141] and there exists a polynomial-time approximation scheme if (V, d) constitutes a Euclidean space [5]. The current best approximation algorithm for the problem achieves an approximation ratio of $\alpha = 2.675 + \varepsilon$ [20]. In terms of negative results, the current best lower bound is the $(1 + 2/e)$-inapproximability result by Guha and Khuller [61].

Laporte reviewed a number of exact and approximate algorithms for DVRP [93]. Almoustafa et al. [3] solved large instances of the variant with asymmetric travel costs using a modified branch-and-bound procedure with random tie-breaking. If we wish to minimise the number of vehicles needed rather than the travel costs, Nagarajan and Ravi [109] provide a 2-approximation on tree metrics and an $(\mathcal{O}(\log \frac{1}{\varepsilon}), \varepsilon)$-bicriteria approximation algorithm on general metrics.

Broader surveys of VRP variants were done by Eksioglu et al. [41], Toth and Vigo [148] and Joubert [82]. In particular, in the Dynamic Vehicle Routing Problem [115], the customers to be visited may appear during execution of the

routes, and the decision maker is tasked with making a route over the known customers and to adjust them whenever new customers appear. This is similar, in some sense, to emergencies occurring and requiring a rescheduling. A key difference is that new Dynamic VRP customers can be incorporated into existing routes at any point, whereas emergencies demand an immediate response. The inherent uncertainty in Dynamic VRP is dealt with in several ways, including Multiple Scenario Approaches [116], a-priori routes [151, 161], rolling horizon approaches [80, 112] and rollout policies [57].

Ichoua et al. [73] use Tabu Search to minimise a weighted sum of travelled distance and lateness to both known and dynamically revealed jobs. One could adapt this to the problem at hand by seeing the revealed jobs as 'emergencies', and assigning zero weight to travelled distance and lateness to known jobs. However, their model would then prescribe that any feasible solution is optimal, as long as it responds to revealed jobs as quickly as possible.

Our research also considers situations where agents must be routed over jobs, but their start and end locations are not the same, despite this being a typical assumption in VRP variants. Therefore, our research makes use of the (S, W)-*path Travelling Salesman Problem* (path-TSP) [66]. In path-TSP, we again observe a finite node set with a symmetric distance matrix. One node is called the 'start node' S; one other node is called the 'end node' W. Path-TSP is the problem of finding the shortest path that starts at S, ends at W, and visits each node exactly once. It can be solved by means of the Mixed Integer Linear Program in A.2, or approximated by the method of Zenklusen [160].

The problem at hand bears a strong resemblance to the k-Server Problem [88]. In the k-Server Problem, requests appear dynamically in a metric space, and whenever this happens, a decision-maker must immediately decide which of k servers to send towards the request and how to reposition the rest. The goal is to minimise the total amount of distance travelled. Typically, probabilities for where requests may appear are not known, and researchers have focused on finding algorithms with small competitive ratio against someone who knows completely when and where the requests will appear.

Farahani and Hekmatfar [46] describe a number of different facility location problems and concepts. In the Dynamic k-Median problems reviewed by Owen and Daskin [111], locations may close and reopen in different time periods to satisfy period-dependent demands. In the Capacitated Mobile Facility Location Problem [124], initial facility locations have already been chosen, but one may relocate facilities against distance-dependent costs. The goal is to minimise a weighted sum of these facility relocation costs and the subsequent cost to serve all clients. The Location-Routing Problem, reviewed also by Drexl and Schneider [40], is concerned with simultaneously deciding delivery routes and the facility locations from which they spring: the goal is to minimise the distance travelled between jobs.

Bertsimas and Van Ryzin [13] study a dynamic Travelling Repairman Problem, where multiple agents may move freely over the Euclidean plane and must

respond to dynamically revealed service requests as soon as possible. They describe policies with costs that are provably within a constant factor of the optimal policy costs.

For ambulances, the combined planning of emergency response and non-emergency patient transportation has received some attention. Kergosien et al. [85] study when and from which hospital to temporarily expend emergency ambulances on non-emergency transportation, as do Van Den Berg and Van Essen [150]. They seek to minimise the temporary loss in emergency response coverage when performing non-emergency transportation. In contrast, Kiechle et al. [86] study a problem where emergencies are responded to by the nearest empty ambulance, including the ones performing non-emergency transportation. Their analysis is mostly focused on comparing whether it is better to arrive at the next job as early as possible or as late as possible.

In other fields, research has been done into combined maintenance-routing, which studies how to jointly determine when to perform maintenance and how to route between maintenance jobs. The maintenance schedule affects the routes, but the routes may also affect the maintenance schedule, depending on the piece of research. Most of the maintenance-routing literature in air transportation [7, 58, 63, 132, 140] and train transportation [103, 114] seems to focus on how to execute a required transportation schedule with a set of vehicles, while ensuring that these vehicles are routed over maintenance stations regularly. Cohn and Barnhart [29] include the subsequent crew scheduling into the optimisation as well.

López-Santana et al. [102] study a combined maintenance-routing problem in the oil and gas industry where the goal is to determine the expected optimal times and frequencies at which to perform maintenance, balancing the fixed cost of performing maintenance against the expected cost incurred when a machine breaks down and remains unrepaired until its next maintenance moment. Aside from determining the optimal times and frequencies in a maintenance planning phase (by optimising over continuous, non-linear functions numerically), they also try to fit feasible repairmen routes on this (using Mixed Integer Linear Programming on a space-time network) in a routing phase, and iterate between the two until some stopping criterion is reached. Fontecha et al. [50] improve upon this work in two notable ways. First, they expand the model to allow for re-planning after breakdowns. Second, they replace the computation method with a more scalable version: that is, they remove the need to iterate between the two phases and they replace the Mixed Integer Linear Program by a matheuristic. They then apply this to case studies in a large-scale sewage cleaning application. Irawan et al. [76] study a maintenance-routing problem for off-shore wind farms, that more closely resembles a Vehicle Routing Problem with Pick-up and Delivery. Inspired by this similarity, they solve it using a Dantzig-Wolfe decomposition method.

Some work has been done in coordinating several unmanned vehicles to provide joint 'coverage' over a region [1, 39, 137, 156]. They focus on mapping out the entire area once with mobile camera's, rather than providing 'emergency

coverage' whilst performing jobs in the region.

We conclude that many similar problems have been studied, but that each differs fundamentally from the problem at hand, and that a new model is required.

2.3 Problem definition

In this section, we formally introduce MRP as a mathematical optimisation problem. We also describe the problem by means of a Mixed Integer Linear Program in Section 2.4.1. In order to give a formal definition of the MRP, we employ the notation listed in Table 2.1.

Set		Description
A		The set of agents
J		The set of jobs
V		The set of nodes
T		The set of time steps, $T = \{0, 1, \ldots, \omega\}$
V_P		The set of nodes where incidents may occur ($V_P \subseteq V$)
V_v		The neighbourhood of $v \in V$
Parameter	Domain	Description
S_a	V	The start location of agent $a \in A$
W_a	V	The end location of agent $a \in A$
L_j	V	The node where job $j \in J$ is located
Q_j	$\mathbb{Z}_{\geq 0}$	The number of time steps job $j \in J$ takes
P_v	$(0, 1]$	The probability that the next emergency happens at node $v \in V_P$
C_{uv}	$\mathbb{Q}_{\geq 0}$	The undiscretised emergency response time from $u \in V$ to $v \in V_P$
Θ_v	$\mathbb{Z}^2$	The coordinates of node $v \in V$
Variable	Domain	Description
x_{avt}	$\{0, 1\}$	Whether or not agent $a \in A$ is at $v \in V$ at time $t \in T$
y_{uvt}	$\{0, 1\}$	Whether or not a potential emergency at $v \in V_P$, time $t \in T$ will be responded to from $u \in V$
z_{ajt}	$\{0, 1\}$	Whether or not agent $a \in A$ starts job $j \in J$ at time $t \in T$

Table 2.1: Notation for the Median Routing Problem.

In MRP, we observe a connected, undirected graph with node set V. The neighbourhood of $v \in V$ is denoted $V_v \subseteq V$. We demand that each node is in its own neighbourhood. Each node v has known coordinates $\Theta_v \in \mathbb{Z}^2$. We discretise time into a finite time horizon $T = \{0, 1, \ldots, \omega\}$.

There is a set of agents A that can move over the network. That is: if agent

a is at node u at time $t \neq \omega$, it can only be at $v \in V$ at time $t+1$ if $v \in V_u$. At time 0, each agent a is at some start location $S_a \in V$. At time ω, each agent must be at a specific end location $W_a \in V$.

There also exists a set of jobs J, distributed over the graph. Each job $j \in J$ has a location $L_j \in V$ and a processing time $Q_j \in \mathbb{Z}_{\geq 0}$. Each job must be processed and the jobs must be processed non-preemptively to succeed: that is, whenever an agent starts processing a job, that agent must stay at that location to process the job for its full duration, unless an emergency occurs. If a job is aborted halfway due to an emergency, the job fails and must be processed entirely anew.

Note that we allow for distinct jobs to be at the same node: for example, the depot may host several equipment maintenance and administrative jobs. Though we could combine all jobs at a given location into one superjob, there exist instances where doing so destroys feasibility. The only exception we make is that if a job has length 0 and there is another job at the same location, we will merge them into one. An alternative could be to place each job on its own virtual node, but we believe having to distinguish between real and virtual nodes that describe the same location is less elegant than simply allowing one node to host multiple jobs.

Finally, emergencies may occur in any node $v \in V_P \subseteq V$. An emergency may occur at any time step $t \in T$ against a time-independent probability. The probability that the next emergency occurs in $v \in V_P$ is $P_v > 0$, with $\sum_{v \in V_P} P_v = 1$. If an emergency at $v \in V_P$ is responded to from an agent at $u \in V$, the emergency response time is $C_{uv} \in \mathbb{Q}_{\geq 0}$. This distance matrix C does not need to be symmetric, and the methods presented in Section 2.4 do not require C to be symmetric, though the benchmark instances discussed in Section 2.5 do all have a symmetric C. Note that, outside of emergency logistics, P_v can be more broadly interpreted as node weights, and C_{uv} can be more broadly interpreted as service costs.

A feasible solution of MRP must tell the agents where to be at each time step and which jobs to start processing when, respecting the above constraints. Moreover, for each $v \in V_P$ and each $t \in T$, a node $u \in V$ must be appointed to 'cover' v in case of an emergency; in any optimal solution, u is always the node with lowest response time C_{uv} that has at least one agent present at time t. We encode any solution to MRP with binary variables x_{avt} indicating whether agent $a \in A$ is at node $v \in V$ at time $t \in T$, binary variables y_{uvt} indicating whether $v \in V_P$ is covered from $u \in V$ at time $t \in T$, and binary variables z_{ajt} indicating whether agent $a \in A$ starts processing job $j \in J$ at time $t \in T$.

Using this notation, we remark that if exactly one emergency occurs at some time t and node $v \in V_P$, the response time equals C_{u^*v} for some $u^* \in V$, which has $y_{u^*vt} = 1$, while the other $u \in V$ have $y_{uvt} = 0$. In other words, the response time equals $\sum_{u \in V} C_{uv} y_{uvt}$. If indeed an emergency happens at time t, it happens at node $v \in V_P$ with probability P_v, meaning the overall expected response time

to an emergency at time t equals

$$\sum_{v \in V_P} P_v \left(\sum_{u \in V} C_{uv} y_{uvt} \right)$$

In MRP, the goal is to minimise $\sum_{t \in T} \sum_{v \in V_P} P_v (\sum_{u \in V} C_{uv} y_{uvt})$ over the feasible (x, y, z), where by the above discussion, the summed expression equals the expected response time to an emergency if exactly one emergency happens at time $t \in T$ and none have happened earlier. Summing this expression over T and dividing by $|T|$ yields the expected response time to the next emergency, given that at most one emergency can happen per time step. Note that dividing by $|T|$ makes no difference to the optimal solution, and we leave out the scalar $1/|T|$ for legibility. Therefore, the objective in MRP is to minimise the expected response time to the next emergency, under the condition that all jobs are processed and that all agents are at their end location at the end of the time horizon.

2.3.1 Example instance

As MRP is the central problem of this thesis, the model is further clarified here by means of a visual example.

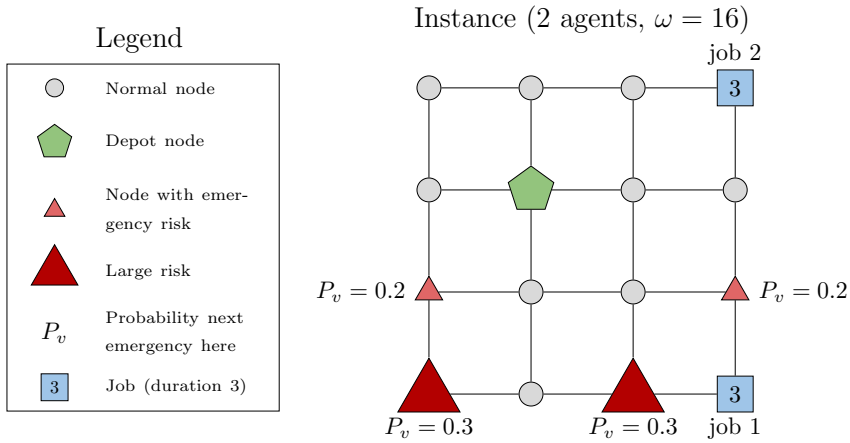


Figure 2.1: An example instance of MRP. Two agents, starting at a depot at $t = 0$, must process all jobs and be back at $t = 16$, whilst minimising average expected emergency response time. The optimal solution is presented in Figure 2.2.

In the MRP example illustrated in Figure 2.1, we may move two agents discretely over a network. They both start and end their work shift at a central node, which acts as a classical 'depot'. At some of the nodes, an emergency may

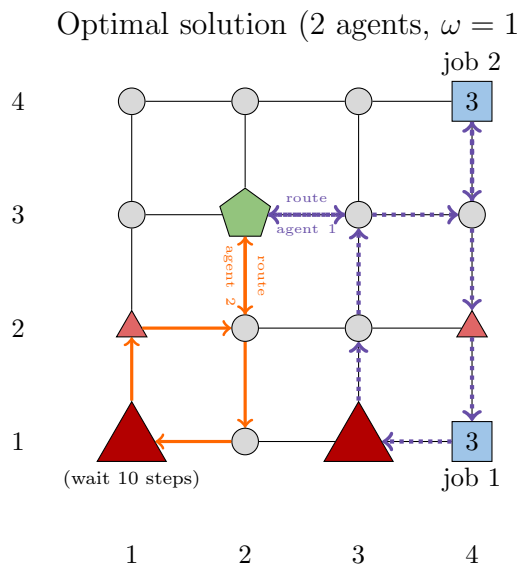


Figure 2.2: The optimal solution to the instance in Figure 2.1, detailed further in Table 2.2. It is optimal to let agent 1 (purple, dotted) process both jobs and be responsible for the 'right half' of the network, and to put agent 2 (orange, solid) in a good position to respond to emergencies in the 'left half'.

occur; each of these nodes has a probability P_v of being the site of the next emergency. There are also two jobs to be processed at specific locations in the 'right half' of the network, each with a processing time of 3. A feasible solution tells all agents where to be at each time step and when to start processing which jobs. The goal is to move the agents such that the weighted distance of all nodes to their nearest agent is minimised, where weights may represent emergency probabilities and distances may represent response times, while ensuring that all jobs get done. In the optimal solution, presented in Figure 2.2, agent 1 processes both jobs, thus giving 'coverage' to the right half of the network; meanwhile, agent 2 processes no jobs but moves to a good position to give coverage to the 'left half' of the network. Both agents make sure to be back at the depot just at the end of the shift.

In the optimal solution, at $t = 3$, there is one agent at (1,1) and one agent at (4,4). This means that the expected response time (assuming Manhattan distances) for an emergency at $t = 3$ equals

$$0.3 \cdot 0 + 0.3 \cdot 2 + 0.2 \cdot 1 + 0.2 \cdot 2 = 1.2$$

as this is, summed over the four possible emergency locations, the probability of the emergency occurring there times the distance of the nearest agent.

Time-step	Location agent 1	Location agent 2	Expected response time
$t = 0$	(2,3), depot	(2,3), depot	2.8
$t = 1$	(3,3)	(2,2)	1.8
$t = 2$	(4,3)	(2,1)	1.2
$t = 3$	(4,4), start job 2	(1,1), wait	1.2
$t = 4$	(4,4), process job 2	(1,1), wait	1.2
$t = 5$	(4,4), process job 2	(1,1), wait	1.2
$t = 6$	(4,4), job 2 finished	(1,1), wait	1.2
$t = 7$	(4,3)	(1,1), wait	1
$t = 8$	(4,2)	(1,1), wait	0.8
$t = 9$	(4,1), start job 1	(1,1), wait	0.7
$t = 10$	(4,1), process job 1	(1,1), wait	0.7
$t = 11$	(4,1), process job 1	(1,1), wait	0.7
$t = 12$	(4,1), job 1 finished	(1,1), wait	0.7
$t = 13$	(3,1)	(1,1)	0.6
$t = 14$	(3,2)	(1,2)	0.8
$t = 15$	(3,3)	(2,2)	1.8
$t = 16$	(2,3), depot	(2,3), depot	2.8
Average			1.247 (optimal)

Table 2.2: A detailed description of the solution illustrated in Figure 2.2.

The solution in Figure 2.2 is feasible because it visits all jobs and everyone is back in time; it is optimal because it has minimal average expected response time among feasible solutions.

Note that it is by no means necessary in MRP that the start and end location for all agents are the same: this is simply the case for this example. In fact, allowing any node to be an agent's start location is necessary to allow for dynamic re-solving after an emergency occurs. Also, in this example, the response time between nodes is equal to the number of steps needed to get there via the graph; in general, response times may follow a different metric.

2.3.2 Discussion of modelling choices

Two non-trivial modelling choices have been made in formulating MRP.

Remark 1. Discretising space-time allows us to use linear optimisation on an approximation of continuous movement. Furthermore, it facilitates legible day plans as output, and we can always reduce the lost accuracy to acceptable levels by discretising more finely, at the cost of more computational effort. One may interpret this discretisation as having agents move around over the set of potential facility locations in an instance of K-MED. Though one could also take the DVRP perspective of directing agents over jobs, rather than over discretised nodes, we believe this would create too much inaccuracy in expected response times when agents traverse long roads from one job to another. As a consequence of a discrete

space-time model, it is possible for agents to choose non-shortest roads between jobs if these give better response times, and to roam the network freely for the sake of coverage in their remaining time: these things would also not be possible when simply routing over jobs.

Remark 2. MRP seeks to minimise the expected response time to the next emergency, but in no way captures the actual processing of emergency events. Instead, when an emergency actually occurs and agents are deployed, it is advised to make a new planning for the rest of the shift and the remaining jobs by observing a new MRP instance, in which the remaining agents have their current location as their start location. In this model, we thus only prepare for the next emergency with optimal expected response time and re-optimise whenever it actually occurs. This 'single coverage'-approach is in a sense similar to a rolling horizon approach, where we keep the uncertainty tractable by only looking so far ahead, except we look towards the next emergency rather than towards some rolling horizon.

2.3.3 Complexity

We discuss two complexity results in this section that guide the design of our solution approaches. First, we remark that MRP on a complete graph without jobs is equivalent to K-MED, implying that MRP is NP-hard. In practice, one could view MRP as much harder than K-MED, because it involves solving $(\omega - 1)$ instances of K-MED, where the decision in any one instance influences the decision space of all other instances. More importantly, we have the following stronger complexity result.

Theorem 1. *Deciding whether an instance of MRP admits a feasible solution is NP-complete in general, even for the case with one agent.*

Proof. First, note that the problem of deciding whether a feasible solution exists for an MRP instance is in NP, because any feasible solution can be stored and checked in polynomial time and size with respect to the input. Next, take any instance of the Hamiltonian Path problem: that is, observe some connected graph with n nodes, some start node S and some end node W; without loss of generality, assume $S \neq W$. It is NP-complete to decide whether this graph admits a Hamiltonian (S, W)-path [52]: that is, an (S, W)-path that visits all nodes exactly once. Transform this into an instance of MRP by observing the same graph, placing a job of length 0 on each node, setting $\omega = n$ and having $|A| = 1$ agent start at S and end at W. Because $\omega = n = |V|$, the only way the agent can reach node W at time ω and process all jobs is if the agent visits all nodes at least once and never twice. Therefore, every feasible solution of this MRP instance corresponds to a Hamiltonian Path, meaning it is NP-complete to decide whether this MRP instance admits a feasible solution. □

Corollary 1. *Unless P=NP, there exists no polynomial-time algorithm that is guaranteed to return a feasible solution if one exists. In particular, no polynomial-time approximation algorithms exist for MRP.*

These results validate the following design choices: that heuristics are needed for MRP, and that these heuristics must contain NP-hard problems themselves if they are to always return a feasible solution.

2.4 Methods

In this section, we describe a solution method and several heuristics for MRP.

2.4.1 Mixed Integer Linear Programming

Denote $\Gamma := \{(j,k) \in J^2 : j \neq k, L_j = L_k\}$. Then in the notation already presented, the MRP can be formulated as the following *Mixed Integer Linear Program* (MILP):

$$\min \sum_{t\in T}\sum_{v\in V_P} P_v \sum_{u\in V} C_{uv} y_{uvt}$$

$$s.t.\ x_{aS_a 0} = 1 \qquad (\forall a \in A) \tag{2.1}$$

$$x_{aW_a\omega} = 1 \qquad (\forall a \in A) \tag{2.2}$$

$$\sum_{v\in V} x_{avt} = 1 \qquad (\forall a \in A)(\forall t \in T) \tag{2.3}$$

$$x_{av(t+1)} \leq \sum_{u\in V_v} x_{aut} \qquad (\forall a \in A)(\forall v \in V)(\forall t \in T\backslash\{\omega\}) \tag{2.4}$$

$$\sum_{u\in V} y_{uvt} = 1 \qquad (\forall v \in V_P)(\forall t \in T) \tag{2.5}$$

$$y_{uvt} \leq \sum_{a\in A} x_{aut} \qquad (\forall u \in V)(\forall v \in V_P)(\forall t \in T) \tag{2.6}$$

$$\sum_{a\in A}\sum_{t\in T} z_{ajt} = 1 \qquad (\forall j \in J) \tag{2.7}$$

$$\sum_{\tau=t}^{t+Q_j} x_{aL_j\tau} \geq (Q_j+1) z_{ajt} \qquad (\forall a \in A)(\forall j \in J)(\forall t \in T) \tag{2.8}$$

$$z_{ajt} + \sum_{\tau=t}^{t+Q_j-1} z_{ak\tau} \leq 1 \qquad (\forall (j,k) \in \Gamma)(\forall a \in A)(\forall t \in T) \tag{2.9}$$

$$x_{avt}, z_{ajt} \in \{0,1\},\ y_{uvt} \in [0,1]$$

Here, the objective equals the expected response time to the next emergency (up to a scalar $|T|$, as explained in Section 2.3).

Constraint (2.1) states that all agents must start at their start location, and that they must end at their end location. Constraint states that an agent can only be in one place at a time.

Constraint (2.4) states that an agent can only be at node v at time $t+1$ if he or she was at some adjacent node u at time t, where the adjacency is indicated by whether or not $u \in V_v$.

Constraint (2.5) states that each emergency node, at each time step, must receive coverage from somewhere. Constraint (2.6) adds, however, that coverage at time t can only be given from some node $u \in V$ if there is someone actually present at node u at time t.

Constraint (2.7) states that each job must be initiated by someone at some point in time. Constraint (2.8) adds that when an agent starts a job j, that agent must stay at location L_j for the duration of the job. The formulation also ensures that the job is started early enough to be finished before the end of the time horizon. For jobs that are in different places, this implies that they cannot be processed at the same time by the same agent.

Recall, however, that we allow for multiple jobs to be hosted at the same node. Suppose some pair of jobs $j \neq k$ exists at the same location $L_j = L_k$. Constraint (2.8) does not forbid one agent to process them simultaneously. We thus need constraint (2.9) to address this fringe case. Suppose some agent a wishes to process j at some time t. The constraint

$$z_{ajt} + \sum_{\tau=t}^{t+Q_j-1} z_{ak\tau} \leq 1$$

then states that a cannot also start processing k at any time between t and $t + Q_j - 1$: k can only be processed after j is done or before j is started. If k is started very briefly before t, say at $t' := t - 1$, then the constraint

$$z_{akt'} + \sum_{\tau=t'}^{t'+Q_k-1} z_{aj\tau} \leq 1$$

ensures that j is not started at t any more. Due to this symmetry, constraint (2.9) ensures that if two jobs are at the same location, an agent cannot process them simultaneously.

Though the variables x_{avt} and z_{ajt} are explicitly constrained to be binary, this is not necessary for the variables y_{uvt}. The reason is as follows. Any feasible solution has $\mathbf{x}$ binary. Therefore, any optimal solution obviously has $y_{uvt} = 1$ for the closest $u \in V$ to a given $v \in V$ that has someone present at $t \in T$. If several nodes are tied for closest, then dividing the coverage fractionally over these nodes would still give a feasible solution with optimal solution value, but breaking the tie arbitrarily would result in a feasible solution with strictly fewer basic variables, meaning the original fractional solution cannot be a vertex of the solution polytope. We conclude that the variables y_{uvt} can be formulated as continuous between 0 and 1 without fear of non-integral optimal solutions. This is fortunate, as the variables y_{uvt} comprise the vast majority of the variables. On a random sample of benchmark instances (which are further described in Section 2.5), this indeed yields an average reduction of 29.1% in computation time.

Though simply plugging this Mixed Integer Linear Program into a *Mixed Integer Linear Programming solver* (MILP solver) will eventually yield the optimal solution, this approach can come with long and unpredictable computation times as the instance size grows. This is indeed observed in Section 2.6 for the more difficult instance classes.

2.4.2 `WAIT-AT-MEDIANS`-heuristic

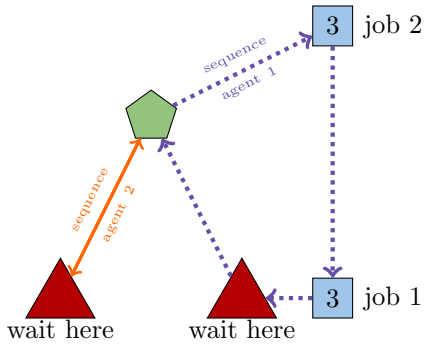


Figure 2.3: The idea behind the `WAIT-AT-MEDIANS`-heuristic, applied to the example in Figure 2.1. First, solve K-MED, which results in identifying the large red triangles as the best places from which to offer emergency response. Then, solve a variant of DVRP to minimise the total time spent travelling, so as to maximise the total time spent waiting at the medians. The sequences thus obtained are shown here; they must still be translated back to a feasible solution in the discretised setting, but this can be done with ease.

In practical applications, solving the MILP proposed in Section 2.4.1 may take excessively long. As an alternative, we discuss some heuristics here, starting with the following simple one.

If there is a large gap of time in which agents do not process jobs, for example when the instance has almost no jobs at all, then the optimal place for the agents to be is at those places given by the solution of K-MED. It may sometimes be costly or infeasible to reach those places within the time window. If the amount of remaining time goes to infinity, however, it will be feasible to reach that steady state, and any costs incurred while getting there are outweighed by the saved cost of being optimally distributed over a long period of time.

One heuristic strategy could be to identify these medians, and spend as much time as possible at these medians. This, indeed, is proposed by the `WAIT-AT-MEDIANS`-heuristic (`WAM`), described by Algorithm 1 and illustrated in Figure 2.3. `WAM` requires solving K-MED and a special variant of DVRP, which are NP-hard problems in their own right; these subroutines themselves may be

approached with heuristics, though the results in Section 2.6 suggest that the subroutines are easy enough to solve to optimality for the studied benchmark instances. Note that, in practice, step 1 and part of step 2 can be done as preprocessing, assuming the network is known beforehand but the daily tasks are not. Though step 2 seems trivial, we remark that computing a distance matrix is not necessary for the MILP in Section 2.4.1, and we include this step in the description of `WAM` for the sake of fair computational comparison.

ALGORITHM 1: `WAIT-AT-MEDIANS`, high-level overview

1: Solve induced K-MED, with $k = |A|$
2: Obtain shortest paths between jobs, medians, starts and ends
3: Solve DVRP with one median and (S_a, W_a) per route
4: Infer $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$, with waiting done only at medians

More in-depth, `WAM` consists of the following:

1. **Solve induced K-MED, with $k = |A|$.** This can be done by solving the MILP formulated by Charikar et al. [25], where the distances are given by C_{uv} and the nodes have weight P_v if they are in V_P and 0 otherwise. Denote the obtained medians $\mathcal{M} \subseteq V$.

2. **Obtain shortest paths between jobs, medians, starts and ends.** This can be done in polynomial time using Dijkstra's algorithm [37] or the Floyd-Warshall algorithm [31]. Denote $D_{ij} = D_{ji}$ the minimum number of steps needed to get from any job, median or start location i to any job, median or end location j.

3. **Solve DVRP with one median and (S_a, W_a) per route.** This can be done by solving the MILP in A.1. The result is a sequence for each $a \in A$, starting at S_a and ending at W_a, such that the sequences together visit all jobs, each median is visited by exactly one agent, each sequence admits a feasible execution with respect to the finite time horizon, and the total time spent travelling is minimised. This allows us to spend as much time as possible on waiting at medians. Note that, if there is enough time to visit all jobs but not enough time to also visit all medians, this subroutine fails.

4. **Infer x, y and z, with waiting done only at medians.** The sequences from the previous step can be translated to a feasible MRP solution easily. For each agent, observe the sequence from step 3, and demand that movement between any two goals follows the shortest path from step 2. If this requires strictly less time than ω, allocate all remaining time to waiting at the median. This directly implies the values of x_{avt} and z_{ajt}. After thus fixing $\mathbf{x}$ and $\mathbf{z}$ completely, set y_{uvt} as follows: for any $v \in V$, $t \in T$, find the closest $u \in V$ with respect to C_{uv} which has someone present, so which has $\sum_{a \in A} x_{aut} > 0$, then set $y_{uvt} = 1$.

Though this heuristic seems intuitive, it comes with some immediately apparent downsides:

- It does not explicitly take coverage into account while routing over jobs, aside from creating as much median-waiting time as possible.
- It does not explicitly take coverage into account when translating routes back to the discrete network; it instead follows arbitrary shortest paths.
- Agents do not take into account where the other agents are.
- There exist feasible MRP instances where this heuristic does not produce a feasible solution, namely when there is not enough time to actually visit all medians.

2.4.3 `MEDIATE-DIVIDE-SEQUENCE-AGREE`-heuristic

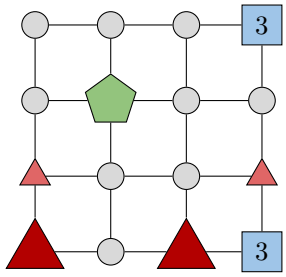
Observing the shortcomings of `WAM`, a heuristic is presented here which attempts to overcome the shortcomings. The `MEDIATE-DIVIDE-SEQUENCE-AGREE` (`MDSA`) heuristic is presented here as Algorithm 2 and illustrated in Figure 2.4.

ALGORITHM 2: `MEDIATE-DIVIDE-SEQUENCE-AGREE`, high-level overview

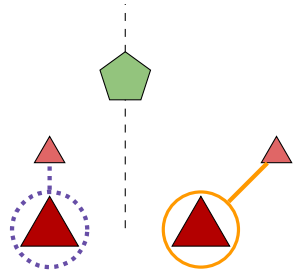
1: Solve induced $|A|$-MED, obtain coverage regions (`MEDIATE`)
2: Obtain shortest paths between jobs, medians, starts and ends
3: Assign medians optimally to nearest agents
4: Solve 'job division' (`DIVIDE`)
5: For each agent, solve path-TSP, take clockwise solution (`SEQUENCE`)
6: For each agent, given the sequence and region, find cheapest space-time path
7: Forget medians, refine to time-dependent regions (`AGREE`)
8: For each agent, given the sequence and time-dependent region, find cheapest space-time path
9: Infer $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$

In more detail, `MDSA` consists of the following steps:

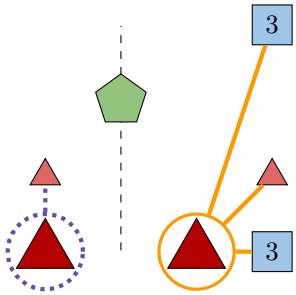
1. **Solve induced $|A|$-MED, obtain coverage regions (`MEDIATE`).** This step is identical to the one in `WAM`, resulting in a set $\mathcal{M}$ of medians. For any $m \in \mathcal{M}$, denote coverage region $V_m \subseteq V_P$ the nodes for which m is the nearest median, breaking ties arbitrarily.
2. **Obtain shortest paths between jobs, medians, starts and ends.** This step is again identical to the one in `WAM`. Denote again $D_{ij} = D_{ji}$ the minimum number of steps needed to get from any job, median or start location i to any job, median or end location j.
3. **Assign medians optimally to nearest agents.** If all agents start and end at one location, like a classical depot, then medians can be assigned



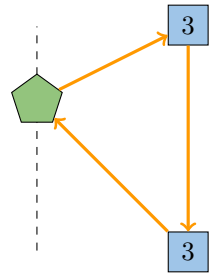
(a) The instance as depicted in Figure 2.1.



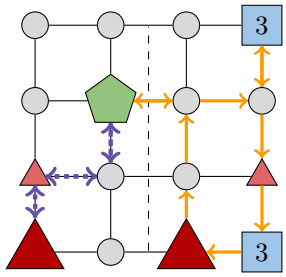
(b) Step 1, 2, 3: Solve K-MED, and split the instance into regions. Assign regions to agents. (`MEDIATE`)



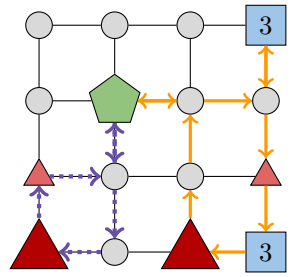
(c) Step 4: Try to assign jobs to the nearest median and its agent. (`DIVIDE`)



(d) Step 5: For each agent, solve path-TSP over the assigned jobs, taking the 'clockwise' solution if possible. (`SEQUENCE`)



(e) Step 6: Find the cheapest path that follows these sequences, with respect to the assigned coverage regions.



(f) Step 7, 8, 9: Check where agents are over time, and redetermine the cheapest paths with respect to where the other agents are. (`AGREE`)

Figure 2.4: An illustration of the `MEDIATE-DIVIDE-SEQUENCE-AGREE`-heuristic.

arbitrarily to agents. Otherwise, each median m has an average distance $(D_{S_a m} + D_{m W_a})/2$ to the start and end point of a given agent $a \in A$, and we can find the optimal assignment of medians to agents in polynomial time using the Hungarian algorithm [91]. Denote by $m(a)$ the median assigned to a, and abbreviate $V_a = V_{m(a)}$.

4. **Solve 'job division' (`DIVIDE`).** In this step, each job is assigned to its nearest median and the corresponding agent, as well as possible. That is, for any $a \in A$, $j \in J$, denote the proxy cost of assigning j to a as $F_{aj} = (Q_j + 1) \cdot \sum_{v \in V_a} P_v C_{L_j v}$; this quantity represents the cost incurred as a processes j, under the assumption that the nodes covered by a are always exactly V_a. Blindly assigning jobs to their nearest median may result in an agent getting more jobs than feasibly executable. Instead, we find the feasible division of jobs over agents with minimal sum F_{aj} as follows: we again solve the MILP in A.1, except that we treat $\mathcal{M}$ as being empty, and we replace the objective with $\sum_{a \in A} \sum_{j \in J} F_{aj} z'_{aj}$. Note that, in contrast to `WAM`, this subroutine does *not* fail when there is not enough time to visit all medians. Denote $J_a \subseteq J$ the jobs assigned to agent $a \in A$.

5. **For each agent, solve path-TSP, take clockwise solution (`SEQUENCE`).** Each agent $a \in A$ now has a set of jobs J_a assigned to him or her. In this step, we decide in which sequence these jobs are visited. We do so by solving the (S_a, W_a)-path Travelling Salesman Problem over J_a, as described in A.2, for each $a \in A$.

   If $S_a = W_a$, then the found sequence in reverse is also optimal. Of these two optimal sequences, we choose the *clockwise* one, which we define as follows. Denote (X_s, Y_s) the 2D-coordinates of S_a. For any $j \in J_a$, denote (X_j, Y_j) the 2D-coordinates of the job location L_j, and define the *angle* of j as $\text{atan2}(Y_j - Y_s, X_j - X_s)$, where $\text{atan2}(y, x)$ is a commonly used function to compute the geometric angle between the vector (x, y) and the vector $(1, 0)$ [36]. When i is followed up by j, we define this move to be *clockwise* if i has a greater angle than j. Of the two optimal sequences, we choose the one that has the largest number of clockwise moves.

6. **For each agent, given the region and sequence, find cheapest space-time path.** If an agent $a \in A$ is assumed to give coverage to a specific set of nodes V_a, then each $u \in V$ has a cost of a being there for one time step, namely, $\sum_{v \in V_a} P_v C_{uv}$. Based on these node costs, we compute for each $a \in A$ the cheapest path starting at the space-time point $(S_a, 0)$ and ending at the space-time point (W_a, ω), such that the jobs J_a are visited in the predetermined sequence. This can be done using a Dynamic Program, Algorithm 12 in A.3, which is essentially a modification of Dijkstra's algorithm [37]. This results in temporary values $\tilde{x}_{avt}$ describing the movement of the agents as they visit the jobs, while trying to keep optimal coverage over V_a. In particular, if $J_a = \emptyset$ for some $a \in A$ and ω is large

enough, then this step results in a moving to median $m(a)$ and staying until it is time to go to end point W_a.

7. **Forget medians, refine to time-dependent regions (`AGREE`).** Up until this point, we assumed that $a \in A$ would always cover a fixed area V_a, so that a starting solution $\tilde{x}_{avt}$ could be constructed. In this step, we define more finely tuned, time-dependent coverage regions V_{at}, by observing for each $v \in V_P$ and $t \in T$ which $a \in A$ is 'nearest'; that is, which $a \in A$ has minimal C_{uv}, where u is the location of a at time t according to the movement $\tilde{\mathbf{x}}$. This leads to the sets V_{at}, which at each time step partition V_P over the nearest agents.

8. **For each agent, given the sequence and time-dependent region, find cheapest space-time path.** We repeat step 6, but with the node cost of $a \in A$ being at $u \in V$ at time $t \in T$ equal to $\sum_{v \in V_{at}} P_v C_{uv}$, rather than $\sum_{v \in V_a} P_v C_{uv}$. This results in final movement values x_{avt}.

9. **Infer x, y and z.** The previous step has produced final values for movement x_{avt}. The corresponding values for z_{ajt} can be easily inferred from Algorithm 12. The values for y_{uvt} can again be determined by checking for each $v \in V_P$ and $t \in T$ which $u \in V$ with someone present has lowest C_{uv}.

The intuition of `MDSA` is less transparent from the algorithm. It is illustrated in Figure 2.4 and described here.

As in `WAM`, we observe that as $\omega \to \infty$, it is optimal to have the agents spend their remaining time at the medians. Therefore, in step 1, we determine where the medians are. In step 2, we obtain the distances between the points of interest. In step 3, we use this to assign the $|A|$ medians to their most suitable agent, with respect to start and end locations S_a and W_a. For the majority of the algorithm, this not only assigns each agent a median, but also the region covered by that median: all agents now have a region $V_a \subseteq V_P$ that they are 'responsible' for.

In step 4, each job is assigned to its nearest median and handled by that median's agent if possible, so that all agents can try and stay in 'their' region; if this leads to too many jobs for one agent, the MILP in step 4 instead tries to divide jobs such that agents can stay as close to 'their' region as possible.

As soon as jobs are divided over the agents, each agent then determines in step 5 in which sequence the jobs will be visited. This is done by solving path-TSP, following again the logic that any time step saved can be used to improve coverage. We synchronise all agents to move 'clockwise' if possible, so that whenever one agent moves away from a node, another will hopefully already be approaching to take over coverage.

In this discretised setting, there may exist many shortest paths between any two nodes. By taking the cheapest space-time path in step 6 with respect to coverage over V_a, the ties between these shortest paths are broken sensibly rather than arbitrarily. In fact, this also allows non-shortest paths with better coverage to be chosen. Moreover, following the cheapest coverage paths is a more robust

way of 'visiting the medians', rather than explicitly demanding they be visited: if there is not enough time, the cheapest coverage path will likely only approach the median the best it can.

However, assuming that some $a \in A$ will always be the most appropriate agent to cover every node in V_a, is a somewhat crude assumption. As agents visit jobs in a clockwise fashion, it may well be that nodes have one agent closest at one point and another at another point. Now that an initial movement profile $\tilde{\mathbf{x}}$ has been created, this can be used to determine time-dependent coverage regions in step 7 that are more fine and realistic than the coverage regions based on the medians. Finding the cheapest space-time paths based on these finer regions in step 8 produces a final movement profile in which agents actually observe where the other agents are at a given time, albeit that they look at $\tilde{\mathbf{x}}$ and hope that the other agents do not deviate too much from $\tilde{\mathbf{x}}$.

We remark that, if the MRP instance admits a feasible solution, then `MDSA` finds a feasible solution as well. The argument is as follows. If a feasible solution exists, then the MILP in step 4 will succeed in finding a feasible way to divide jobs over agents. By construction of step 4, every instance of path-TSP in step 5 admits a feasible solution, which will be found by the MILP. By the feasibility of step 5, the Dynamic Programs in the remaining steps will terminate successfully.

This guaranteed feasibility is one of the ways in which `MDSA` improves over `WAM`. Furthermore, `MDSA` prioritises coverage over travel time when dividing jobs, it breaks ties between fastest (and even non-fastest) paths by taking the cheapest ones with respect to a coverage profile, and it allows agents to respond to what the other agents have done in the final steps.

We acknowledge that this heuristic involves solving some NP-hard problems, as well as pseudo-polynomial Dynamic Programs with a running time that depends on ω. However, we observe in Section 2.6 that `MDSA` needs mere seconds to run for the studied instances. An explanation for this speed is that, aside from step 1, the NP-hard subproblems involve routing over a relatively tiny set of jobs, rather than over a network with many nodes. In fact, one could say that the point of `MDSA` is that it first takes the most restricting decisions on a very small decision space with proxy costs, and then makes the best out of those decisions in pseudo-polynomial time. If one still chooses to replace the complex subroutines by heuristics, the guaranteed feasibility may be lost. As long as the bottleneck in studying heuristics for larger instances of MRP lies in finding the optimal solutions to compare against, developing a fully polynomial version of `MDSA` is left as a point for future research.

2.4.4 Partial versions of `MDSA`

The `MDSA`-heuristic involves a number of complex steps, the added value of which may not be immediately clear. To investigate this, the heuristic was modified slightly to allow switching steps on and off. This resulted in the heuristics `MDS`, `MDA`, `DSA`, `MD`, `DS`, `DA` and `D`, where for example `MDA` performs all steps except

the SEQUENCE-step. That is to say, these seven heuristics are identical to MDSA, except:

- If 'M' is not in the name, the MEDIATE-step is skipped, or in particular step 1 and step 3 are skipped. It also implies that in step 4, a normal DVRP is solved, so simply the travelled distances are minimised; that step 5 can be skipped because of this; and that in step 6, translation back to the discrete graph is done by means of shortest paths, rather than paths that give the best coverage to a predetermined region.
- If 'S' is not in the name, the SEQUENCE-step is skipped. The sequences in which agents visit their jobs are then taken directly from step 4.
- If 'A' is not in the name, the AGREE-step is skipped. No re-optimisation to observed coverage regions is done, or put simply, steps 7 and 8 are skipped.

Note that the DIVIDE-step is never skipped, as the division of jobs over agents cannot be arbitrary. This procedure of skipping steps produces heuristics that are faster and conceptually easier, but on average perform worse, as can be observed in Section 2.6. In particular, D simply performs the jobs and returns to the depot as quickly as possible. We view this as a benchmark heuristic, that models how agents would move if they only cared about getting their jobs done as quickly as possible.

2.5 Experimental setup

In Section 2.4, we discussed a solution method and several heuristics for MRP. To compare how well these work in practice, it would be insightful to apply them to a set of benchmark problems. Furthermore, it would be of interest to see what typical solutions look like. In Section 2.2, however, we concluded that the MRP or similar problems have received little academic attention. To the best of our knowledge, benchmark instances for this problem did not yet exist prior to this research. Therefore, we created benchmark instances from both a case study and from an instance generator and compared the methods on these.

The code used to generate the benchmark instances is publicly available, as is the code to perform the experiments.

2.5.1 Used instances

We compared the methods on benchmark instances from two sources.

First, we obtained six benchmark instances from a case study with a European railway infrastructure manager. These six instances, denoted I_R, are defined on the same piece of the railway network, with emergency probabilities based on historical incident data. The jobs have been sampled from a database of typical jobs for this area. The six instances differ in which jobs have been sampled and

how many. The instances have 143 nodes, 4 agents and $\omega = 16$. They have 3, 4, 5, 7, 8 and 9 jobs, respectively.

Second, Algorithm 13 was developed to generate benchmark instances, in order to also study algorithmic behaviour under variation of problem features. The idea is simple: randomly draw node coordinates on the Euclidean plane, connect them if their Euclidean distance is under a given threshold, let all agents start and end at a central depot, scatter jobs randomly over the network and check if the result admits a feasible solution. Because these ideas are simple and the created instances are publicly available, further details on how the benchmark instances were generated have been moved to A.4.

The classes we created are described in Table 2.3. The details of their parameters are described in A.4, but their features roughly represent the following:

- 'Small' problems have 3 agents and 20 nodes, where 'medium' problems have 4 agents and 100 nodes;
- 'Productive' problems have more than two jobs per agent, where 'unproductive' problems have fewer than one;
- 'Sparse' problems have smaller node neighbourhoods than 'dense' problems, meaning it will typically take more time steps to get from any one node to another.

Class	Amount	Type	Size	Productivity	Sparseness
I_1	50	random	small	productive	sparse
I_2	50	I_1-derived	small	unproductive	sparse
I_3	50	I_1-derived	small	productive	dense
I_4	50	I_1-derived	small	unproductive	dense
I_5	50	random	medium	productive	sparse
I_6	50	I_5-derived	medium	unproductive	sparse
I_7	50	I_5-derived	medium	productive	dense
I_8	50	I_5-derived	medium	unproductive	dense
I_R	6	real-life	$\pm$ medium	both	$\pm$ dense

Table 2.3: Brief description of the benchmark instance classes used.

Note that the medium size problems are of a comparable size to the real-life instances in class I_R.

In order to purely observe the differences in performance under variation of features, only classes I_1 and I_5 were randomly generated: the unproductive problems were created from the productive problems by deleting jobs, and the dense problems were created from the sparse problems by updating adjacency for a higher adjacency threshold. By construction, these operations preserve feasibility of the instances.

2.5.2 Metrics and methods for solution structure

For each instance and method, we computed the expected response time and the computation time. On a selection of methods and a random sample of the instances, we also computed three metrics that give more insight into the structure of solutions.

First, we compare the response time to how much it would have been if there had been no jobs at all. This indicates how much 'response power' we have sacrificed to do jobs. Second, the solutions from the different methods are compared by total travel distance, measured in the total number of 'hops'. Third, making many tiny steps back and forth to continually compensate the movement of others can yield a marginal cost improvement, but may be irritating for the agent. To measure how 'jittery' a solution is, we also counted the total number of *shortcuts*, where a shortcut is defined as any occurrence of an agent visiting the distinct locations u, v and w at times t, $t+1$ and $t+2$ respectively, while the agent could also have gone from u to w directly. If the agent did so to process a job of length 0 at location v, this is not counted as a shortcut. This definition of a shortcut also accounts for an agent moving towards a destination at an irritatingly slow pace.

To make a more meaningful comparison of these metrics, we applied two additional solution methods to these instances. `Optimal jobless` means we delete the jobs and run the MILP: this does not give a feasible solution, but does provide a lower bound, and illustrates what solutions would have looked like if we did not care about jobs. Recall that `D` illustrates what solutions would have looked like if we did not care about coverage and simply solved VRP. `Split fleet` means we cut the fleet in half, where the larger half is only concerned with jobs, and the smaller half is only concerned with coverage. We apply `D` to the larger half and `Optimal jobless` to the smaller half, with the two halves ignoring each other's existence. In the case of an emergency, however, the nearest agent is still called upon, regardless of what half they are in. This illustrates what solutions would look like if, with the same resources, we decided for simplicity not to coordinate emergency response and non-urgent job processing jointly. Note that this method may also fail to find a feasible solution. It is also possible, of course, to split the fleet into portions of unequal size, but we believe examining this fifty-fifty split should suffice for our goal of better understanding solution structures.

2.5.3 Hardware specifications

All experiments were conducted on the Lisa Cluster, a computing cluster hosted by Surfsara. Each benchmark instance was solved on its own 16-core 2.60 GHz node with 64 GB QPI 8.00 GT/s memory. In particular, experiments were queued until they could get a node of their own, meaning all processing power of the node was dedicated to the experiment, and that clock times correspond to CPU times. Here, an 'experiment' means either solving a benchmark instance with a MILP

solver, or applying all heuristics on it. Gurobi 8.0.1 was used as a MILP solver, using all 16 cores.

2.6 Results

The ten methods were applied to the nine classes of benchmark instances. The running time of these methods was recorded, as well as how large the gap was between the produced solution value and the optimal solution value, as a percentage of the optimal solution value. The full results are presented in Table 2.4, Table 2.5 and Table 2.6 For ease of reading, some of the entries in this table have been bold-faced: most notably, the best performing heuristics with respect to average solution quality and worst solution quality, and the computation times of the MILP and `MDSA`. A summarised version of these tables is presented in Table 2.7. Table 2.8 presents comparisons of the other metrics. From the results in these tables, we make several observations.

Method	number of feasible solutions	average gap (%)	average time (s)	worst gap (%)	worst time (s)
I_R	(real-life instances)				
`opt`	6	0	**1202.7**	0	2871
`WAM`	6	5.9	14.0	9.3	62.9
`D`	6	27.0	0.4	40.2	0.9
`MD`	6	4.4	3.7	8.0	4.3
`DS`	6	27.4	0.7	40.2	2.0
`DA`	6	13.6	0.8	24.6	1.3
`MDS`	6	4.4	3.9	7.4	4.4
`MDA`	6	**3.1**	4.4	**4.9**	5.3
`DSA`	6	12.3	0.9	24.6	1.4
`MDSA`	6	3.4	**4.5**	7.4	5.5

Table 2.4: Results of applying the ten solution methods on the six real-life instances.

In almost every class, `MDSA` is the best scoring heuristic in both average optimality gap and worst optimality gap. Class I_5 is the most difficult class to solve for Gurobi, at an average of 140 minutes. Here, `MDSA` supplies a solution that is on average 3.5% away from optimal in an average 4.1 seconds. Taken over all 406 instances, Gurobi needs an average of 39 minutes to solve MRP, while `MDSA` needs 2.4 seconds to find a solution that is 3.2% away from optimal. It appears that simultaneously timetabling jobs and managing coverage is difficult, but that decomposing these decisions into different decision stages makes for an effective heuristic.

Averaged over all instances, the benchmark algorithm `D` is 127.4% away from optimal. The average optimality gap increases as the problem becomes denser

Method	number of feasible solutions	average gap (%)	average time (s)	worst gap (%)	worst time (s)
I_1	(small, productive, sparse)				
opt	50	0	**29.1**	0	353
WAM	50	20.7	1.2	106.0	12.8
D	50	102.0	0.3	241.6	0.4
MD	50	9.3	0.4	35.3	0.8
DS	50	101.7	0.4	238.0	0.6
DA	50	30.0	0.3	85.3	0.5
MDS	50	8.4	0.6	**34.2**	0.9
MDA	50	8.0	0.5	35.3	0.8
DSA	50	32.9	0.4	85.3	0.6
MDSA	50	**7.0**	**0.6**	**34.2**	0.9
I_2	(small, unproductive, sparse)				
opt	50	0	**2.7**	0	9
WAM	50	10.5	0.3	80.8	0.5
D	50	167.2	0.1	314.6	0.1
MD	50	4.4	0.2	**25.7**	0.3
DS	50	167.2	0.1	314.6	0.2
DA	50	53.8	0.1	162.5	0.2
MDS	50	4.4	0.3	**25.7**	0.4
MDA	50	**3.5**	0.2	**25.7**	0.3
DSA	50	55.0	0.1	162.5	0.2
MDSA	50	**3.5**	**0.3**	**25.7**	0.5
I_3	(small, productive, dense)				
opt	50	0	**5.7**	0	48
WAM	50	29.4	3.2	140.5	16.6
D	50	180.7	0.2	314.5	0.4
MD	50	5.4	0.4	45.9	0.6
DS	50	180.6	0.3	314.5	0.5
DA	50	63.4	0.3	160.7	0.5
MDS	50	5.3	0.5	43.2	0.7
MDA	50	5.0	0.5	45.9	0.7
DSA	50	64.4	0.4	158.8	0.6
MDSA	50	**4.3**	**0.6**	**43.0**	0.8
I_4	(small, unproductive, dense)				
opt	50	0	**0.7**	0	5
WAM	50	8.0	0.3	56.6	0.6
D	50	258.9	0.1	520.5	0.1
MD	50	2.4	0.3	24.5	0.4
DS	50	258.9	0.1	520.5	0.2
DA	50	107.9	0.2	281.9	0.2
MDS	50	2.4	0.3	24.5	0.5
MDA	50	**2.0**	0.3	**24.2**	0.4
DSA	50	106.5	0.2	284.5	0.3
MDSA	50	**2.0**	**0.4**	**24.2**	0.5

Table 2.5: Results of applying the ten solution methods on the 200 small problems.

Method	number of feasible solutions	average gap (%)	average time (s)	worst gap (%)	worst time (s)
I_5	(medium, productive, sparse)				
opt	50	0	**8739.1**	0	132750
WAM	50	7.7	11.4	24.0	101.8
D	50	60.5	0.5	82.3	0.8
MD	50	4.7	3.2	14.5	7.1
DS	50	60.4	0.7	83.0	1.1
DA	50	30.3	0.9	49.8	1.6
MDS	50	4.5	3.4	14.6	7.3
MDA	50	3.7	3.9	14.4	8.3
DSA	50	29.4	1.0	54.0	1.6
MDSA	50	**3.5**	**4.1**	**14.3**	8.1
I_6	(medium, unproductive, sparse)				
opt	50	0	**1476.1**	0	12839
WAM	50	4.6	2.4	21.7	3.6
D	50	88.0	0.1	115.6	0.3
MD	50	2.3	2.7	18.7	3.3
DS	50	88.1	0.2	115.6	0.3
DA	50	51.1	0.5	80.8	0.8
MDS	50	2.0	2.7	12.1	3.3
MDA	50	**1.4**	3.2	**11.9**	4.1
DSA	50	51.1	0.5	80.8	0.8
MDSA	50	1.5	**3.3**	**11.9**	3.9
I_7	(medium, productive, dense)				
opt	50	0	**8390.3**	0	72703
WAM	50	11.3	**438.5**	35.1	4749.7
D	50	73.9	1.1	101.9	6.0
MD	50	3.9	3.6	14.3	5.0
DS	50	74.0	1.2	101.9	6.1
DA	50	39.1	1.8	63.4	7.0
MDS	50	4.0	3.9	14.1	5.2
MDA	50	**2.8**	5.1	13.8	7.8
DSA	50	38.4	1.9	63.4	7.2
MDSA	50	**2.8**	**5.4**	**13.3**	7.8
I_8	(medium, unproductive, dense)				
opt	50	0	**168.3**	0	2963
WAM	50	4.1	2.6	12.6	3.5
D	50	100.1	0.2	128.1	0.6
MD	50	1.3	3.5	6.1	3.8
DS	50	100.0	0.2	128.1	0.4
DA	50	58.8	1.0	103.4	1.2
MDS	50	1.3	3.5	6.1	4.0
MDA	50	**0.9**	4.5	**5.1**	6.6
DSA	50	59.0	1.0	112.6	1.3
MDSA	50	**0.9**	**4.6**	**5.1**	5.2

Table 2.6: Results of applying the ten solution methods on the 200 medium-sized problems.

Method	average gap (%)	average time (s)	worst gap (%)	worst time (s)
opt	0	2334.5	0	132750.0
WAM	11.7	56.8	140.5	4749.7
D	127.4	0.3	520.5	6.0
MD	4.2	1.8	45.9	7.1
DS	127.4	0.4	520.5	6.1
DA	53.7	0.6	281.9	7.0
MDS	4.0	1.9	43.2	7.3
MDA	3.4	2.3	56.6	8.3
DSA	54.0	0.7	284.5	7.2
MDSA	3.2	2.4	43.0	8.1

Table 2.7: Summarised results of applying the ten solution methods on the 200 small instances, 200 medium instances and 6 real-life instances.

and less productive. This is easily explained: in D, all agents go to their end point as fast as possible and stay there. In the benchmark instances, this end point is the same for all agents, meaning that near the end of the time horizon, everyone is in the same place, which is bad for coverage. The more unproductive the problem is and the denser the network, the sooner in the time horizon this occurs.

There is a large difference between the average optimality gap of heuristics D and MDSA. Of this difference, examining marginal contributions shows that 70.0% is due to step M, 29.95% is due to step A and 0.05% is due to step S. For the difficult class I_5, the absence of M, S and A explain 71.8%, 0.4% and 27.8% of the difference respectively. In the case study instances I_R, these marginal contributions are on average 65.6%, −0.7%, and 35.1%, respectively.

The added value of step S is apparently small in the observed instances. Unfortunately, the average added value of S is even negative in the case study instances, though this is caused largely by one outlier. In particular, on a random sample of the instances, the clockwising feature gives a strict improvement in only 13.3% of the cases, and the average objective improvement is less than 0.01%. On the studied instances, clockwising may not be worth its implementational effort. The ineffectiveness of S is understandable for unproductive problems, where by construction agents are unlikely to get more than one job in the first place, so there is not much to re-sequence. In general, we expect the added value of S to increase with the average number of jobs per agent. We note, moreover, that the added value of step S is at its largest in productive, sparse problems, which are the most difficult to solve in terms of Gurobi time.

The added value of step A is more immediately apparent. Comparing heuristic D to DA or DS to DSA indicates that in step A, taking routes that are good for coverage rather than generic shortest routes has a significant amount of added value. However, because step M is skipped, the coverage profiles are based on earlier movement, and the earlier movement still sends everyone back to the

depot as quickly as possible, meaning that near the end of the time horizon, only one of the agents bunched up at the depot is given a coverage instruction in step `A`. Indeed, we see that the difference between `DSA` and `MDSA` becomes larger as the problem becomes denser and less productive, meaning agents bunch up at the depot earlier. Therefore, the difference between `MDS` and `MDSA` is much smaller: because of step `M`, agents have prioritised staying as well positioned in a sub-region as possible over being back at the depot as quickly as possible.

This may well explain the large added value of step `M`: it gives agents a way to use their remaining time more fruitfully than bunching up at the depot, and this way of spending remaining time is clearly optimal as the amount of remaining time goes to infinity. However, this alone is not a sufficient explanation for the success of `MDSA`, as `WAM` operates by the same logic but does not perform as well. First, it must be reiterated that `WAM` does not always succeed at finding a feasible solution because it may not always be possible to actually visit a median in the length of a shift; in such cases, `MDSA` is capable of merely approaching the median as best as it can, among other options. More importantly, it appears that agents in `WAM` have less 'restraint' in selecting jobs; job selection in `MDSA` seeks to keep agents as close as possible to their respective medians, where agents in `WAM` are encouraged to take on jobs anywhere if they can reduce the total amount of distance travelled this way, even if this means taking on jobs that are closer to the median of someone else. This can create unbalanced situations where some agents move around a lot while others get no jobs; the agents without jobs can cover their neighbourhood well, but other parts of the network can become deserted because 'their' agent is out doing jobs everywhere. This argument is supported by the fact that `WAM` performs worst for productive, dense problems, where there exists most opportunity for agents to take on many jobs and spend time away from 'their' median, thus creating poorly covered parts of the network.

We conclude that `MDSA` is likely a successful heuristic for two reasons: step `M` allows agents to spend their remaining time in a way that is asymptotically optimal, as opposed to spending it bunched up at the depot; and dividing jobs by distance to medians rather than by total amount of distance travelled better ensures that no part of the network is completely deserted for the sake of jobs, unless absolutely necessary.

As a point for improvement, it deserves noting that while `MDSA` finds solutions with near-optimal values, it only succeeds in finding a completely optimal solution in 49 out of 406 instances, 22 of which are in the 'easiest' class I_4. In fact, the partitioning of jobs over agents found by `MDSA` only corresponds with the partitioning in the optimal solution in 125 out of 406 instances. Further improvements to the heuristic may be achieved by better understanding the logic by which jobs should be divided over agents.

The success of `MDSA` prompted us to see if the produced solutions are useful for warm-starting: that is, to see if the MILP running time could be significantly reduced by offering a good feasible solution to start from. For the easier problem instances, this is not the case: apparently, the overhead of running `MDSA` and

Method	Average gap to jobless (%)	Average number of hops	Average number of shortcuts	Average cost decrease versus split fleet (%)	Average hop increase versus split fleet (%)
`opt`	11.5	20.8	1.7	28.3	69.9
`Jobless`	0	11.3	0	–	–
`D`	158.0	9.8	0	-53.2	-33.0
`WAM`	23.0	16.5	0.7	21.7	34.3
`MDSA`	13.0	19.1	1.2	27.3	52.5
`Split fleet`	69.1	13.5	0.5	0	0

Table 2.8: Differences in solution structure when applying different methods on a sample of instances.

feeding this solution into Gurobi is hardly worth the saved Gurobi run time, which is often already in the order of a few seconds. For the difficult classes I_5, I_7 and I_R, however, the average computation time reduction is quite significant. The average computation time in I_5 goes down from 8739.1 seconds to 4949.7 seconds, which is a reduction of 43.4%. I_7 drops 81.8% from 8390.3 seconds to 1523.1 seconds, and I_R drops 18.7% from 1202.7 seconds to 977.5 seconds. These average savings are largely incurred by the hardest instances suddenly being much easier to solve. There exist instances, however, where warm-starting increases the total run time by more than twenty minutes, and it is hard to predict whether warm-starting will be helpful for any given instance. This could be dealt with by solving instances in parallel, where one thread uses a warm start and the other does not, and terminating when either finish. On average, though, warm-starting seems very useful for difficult instance classes.

Looking at Table 2.8, we can see that the solutions produced by the different methods have structural differences. Comparing the hops between `D` and `opt`, we could observe that 'caring about coverage' more than doubles the number of hops. Similarly, comparing `Jobless` and `opt`, we could observe that 'caring about jobs' increases emergency response time by 11.5%. However, in making such comparisons, we implicitly assume that there are $|A|$ agents available solely for jobs or for coverage, while in the `opt`-solutions, we only have $|A|$ agents available in total for both jobs and coverage. It is more fair, therefore, to compare these metrics against the `Split fleet`-solutions. Then, we can observe that joining the coverage fleet and the job fleet into one increases the number of hops by 69.9%, but we also cut our primary objective of expected response time by 28.3%. The increase in hops is explained as follows: in a split fleet, we would run `D` on half the agents and `Jobless` on the other, which are both very efficient methods with respect to hops, as `Jobless`-solutions mainly go to medians and back. The improvement in response time is due to having twice as many agents available to be spread for effective emergency response. We remark that this trade-off between increased travel costs and decreased emergency response costs is less extreme when running `MDSA` instead of `opt`, and least extreme when running `WAM`

instead of opt. Following the same trend, opt-solutions are most 'jittery' with respect to the average number of shortcuts, MDSA-solutions are less jittery and WAM-solutions are least jittery among these three. D is most efficient with respect to hops and shortcuts, but offers even poorer emergency response than splitting the fleet does.

As a final observation, we acknowledge that the average running time of WAM sees a remarkable spike in class I_7. Class I_7 has the largest decision space for WAM, as medium, productive problems have most variables, while dense problems have more feasible solutions than sparse ones. This, apparently, is reflected in Gurobi needing much more time to verify optimality of found routings. We presume that this difficulty is not observed in step D because D does not demand that each route has exactly one median, and because step M makes routes more distinct in solution value. As WAM is typically outperformed by MDSA, resolving this issue was not further considered in this research.

2.7 Conclusions

The problem of scheduling non-urgent jobs in a way that provides optimal emergency response time has received little academic attention, although it is an interesting problem from both academic and applied viewpoints.

We proposed the Median Routing Problem as a mathematical model for this problem. An optimal solution method using Mixed Integer Linear Programming was presented, as well as several heuristics. Of these, MDSA typically worked best: on the benchmark instances, it found solutions that were on average 3.2% away from optimal in 2.4 seconds, where solving them with a MILP solver takes 39 minutes. Using the MDSA solution to warm-start the MILP is especially useful for the most difficult problem instances.

By studying partial versions of MDSA under parametric variation in the benchmark instances, and comparing it with the simpler WAM-algorithm, our experiments suggest that the success MDSA is due to two factors, both of which rely on the MEDIATE-step. First, the MEDIATE-step allows agents to spend their remaining time in a way that is asymptotically optimal as the amount of free time goes to infinity. Second, creating coverage subregions and trying to divide jobs as well as possible into these subregions helps guarantee that no subregion is ever completely abandoned for the sake of jobs, unless absolutely necessary. The speed of MDSA was due to first taking the most restrictive decisions in a very compressed decision space, then making the best out of those decisions in pseudo-polynomial time. Though MDSA contains NP-hard subroutines, this was not yet found to be a bottleneck, even for the practical case study instances. Studying marginal contributions showed that the MEDIATE-step contributed most; it is essential to both success factors of MDSA.

Finally, we observed that combining an emergency response fleet and a job processing fleet into one joint fleet produced a reduction in expected emergency response time of 28.3%. This comes at the cost, however, of an increase in

total travelled distance, as the optimal solutions for the separated problems are typically very efficient in travelled distance. This trade-off is strongest in optimal solutions, then in `MDSA`-solutions, and mildest in `WAM`-solutions.

As venues for future research, we propose the following. Our model discretises space and time so that joint coverage can be measured tractably, but other strategies than discretisation may exist. `MDSA` contains NP-hard subroutines, and for (much) larger instances, a fully polynomial version may be needed, though Corollary 1 implies that we lose the guarantee of feasibility in the process. As mentioned earlier, better understanding the logic by which optimal solutions cluster jobs may lead to even stronger heuristics. The single-agent case of MRP is much easier to analyse, as its coverage profile is always known, and interesting solution properties may be discovered. We have studied a 'single coverage' problem, and it merits research how to well prepare for more emergencies than just the next one. Perhaps most importantly, the set of jobs to be processed was given as a hard constraint: it is an interesting and non-trivial question to decide which jobs will be selected as input for a series of shifts, and what role multi-objective optimisation can play in weighing responsiveness against productivity.

Chapter 3

The Travelling k-Median Problem

3.1 Introduction

In the previous chapter, Corollary 1 made clear that no polynomial-time algorithm exists for MRP in general unless P=NP. The main difficulty lies in the fact that it is NP-complete to decide whether a mandatory set of jobs can be visited within a hard time limit. However, in order to explore the boundary of what *can* be achieved in polynomial time, this chapter will focus on a special case of MRP: namely, instances that feature no jobs. Without jobs, it is no longer NP-complete to check for feasibility, as feasibility is simply a matter of whether all agents can move from their start point to their end point within the fixed time horizon.

It may seem counterintuitive to first add jobs to the k-Medians Problem and then remove them again. However, not only jobs were added, but also discrete movement. In this chapter, we will essentially add a notion of discrete movement to the k-Medians Problem, and obtain algorithmic results for this unusual and already challenging extension of the k-Medians Problem. Results for this special case of MRP, while not directly reusable for MRP itself, can help us develop algorithmic techniques and insights for MRP.

Specifically, we consider the following natural extension of the k-Median Problem, which we term the *Travelling k-Median Problem (TkMP)*: Suppose we are given a fixed time horizon $T = \{0, \dots, \omega\}$ and a set $A = \{1, \dots, k\}$ of k medians (called *agents* in this context) which can move through the graph $G = (V, E)$ in discrete steps. Each agent $a \in A$ starts at time $t = 0$ at a designated start node $S_a \in V$, and ends at time $t = \omega$ at a designated end node $W_a \in V$. In each time step, an agent may either hop from her current location $u \in V$ to an adjacent node $v \in V$, or simply stay at u. Our goal is to determine a set of k walks (one for each agent) such that the total coverage costs (summed over all time steps) is minimised. Here, the *coverage cost* at time t is defined the same way as in the k-Median Problem (with respect to the locations of the agents at time t).

Aside from its relation to MRP, this problem in isolation already has merit for being studied. Intuitively, in the Travelling k-Median Problem one wants to

distribute the agents over the graph such that they guarantee a low coverage cost, ideally positioning them at the optimal k-median locations. However, in many real-life applications we cannot simply assume that the agents can be 'teleported' to their ideal locations. Instead, the agents might have to move there over time, which in turn influences the achievable coverage. For example, the agents might correspond to an 'emergency fleet' (e.g., police patrol force, emergency medical coverages, railway emergency responders) that one needs to schedule to provide a good emergency response time over a pre-specified time horizon. This is especially relevant in the current day, where advances in communication allow emergency responders and their coordinators to communicate in real-time while on the move. This means, namely, that emergency response organisations have the possibility to optimise responsiveness even before the steady state coverage positions are attained. Assuming that the time horizon is fixed adds some new characteristics to the problem which are relevant to several applications in emergency logistics.

In this chapter, we study the Travelling k-Median Problem (TkMP). We initiate the study of this topic by focusing on the uniform case (referred to as U-TkMP), i.e., when all edge costs are uniform and all agents start from the same start location and end at the same end location. The contribution of this chapter is fivefold. First, we first argue that even U-TkMP is NP-hard in general by a simple reduction from the Set Cover Problem. We then show that the problem can be solved optimally in $\mathcal{O}(\omega^2 n^{2k})$ time. Second, we derive a 5-approximation algorithm for U-TkMP if the number of agents is large, i.e., $k \geq n/2$. This allows us to assume $k \leq n/2$ in the remainder. Basically, the idea here is to let each agent take care of an edge in a maximum matching of the underlying graph. Third, we introduce a novel way of approximating U-TkMP by rounding an (approximate) solution to a 'continuous movement' relaxation of the problem to a discrete solution, thereby incurring a bounded loss in the approximation factor. Fourth, we use this Rounding Theorem to derive 10-approximation algorithms for U-TkMP on path graphs and cycle graphs. (These results are given in Section 3.6). Fifth and final, for U-TkMP on general graphs we obtain the following results: If the shortest (S, W)-path distance $d(S, W)$ satisfies $d(S, W) \leq 2\sqrt{\omega}$, then we obtain an $\mathcal{O}(\min\{\sqrt{\omega}, n\})$-approximation algorithm. If $d(S, W) > 2\sqrt{\omega}$, we instead have an $\mathcal{O}(d(S, W) + \sqrt{\omega})$-approximation algorithm. This is useful for those graph topologies for which we do not yet know how to solve or approximate their continuous counterparts.

The remainder of this chapter is structured as follows. In Section 3.2, literature related to this problem is outlined. In Section 3.3, TkMP and its variants are introduced formally, as well as some useful notation. In Section 3.4, the U-TkMP and TkMP are shown to be NP-hard, but constant-factor approximation algorithms are offered for special cases: this allows us to, for the remainder of the chapter, make assumptions on the number of agents and time steps the problem instances will feature. Using this, in Section 3.5, a Rounding Theorem is proven that provides a constant-factor approximation algorithm for U-TkMP on 'roundable' graph topologies. How to apply this Rounding Theorem is demonstrated

for two topologies in Section 3.5. For other topologies, a $\mathcal{O}(d(S,W) + \sqrt{\omega})$-approximation algorithm is given in Section 3.7. Finally, some brief conclusions are presented in Section 3.8.

3.2 Related literature

To the best of our knowledge, the Travelling k-Median Problem introduced in this chapter has not been studied in the literature before. The following are the key distinguishing features of TkMP with respect to other problems considered in the literature: (i) We consider a fixed time horizon and all agents have to start and end at designated nodes; in particular, this restricts the walks that can be chosen by each median. (ii) The coverage cost accounts for the total connection cost of each node to its closest median (K-MED-objective). However, there are several other optimisation problems that are related to TkMP. Below is a partial list of related problems.

At the core of TkMP lies the k-Median Problem which has been studied extensively in the literature. An overview of literature related to this specific problem was already presented in Section 2.2.

TkMP is also related to classic online problems such as the *k-Server Problem* [89] or, more generally, the *Metrical Task Systems* [16]. In the k-Server Problem [89], there are k servers that can move through a graph and have to serve a sequence of requests. Also here, the servers incur some movement costs (given by a metric). In each step a node is requested and a server has to move there. The main difference with respect to TkMP is that the time horizon is not limited and that there is no coverage cost. In Metrical Task Systems [16], an online algorithm starts at a designated node of a given graph and has to cover a sequence of tasks. Also here, the algorithm incurs some coverage cost and some movement cost (given by a metric) to serve a sequence of tasks. However, the coverage cost is given in terms of a non-negative cost incurred at the node in which the algorithm resides; this is different from TkMP, where the coverage cost accounts for the total connection cost of the nodes to the medians.

Dynamic facility location models are often on a strategic level [44]. Wesolowsky [157] extends several static facility location models, including K-MED, to a problem where facility locations have to be chosen for several periods. If the facility locations differ from one period to the next, then a period-dependent cost is incurred, independent of how many facilities are relocated and how far.

Of the many dynamic facility location problems reviewed and classified by Farahani et al. in 2012 [44], none fits our setting of one configuration determining the decision space in the next time step. Distance-dependent relocation costs appear to be less commonly discussed than time-dependent costs, but they are studied by Huang et al. [68] and Gendreau et al. [55]. Galvão and Santibanez-Gonzalez, instead, charge period-dependent costs for having a facility open [51]. Melo et al. consider a combination of facility opening, closing and maintenance costs, and costs of reallocating capacity between facilities [105].

On a more operational level, ambulance relocation models seek to deploy an ambulance to a revealed emergency and relocate the rest for optimal response to the next emergency [56, 79]. Even these operational models assume 'teleportation' from one steady-state configuration to the next. The same holds on the online level for the classical k-Server Problem [88].

From a game-theoretical standpoint, Hotelling [67] discusses the *Ice-Cream Man Problem*, in which two competing ice cream vendors position themselves on the line to capture the largest portion for themselves. When moving to increase the portion of the line they cover, they surprisingly reach an equilibrium when their distance approaches zero. However, an equilibrium is no longer reached for three vendors [143].

3.3 Problem definition

In the *Travelling k-Median Problem* (TkMP), we are given a connected graph $G = (V, E)$ with n nodes, a set $A := \{1, \dots, k\}$ of k agents, and a discrete time horizon $T := \{0, \dots, \omega\}$. Each agent $a \in A$ has their own start location $S_a \in V$, where they must be at time 0, and an end location $W_a \in V$, where they must be at time ω. Between each time step and the next, agents may stay where they are or 'hop' to adjacent nodes.

At each time step, we evaluate how well 'spread' the agents currently are. A *configuration* $\sigma = (\sigma_1, \dots, \sigma_k) \in V^A$ specifies for each agent $a \in A$ the node $\sigma_a \in V$ at which a resides. Let $d(u, v)$ denote the cost function for covering $v \in V$ from $u \in V$. We assume that every node will be covered from its nearest median in σ, i.e., the cost of node v is $D(\sigma, v) := \min_{a \in A} d(\sigma_a, v)$, and the total cost of this configuration is $D(\sigma) := \sum_{v \in V} D(\sigma, v)$. We say that a configuration σ is *optimal* if it minimises the total cost $D(\sigma)$ (over all possible configurations).

A *feasible solution* $\mu \in V^{A \times T} = (\mu^0, \dots, \mu^\omega)$ is a sequence of $(\omega + 1)$ configurations, with μ^0 and μ^ω given by the start and end locations, respectively, and each configuration μ^t being *adjacent* to the previous configuration μ^{t-1} for every $t \in \{1, \dots, \omega\}$. We say that μ^t is adjacent to configuration μ^{t-1} if for each agent $a \in A$, either $\mu_a^{t-1} = \mu_a^t = u$ (agent a remains at node u), or $\mu_a^{t-1} = u$, $\mu_a^t = v$ and $(u, v) \in E$ (agent a moves along edge (u, v) from u to v). For any feasible solution μ, we define the cost as $D(\mu) := \sum_{t=0}^{\omega} D(\mu^t)$. TkMP is the problem of finding a solution μ with minimal $D(\mu)$.

Given a feasible solution μ, we call the sequence $(\mu_a^0, \dots, \mu_a^\omega)$ of nodes visited by agent a also the *walk* of a. The cost of the walk of agent $a \in A$ with respect to some fixed node v is defined as $\sum_{t=0}^{\omega} D(\mu_a^t, v)$.

In the *Uniform Travelling k-Median Problem* (U-TkMP), which is a special case of TkMP, we make the following three assumptions: (i) d is defined as the shortest path distance (with respect to the number of edges) induced by the underlying graph G, (ii) all agents start at the same node S, and (iii) all agents end at the same node W.

3.4 Hardness and bounds

In this section, we will derive some lemmas and observations that will serve as instrumental building blocks for the approximation algorithms in this chapter. As a first result, it deserves mentioning that even U-TkMP is NP-hard.

Theorem 2. *U-TkMP and its generalisation TkMP are NP-hard.*

Proof. U-TkMP is obviously in NP. We will prove that U-TkMP is at least as hard as the Set Cover Problem, which is NP-hard. Let I_1 be the following instance of the decision version of the Set Cover Problem: given universe U and a collection S of subsets of U, can we find k (or fewer) members of S which have union U? Let I_2 be an instance of the decision version of U-TkMP that we construct as follows. Let the node set be $V := V_U \cup V_S \cup \{S\}$, where each element in U is represented by its own node in V_U, and each member in S is represented by its own node in V_S, and S is some additional node. In the graph $G = (V, E)$, we have $(u, v) \in E$ for nodes $u, v \in V$ exactly when either of these two conditions holds: $u, v \in V_S \cup \{S\}$, or $u \in V_U$ and $v \in V_S$ and the element that corresponds to u is contained in the subset that corresponds to v. In I_2, we set $\omega = 2$, and let all k agents start and end at S, and we ask: can we find a feasible solution with cost at most $2|S| + 4|U| + (n - k)$? I_2 can be constructed in polynomial time and space with respect to the input of I_1. In any feasible solution, all agents are at S at times $t = 0$ and $t = 2$, so we incur a cost of $|S| + 2|U|$. So the answer to I_2 is 'yes' if and only if we can find a configuration at $t = 1$ with cost $(n - k)$, meaning every node is adjacent to at least one agent position. The agents can only be in $V_S \cup \{S\}$, which is a clique. Adjacency to all nodes in V_U is achieved if and only if the chosen positions in V_S correspond to members of S which have union U. (If agents stay at S, this corresponds to choosing fewer than k subsets in I_1.) So any 'yes'-answer to I_1 supplies positions that will give a 'yes'-answer to I_2, and any 'yes'-answer to I_2 gives a 'yes'-answer to I_1 by reading out the positions at $t = 1$. So I_1 and I_2 are equivalent, implying U-TkMP is NP-hard. □

Theorem 3. *TkMP is solvable in $\mathcal{O}(\omega^2 n^{2k})$.*

Proof. Let $G = (V, E)$ be the original graph of our TkMP-instance. We construct a weighted, expanded graph $G^+ = (V^+, E^+)$ as follows. First, make a 'directed looping version' of G: take V, and for every $u, v \in V$, include arc (u, v) if edge $\{u, v\} \in E$ or $u = v$. Let this digraph be denoted by G'. Next, let $\vec{T}$ be the digraph with node set T, and arc set $\cup_{t \in T \setminus \{\omega\}} \{(t, t + 1)\}$, so $\vec{T}$ is a path digraph representing the time horizon.

Then, let G^+ be the tensor product $(G')^k \times \vec{T}$. So every node $(u_1, u_2, \ldots, u_k, t_1)$ in G^+ represents a position for each agent, and a time step. There is an arc in G^+ from node $(u_1, u_2, \ldots, u_k, t_1)$ to node $(v_1, v_2, \ldots, v_k, t_2)$ if and only if the arc (u_a, v_a) is in G' for each agent a, and $t_2 = t_1 + 1$.

Finally, let each such arc in G^+ have weight $D(\sigma_2)$, where σ_2 is the visited configuration $(v_1, v_2, \ldots, v_k)$. This means that, walking through G^+, we incur

at each time step $t = 1, \ldots, \omega$ the cost of the configuration currently visited. The cost made at $t = 0$ is a constant that is the same for each feasible solution. Therefore, finding an optimal TkMP-solution is equivalent to finding a cheapest $(S_1, S_2, \ldots, S_k, 0) - (W_1, W_2, \ldots, W_k, \omega)$-path. Because G^+ has $\mathcal{O}(\omega n^k)$ nodes, finding this path with Dijkstra's algorithm has time complexity $\mathcal{O}(\omega^2 n^{2k})$. □

However, if we want to determine a minimum cost walk of one agent with respect to a given node v, we can do this quite easily.

Lemma 1. *In U-TkMP, given any node v and an agent $a \in A$, we can find a minimum cost (S, W)-walk $\pi = (\pi^0, \ldots, \pi^\omega)$ of agent a with respect to v in $\mathcal{O}(n^4)$ time.*

Proof. Suppose first that $\omega \geq 2n$. Then, certainly, a walk exists from S to v and then W. We take a shortest (S, v)-walk and a shortest (v, W)-walk, and spend all remaining time 'looping' at v. This is obviously a walk π that minimises $\sum_{t=0}^{\omega} d(\pi^t, v)$, because at every time step, the agent is as near to v as the start and end conditions allow.

Suppose now that $\omega < 2n$. In this case, we construct G^+ exactly as in the proof of Theorem 3, setting $k = 1$. Each arc $((u, t), (u', t + 1))$ in G^+ now has weight $d(u', v)$. Because $\omega < 2n$, we find our walk in $\mathcal{O}(n^4)$ by applying Dijkstra's algorithm on G^+ from $(S, 0)$ to (W, ω). □

We now provide several bounds on performance and parameters.

Lemma 2. *Consider U-TkMP. Then an optimal configuration σ^* has cost at least $D(\sigma^*) \geq (n - k)$. Further, any feasible solution μ has cost at least $D(\mu) \geq (\omega + 1)D(\sigma^*) \geq (\omega + 1)(n - k)$.*

Proof. In any configuration σ, at most k nodes can be occupied. All other nodes account for at least a cost of 1, implying $D(\sigma^*) \geq (n - k)$. Because any solution μ decomposes into $(\omega + 1)$ feasible configurations, $D(\mu) \geq (\omega + 1)D(\sigma^*)$. □

Theorem 4. *In U-TkMP, if $k \geq n/2$, we can find a 5-approximation in $\mathcal{O}(n^5)$ time.*

Proof. We construct the approximate solution μ as follows. Using a greedy $\mathcal{O}(n^2)$ algorithm, find any (inclusion-wise) maximal matching on G. For every edge in this matching, assign an agent to one of its end points: we have enough agents to do this, because no matching can consist of more than $n/2$ edges. Assign the remaining agents to W. For every agent, determine a minimum cost walk with respect to the assigned node by using Lemma 1 (which requires at most $\mathcal{O}(n^4)$ time). We need to determine at most $n/2 + 1$ such walks, resulting in an algorithm that needs at most $\mathcal{O}(n^5)$ time.

We can compare μ to the hypothetical solution μ^n where the number of available agents is n, and we assign each node its own agent. In μ^n, we again send each agent over a walk that minimises the total distance to their picked

node. Note that $D(\mu^n)$ is minimal, because every node that gets its own agent is served optimally.[1] If node v is 'picked' by μ, v contributes this same optimal amount to $D(\mu)$. If v is not 'picked', we show here that the cost increases by at most 4 per time step.

Take any node v and time step t, and let us examine the contribution of (v,t) to the cost function. Let a be the agent in μ^n assigned to v. Let $u \in V$ be the location of a at time t. In μ, there may or may not be someone at u at time t. If there is, the request (v,t) has the same cost contribution to $D(\mu)$ as to $D(\mu^n)$, remarking again that nodes with their own agent are served optimally. If not, there is a node $u' \in V$ that was picked, at most 2 hops away from u; otherwise, the underlying matching would not be maximal. If the agent in μ assigned to u' is still headed for it, that agent is at most 2 hops away from u', because $d(S,u') \leq d(S,u) + d(u,u') \leq t+2$. Similarly, if the agent is moving away from u' towards W, the agent is at most 2 hops away from u' because $d(u',W) \leq d(u',u) + d(u,W) \leq (\omega - t) + 2$. That is, there are only $\omega - t$ hops left to get to W, and while this is no problem when $d(u',W)$ is small, the agent will be pulled away by at most 2 hops if $d(u',W)$ is large. So indeed, by having fewer agents then in μ^n, each unpicked node v (at most $n-k$) contributes at most 4 extra per time step. Thus, if μ^* is some optimal solution, then

$$D(\mu) \leq D(\mu^n) + 4(\omega+1)(n-k) \leq D(\mu^*) + 4(\omega+1)(n-k) \leq 5D(\mu^*). \quad \square$$

As a result of Theorem 4, we will often assume in the remainder that $k \leq n/2$, which yields the following useful refinement of Lemma 2.

Corollary 2. *Consider U-TkMP and assume that $k \leq n/2$. Then an optimal configuration σ^* has cost at least $D(\sigma^*) \geq \frac{1}{2}n$. Further, any feasible solution μ has cost at least $D(\mu) \geq (\omega+1)D(\sigma^*) \geq \frac{1}{2}(\omega+1)n$.*

Lemma 3. *In U-TkMP, for any feasible solution μ and time steps t_1 and t_2, we have $D(\mu^{t_2}) \leq D(\mu^{t_1}) + |t_2 - t_1|n$.*

Proof. For any $v \in V$, let a be the agent nearest to it in μ^{t_1}. The contribution of v to $D(\mu^{t_2})$ cannot exceed the distance of v to a, who cannot have moved more than $|t_2 - t_1|$ hops away from their position in μ^{t_1}. $\square$

Theorem 5. *Suppose we have an α-approximation algorithm for K-MED that runs in $\mathcal{O}(kmed(n))$ time. In U-TkMP, if $k \leq n/2$ and $\omega \geq 4n^2 + 2n$, we can find an $(\alpha+1)$-approximation in $\mathcal{O}(kmed(n) + n^5)$ time.*

Proof. We construct our solution as follows. First, if σ^* is an optimal K-MED-configuration, we use the α-approximation algorithm to find a good configuration

[1] That is, for every node v, there exists an agent who, through Lemma 1, is always as nearby as possible, so far as the start and end conditions allow. While it may seem nontrivial which nodes this agent could give coverage to on the way to v, in fact those intermediate nodes have agents of their own serving them optimally, and the agent never has to cover anything else but v.

σ with $D(\sigma) \leq \alpha D(\sigma^*)$, requiring a running time of $\mathcal{O}(kmed(n))$. Next, we assign these medians to agents arbitrarily. Finally, for each of the $k \leq n/2$ agents, we find in $\mathcal{O}(n^4)$ time the (S, W)-walk that minimises the total distance to that agent's median, using Lemma 1. This adds a running time of $\mathcal{O}(n/2 \cdot n^4)$, for a total of $\mathcal{O}(kmed(n) + n^5)$.

We now prove that this yields an $(\alpha + 1)$-approximation. Let μ be the found solution, and μ^* an optimal solution. Obviously, μ consists of shortest paths from S to points in σ, a lot of waiting in σ, then shortest paths to W.

Observe any time step t. If $n - 1 \leq t \leq (\omega + 1) - (n - 1)$, then clearly all agents have reached their position in σ and have not departed them yet. In each of those time steps, we have $D(\mu^t) \leq \alpha D(\sigma^*)$. In the other time steps t, we at least know that $D(\mu^t) \leq n^2$, because each of the n nodes has an agent at most n hops away.

So, recalling Corollary 2, we find

$$\frac{D(\mu)}{D(\mu^*)} \leq \frac{2n \cdot n^2}{\frac{1}{2}\omega n} + \frac{(\omega + 1 - 2n) \cdot \alpha D(\sigma)}{(\omega + 1) D(\sigma^*)} \leq \frac{2n \cdot n^2}{\frac{1}{2} \cdot 4n^2 \cdot n} + \alpha = \alpha + 1,$$

meaning that μ is an $(\alpha + 1)$-approximation. □

Note that this theorem helps us in the following way: instances with $\omega \geq 4n^2 + 2n$ are 'easy', and we can assume in the remainder that $\omega \leq 4n^2 + 2n$. This, in turn, means that algorithms with runtime depending on ω (polynomially) can still be considered polynomial-time.

We also remark that by combining Theorem 3 and Theorem 5, U-TkMP can be solved in $\mathcal{O}(n^6)$ if there is only one agent.

3.5 Rounding continuous graph movement

A related problem to U-TkMP is the *Continuous Uniform Travelling k-Median Problem* (CU-TkMP), which we can use to approximate U-TkMP. In CU-TkMP, the allowed positions for agents are not only the nodes, but also any point on the edges.

Technically, we define CU-TkMP as follows. Given an unweighted, undirected graph $G = (V, E)$, we construct its 'continuous version' as the following metric space $(\tilde{E}, c)$. Assuming an arbitrary orientation on the edges of E, we build $\tilde{E}$ out of V and all points 'strictly on' edges: $\tilde{E} := V \cup E \times (0, 1)$, where a point $((u, v), \theta) \in E \times (0, 1)$ represents the point that is on the oriented edge (u, v), at a fraction θ from u to v.

As for the distance metric c, we base it on a continuous version of the shortest path distance as follows. Take any $x \in \tilde{E}$ and $y \in \tilde{E}$. If both are in V, then $c(x, y) = d(x, y)$, where d is the shortest path distance on G. If $x \in V$ but $y \notin V$, let $y = ((u, v), \theta)$. Then $c(x, y) = \min\{\theta + d(u, x), 1 - \theta + d(v, x)\}$. We uphold an analogous definition if $y \in V$ but $x \notin V$. Finally, if both points are not in V, let again $y = ((u, v), \theta)$. Then $c(x, y) = \min\{\theta + c(u, x), 1 - \theta + c(v, x)\}$.

Now, in CU-TkMP, we are again given k agents and a discrete time horizon $\{0, \ldots, \omega\}$. The agents must start at $S \in V$ at time $t = 0$ and must be at their end location $W \in V$ at time $t = \omega$. If an agent is at position $x \in \tilde{E}$ at time t, then at time $t+1$, they may only be at a position $y \in \tilde{E}$ with $c(x, y) \leq 1$. The continuous cost of a configuration $\sigma \in \tilde{E}^A$ is not the discrete sum of distances to nodes, but the integrated distance to all points in $\tilde{E}$, i.e., $C(\sigma) := \int_{\tilde{E}} \min_{x \in C} c(x, y) \mathrm{d}y$. The continuous cost of a solution $\nu \in \tilde{E}^{A \times T}$ is again the sum over time, i.e., $C(\nu) := \sum_{t \in T} C(\nu^t)$.

Suppose G has maximum degree Δ, and $k \leq n/2$. Suppose also we can find a β-approximate solution ν to CU-TkMP. We prove that we can 'round' ν to a $(4\beta\Delta + 2)$-approximate solution $\tilde{\nu}$ for the original, 'discrete' problem.

We first show that the cost functions D and C are related. Remark that, if ρ is a continuous configuration, we mean by $D(\rho)$ that we sum distances to V rather than integrating over $\tilde{E}$: we define $D(\rho) := \sum_{v \in V} \min_{u \in \rho} c(u, v)$. If ν is a continuous solution, we define $D(\nu)$ similarly.

Lemma 4. *Let $\rho \in \tilde{E}^A$ be a continuous configuration. Then $D(\rho) \leq 2C(\rho) + k$.*

Proof. Given ρ, we split E into three classes. Denote by $E_0 \subset E$ all edges $\{u, v\}$ that have at least one agent 'strictly' on them, meaning we disregard agents on u and v. Note that $|E_0| \leq k$. Let $E_1 \subset E \setminus E_0$ denote all unoccupied edges $\{u, v\}$ with $d(\rho, u) \neq d(\rho, v)$, oriented as (u, v) if $d(\rho, u) < d(\rho, v)$ and as (v, u) otherwise. Let $E_2 \subset E \setminus E_0$ refer to all unoccupied edges $\{u, v\}$ for which $d(\rho, u) = d(\rho, v)$. We orient E_0 and E_2 arbitrarily.

For any edge $(u, v) \in E_0$, we have $d(\rho, u) + d(\rho, v) \leq 1$, as any one agent on (u, v) is $0 < \theta < 1$ away from u and $1 - \theta$ away from v. Notice that $\{u : (u, v) \in E_0\} \cup \{v : (u, v) \in E_0\} \cup \{v : (u, v) \in E_1\} = V$, as any node $v \in V$ is either incident to E_0, or has a neighbour u on the shortest (v, ρ)-path, thus $(u, v) \in E_1$. Thus summing over these three node sets is sufficient, though we may be over-counting:

$$D(\rho) \leq \sum_{(u,v) \in E_0} (d(\rho, u) + d(\rho, v)) + \sum_{(u,v) \in E_1} d(\rho, v) \leq |E_0| + \sum_{(u,v) \in E_1} (d(\rho, u) + 1)$$

Each $(u, v) \in E_1$ has $\int_u^v d(\rho, w) \mathrm{d}w = d(\rho, u) + \int_0^1 w \mathrm{d}w = d(\rho, u) + \frac{1}{2}$. So

$$C(\rho) = \sum_{(u,v) \in E} \int_u^v d(\rho, w) \mathrm{d}w \geq \sum_{(u,v) \in E_1} (d(\rho, u) + \tfrac{1}{2}) \geq \tfrac{1}{2}|E_1|$$

Putting this together, we find

$$D(\rho) \leq \sum_{(u,v) \in E_1} (d(\rho, u) + \tfrac{1}{2}) + \tfrac{1}{2}|E_1| + |E_0| \leq 2C(\rho) + k.$$

□

Lemma 5. *Let $\sigma \in V^A$ be a discrete configuration and $n \geq 2k$. Then $C(\sigma) \leq 2\Delta D(\sigma)$.*

Proof. Because σ is a discrete configuration, $E_0 = \emptyset$. Each $(u,v) \in E_1$ has $\int_u^v d(\sigma,w)\mathrm{d}w = d(\sigma,u) + \int_0^1 w\mathrm{d}w = d(\sigma,u) + \frac{1}{2}$. Each $(u,v) \in E_2$ has $\int_u^v d(\sigma,w)\mathrm{d}w = d(\sigma,u) + 2\int_0^{\frac{1}{2}} w\mathrm{d}w = d(\sigma,u) + \frac{1}{4}$.

Observe some oriented version $\vec{E}$ of E, such that if $(v,u) \in \vec{E}$, then $d(\sigma,u) \leq d(\sigma,v)$. For any $u \in V$, let $\delta^-(u)$ be the edges in E such that, in $\vec{E}$, they point at u. Then $|\delta^-(u)| \leq \Delta$ for any $u \in V$.

$$\begin{aligned} C(\sigma) &= \sum_{(u,v)\in E_1} (d(\sigma,u) + \tfrac{1}{2}) + \sum_{(u,v)\in E_2} (d(\sigma,u) + \tfrac{1}{4}) \leq \sum_{u\in V} |\delta^-(u)|(d(\sigma,u) + \tfrac{1}{2}) \\ &\leq \Delta \sum_{u\in V} d(\sigma,u) + \tfrac{1}{2}\Delta n \leq \Delta D(\sigma) + \Delta \cdot (n-k) \leq 2\Delta D(\sigma) \end{aligned}$$

□

Suppose we have some continuous solution $\nu = (\nu_0, \ldots, \nu_\omega)$, which has $C(\nu) \leq \beta C^*$, with C^* the optimal coverage solution value for CU-TkMP. We can for each agent a find a continuous movement line across $(\nu_0)_a, \ldots, (\nu_\omega)_a$ by taking shortest walks through $\tilde{E}$ from each $(\nu_t)_a$ to $(\nu_{t+1})_a$, waiting at $(\nu_{t+1})_a$ until $t+1$ if necessary. We can obtain a 'rounded solution' $\tilde{\nu}$ by setting $(\tilde{\nu}_t)_a$ equal to $(\nu_t)_a$ if that is a node. If instead it is a point on an edge (u,v), because the agent is moving from u to v in the continuous movement line, then we set $(\tilde{\nu}_t)_a$ to the closest of these two to $(\nu_t)_a$. If the point is exactly halfway between u and v, we set $(\tilde{\nu}_t)_a$ to the destination v. Assuming again that $k \leq n/2$ and $\omega \leq 4n^2$, we can perform this rounding in $\mathcal{O}(k\omega) = \mathcal{O}(n^3)$.

Theorem 6 (Rounding Theorem)**.** *If $n \geq 2k$, then rounding ν to the solution $\tilde{\nu}$ yields a $(4\beta\Delta + 2)$-approximation with respect to $D(\mu^*)$, where μ^* is an optimal solution.*

Proof. Take some instance of U-TkMP with $n \geq 2k$. Let μ^* be an optimal solution for this instance, and ν^* an optimal solution for its CU-TkMP-equivalent. Let ν be a β-approximate solution to the CU-TkMP-instance, and let $\tilde{\nu}$ be the result of rounding ν in the way described above.

Solution $\tilde{\nu}$ is feasible for the discrete problem. That is, each agent still starts at their start point and ends at their end point; we show that they also only move from nodes to neighbouring nodes. Take any agent a and any time step $t < \omega$. Define $q := (\nu_t)_a$ and $r := (\nu_{t+1})_a$. By feasibility of ν in CU-TkMP, $d(q,r) \leq 1$. If q and r are on the same edge (u,v), including the end points, then they are rounded to u and/or v, which are neighbouring. If q is on (u,v) and r is not, then r is on some (v,w). If q or r is rounded to v, then q and r are rounded to neighbouring nodes. Suppose not, so q is rounded to u and r to w. Then $d(q,v) > \frac{1}{2}$ and $d(v,r) \geq \frac{1}{2}$, so $d(q,r) = d(q,v) + d(v,r) > 1$, but this contradicts feasibility of ν in CU-TkMP.

It remains to show the approximation factor. Using Corollary 2 for OPT, we

have

$$OPT := D(\mu^*) \geq (\omega+1)(n-k) \geq (\omega+1)\tfrac{n}{2} \geq (\omega+1)k$$

$$D(\tilde{\nu}) \leq D(\nu) + (\omega+1)n \cdot \tfrac{1}{2} \leq D(\nu) + OPT \tag{3.1}$$

$$D(\nu) \leq \sum_{t=0}^{\omega} (2C(\nu_t) + k) \leq 2C(\nu) + OPT \tag{3.2}$$

$$C(\nu) \leq \beta C(\nu^*) \leq \beta C(\mu^*) \tag{3.3}$$

$$C(\mu^*) \leq 2\Delta D(\mu^*) = 2\Delta \cdot OPT \tag{3.4}$$

(3.1) follows from the fact that every demand point (v,t) has its median shifted away at most $\frac{1}{2}$ in rounding. (3.2) follows from Lemma 4. (3.3) follows from the fact that ν is β-approximate for CU-TkMP, while μ^* is feasible. (3.4) follows from Lemma 5. □

3.6 Topologies with constant-factor guarantees

In this section, we show how to compute an optimal solution to the continuous problem CU-TkMP on two simple graph topologies, namely on a path and on a cycle. In both cases, we need to solve a Linear Program with a convex minimisation objective, which can be done in polynomial time (see, e.g., [110]). Throughout this section, we use $convex(k_1, k_2)$ to refer to the time that is needed to solve such a program consisting of k_1 variables and k_2 constraints. Recall that in light of Theorem 5, we can assume that $k \leq n/2$ and $\omega \leq 4n^2 + 2n$; in particular, this implies that $\mathcal{O}(k\omega) = O(n^3)$.

3.6.1 Path case

We show that one can compute an optimal solution to the continuous problem CU-TkMP if the underlying topology is a line of length L, i.e., $\tilde{E} = [0, L]$. By applying our Rounding Theorem (Theorem 6) to the optimal solution, we obtain the result stated in Theorem 7 below (with $\beta = 1$ and $\Delta = 2$).

Theorem 7. *Let $n \geq 2k$. Then there is a 10-approximation algorithm for U-TkMP on the path graph that runs in $\mathcal{O}(convex(k_1, k_2) + n^3)$ time, where $k_1 = k_2 = \mathcal{O}(k\omega)$.*

A key observation that we can exploit in the path case is that we can impose an order on the positions of the agents. More precisely, let $\sigma \in \tilde{E}^A$ be a configuration. Assume that the agents in $A = \{1, \ldots, k\}$ are indexed such that $\sigma_1 \leq \sigma_2 \leq \cdots \leq \sigma_k$ (break ties arbitrarily). For notational convenience, we introduce two dummy agents 0 and $k+1$ which are fixed at $\sigma_0 = 0$ and $\sigma_{k+1} = L$, respectively. Further, let λ_a be a scalar that is defined as $\lambda_a = \frac{1}{4}$ for all $a \in \{1, \ldots, k-1\}$ and $\lambda_0 = \lambda_k = \frac{1}{2}$.

Note that we do not have to worry about agents 'switching' their index: because everyone has the same start and end location, there obviously exists an optimal solution in which they do not switch their index. This is because for any feasible solution with a switch, there also exists a feasible solution with identical cost in which the switch is not made, namely the solution in which the agents approach each other for the switch but go back again. Therefore, we are free to enforce in our Convex Program that agents will always keep their index.

The following lemma shows that the coverage cost reduces to a function that depends on the distances between consecutive agents.

Lemma 6. *Consider a continuous configuration $\sigma \in \tilde{E}^A$ and suppose that the agents are indexed as stated above. Then the coverage cost is*

$$C(\sigma) = \sum_{a=0}^{k} \lambda_a (\sigma_{a+1} - \sigma_a)^2$$

Proof. Consider the cost of an interior line segment $[\sigma_a, \sigma_{a+1}]$ between two consecutive agents σ_a and σ_{a+1} with $a \in \{1, \dots, k-1\}$. In this case, each point of the segment is covered by the closest agent among σ_a and σ_{a+1}. Thus, the coverage cost is

$$\int_{\sigma_a}^{\sigma_{a+1}} \min_{u \in \{\sigma_a, \sigma_{a+1}\}} c(u, v) \mathrm{d}v = 2 \int_0^{\frac{1}{2}(\sigma_{a+1} - \sigma_a)} v \mathrm{d}v = \frac{1}{4}(\sigma_{a+1} - \sigma_a)^2$$

Next, consider the first line segment $[0, \sigma_1]$. In this case, each point of the segment is covered by σ_1. Thus, the coverage cost is

$$\int_0^{\sigma_1} c(\sigma_1, v) \mathrm{d}v = \int_{\sigma_0}^{\sigma_1} v \mathrm{d}v = \frac{1}{2}(\sigma_1 - \sigma_0)^2$$

A similar argument proves that the coverage cost of the last line segment $[\sigma_k, \sigma_{k+1}]$ is $\frac{1}{2}(\sigma_{k+1} - \sigma_k)^2$. Summing over all line segments proves the claim. □

Exploiting the above observation, we can solve CU-TkMP for the path case simply by formulating the problem as a Linear Program with a convex (in fact quadratic) objective function.

We introduce a variable $x_a^t \in [0, L]$ for each $a \in \{0, \dots, k+1\}$ and time step $t \in T$. Our interpretation is that x_a^t represents the position of the a-th leftmost agent (breaking ties arbitrarily). The following Convex Program then captures the continuous version of the problem:

$$\begin{array}{lrcll}
\min & \sum_{t \in T} \sum_{a=0}^{k} \lambda_a (\sigma_{a+1} - \sigma_a)^2 & & & \text{(QP1)} \\
\text{s.t.} & 0 = x_0^t \le x_1^t & \le \cdots \le & x_k^t \le x_{k+1}^t = L & \forall t \in T \\
& x_a^t - x_a^{t-1} & \le & 1 & \forall a \in A \;\; \forall t \in T \\
& -x_a^t + x_a^{t-1} & \le & 1 & \forall a \in A \;\; \forall t \in T \\
& x_a^t & \in & [0, L] & \forall a \in A \;\; \forall t \in T
\end{array}$$

Note that the second and third set of constraints together ensure that the distance that each agent $a \in A$ (more precisely, the a-th leftmost agent) moves in each time step $t \in T$ is at most 1, i.e., $|x_a^t - x_a^{t-1}| \leq 1$.

(QP1) is a Linear Program with a convex minimisation objective and can thus be solved in polynomial time, e.g., by using standard interior point methods (see [110]). Let $\nu \in \tilde{E}^{A \times T}$ be the continuous solution that we obtain by solving (QP1). If we then round ν according to our Rounding Theorem, we obtain a discrete solution which is a 10-approximation for the corresponding U-TkMP problem on the path graph. This completes the proof of Theorem 7.

3.6.2 Cycle case

We show that ideas similar to the ones used for the path case above, can be used to compute an optimal solution for the continuous problem CU-TkMP if the underlying topology is a cycle of length L, i.e., $\tilde{E} = ([0, L] \mod L)$.[2]

We obtain the following result from our Rounding Theorem (with $\beta = 1$ and $\Delta = 2$):

Theorem 8. *Let $n \geq 2k$. Then there is a 10-approximation algorithm for U-TkMP on the cycle graph $\mathcal{O}(convex(k_1, k_2) + n^3)$ time, where $k_1 = k_2 = \mathcal{O}(k\omega)$.*

As before, given a configuration $\sigma \in \tilde{E}^A$ we assume that the agents in $A = \{1, \ldots, k\}$ are indexed such that $\sigma_1 \leq \sigma_2 \leq \cdots \leq \sigma_k$. We introduce a single dummy agent $k+1$ whose position is defined as $\sigma_{k+1} = L + \sigma_1$. We can again assume that the agents will never want to switch index, because there exists an optimal solution in which they never switch. Note that we have $(\sigma_{k+1} \mod L) = \sigma_1$.

Lemma 7. *Consider a continuous configuration $\sigma \in \tilde{E}^A$ and suppose that the agents are indexed as stated above. Then the coverage cost is*

$$C(\sigma) = \frac{1}{4} \sum_{a=1}^{k} (\sigma_{a+1} - \sigma_a)^2$$

Proof. As in the proof of Lemma 6, we subdivide the cycle into k line segments and argue for each segment separately. Using the same arguments as in the proof of Lemma 6, the cost of each line segment $[\sigma_a, \sigma_{a+1}]$ with $a \in \{1, \ldots, k-1\}$ is $\frac{1}{4}(\sigma_{a+1} - \sigma_a)^2$.

It remains to consider the (cyclic) segment $([\sigma_k, \sigma_{k+1}] \mod L)$ covered by agents k and 1. It is not hard to see that the coverage cost of this segment is

$$\int_{\sigma_k}^{\sigma_{k+1}} \min_{u \in \{\sigma_k, \sigma_{k+1}\}} |v - u| \mathrm{d}v = 2 \int_0^{\frac{1}{2}(\sigma_{k+1} - \sigma_k)} v \mathrm{d}v = \frac{1}{4}(\sigma_{k+1} - \sigma_k)^2$$

Summing over all line segments proves the claim. □

[2]Note that the modulo operator ensures that 0 and L correspond to the same point.

In light of the above lemma, we can now adapt the Convex Program for the path case to the cycle case as follows:

$$
\begin{array}{lrcll}
\min & \frac{1}{4}\sum_{t\in T}\sum_{a=1}^{k}(\sigma_{a+1}-\sigma_a)^2 & & & & \text{(QP2)}\\
\text{s.t.} & 0 \le x_1^t & \le \cdots \le & x_k^t \le L & \forall t \in T & \\
& x_{k+1}^t & = & L + x_1^t & \forall t \in T & \\
& x_a^t - x_a^{t-1} & \le & 1 & \forall a \in A \;\; \forall t \in T & \\
& -x_a^t + x_a^{t-1} & \le & 1 & \forall a \in A \;\; \forall t \in T & \\
& x_a^t & \in & [0, L] & \forall a \in A \;\; \forall t \in T &
\end{array}
$$

As before, we can solve this program efficiently and use our Rounding Theorem to obtain a 10-approximation for U-TkMP in the cycle case.

3.7 Approximation algorithms for general graphs

Our Rounding Theorem can be used whenever we can solve (or approximate) the corresponding continuous version CU-TkMP of the problem. In this section, we derive approximation algorithms that do not rely on this rounding scheme. We first derive an $\mathcal{O}(\min\{\sqrt{\omega}, n\})$-approximation algorithm if the distance between S and W satisfies $d(S, W) \le 2\sqrt{\omega}$. We then show that we can obtain an $\mathcal{O}(d(S, W) + \sqrt{\omega})$-approximation algorithm if $d(S, W) > 2\sqrt{\omega}$.

Our first algorithm is described in Algorithm 3. The idea of this algorithm is as follows: We first find approximate medians, and then try to stay as close as possible to these medians. We do not only try to find good medians for all of V, but also for a smaller subset Ω that we can cover well without moving more than $\sqrt{\omega}$ hops from S and W. If the 'optimal medians' are indeed within $\sqrt{\omega}$ hops from S and W, then the V-medians will give us a constant-factor approximation; otherwise, the Ω-medians will give us an $\mathcal{O}(\sqrt{\omega})$-approximation.

We assume we have access to an α-approximation algorithm that runs in $\mathcal{O}(kmed(n))$ time for K-MED for some given facility and client sets; e.g., like the 6-approximation by Jain and Vazirani [81].

ALGORITHM 3: A 'mediate at $\sqrt{\omega}$'-algorithm for U-TkMP if $d(S, W) \le 2\sqrt{\omega}$

1: Find the reachable set $\Omega := \{v \in V : d(S, v) \le \theta \wedge d(v, W) \le \theta\}$, with $\theta := \lfloor\sqrt{\omega}\rfloor$.
2: Find medians σ by approximating K-MED with facilities V and clients V.
3: Take any bijection between agents and medians in σ, then form solution μ by minimising each agent's summed distance to their median, using Lemma 1.
4: Find medians $\sigma^{[\Omega]}$ by approximating K-MED with facilities Ω and clients Ω.
5: Take any bijection between agents to medians in $\sigma^{[\Omega]}$, then form solution $\mu^{[\Omega]}$ by minimising each agent's summed distance to their median, using Lemma 1.
6: **return** $\mu \leftarrow \arg\min\{D(\mu), D(\mu^{[\Omega]})\}$

Lemma 8. *Algorithm 3 runs in $\mathcal{O}(kmed(n) + n^5)$ time.*

Proof. Line 1 costs $\mathcal{O}(n)$ time. Lines 2 and 4 cost $\mathcal{O}(kmed(n))$ time. In lines 3 and 5, we seek $k \leq n/2$ walks, which according to Lemma 1, each cost $\mathcal{O}(n^4)$ time to find. □

Theorem 9. *If there exists an optimal configuration σ^* that is within $\lfloor\sqrt{\omega}\rfloor$ hops from the starting configuration and the ending configuration, then Algorithm 3 is an $(\alpha + 2 + \sqrt{2})$-approximation that runs in $\mathcal{O}(kmed(n) + n^5)$ time.*

Proof. Let OPT be the optimal solution value, and ALG the value of the solution from Algorithm 3. In μ, the agents are at some configuration σ (as determined by the algorithm) throughout $t = \theta, \theta + 1, \ldots, \omega + 1 - \theta$. We assumed that there is some optimal configuration $\sigma^* \subseteq \Omega$, so $D(\sigma) \leq \alpha D(\sigma^*)$. With Lemma 3 and Corollary 2, we obtain

$$\begin{aligned} D(\mu) &\leq (\omega + 1)D(\sigma) + n \cdot \sum_{t=1}^{\theta} t + n \cdot \sum_{t=1}^{\theta} t = (\omega + 1)D(\sigma) + (\theta + 1)\theta n \\ &\leq \alpha \cdot (\omega + 1)D(\sigma^*) + \omega n + \sqrt{\omega} n \leq (\alpha + 2 + \sqrt{2})OPT \end{aligned}$$

where the last inequality holds because $\frac{\sqrt{\omega}n}{\frac{1}{2}(\omega+1)n} \leq \frac{2}{\sqrt{\omega}} \leq \sqrt{2}$ for $\omega \geq 2$ (which we can assume). We conclude that $ALG \leq D(\mu) \leq (\alpha + 2 + \sqrt{2})OPT$. □

Theorem 10. *If $d(S, W) \leq 2\sqrt{\omega}$, Algorithm 3 is an $\mathcal{O}(\sqrt{\omega})$-approximation algorithm that runs in $\mathcal{O}(kmed(n) + n^5)$ time.*

Proof. For any configuration σ and $U \subseteq V$, define $D(\sigma, U) := \sum_{u \in U} d(\sigma, u)$. Let $V' := V \setminus \Omega$ be the nodes that are not reachable in 2θ hops, with Ω as defined in Algorithm 3. Also let $\vec{D} := \sum_{v \in V'} \min_{u \in \Omega} d(u, v)$ be the summed distance of V' to Ω. Let σ^* be an optimal configuration, and σ^*_Ω be a minimiser of $D(\sigma, \Omega)$.

We can split the value of some optimal solution μ^* over its contribution to Ω and V'. The former can be lower-bounded with a configuration σ^*_Ω that minimises $D(\sigma, \Omega)$ instead of $D(\sigma)$. The latter can be lower-bounded by the fact that there can be no agents in V' if $t \leq \sqrt{\omega} - 1$ or $t \geq \omega + 1 - (\sqrt{\omega} - 1)$, so in those time steps, at least a cost $\vec{D}$ is incurred. If $\vec{D} = 0$ or $\theta = 0$, then Theorem 9 gives us a constant-factor approximation, so assume $\vec{D} > 0$. Then

$$OPT \geq (\omega + 1)D(\sigma^*, \Omega) + D(\mu^*, V') \geq (\omega + 1)D(\sigma^*_\Omega, \Omega) + (\sqrt{\omega} + \sqrt{\omega})\vec{D}.$$

In $D(\mu^{[\Omega]})$, the agents will be at $\sigma^{[\Omega]}$ throughout $t = \theta, \theta + 1, \ldots, \omega + 1 - \theta$. By Lemma 3, we again know that the total cost cannot exceed $(\omega + 1)D(\sigma^{[\Omega]}) + (\theta + 1)\theta n$. While we know that $\sigma^{[\Omega]}$ is favourable for serving Ω, we need to upper-bound the cost of V' as well. For any $v \in V'$, if u is the nearest node in Ω to v, then we know that there is a median m at most 2θ hops away, because

$d(u,m) \leq d(u,S) + d(S,m) \leq 2\theta$. So $D(\sigma^{[\Omega]}) \leq D(\sigma^{[\Omega]},\Omega) + \vec{D} + |V'| \cdot 2\theta$. We have

$$\begin{aligned} ALG &\leq D(\mu^{[\Omega]}) \leq (\omega+1)D(\sigma^{[\Omega]}) + (\theta+1)\theta n \leq (\omega+1)D(\sigma^{[\Omega]}) + \omega n + \sqrt{\omega}n \\ &\leq (\omega+1)\left(D(\sigma^{[\Omega]},\Omega) + \vec{D} + |V'| \cdot 2\sqrt{\omega}\right) + \omega n + \sqrt{\omega}n, \\ \frac{ALG}{OPT} &\leq \frac{(\omega+1)D(\sigma^{[\Omega]},\Omega)}{(\omega+1)D(\sigma^*_\Omega,\Omega)} + \frac{(\omega+1)\vec{D}}{(2\sqrt{\omega})\vec{D}} + \frac{(\omega+1)|V'| \cdot 2\theta}{|V'| \cdot \frac{1}{2}(\theta+1)\theta} + \frac{\omega n + \sqrt{\omega}n}{\frac{1}{2}(\omega+1)n} \\ &\leq \alpha + \frac{\omega+1}{2\sqrt{\omega}} + \frac{4(\omega+1)}{\sqrt{\omega}+1} + 2 + \sqrt{2} = \mathcal{O}(\sqrt{\omega}) \end{aligned}$$

□

Though it is true that $\sqrt{\omega}$ can become arbitrarily large, we can always apply Theorem 9 when $\sqrt{\omega}$ exceeds n. Therefore, these results together make Algorithm 3 an $\mathcal{O}(\min\{\sqrt{\omega}, n\})$-approximation algorithm.

Next, we consider the case $d(S,W) > 2\sqrt{\omega}$. Then it may occur that Ω is empty. In that case, we resort to Algorithm 4, the performance of which is $\mathcal{O}(d(S,W) + \sqrt{\omega})$. Analogous statements of Lemma 8 and Theorem 9 clearly hold for Algorithm 4, but we explain the weaker performance bound in Theorem 11.

ALGORITHM 4: A 'mediate at $\sqrt{\omega}$'-algorithm for U-TkMP if $d(S,W) > 2\sqrt{\omega}$

1: Find the reachable set $\Psi := \{v \in V : d(S,v) + d(v,W) \leq d(S,W) + \theta\}$, with $\theta := \lfloor\sqrt{\omega}\rfloor$.
2: Find medians σ by approximating K-MED with facilities V and clients V.
3: Take any bijection between agents and medians in σ, then form solution μ by minimising each agent's summed distance to their median, using Lemma 1.
4: Find medians $\sigma^{[\Psi]}$ by approximating K-MED with facilities Ψ and clients Ψ.
5: Take any bijection between agents to medians in $\sigma^{[\Psi]}$, then form solution $\mu^{[\Psi]}$ by minimising each agent's summed distance to their median, using Lemma 1.
6: **return** $\mu \leftarrow \arg\min\{D(\mu), D(\mu^{[\Psi]})\}$

Theorem 11. *Assuming $d(S,W) > 2\sqrt{\omega}$, Algorithm 3 is an $\mathcal{O}(d(S,W) + \sqrt{\omega})$-approximation algorithm that runs in $\mathcal{O}(kmed(n) + n^5)$ time.*

Proof. We follow a similar argument as in Theorem 10. That is, we separately bound the cost that Ψ incurs in $\mu^{[\Psi]}$ and that $\tilde{V}' := V \backslash \Psi$ incurs. Denote $\vec{\Psi} := \sum_{v \in \tilde{V}'} \min_{u \in \Psi} d(u,v)$ the summed distance of $\tilde{V}'$ to Ψ. Note first that any $u \in \Psi$ and $v \in \Psi$ can have distance at most $d(u,v) \leq d(S,W) + \sqrt{\omega}$:

$$2d(u,v) \leq d(u,S) + d(S,v) + d(u,W) + d(W,v) \leq 2(d(S,W) + \sqrt{\omega})$$

Let μ^* be an optimal solution, let $\sigma^{*[\Psi]}$ be an optimal K-MED-configuration if the client set and facility set are both Ψ, and let $\mu^{*[\Psi]}$ be an optimal U-TkMP-solution for the induced subgraph $G[\Psi]$. Note also that $OPT := D(\mu^*) \geq \sqrt{\omega}\vec{\Psi}$, because for at least $\sqrt{\omega}$ time steps, no agent can be in $\tilde{V}'$. Therefore:

$$ALG \leq D(\mu^{[\Psi]}) = D(\mu^{[\Psi]}, \Psi) + D(\mu^{[\Psi]}, \tilde{V}') \tag{3.5}$$

$$D(\mu^{[\Psi]}, \Psi) \leq (\omega+1)D(\sigma^{[\Psi]}, \Psi) + \frac{(d(S,W)+\sqrt{\omega})(d(S,W)+\sqrt{\omega}+1)n}{2} \tag{3.6}$$

$$D(\sigma^{[\Psi]}, \Psi) \leq \alpha D(\sigma^{*[\Psi]}, \Psi) \leq \frac{\alpha D(\mu^*)}{(\omega+1)} \tag{3.7}$$

$$\frac{D(\mu^{[\Psi]}, \Psi)}{OPT} \leq \frac{\alpha D(\mu^*)}{D(\mu^*)} + \frac{\frac{1}{2}n(d(S,W)+\sqrt{\omega})(d(S,W)+\sqrt{\omega}+1)}{\frac{1}{2}n(\omega+1)} \tag{3.8}$$

$$\leq \alpha + \frac{d(S,W)^2 + 2d(S,W)\sqrt{\omega} + \omega + d(S,W) + \sqrt{\omega}}{\omega} \tag{3.9}$$

$$D(\mu^{[\Psi]}, \tilde{V}') \leq (\omega+1)(\vec{\Psi} + (d(S,W)+\sqrt{\omega})n) \tag{3.10}$$

$$\frac{D(\mu^{[\Psi]}, \tilde{V}')}{OPT} \leq \frac{(\omega+1)\vec{\Psi}}{\sqrt{\omega}\vec{\Psi}} + \frac{(\omega+1)n(d(S,W)+\sqrt{\omega})}{\frac{1}{2}(\omega+1)n} \tag{3.11}$$

In (3.6), we use Lemma 3 repeatedly, knowing that movement occurs in at most $(d(S,W)+\sqrt{\omega})$ hops. In (3.10), we use the fact that the diameter of Ψ does not exceed $d(S,W)+\sqrt{\omega}$. So if $v \in \tilde{V}'$ and $u = \min_{u \in \Psi} d(u,v)$, then the cost contribution of v cannot exceed $d(u,v) + d(S,W) + \sqrt{\omega}$. So in every time step, the nodes of $\tilde{V}'$ together cannot contribute more than $\vec{\Psi} + n(d(S,W)+\sqrt{\omega})$ to the cost. Put together, this means that $\frac{ALG}{OPT} \leq \frac{D(\mu^{[\Psi]},\Psi)}{OPT} + \frac{D(\mu^{[\Psi]},\tilde{V}')}{OPT}$ is an $\mathcal{O}(\sqrt{\omega} + d(S,W))$-approximation factor. □

Studying the approximation factor in the proof of Theorem 11, it would seem that Algorithm 4 performs poorly for high $d(S,W)$ and low ω. However, we remark that the most extreme version of this is observed only in the path graph, which is again 'easy'. It could be interesting to see if this 'gradual transition to path graphs' can help strengthen the performance bound.

3.8 Conclusions

We introduced TkMP and its uniform variant, U-TkMP. We provided several approximation algorithms for U-TkMP, including a novel method in which we round back an optimal or β-approximate solution to a 'continuous graph movement relaxation'.

As venues for future research, we suggest the following. There are likely more topologies than just path graphs and cycle graphs for which we can solve or approximate CU-TkMP. Extending the results of this chapter to heterogeneous start and end locations seems imaginable, once an equivalent to Lemma 4 can be found that allows us to again lower-bound $D(\sigma)$ by some function of only n. (It seems more difficult to let go of the edge lengths being uniform, because

this makes rounding back non-trivial, and we can no longer bound distances to unoccupied nodes to be between 1 and n.)

Chapter 4

Representative Distance-preserving Graph Sparsifiers

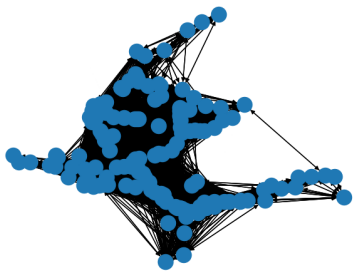
4.1 Introduction

In the previous chapter, we have explored the boundary of guaranteed approximation algorithms for MRP, by focussing on several special cases of MRP. From this chapter onward, instead of a special case, the thesis will move towards a generalisation of MRP: namely, a model that is rich enough to model the practical considerations that go into using MRP in a realistic application. This enriched problem will be the focus of the next chapter, Chapter 5. However, a key part of the algorithm proposed in that chapter is a graph sparsification subroutine that is already involved enough to merit a chapter of its own. In this chapter, therefore, we discuss this graph sparsification subroutine and demonstrate its use for MRP. Although on its own it does not improve a stronger algorithm for MRP than `MDSA`, we discuss the subroutine in the context of MRP instead of the generalisation it will be used for. This is because it is less convoluted to explain how this subroutine could be used for MRP, and once this is understood, it is not difficult to understand how it would be used for the generalisation of MRP.

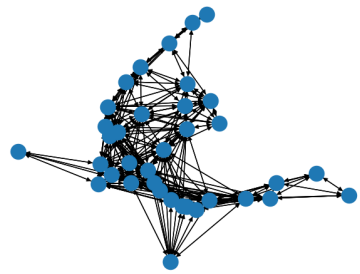
In many network optimisation problems, having an overfitted network may cost us a lot of computation time, while offering only marginal improvements to accuracy. In the Uncapacitated Facility Location Problem [153], for instance, if two potential facilities are very close together, it should hardly matter for the optimal solution cost if we delete one of these two facilities from the instance (namely the one with higher opening cost). In fact, if we can prune all facilities that are very close together and leave only one as local representative, we can probably strongly reduce computation time, while only marginally sacrificing solution quality. This idea, of replacing node clusters with just one representative

to save computation time while losing only little solution quality, seems sensible for many NP-hard problems, including the Vehicle Routing Problem [18] and its variants, various Facility Location Problems [153], the Travelling Salesman Problem [49] and variants of the Orienteering Problem [152]. It appears an especially promising idea in real-life applications such as delivery logistics, emergency response systems and road-side assistance, because these tend to contain many closely clustered data points arising from cities and settlements.

However, whatever problem we choose to apply this pruning on, we should be careful that our pruned network preserves feasibility. That is, if a feasible solution to our problem exists in the original network, there should still exist a feasible solution in the simplified network.



(a) Original graph (143 nodes, 8312 edges).



(b) A simpler, 'representative' subgraph (36 nodes, 530 edges).

Figure 4.1: A network from a Median Routing Problem use case, and a sparsified version.

In this chapter, essentially, we will transform the graph in Figure 4.1a into the simpler graph in Figure 4.1b, and discuss the methods by which we do so. Solving MRP on the simpler graph should lead to significantly shorter computation times, while hardly changing the optimal solution value. We will not only validate this empirically, but we will also compute a *proven bound* on how much solution quality we lose: that is, given a set of possible problem instances and a sparsified network, we will compute a tight upper bound on how much worse the optimal solution value can be when solving the problem instance on the sparsified network instead of the original network.

While one could simply interpret data in a coarser graph, we require the following four non-trivial characteristics of our sparsification: (1) the graph distances between kept nodes should not change (because we need to preserve feasibility), (2) a given set of nodes that are essential for feasible solutions (like a 'depot node') must be kept, (3) the number of kept nodes should be as small as possible (because we want optimal speed-up), and (4) how much accuracy we lose in our simplification should be quantified. To enable the latter, we will introduce a framework that allows us to compute worst cases.

Put together, this chapter addresses the fundamental question of how we can simplify graphs for NP-hard network problems and bound the loss of accuracy in doing so.

The contribution of this chapter is fivefold. First, *representative distance-preserving graph sparsifiers* are introduced, as well as a useful equivalence theorem for determining distance preservation. Second, we introduce four methods to compute representative distance-preserving graph sparsifiers. Third, we compare these methods on a Median Routing Problem use case, and see that we can indeed save 91.2% computation time while losing only 3.3% solution quality. Fourth, we introduce the Mixed Integer Jester Game framework for worst-case computation. Lastly, using this framework, we prove that for any allowed realisation in our Median Routing Problem use case, using a sparsified network instead of the original use case network leads to increases in optimal solution value that cannot exceed a factor 2.175.

The remainder of this chapter is structured as follows. In Section 4.2, we review related graph sparsification literature. In Section 4.3, we formally define our problem and introduce notation. In Section 4.4, we define distance-preserving graph sparsifiers and demonstrate four ways to determine them. In Section 4.5, we introduce Mixed Integer Jester Games. In Section 4.6, we briefly discuss for several OR problems how we could use such an RDGS to quickly compute heuristic solutions. In Section 4.7, we show how to use the MIJG framework to bound the performance loss for our chosen network optimisation problem. In Section 4.8, we describe the setup of our experiments, and in Section 4.9, we discuss the results. We end with our conclusions and outlook in Section 4.10.

4.2 Related literature

There are many applications for which the simplification of graphs has been studied, as outlined by Liu et al. [100]. These include social network analysis [104], improving privacy of social network data [96], reducing data storage and transfer [83, 158], clearer visualisation of graphs [75], solving linear systems [27, 87, 139], noise elimination [136], and most importantly for us, speeding up graph algorithms. In particular, graph sparsification has been used to speed up the computation of all-pairs shortest paths [47], k-skip shortest paths [142], maximum bipartite matchings [47], vertex and edge connectivity [47], approximate max-cut and min-flows [27], network flows [14, 106], flows and effective resistances in electrical networks [27], cover times of random walks [38], and for faster learning of labelled graph data [162].

Liu et al. categorise four different approaches to simplifying a graph [100]: graph aggregation, bit compression, graph sparsification and influence description. Technically, we will consider a graph sparsification problem, meaning that we will simplify a given graph merely by deleting nodes and edges from it. However, our goal is akin to that of graph aggregation, where nearby nodes are combined into supernodes. We will discuss the advances in both fields.

In the sparsification literature, many authors focus on spectral sparsification; that is, finding a subgraph that is 'σ-spectrally similar' [9] to the original graph, but only has a given fraction of the nodes. Many of the sped-up algorithms mentioned above make use of spectral sparsification. It was also used by Sadhanala et al. [130], who remove edges through spectral sparsification or sampling sparsification to perform Laplacean smoothing against much less computational effort, with proven bounds on accuracy loss. One of the currently most powerful spectral sparsifiers is the $(1 + 2\varepsilon, 4/\varepsilon^2)$-sparsifier by Batson et al. [8].

In distance-preserving sparsification and simplification, we see that many authors allow small mistakes in the distance preservation. Peleg and Schäffer [113] first introduced t-spanners: subgraphs obtained by removing edges, but such that the shortest-path distance between any pair of nodes does not increase by more than a factor t. Since then, many advances have been made in this field, as summarised recently by Ahmed et al. [2]. The recent work of Bodwin [15] discusses upper bounds on how many edges we need for a subgraph that preserves distance exactly, as well as for '$+k$-spanners', by which we mean that an additive error of k is allowed. Salim et al. [131] iteratively merge nodes and determine the new lengths of edges by solving a linear system of equations that minimises the shortest path error, achieving an average error of only 1%. On the contrary, Ruan et al. seek to preserve distances exactly between any pair of nodes [129] and seek a smallest 'gate vertex graph': a representation such that, for any pair of nodes in the original graph, there exists a path through the gate vertex graph where each next node is at most some threshold ε away from the current node, meaning we can transform a shortest path in the gate vertex graph back to a shortest path in the original graph by finding 'local' shortest paths of length at most ε.

We also outline here some relevant results in graph aggregation. LeFevre and Terzi [96] formalised the answering of queries on summarised graphs, like adjacency, degree and eigenvector centrality, using a random-worlds framework. In addition, they developed different algorithms to find summarised graphs with a focus on either data compression or strong protection of privacy. These algorithms make use of an expected adjacency matrix in a more or less greedy way. Riondato et al. [127] improved on their GraSS-algorithm by finding summaries with guaranteed quality in polynomial time. Toivonen et al. [146] seek to aggregate nodes into supernodes and weighted edges into weighted superedges, such that given any desired number of superedges, the 'Euclidean' distance between the original graph and the summarised graph is minimal after decompression. This 'Euclidean' distance is initially based on how similar the weights are on edges between any two nodes, but they generalise this notion to seeking similar path distances between nodes. Using their semi-greedy method, among others, they manage to compress a 16,000 edge graph in two seconds with little loss of information.

We conclude that our problem is atypical against the common literature, in two ways. First, instead of summarising massive graphs for relatively 'easy' graph

queries, we aim to summarise modestly sized graphs for NP-hard queries. This means our focus is less on the complexity of simplifying the graph, but more on the approximation guarantees for the optimisation problem on that graph. Second, we are allowed to completely disconnect and disregard nodes (except for a given set), but do not allow any changes in distance between the nodes we *do* keep, in order to maintain feasibility.

4.3 Problem definition

Our goal, in loose terms, is to simplify a given minimisation problem on a graph by limiting ourselves to solutions that only use a 'representative' subgraph. This way, we want to compute a solution in much less time, with this solution hopefully being not far from optimal.

Suppose we have a graph $G = (V, E, d)$, with d the (potentially unit) weights of the edges. Denote $n := |V|$. We do not need G to be connected, directed or undirected, and we do not need d to be non-negative. However, we assume that no negative cost cycles exist, which can be checked in polynomial time [97]. For convenience, denote by $D_G(u, v)$ the shortest path distance in G from any node $u \in V$ to node $v \in V$, which equals ∞ if v is not reachable from u. For any $V' \subseteq V$, denote $G[V']$ the induced subgraph of V'.

Suppose also that we want to solve a scenario-based minimisation problem on this graph. That is, the problem instance is drawn from some set of scenarios Z. We assume Z can be described as the set of feasible vectors of a Mixed Integer Linear Program. When the scenario $z \in Z$ is revealed, we want to use a Mixed Integer Linear Programming solver to solve the minimisation problem

$$\Pi(z) := \min_x \{f^T x | Ax \leq b + \tilde{A}z, x \text{ mixed integer}\}$$

That is, we have some formulation for our minimisation problem as a Mixed Integer Linear Program, with z only affecting the right-hand side of the constraints. For convenience of definitions, we make the natural assumption that $\Pi(z) > 0$ for all $z \in Z$.

In order to speed-up computation, suppose we decide to only 'use' a subgraph G' of G in our solutions. For instance, in the Uncapacitated Facility Location Problem, we decide to disregard a subset of possible facility locations. More generally, we have a 'mechanism' that modifies $\Pi(z)$ based on our decision to only use G', and instead of solving $\Pi(z)$, we solve

$$\Pi_{G'}(z) := \min_x \{f_{G'}^T x | A_{G'} x \leq b_{G'} + \tilde{A}_{G'} z, x \text{ mixed integer}\}$$

How the matrices $f_{G'}$, $A_{G'}$, $b_{G'}$ and $\tilde{A}_{G'}$ differ from f, A, b and $\tilde{A}$ will depend on the problem and sparsification strategy. However, in many applications, $\Pi_{G'}(z)$ will only differ from $\Pi(z)$ in that it has additional constraints that fix certain variables to 0, because they 'use' nodes and edges outside of G'. Regardless of

what mechanism is used, we also need a function $\psi_{G'}(x)$ that translates feasible solutions of $\Pi_{G'}(z)$ back to feasible solutions of $\Pi(z)$; in many applications, the identity function suffices. In Section 4.6, examples of sparsification mechanisms for different problems are proposed.

Given this graph, problem and scenario set, we want to design a sparsification mechanism $(\Pi_{G'}, \psi_{G'})$ and obtain a factor $\alpha \geq 1$ for which the following hold:

1. For any $z \in Z$, if $\Pi(z)$ has an optimal solution x^*, then an optimal solution $\hat{x}$ of $\Pi_{G'}(z)$ exists and $x' := \psi_{G'}(\hat{x})$ is feasible for $\Pi(z)$.

2. Empirically, we need much less time after $z \in Z$ is revealed to find x' than to find x^*.

3. Empirically, $f^T x'$ is not much larger than $f^T x^*$.

4. Provably, $f^T x' \leq \alpha f^T x^*$ for any scenario $z \in Z$.

We distinguish between two approaches to attain these goals. In *reactive sparsification*, we 'wait until $z \in Z$ is revealed': that is, we take the revealed z as part of the input in determining $(\Pi_{G'}, \psi_{G'})$. In *universal sparsification*, we already determine $(\Pi_{G'}, \psi_{G'})$ before $z \in Z$ is revealed. The upside of reactive sparsification is that we have more information to perform our sparsification with. The downside is that, with respect to the goal of saving time, any computations needed to determine $(\Pi_{G'}, \psi_{G'})$ are counted towards the time needed to determine x'. Put differently, the upside of universal sparsification is that we are allowed to count computing $(\Pi_{G'}, \psi_{G'})$ as a 'preprocessing' step.

4.4 Four ways to compute representative subgraphs

In this section, we will formally define *representative distance-preserving graph sparsifiers*, and discuss four algorithms by which to obtain them.

Given our graph $G = (V, E, d)$, suppose we choose $R \in \{0, 1\}^{V \times V}$ as a 'representation matrix': that is, $R_{uv} = 1$ if and only if node $u \in V$ can 'represent' node $v \in V$. For example, we could say that u and v can represent each other if their Euclidean distance is below some threshold. If some node u is crucial for feasible solutions, like the 'depot' node, we can encode this by letting the only node that can represent u be u itself. We define an induced subgraph $G[V']$ to be *R-representative* of G if, for every $v \in V$, there exists a $u \in V'$ with $R_{uv} = 1$. In the remainder of this thesis, we will refer to subgraphs as simply being 'representative' instead of R-representative, trusting that which R they are representative of is clear from the context. We also define an induced subgraph $G[V']$ to be *distance-preserving* if, for every pair of nodes $u \in V', v \in V'$, the shortest path length from u to v is the same in both G and G', so $D_G(u, v) = D_{G'}(u, v)$.

Definition 1. *Given a weighted directed graph $G = (V, E, d)$, a representation matrix $R \in \{0,1\}^{V \times V}$ and a subset $V' \subseteq V$, we will say that $G[V']$ is a* representative distance-preserving graph sparsifier *(RDGS) if $G[V']$ is distance-preserving and R-representative.*

In this chapter, we will look for an RDGS that is 'good' to solve an NP-hard network problem on. This means that we want $G[V']$ to be as small as possible, so as to speed up computation, but that we must keep certain nodes that each feasible solution uses (like the 'depot node'). Moreover, the graph distances between these mandatory nodes should not increase, because it may be that feasible solutions only exist if shortest paths are used. That is, in many applications, an RDGS will preserve feasibility because the mandatory nodes can still be visited and the distance between them does not change. In addition to finding an RDGS that saves as much computation time as possible, we also want to lose as little solution quality as possible when solving an NP-hard problem on this RDGS.

We present here four different algorithms that can be used to obtain an RDGS. The first two are geared more towards *universal* sparsification, the other two towards *reactive* sparsification. That is, the latter two aim to have lower build time, accepting that they may produce a larger subgraph than more extensive methods would. All four methods, however, can be used both for universal and reactive sparsification.

Remark that it is NP-hard to find a smallest RDGS. In fact, it is already NP-hard to find a smallest subset V' such that $G[V']$ is R-representative for general R, as well as finding a smallest completion $V'' \subseteq V'$ such that $G[V'']$ is distance-preserving for a general (possibly mandatory) starting set V'. These facts are proven in Theorems 12 and 13 respectively.

Theorem 12. *Given a representation matrix $R : V^2 \to \{0,1\}$, finding a smallest subset $V' \subseteq V$ such that $(\forall v \in V)(\exists u \in V' : R_{uv} = 1)$ is NP-hard in general.*

Proof. Take any instance of the Set Cover Problem, that is, a universe U and subsets $S \subseteq \mathcal{P}(U)$. Create an instance of the above problem by creating an 'element node' for each $u \in U$ and a 'set node' for each $s \in S$. Let $R_{ij} = 1$ if $i \in S$ and $j \in i$, or if $j \in S$; if neither, then $R_{ij} = 0$. The '$j \in i$'-case ensures that we only cover U when we add set nodes i to V' that together span U. The '$j \in S$'-case implies that once we add one set node to V', all set nodes have a representative and are no longer a worry. We implicitly assume that there exists no Set Cover of size 0, but this follows under the natural assumption that $U \neq \emptyset$. In conclusion, creating this problem instance can be done in polynomial time with respect to the Set Cover Problem input, the problem is in NP, and the two problem instances are equivalent. □

Theorem 13. *Suppose we are given a weighted digraph $G = (V, E, d)$ and some initial, 'mandatory' nodes $V' \subset V$. Suppose we want to add as few nodes to V' as possible such that $G[V']$ is distance-preserving. This problem in NP-hard in general.*

Proof. We construct an instance of this problem based on an arbitrary instance of the Set Cover Problem: that is, take any universe U and collection of non-empty subsets $S \subseteq \mathcal{P}(S)$. Now, create the following undirected, unit weighted graph G. Take as node set $V = U \cup V_2$, with $V_2 := \bigcup_{s \in S} \{s_1, s_2\}$. Create the edge set E by, for each $u \in U$ and $s \in S$, adding edges $\{u, s_1\}$ and $\{u, s_2\}$ to E if and only if $u \in s$. Finally, add $\{u, v\}$ to E for every $u \in U$ and $v \in U$, so that U is a clique in G. Note that this instance construction can be done in polynomial time with respect to the set cover instance, and that the problem is in NP.

Take as initial, 'mandatory' node subset $V' \leftarrow V_2$. Note that, at this point, $D_{G[V']}(u, v) = \infty$ for every $u \in V'$ and $v \in V'$ with $u \neq v$, so $G[V']$ is not distance-preserving. We will prove that adding the fewest nodes such that $G[V']$ is distance-preserving is equivalent to finding a smallest set cover.

Remark first that distances in G are as follows, for any $u \in V$ and $v \in V$ with $u \neq v$:

- If $u \in U$ and $v \in U$, then $D_G(u, v) = 1$ (because U is a clique in G);
- If $u \in U$ and $v = s_i \in V_2$, then $D_G(u, v) = D_G(v, u)$ equals 1 if $u \in s$ and 2 otherwise (because $\{u, s_i\} \notin E$, but for any $w \in s$, the path (s_i, w, u) exists);
- If $u = s_i \in V_2$ and $v = t_j \in V_2$, then $D_G(u, v)$ equals 2 if $s \cap t \neq \emptyset$ (because $\{s_i, t_j\} \notin E$, but for any $w \in s$, the path (s_i, w, t_j) exists) and 3 otherwise (because there is no $w_1 \in V$ with $\{s_i, w_1\} \in E$ and $\{w_1, t_j\} \in E$, but if $w_1 \in s$ and $w_2 \in t$, then the path (s_i, w_1, w_2, t_j) exists).

While, for any V' that is a strict superset of V_2, we have the following distances in $G' := G[V']$ for any $u \in V'$ and $v \in V'$ with $u \neq v$:

- If $u = s_i \in V_2$ and $V' \cap s = \emptyset$, then $D_{G'}(u, v) = D_{G'}(v, u) = \infty$ for all $v \in V'$;
- If $\neg(u = s_i \in V_2 \wedge V' \cap s = \emptyset)$ and $\neg(v = t_j \in V_2 \wedge V' \cap t = \emptyset)$, then if $D_G(u, v) = d$ due to some argument given above, $D_{G'}(u, v) = d$ due to the same argument, except we replace all instances of "$w \in s$" by "$w \in V' \cap s$", and likewise for "$w_1 \in s$" and "$w_2 \in t$".

So indeed, $G[V']$ is distance-preserving if and only if $V' \cap U$ is a set cover. Therefore, finding a smallest $V' \supset V_2$ such that $G[V']$ is distance-preserving is equivalent to solving the Set Cover Problem. □

4.4.1 The `Smallest` RDGS algorithm

Because it is NP-hard to find a smallest RDGS, we will first describe a Mixed Integer Linear Program (MILP) approach to find one.

When formulating a MILP for finding a smallest RDGS, it is non-trivial how to encode that the subgraph should be distance-preserving. It may be tempting

to do so by demanding that, for every $u \in V'$ and $v \in V'$, a unit (u, v)-flow must go from u and v and that the total weight of edges used for this does not exceed $D_G(u, v)$. However, such a formulation would require the inclusion of $\mathcal{O}(|V|^4)$ flow variables into the MILP. Instead, our first contribution is that we can write the distance-preservation constraint much more leanly using the following equivalence theorem with 'on-ramps'.

Definition 2. *Nodes $u \in V$ and $v \in V$ are* distant *if v is reachable from u in G, $u \neq v$ and $(u, v) \notin E$. Given graph G, we denote by $\vec{F} \subset V^2$ its set of distant nodes.*

Definition 3. *For any distant nodes $(u, v) \in \vec{F}$, we define the set of* on-ramps from u to v *as $\Lambda(u, v) := \{w \in V : (u, w) \in E,\ D_G(u, v) = d(u, w) + D_G(w, v)\}$.*

In words, node w is an on-ramp from u to v if there exists a shortest (u, v)-path that has w as its first stop after u. See also Figure 4.2. The name 'on-ramp' is symbolic to the idea that if one wishes to drive to another city, the first target in the road network is likely one of the on-ramps onto the motorway; however, one will not use the on-ramps on the opposite side of the city, or any on-ramps at all if the destination is within the same city.

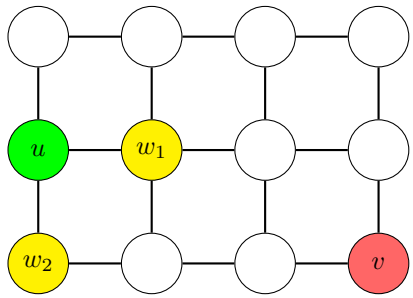


Figure 4.2: In this undirected graph with unit edge length, w_1 and w_2 are (all of the) on-ramps from u to v. In other words, $\Lambda(u, v) = \{w_1, w_2\}$. Indeed, all shortest (u, v)-paths first pass through w_1 or w_2.

Theorem 14 (On-ramp equivalence)**.** *$G' = G[V']$ is distance-preserving if and only if, for every $u \in V'$ and $v \in V'$ with $(u, v) \in \vec{F}$, $\Lambda(u, v) \cap V' \neq \emptyset$.*

Proof. Suppose $G[V']$ is distance-preserving. Take any $u \in V'$ and $v \in V'$ with $(u, v) \in \vec{F}$. Observe any shortest (u, v)-path in G', and denote by $w \in V'$ the first node visited after u. Because $G[V']$ is distance-preserving, this path is also a shortest path in G, meaning $w \in \Lambda(u, v)$. Therefore, $\Lambda(u, v) \cap V' \neq \emptyset$.

Suppose, for every $u \in V'$ and $v \in V'$ with $(u, v) \in \vec{F}$, $\Lambda(u, v) \cap V' \neq \emptyset$. Pick some $u \in V'$ and $v \in V'$. If $u = v$ or $(u, v) \in E$ or v is not reachable from u in G, the distance from u to v is trivially preserved in G'.

Otherwise, $(u, v) \in \vec{F}$, and we construct a (u, v)-path p in $G[V']$ through the following recursion. Initialise p node-wise as (u). Denote by u' the currently last entry of p. If $(u', v) \in E$, we add v to p and p is completed. Otherwise, $(u', v) \in \vec{F}$, so we take some $w \in \Lambda(u', v) \cap V'$ and append w to p, repeating until p is completed. By definition of B, every arc (w_i, w_{i+1}) traversed by this p has length $d(w_i, w_{i+1}) = D_G(w_i, v) - D_G(w_{i+1}, v)$. Substituting this expression into $\sum_{i=1}^{|p|-1} d(w_i, w_{i+1})$, we find that the total length of p equals $D_G(u, v) - D_G(v, v) = D_G(u, v)$. So $D_{G'}(u, v) \leq D_G(u, v)$. G' is a subgraph, so $D_{G'}(u, v) \geq D_G(u, v)$. So $D_G(u, v) = D_{G'}(u, v)$ and G' is distance-preserving. □

Interestingly, this equivalence theorem is similar to an observation made by Ruan et al. [129] for 'gate vertices'. Because of Theorem 14, we can find a smallest subgraph with the following MILP, of only n binary variables and fewer than $n^2 + n$ constraints. Let variable $x_v \in \{0, 1\}$ denote whether node $v \in V$ is included into V'. Then we solve:

$$\begin{aligned}
\min \ & \sum_{v \in V} x_v \\
s.t. \ & \sum_{u \in V} R_{uv} x_u \geq 1 && (\forall v \in V) && (4.1) \\
& \sum_{w \in \Lambda(u,v)} x_w \geq x_u + x_v - 1 && (\forall (u, v) \in \vec{F}) && (4.2) \\
& x_u \in \{0, 1\} && (\forall u \in V)
\end{aligned}$$

Constraint (4.1) ensures that $G[V']$ is R-representative, and constraint (4.2) ensures that it is distance-preserving, as implied by Theorem 14. If V' is the subset of nodes selected by this MILP, we name $G[V']$ the `Smallest` RDGS.

4.4.2 The `Realigned Smallest` RDGS algorithm

The outcome of the Smallest RDGS algorithm is typically not unique. That is, if n' is the number of nodes in any smallest RDGS, there are likely several solutions of n' nodes. Recalling our goal of finding a subgraph that represents the original graph as well as possible, we will try to break this tie by choosing an RDGS which is 'as near as possible' to the original graph. We assume we have some notion of distance C_{uv} between any two nodes $u \in V$ and $v \in V$: depending on the application, this could for instance be the Euclidean distance or the graph distance D_G. We then find a 'nearest' RDGS by solving an n'-Median Problem [35], with an additional representation constraint and distance-preservation constraint. That is, if we let binary variable y_{uv} indicate whether $u \in V$ is the nearest selected node to $v \in V$, we find an RDGS by solving the following MILP:

$$
\begin{aligned}
\min \quad & \sum_{u \in V, v \in V} C_{uv} y_{uv} \\
s.t. \quad & (4.1) - (4.2) \\
& \sum_{u \in V} x_u \leq n' && (4.3) \\
& \sum_{u \in V} y_{uv} = 1 \qquad (\forall v \in V) && (4.4) \\
& y_{uv} \leq x_u \qquad (\forall u \in V)(\forall v \in V) && (4.5) \\
& x_u, y_{uv} \in \{0, 1\} \qquad (\forall u \in V)(\forall v \in V)
\end{aligned}
$$

Constraints (4.1) and (4.2) still ensure that the subgraph is representative and distance-preserving, respectively. The objective and constraints (4.3) – (4.5) form the typical k-Median Problem representation: we want to minimise the total distance from unpicked nodes to our picked nodes or 'medians'. Constraint (4.3) tells us that we can pick at most n' nodes; constraint (4.4) implies that each client needs 'coverage' from somewhere, but constraint (4.5) ensures that coverage can only be given from nodes that were picked as medians.

We name the resulting subgraph the `Realigned Smallest` RDGS. We could also, at this stage, allow more medians than just n' to get a subgraph $G[V']$ with less loss of accuracy, at the cost of including more nodes.

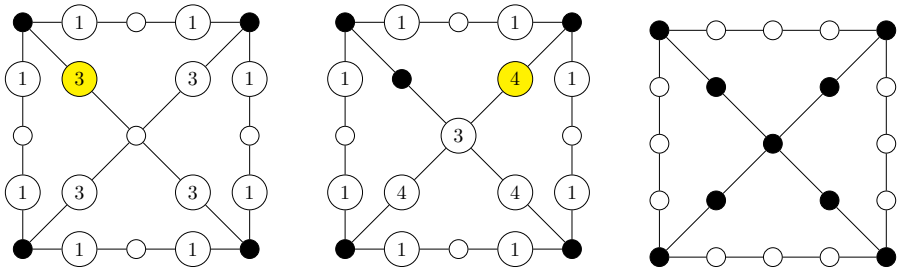
4.4.3 The `Greedy On-Ramps` RDGS algorithm

Despite the fact that we focus on simplifying 'modestly sized' graphs, we still have reasons to study how to construct an RDGS heuristically. First, including more scalable methods in our study makes our results generalisable to a larger range of applications. Second, if a user wants to heuristically solve a network problem as quickly as possible, sparsifying the network only heuristically may save a lot of valuable computation time. For these two reasons, we will discuss two greedy sparsification algorithms.

In the `Greedy On-Ramps` RDGS algorithm, we will again use Theorem 14 to pursue distance-preservation. First, we greedily construct a subset that is R-representative, using Algorithm 5. Next, as long as our subgraph contains pairs $(u, v) \in \vec{F}$ for which the distance is not yet preserved, we repeatedly add the on-ramp that benefits most of these unfulfilled pairs. This procedure is described in Algorithm 6 and illustrated in Figure 4.3.

4.4.4 The `Greedy Centrality` RDGS algorithm

The Greedy On-Ramps RDGS algorithm has quite a naive approach to extending a representative subgraph into an RDGS. That is, if some $(u, v) \in \vec{F}$ still needs its shortest path to be included, we literally look only one step ahead by adding an



(a) First pick, assuming the four black nodes resulted from `Greedy Cover`.

(b) Next pick.

(c) End result.

Figure 4.3: `Greedy On-Ramps` selection for an undirected graph with unit edge lengths. The black nodes are currently included in V'. Each white 'on-ramp' node has a number in it, indicating for how many unconnected pairs it is an on-ramp. Indeed, the first yellow node is an on-ramp for three connections: it's the first stop in a shortest path from the top-left black node to the top-right black node, from top-left to bottom-right, and from top-left to bottom-left. In each iteration, the yellow node is the one we greedily pick to include next, because it is an on-ramp for the most unconnected pairs of black nodes.

on-ramp of (u, v). Moreover, though we try to add a node w that is an on-ramp for as many unconnected pairs as possible, we expect this to result in arbitrary tie-breaking very often.

Therefore, in our final RDGS algorithm, we try to look slightly further ahead in our greedy selection: we will add the node with highest 'centrality', that is, the node featured on a shortest path for as many unconnected pairs as possible. This approach, which we name the `Greedy Centrality` RDGS algorithm, is described in Algorithm 7 and illustrated in Figure 4.4.

ALGORITHM 5: The `Greedy Cover` algorithm.

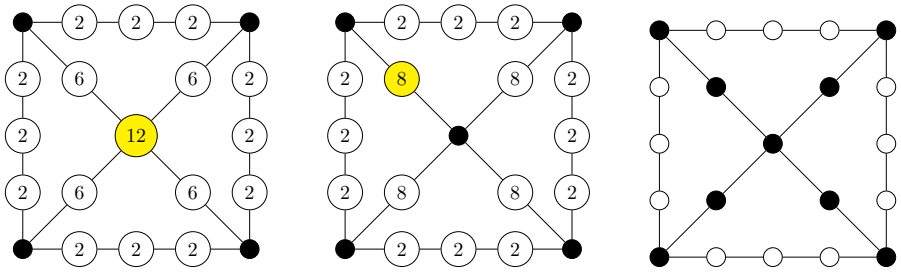
1: Initialise the picked nodes as $V' \leftarrow \emptyset$
2: Initialise the nodes still in need of coverage as $U \leftarrow V$
3: **while** $U \neq \emptyset$ **do**
4: Find $u = \arg\max_{u' \in V \setminus V'} \sum_{v \in U} R_{u'v}$, the unpicked node that covers most uncovered nodes
5: Add u to V', and remove all nodes represented by u from U
6: **end while**
7: **return** V'

ALGORITHM 6: The `Greedy On-Ramps` RDGS algorithm.

Initialise the picked nodes as $V' \leftarrow$ `Greedy Cover`
Initialise the unconnected pairs as
$U \leftarrow \{(u,v) \in \vec{F} : u \in V', v \in V', \Lambda(u,v) \cap V' = \emptyset\}$
while $U \neq \emptyset$ **do**
    Find $w = \arg\max_{w' \in V \setminus V'} \mid \{(u,v) \in U : w' \in \Lambda(u,v)\} \mid$, the unpicked node that is an on-ramp for the most unconnected pairs
    Add w to V', update $U \leftarrow \{(u,v) \in \vec{F} : u \in V', v \in V', \Lambda(u,v) \cap V' = \emptyset\}$
end while
return $G[V']$

ALGORITHM 7: The `Greedy Centrality` RDGS algorithm.

Initialise the picked nodes as $V' \leftarrow$ `Greedy Cover`
Initialise the unconnected pairs as
$U \leftarrow \{(u,v) \in V' \times V' : D_{G[V']}(u,v) > D_G(u,v)\}$
Initialise $S(u,v) \leftarrow \{w \in V \setminus V' : D_G(u,w) + D_G(w,v) = D_G(u,v)\}$ for each $(u,v) \in U$, the unpicked nodes on shortest (u,v)-paths of unconnected (u,v)
while $U \neq \emptyset$ **do**
    Find $w = \arg\max_{w' \in V \setminus V'} \mid \{(u,v) \in U : w' \in S(u,v)\} \mid$, the unpicked node that is on a shortest path for the most unconnected pairs
    Add w to V'
    Update $U \leftarrow \{(u,v) \in V' \times V' : D_{G[V']}(u,v) > D_G(u,v)\}$, checking only those (u,v) with $w \in \Lambda(u,v)$
    Update $S(u,v) \leftarrow \{w \in V \setminus V' : D_G(u,w) + D_G(w,v) = D_G(u,v)\}$ for each $(u,v) \in U$
end while
return $G[V']$



(a) First pick, assuming the four black nodes resulted from `Greedy Cover`.

(b) Next pick.

(c) End result.

Figure 4.4: `Greedy Centrality` selection for an undirected graph with unit edge lengths. The black nodes are currently included in V'. Each white node has a number in it, indicating for how many unconnected pairs it is on a shortest path. The yellow node is the one we greedily pick to include next, because it is on a shortest path for the most unconnected pairs of black nodes, namely all twelve considered pairs.

4.5 Mixed Integer Jester Games

In this study, we are not only interested in sparsifying a graph: we also want to know how this sparsification can affect the solution quality of network problems solved on this graph. In order to track down the worst-case quality loss, we introduce in this section a new algorithmic methodology.

We introduce here the notion of a *Mixed Integer Jester Game (MIJG)*. In this competitive game, we have three players, each solving their own optimisation problem. Player 1 is the *king*, who has access to an MILP solver with which to solve his minimisation problem. Player 2 is the *usurper*, who only has access to an LP solver with which to solve his minimisation problem. Player 0 is the *jester*: she controls some background variables z^0 which influence the problem that the king and the usurper have to solve. Let Z be the set of vectors z^0 she can choose from, namely the scenario set defined in Section 4.3. Her goal is to set the background variables such that the king does well and the usurper does poorly. That is, if $M^1(z^0)$ is the solution value of the king given her choice of z^0, and $M^2(z^0)$ is the solution value of the usurper given z^0, then the jester wants to find the z^0 which maximises $M^2(z^0) - \alpha M^1(z^0)$ for some given factor $\alpha \geq 1$. Note that the jester and the king can communicate freely to achieve this goal together. Eventually, our goal is to claim that what the usurper does is always an 'α-approximation' of what the king can do. That is, we want to show that $M^2(z^0) \leq \alpha M^1(z^0)$ for every possible scenario $z^0 \in Z$. Given any feasible $z \in Z$, denote

$$M^1(z^0) := \min_{\text{mixed integers } x^1} \left\{ (c^1)^T x^1 \mid A^1 x^1 \geq b^1(z) \right\}$$

and

$$M^2(z^0) := \min_{\text{continuous } x^2} \left\{ (c^2(z^0))^T x^2 \mid A^2 x^2 \geq b^2 \right\}$$

to be the optimisation problems of the king and the usurper, respectively. We do not allow z to appear in the cost coefficients c^1 of M^1, because we wish to avoid Quadratic Programming. We do not allow z to appear in the right-hand side b^2 of M^2 for a similar reason, which we will discuss later in this section.

The question is: how do we determine $\max_{z^0}\{M^1(z^0) - \alpha M^2(z^0)\}$? Unfortunately, this is not just a matter of inputting $\max\{(c^2(z^0))^T x^2 - \alpha (c^1)^T x^1 \mid A^1 x^1 \geq b^1(z^0), A^2 x^2 \geq b^2\}$ into an MILP solver. Indeed, $(c^1)^T x^1$ would behave like a minimum, because to maximise the overall expression, $(c^1)^T x^1$ should be low. But $c^2(z^0)^T x^2$ would *not* behave like a minimum, because for the overall expression to be maximal, $c^2(z^0)^T x^2$ should be high instead of low. But because we defined $M^2(z^0)$ to use exclusively continuous variables, we can solve this issue by replacing $M^2(z^0)$ by its dual,

$$\tilde{M}^2(z^0) := \max_{\text{continuous } y^2} \left\{ (b^2)^T y^2 \mid (A^2)^T y^2 \leq c^2(z^0) \right\}$$

which means that we can solve, for any $\alpha \geq 1$,

$$M^0(\alpha) := \max_{z^0, x^1, y^2} \left\{ (b^2)^T y^2 - \alpha (c^1)^T x^1 \mid A^1 x^1 \geq b^1(z^0), (A^2)^T y^2 \leq c^2(z^0) \right\}$$

This is something we can input into an MILP solver. Note that z^0 does not show up in the coefficients, because we only allowed it in the right-hand side of $M^1(z^0)$ and in the objective of $M^2(z^0)$. However, if we want to assign player 2 a problem which has z^0 in the right-hand side of some constraints, this is not an issue: we can simply apply Lagrangian relaxation to move those constraints into the objective, and set the Lagrangian multipliers arbitrarily high to ensure the constraints are still abided by.

4.5.1 α-iteration

How can we find the strongest approximation factor α? Suppose that, for some given $\alpha \geq 1$, we solve $M^0(\alpha)$ and find that $M^0(\alpha) < 0$. Then we have succeeded in finding an α with $M^2(z^0) < \alpha M^1(z^0)$ for every possible scenario z^0. In other words, we have validated that for every possible scenario z^0, the usurper's algorithm 'is an α-approximation' of the king's algorithm.

One can easily find a first α_0 for which this holds: namely, $\alpha_0 := 1 + M^0(1)/\min_z M^1(z)$. Indeed, for every scenario z^0, we have $M^2(z^0) \leq \alpha_0 M^1(z^0)$:

$$\begin{aligned} M^2(z^0) - M^1(z^0) &\leq M^0(1) \\ M^2(z^0) &\leq M^1(z^0) + M^0(1) = \left(1 + \frac{M^0(1)}{M^1(z^0)}\right) \cdot M^1(z^0) \\ &\leq \left(1 + \frac{M^0(1)}{\min_z M^1(z)}\right) M^1(z^0) = \alpha_0 M^1(z^0) \end{aligned}$$

Can we make our approximation claim stronger by finding a smaller α' for which it holds? In fact, can we track down the smallest α^* with $M^2(z^0) \leq \alpha^* M^1(z^0)$? If we ever find a pair (α^*, z^*) with $M^0(\alpha^*) = \max_{z^0} M^2(z^0) - \alpha^* M^1(z^0) = M^2(z^*) - \alpha^* M^1(z^*) = 0$, then we have found a 'tight' α^*: namely, because $M^2(z) - \alpha^* M^1(z) \leq 0$ for all z, and every smaller $\alpha' < \alpha^*$ would give us $M^2(z^*) - \alpha M^1(z^*) > 0$. This means that, to find the strongest approximation claim, we need to find the root α^* of $M^0(\alpha)$. Fortunately, under some natural assumptions on M^1 and M^2, the Newton-Raphson method works quite well for this. We describe this process as Algorithm 8, illustrate it in Figure 4.5 and prove that it performs well in Theorem 15.

ALGORITHM 8: The α-iteration algorithm for Mixed Integer Jester Games.

```
Initialise α ← 1 + M^0(1)/ min_z M^1(z)
Initialise tight ← false
while ¬tight do
    Compute M^0(α), denote z ← arg M^0(α)
    Update α ← M^2(z)/M^1(z)
    if M^0(α) = 0 then
        tight ← true
    end if
end while
return α
```

Theorem 15. *Suppose $M^2(z) \geq M^1(z) > 0$ for every scenario $z \in Z$. Then Algorithm 8 finds a tight pair (α^*, z^*) for the inequality $M^2(z) \leq \alpha^* M^1(z)\ \forall z \in Z$ after no more than $|Z|$ iterations.*

Proof. As earlier discussed, a pair (α^*, z^*) is tight if and only if α^* is the root of the function $M^0(\alpha) = \max_{z \in Z} M^2(z) - \alpha M^1(z)$, so if $M^2(z^*) - \alpha^* M^1(z^*) = 0$. Note that, for every fixed $z \in Z$, the function $\alpha \to M^2(z) - \alpha M^1(z)$ is a descending line that starts at $M^2(z) > 0$. Therefore, the function $M^0(\alpha)$ is the maximum over $|Z|$ such descending lines, which is a convex and piece-wise linear function that has exactly one root.

Observe, therefore, that there exists an interval I around α^* on which the function $\alpha \to M^0(\alpha)$ equals the line $\alpha \to M^2(z^*) - \alpha M^1(z^*)$. Suppose our current guess α is in I, meaning $\arg M^0(\alpha) = z^*$. Then our next guess, $\alpha^* = M^2(z^*)/M^1(z^*)$, has $M^0(\alpha^*) = M^2(z^*) - \alpha^* M^1(z^*) = 0$, meaning (α^*, z^*) is a tight pair. Suppose instead our current guess α is greater than any value in I. Then, because $M^0(\alpha)$ consists of line pieces that become gradually less steep as α grows, the next guess α' has $\alpha' < \alpha^*$. Suppose finally our current guess α is smaller than any value in I. Then, because $M^0(\alpha)$ consists of line pieces that become gradually less steep as α grows, the next guess α' has $\alpha < \alpha' < \alpha^*$.

Putting this together, we know that if we are ever on a line piece on the right side of I, we go over to the left side of I and never come back. If we are on the

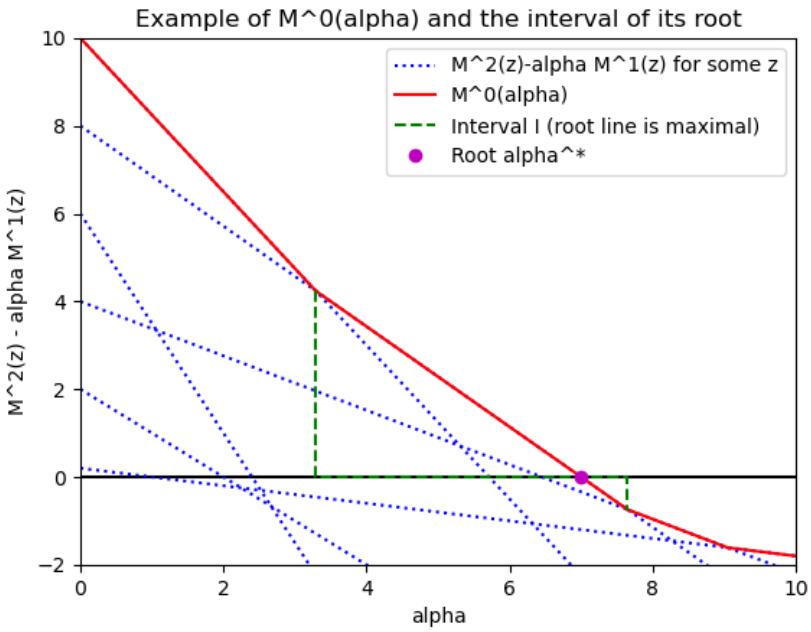


Figure 4.5: Example of how to find the root α^* of $M^0(\alpha)$, given six scenarios. Each scenario $z \in Z$ is depicted as a blue dotted line $\alpha \to M^2(z) - \alpha M^1(z)$ which always starts above 0 and descends. Our first guess of α^* is very high, namely $\alpha_0 = 41.25$. The least descending line is maximal (red) there, and we follow it to its root, ending up with a next guess $\alpha_1 = 1$ which is probably too low. We keep checking which line is maximal at the current α, find its root, thus moving further to the right. After at most $|Z|$ iterations, a guess will be in the interval I, after which the next guess is exactly $\alpha^* = 7$. Remark that two of these six scenarios are never maximal ('red') for $\alpha \geq 1$: they are 'dominated' by other scenarios.

left side of I, we move to a line piece further to the right, thus closer to I. In the worst case, we visit all of the $|Z|$ line pieces once. But we must eventually reach I, and when we do, we find (α^*, z^*). □

Theoretically, it is true that the worst case of $|Z|$ iterations is no better than simply testing all scenarios $z \in Z$ and determining α^* from the outcomes. However, Algorithm 8 only considers scenarios z that are Pareto optimal with respect to low $M^1(z)$ and high $M^2(z)$, and these will typically be much fewer than $|Z|$. Moreover, we will see in Section 4.9 that we tend to need much fewer iterations than $|Z|$ to find the optimal α^*.

4.5.2 LP-rounding algorithms for usurpers

As a further extension of the MIJG framework, suppose that the usurper has a problem that can not be written as a (continuous) Linear Program, but that *can* be written as a Mixed Integer Linear Program with known integrality gap γ. For instance, it is known of the metric k-Median Problem that there always exists a feasible solution with cost at most $\gamma = 6\frac{2}{3}$ times the cost of the Linear Programming relaxation [25]. Then we can use this Linear Programming relaxation as a usurper problem $M^2(z)$. If this yields an approximation factor α^* over the scenario set Z, and if $OPT(z)$ is the optimal solution cost for scenario $z \in Z$, then we know $OPT(z) \leq \gamma M^2(z) \leq \alpha^* \gamma M^1(z) \forall z \in Z$. Therefore, for any problem with known integrality gap γ and any scenario set, we can immediately apply the MIJG framework to compute an upper bound $\alpha^* \gamma$ on the solution quality lost through sparsification.

4.6 Examples of sparsification strategies

Recall that the goal of this chapter is to sparsify a given network minimisation problem by means of an RDGS, and to bound the loss of performance in doing so by an MIJG. In this section, we briefly discuss how sparsification could be applied to various Operations Research problems.

4.6.1 Facility Location Problems

The idea that we can delete or aggregate nodes to save computation time while losing comparatively little on the optimal solution cost, seems sensible for classical Facility Location Problems. In the k-Median Problem, for instance, one can easily imagine that deleting nearby potential facility locations should not influence the optimal solution cost by much, while allowing a substantial fraction of the solution space to be eliminated. That is, a distance threshold should work well as a representation relation between facilities. Our strategy would thus be to compute an RDGS with a threshold representation relation to find some sparsified $G[V']$, then define the simplified problem $\Pi_{G[V']}(z)$ as solving k-Median but not allowing facility locations not present in $G[V']$, and use its solution directly as a feasible solution for the original problem $\Pi(z)$. That is, the function ψ that translates feasible solutions of $\Pi_{G[V']}(z)$ back to feasible solutions of $\Pi(z)$ can simply be the identity function. Note that the original k-Median Problem has no concept of scenarios within it: we could say its scenario set consists of exactly one instance. On the other hand, one could easily imagine uncertain versions of k-Median in which, for instance, it is not yet known which clients will need coverage.

As a next step, in the Uncapacitated Facility Location Problem (UFLP) [153], it makes sense to keep the facility that is cheapest: we could state that facility location u can represent facility location v if the distance is below some threshold

and u has lower opening cost than v. Afterwards, we repeat the same sparsification strategy as for k-Median. Finally, in the Capacitated Facility Location Problem (CFLP) [153], we could additionally ensure that the remaining facilities have enough capacity to admit feasible solutions, by creating a representation relation between facilities and clients as follows: for each facility, we assume it supplies the full demand of the nearest client, then the second nearest, et cetera until the capacity is depleted. Then we say a facility represents a client if and only if the demand of the client is fully supplied this way. This not only ensures that there is enough capacity to admit feasible solutions, but also steers the sparsification such that the demands will likely be satisfied locally.

4.6.2 Classical routing problems

For routing problems, we may need a more sophisticated function ψ to translate sparsified solutions back to feasible solutions for the original problem. For the Travelling Salesman Problem (TSP) [49], we could let one client represent another if their distance is below some threshold, and let a potential depot be represented only by itself. In a sense, this would transform these routing problems into a parallelisable two-stage heuristic. In the first stage, we would obtain an RDGS $G[V']$, meaning we omit many clients who are close together. Our simplified problem $\Pi_{G[V']}(z)$ would then be to solve TSP on $G[V']$. Its solution, $\hat{x}$, would not be feasible for the original problem $\Pi(z)$, because many clients are skipped by it. However, computing $\hat{x}$ would allow us to more quickly make routing decisions on the macro level, trusting that the micro-level details can be accounted for in the second stage. In this second stage, we could base a feasible solution x' on $\hat{x}$ by the following process ψ. First, for each $u \in V'$, we would denote V_u the set of all clients for which the nearest node in $G[V']$ is u. Next, for each (u, v) subsequently visited by $\hat{x}$, we would determine the 'connection points' $(q, r) := \arg\min_{q' \in V_u, r' \in V_v}\{D_G(q, r)\}$. Finally, knowing all connection points between any node cluster V_u and the next, we could solve (path-)TSP [66] on $G[V_u]$ and patch all results together into the feasible solution x'. Though this would require solving $|V'| + 1$ instances of (path)-TSP, these would be much smaller than the original, and the work done by ψ (the 'second stage function') can be divided over $|V'|$ machines.

We can solve the same rough process for the Multiple Travelling Salesman Problem (mTSP) [59] and the Capacitated Vehicle Routing Problem (CVRP) [18]. In the latter, we would only need the additional modification to $\Pi_{G[V']}(z)$ that visiting $u \in V'$ has a capacity requirement equal to the summed capacity requirements of V_u. Technically speaking, trouble may arise if any set V_u requires more capacity than any one vehicle can deliver. Even more technically speaking, one could resolve this by solving the Capacitated Facility Location Problem on these overloaded neighbourhoods V_u, because the resulting $G[V']$ would still be an RDGS if the original graph is complete; however, we consider it out of scope for this chapter to describe this process further.

In all of this, we again remark that the traditional TSP and mTSP and CVRP do not have scenario structures within them, meaning the scenario set would have size 1. But versions with uncertainty exist, such as the universal and a-priori TSP [134].

4.6.3 Orienteering Problems

Another field of routing concerns the Orienteering Problem (OP) [152], which is similar to path-TSP, except we are rewarded for visiting clients instead of being forced, and one seeks to collect the highest reward in a given distance limit. Though we can again find an RDGS using a distance threshold representation relation, it becomes non-trivial to assume that if one node $u \in V'$ is visited, all nearby nodes V_u are visited: namely, visiting all of V_u does not require the easily determined sum of node capacities, but the edge length of a path-TSP solution over V_u, which is NP-hard to determine. Instead, we propose finding an RDGS $G[V']$, reserving only part of the allowed distance for the first 'macro-level' stage by modifying the distance limit in $\Pi_{G[V']}(z)$. Then in the second stage, we would solve $|V'|$ 'micro-level' instances of the Orienteering Problem on each visited cluster V_u, where the distance limit for these micro-instances follows from dividing the remaining allowed distance 'proportionally' over the instances. A good division strategy is certainly non-trivial, and we consider exploring it out of scope for this chapter: we merely wish to sketch how a sparsification strategy for the classical Orienteering Problem could be designed.

Two stochastic variants of the Orienteering Problem include the Orienteering Problem with Stochastic Profits [74] and the Orienteering Problem with Stochastic Edge Costs [144]. In the former, the collectable profits are stochastic and the goal is to maximise the probability that a target amount of profit or better is collected. In the latter, instead the lengths of edges are distributed with independent, non-identical probabilities. Recently, node aggregation has in fact proven successful for this latter problem.

4.6.4 The Two-Stage Stochastic Steiner Tree Problem

A problem in which graphs are typically not complete, is the Steiner Tree Problem, of finding a cheapest tree that spans a given subset of 'terminal' nodes. In this problem, distance-preservation is relevant, because cheapest paths are useful (but not mandatory) for cheap trees. Gupta et al. [62] included a stochastic element in the Steiner Tree Problem by allowing edges to be bought cheaply in a first stage, when knowing only the probability for each node that it will become a terminal, then revealing the terminals and allowing additional edges to be bought in a recourse stage. This Two-Stage Stochastic Steiner Tree Problem (2S-STP) thus appears interesting from a lens of distance-preservation and bounded loss over stochastic scenarios. It may also benefit from sparsification: we can let a potential root node represent itself, and otherwise let nodes represent each other

if their graph distance is below some threshold. One would again obtain a heuristic with a high-level problem and several parallelisable zoomed problems. Each zoomed problem would involve solving another instance of 2S-TSP, taking as root the included vertex that we zoom on, and as remaining nodes the omitted nodes represented by this root. As such, the 2S-TSP could make another interesting candidate to showcase our RDGS and MIJG methodology, though it will not be the problem this chapter focuses on.

4.7 Bounding MRP performance loss as a Mixed Integer Jester Game

In this section, we demonstrate how to use the MIJG framework to compute an upper bound on how much solution quality we can lose due to solving a given optimisation problem on a sparsified network instead of the original network. As a showcase problem, we focus on the Median Routing Problem. One of the reasons for this is that, more so than the problems discussed in Section 4.6, this problem is a good example of a problem where exact distance preservation may be necessary to preserve feasibility, because mandatory targets must be visited within a hard time limit.

Suppose we want to perform universal sparsification for MRP, using any of the algorithms in Section 4.4. That is, we know the graph G and its emergency probabilities, and we know how many agents will be deployed, but there are many things we do not yet know: we do not know where the agents will start, how many non-urgent jobs will be scheduled, what the location and duration of those jobs will be, and we only know an upper bound on the length of the time horizon. This may be the case, for instance, if a daytime emergency logistics organisation decides to perform universal sparsification during the night, so that MRP plans and updated plans can be quickly computed throughout the next day.

Let $\hat{V} \subset V$ be the set of all locations where an agent or job *might* be placed, so each $v \in V$ for which there exists a scenario $z \in Z$ in which an agent a has start location $S_a = v$ or end location $E_a = v$, or in which a job j has location $L_j = v$. Let C_{uv} give the emergency response time from node u to the potential emergency node $v \in V_P$. Let τ be a threshold on response time. That is, we would like every potential emergency node $v \in V_P$ to be represented by a node $u \in V$ within τ minutes of it. For nodes with emergency weight $P_v = 0$, it is all right if they are not explicitly represented: they should only be included if they are useful for covering emergency nodes or for preserving graph distances. Then we can use the following representation matrix $R \in \{0,1\}^{V \times V}$ to perform our sparsification:

- If $v \in \hat{V}$, $R_{uv} = 1$ if and only if $u = v$.
- Otherwise, if $v \in V_P$, $R_{uv} = 1$ if and only if $C_{uv} \leq \tau$.

- Otherwise, $R_{uv} = 1$ for every $u \in V$, meaning v imposes no representation constraints.

While it is possible to compute a sparsified graph this way and evaluate afterwards how much better the solution would have been if the original graph had been used, our MIJG framework allows us to already bound this quality loss beforehand. We describe here an approach that comes with some slack, but which is computationally manageable.

Suppose that, given some optimal MRP solution in the original graph, we know that we can stay 'nearby' in the sparsified graph. Then, intuitively, the optimal coverage in the sparsified network should not be much worse.

We define the *projection radius* ρ of G to G' in $T := \{0, \ldots, \omega\}$ under distance function $D : V^2 \to [0, \infty)$ as follows: it is the smallest number such that, for every path in G of length at most ω hops that starts and ends in G', there exists a path in G' that is at most ρ away from it at every time step, with respect to D. For our MRP case, we will take D to be the Euclidean distance between any two nodes. Fortunately, we can compute ρ in $\mathcal{O}(\omega n^6)$ time, using the Dynamic Program in Algorithm 9. In it, we try increasing values of ρ from among the $\mathcal{O}(n^2)$ options, and compute states $F(u, v, t)$, indicating whether any path to (v, t) can be followed by a path to (u, t) which stays within ρ. If in line 10 we find that there exists a path to (w, t) that cannot be followed in $G[V']$, then apparently, ρ is not large enough yet and we try its next value. If at some point ρ becomes at least $\max_{u \in V, v \in V} D_{uv}$, then we are certainly done: after all, every feasible path is within $\max_{u \in V, v \in V} D_{uv}$ from any other feasible path. Therefore, Algorithm 9 finds the projection radius after $\mathcal{O}(n^2 \cdot \omega n^4)$ time.

ALGORITHM 9: Computing the projection radius ρ of G to G' in $T := \{0, \ldots, \omega\}$.

```
1: Compute D⃗, the different distances in D_uv in ascending order
2: for ρ ∈ D⃗ do
3:    Initialise F(u, v, t) ← 0 (∀u ∈ V', V ∈ V, t ∈ T)
4:    Set F(u, u, 0) ← 1 (∀u ∈ V')
5:    for t ∈ {1, ..., ω} do
6:       for (v, w) ∈ E : (∃u ∈ V' : F(u, v, t − 1) = 1) do
7:          for u ∈ V, u' ∈ V : F(u, v, t − 1) = 1, (u, u') ∈ E', C_u'w ≤ ρ do
8:             F(u', w, t) ← 1
9:          end for
10:         if ∄u' ∈ V' : F(u', w, t) = 1 then
11:            continue ρ
12:         end if
13:      end for
14:   end for
15:   return ρ
16: end for
```

With this projection radius ρ, we can immediately make the following claim:

for every feasible MRP solution in the original graph G, there exists a feasible solution in $G[V']$ in which every agent a always stays within a distance ρ of where they are in the G-based solution. Note that this claim disregards the fact that the agents have to visit jobs, which causes them to converge again in G and G', but that fact can only improve the projection radius.

If we can indeed always stay within ρ from a G-based solution, how much worse can the coverage in G' be at any given time? This is hard to bound algebraically, because D and C may be completely unrelated. However, with the use of the MIJG framework, we can answer this question. That is, we will search for the largest difference in coverage between a configuration $\mu \in V^A$ and a configuration $\mu' \in (V')^A$, knowing that $D(\mu_a, \mu'_a) \leq \rho$ for every agent a.

In this MIJG, both the king and the usurper would like to try to find a k-Median optimum on V and V', respectively. However, the usurper only has access to an LP-solver, not a MILP-solver, so he cannot solve k-Median. Instead, the jester will choose median locations for the usurper, and the usurper only gets to draw the best connections from nodes to medians. If given absolute freedom, the jester would choose terrible medians for the usurper to work with. Instead, we demand that if the king chooses a median location $u \in V$ for agent a, the jester can only choose a median location $v \in V'$ for agent a that has $D(u, v) \leq \rho$, since we know by the definition of ρ that the king's agents can always be tracked through V' up to a distance ρ. To conceptually fit our framework, we will say that the jester first chooses medians for the usurper, then the king gets to choose his medians at most ρ away.

Let $x^0_{au} \in \{0, 1\}$ denote whether or not the jester places agent $a \in A$ at node $u \in V'$. Let $m^0_u \in \{0, 1\}$ denote whether any such agent is placed at u. Then we denote all possible jester inputs as $Z = \{(x^0, m^0) \in \{0, 1\}^{A \times V'} \times \{0, 1\}^{V'} : \sum_{u \in V'} x^0_{au} = 1 \ \forall a \in A, m^0_u = 1 \Leftrightarrow \sum_{a \in A} x^0_{au} \geq 1\}$. Let $x^1_{au} \in \{0, 1\}$ denote whether the king places agent $a \in A$ at node $u \in V$, and let $y^1_{uv} \in \{0, 1\}$ denote whether he provides coverage for $v \in V_P$ from node $u \in V$. Given any $z^0 = (x^0, m^0) \in Z$, we define the king's problem as follows:

$$
\begin{aligned}
&M^1(z^0) = \\
&\quad \min_{x^1,y^1} \sum_{u\in V}\sum_{v\in V_P} P_v C_{uv} y^1_{uv} \\
&\quad s.t. \sum_{u\in V} y^1_{uv} = 1 && (\forall v \in V_P) && (4.6)\\
&\qquad y^1_{uv} - \sum_{a\in A} x^1_{au} \le 0 && (\forall u \in V)(\forall v \in V_P) && (4.7)\\
&\qquad \sum_{u\in V} x^1_{au} = 1 && (\forall a \in A) && (4.8)\\
&\qquad x^1_{av} + \mathbb{1}[D(u,v) > \rho] x^0_{au} \le 1 && (\forall a \in A)(\forall u \in V')(\forall v \in V) && (4.9)\\
&\qquad x^1_{au}, y^1_{uv} \in \{0,1\} && (\forall u \in V)(\forall v \in V_P)
\end{aligned}
$$

The objective function, as well as constraints (4.6)–(4.8), are typical for a k-Median Problem. The only addition is (4.9), which demands that the king places his agents within ρ from their positions chosen by the jester.

The usurper's problem is much simpler, because the jester has already chosen agent positions. Ideally, we would use the same formulation as in $M^1(z^0)$; however, the MIJG demands that we move the jester's variables from the right-hand side into the objective. Therefore, denote $\bar{M} := \max_{uv} P_v C_{uv} + 1$ an arbitrarily large penalty for giving coverage from a node that is not occupied by any agents. Then we define the primal version of the usurper's problem as follows:

$$
\begin{aligned}
&\min_{y^2} \sum_{u\in V'}\sum_{v\in V_P} (P_v C_{uv} + (1 - m^0_u)\bar{M}) y^2_{uv} \\
&s.t. \sum_{u\in V'} y^2_{uv} \ge 1 && (\forall v \in V_P) && (4.10)\\
&\quad y^2_{uv} \ge 0 && (\forall u \in V)(\forall v \in V_P)
\end{aligned}
$$

which essentially does nothing more than setting $y^2_{uv} = 1$ instead of 0 if and only if $u = \arg\min_{u'\in V': m^0_{u'}=1} C_{u'v}$. Remark that this follows from the fact that every column of the constraint matrix consists of zeroes and one entry 1, meaning the matrix is totally unimodular [135], meaning the optimal y^2 will be integer-valued regardless of objective. Each element of the cost vector is non-negative, so there exists an optimal solution where (4.10) is satisfied tightly for each $v \in V_P$, meaning for each $v \in V_P$ we choose exactly one $u \in V$ for which to set $y^2_{uv} = 1$. Choosing a $u' \notin V'$ incurs cost at least $\bar{M}$, so it is always better to choose a $u' \in V'$: by interchange, we always choose $u = \arg\min_{u'\in V': m^0_{u'}=1} C_{u'v}$. Fortunately, therefore, we can conclude that the usurper indeed has a problem that can be solved solely with continuous variables. This problem has the following dual:

$$
\begin{aligned}
&M^2(z^0) = \\
&\quad \max_{\lambda^2} \sum_{v \in V_P} \lambda_v^2 \\
&\quad s.t.\ \lambda_v^2 \leq P_v C_{uv} + (1 - m_u^0)\bar{M} && (\forall u \in V')(\forall v \in V_P) \qquad (4.11)\\
&\qquad \lambda_v^2 \geq 0 && (\forall v \in V_P)
\end{aligned}
$$

Put together, we obtain the MIJG problem

$$
\begin{aligned}
&M^0(\alpha) = \\
&\max_{x^0, m^0, x^1, y^1, \lambda^2} \sum_{v \in V_P} \lambda_v^2 - \alpha \sum_{u \in V} \sum_{v \in V_P} P_v C_{uv} y_{uv}^1 \\
&\quad s.t.\ (4.6) - (4.9), (4.11) \\
&\qquad \sum_{u \in V'} x_{au}^0 = 1 && (\forall a \in A) \qquad (4.12)\\
&\qquad m_u^0 \geq x_{au}^0 && (\forall a \in A)(\forall u \in V') \qquad (4.13)\\
&\qquad x_{au}^0, m_u^0, x_{au}^1, y_{uv}^1 \in \{0,1\}, \lambda_v^2 \geq 0 && (\forall a \in A)(\forall u \in V)(\forall v \in V_P)
\end{aligned}
$$

where the new constraint (4.12) indicates that the jester must choose one location for each agent, and (4.13) that the jester must admit someone is present at node u if she chooses at least one agent a to be at u.

Suppose that we apply Algorithm 8 with $M^0(\alpha)$ and obtain some α^*. Then we know, for each configuration in the original graph G, that there exists a configuration in G' which costs no more than α^* times as much.

In fact, it immediately follows from the definition of ρ that for each feasible MRP solution in the original graph G, there exists a feasible solution in the sparsified graph G' that costs at most α^* times as much. This is because we can stay within ρ throughout the entire solution, so the factor α^* holds at each time step. Note that it is completely irrelevant where in $\hat{V} \subseteq V'$ the job locations and agent start locations are realised: whatever movement this incurs for the optimal MRP solution in G, it can be matched with an α^*-approximate feasible solution in G'.

With this, we have proven that we can bound the performance loss from a universal sparsification by a factor α^*, which we compute through Algorithm 8 and the MIJG Program $M^0(\alpha)$.

4.8 Experiments

In order to test the methods developed in this research, we made use of existing MRP benchmark instances [70]. We ran three series of experiments, outlined below.

First, we applied universal sparsification on the six instances in the MRP class I_R. That is, these six instances act on the same network G with the same number of agents and time steps, but the jobs differ per instance. We took the locations of all jobs and the depot as mandatory locations $\hat{V}$, so that whichever combination of jobs would appear in the realisation, the RDGS would be capable of producing a feasible solution for it. We applied all four RDGS methods on G and $\hat{V}$. Aside from measuring the time of running the RDGS algorithm and the number of nodes that remained after sparsification, we also optimised all six MRP instances on each RDGS to see the median speed-up and quality loss, and we used the method from Section 4.7 to compute a guaranteed bound α^* on this quality loss.

Second, to give more context to these results, we also applied reactive sparsification on the six instances in I_R: that is, we computed all four RDGS networks for each instance separately, and measured the speed-up and quality loss. This could give insight to the trade-off between universal and reactive sparsification. Note that the guaranteed bound α^* computed for universal sparsification also holds for reactive sparsification, because the instances we sparsify for are instances we have prepared for in universal sparsification.

Third, to diversify the measurements on more graphs than just the one belonging to I_R, we performed reactive sparsification on MRP benchmark instance $I_{1,1}$, $I_{2,2}$, et cetera until $I_{8,8}$. We recall from [70] that these eight classes vary in size, density and productivity. It did not make sense to perform universal sparsification on any of these instances, because there existed no benchmarks that shared a graph, except those where the job set or edge set of one instance were a subset of those in the other instance. Likewise, we did not compute guaranteed bounds for these instances, as there are only two benchmark instances on each such network, with one job set being a subset of the other.

All experiments were performed on a ThinkPad L470, running an Intel Core i5-7200U CPU processor with 2.50GHz and 8GB RAM. The MILPs were solved using Gurobi 9.0.2.

4.9 Results

We present the results of our universal sparsification experiments on the MRP instance class I_R in Table 4.1. In Figure 4.1, we can see the result of applying universal sparsification through the `Smallest Realigned` method on this network. In Table 4.2, we present the results of our reactive sparsification experiments on the same class. Finally, in Table 4.3 we see the results of reactive sparsification on a wider sample of MRP benchmark instances. Rather than average observations, we provide median observations to limit the effect of outliers, but these numeric medians should not be confused with median nodes. For completeness, the raw results that these three tables are based on are given in Tables 4.4, 4.5 and 4.6, respectively.

Table 4.1 shows that all four RDGS methods are quite successful in finding

	Creation time (s)	Size ($\|V'\|$)	Median time saved (%)	Median quality loss (%)	Proven bound
Original	0	143	0	0	1.443
`Smallest`	11.4	36	91.2	3.3	2.175
`Smallest Realigned`	12.6	36	89.5	3.3	2.175
`Greedy On-Ramps`	7.7	37	88.3	3.2	2.175
`Greedy Centrality`	0.9	37	88.9	3.2	2.175

Table 4.1: Results of anticipatory sparsification testing for the MRP instance class I_R.

	Median node reduction (%)	Median time saved (%)	Median quality loss (%)	Proven bound
Original	0.0	0.0	0.0	1.443
`Smallest`	93.0	97.1	7.29	2.175
`Smallest Realigned`	93.0	94.8	6.02	2.175
`Greedy On-Ramps`	91.6	92.8	7.54	2.175
`Greedy Centrality`	91.6	93.9	7.54	2.175

Table 4.2: Results of reactive sparsification testing for the MRP instance class I_R.

	Median node reduction (%)	Median time saved (%)	Median quality loss (%)
Original	0	0	0
`Smallest`	63.5	54.0	4.44
`Smallest Realigned`	63.5	36.2	4.19
`Greedy On-Ramps`	59.5	55.3	4.03
`Greedy Centrality`	59.0	56.1	4.06

Table 4.3: Results of reactive sparsification testing for a sample of MRP benchmark instances.

good MRP solutions in much less time. Even the `Smallest` algorithm and the `Smallest Realigned` algorithm run in 12.6 seconds, possibly due to the lean formulation that the on-ramp equivalence theorem allows. The found networks hardly differ in size, possibly because they share a mandatory set $\hat{V} \subset V$ that pushes the solution in a certain direction. All four RDGS methods produce a median quality loss of only around 3.3%, while saving a median 88.3% computation time or better. This supports our initial intuition that, indeed, we can save a lot of time through sparsification while only marginally giving up on solution quality.

Using our MIJG framework, we can prove that for *any* realisation of the MRP data on this network, as long as the agent start and end locations and job

	$I_{R,0}$	$I_{R,1}$	$I_{R,2}$	$I_{R,3}$	$I_{R,4}$	$I_{R,5}$
Original value	268.817	282.985	283.819	310.541	325.609	301.185
Original time (s)	177.428	673.057	3172.097	17691.455	7439.987	34072.066
`Smallest` value	288.117	298.618	294.924	318.303	333.454	309.15
`Smallest` time (s)	81.783	70.434	82.267	945.475	1185.634	1536.373
`Realigned` value	288.117	298.618	294.924	318.303	333.536	309.15
`Realigned` time (s)	144.288	80.294	96.028	727.933	1537.254	2322.702
`On-Ramps` value	287.124	298.025	294.606	317.881	331.215	308.962
`On-Ramps` time (s)	98.212	84.95	110.763	813.864	1061.341	3163.737
`Centrality` value	287.124	298.025	294.606	317.881	331.215	308.962
`Centrality` time (s)	97.656	87.027	109.692	806.712	1052.682	3087.618

Table 4.4: Optimal solution value and computation time of the instances in I_R, computed on the four different anticipatory RDGS networks. The network build time and resulting number of nodes can be found in Table 4.1.

locations are among those found in $I_{R,0}$ through $I_{R,5}$ and the number of agents is the same or more and the length of the time horizon is the same or shorter, the quality loss can never exceed 117.5%, or a factor 2.175. For every RDGS method, this is a tight factor found in merely 3 iterations of Algorithm 8, including the computing of the initial factor. Seeing how the best-known approximation factor

	$I_{R,0}$	$I_{R,1}$	$I_{R,2}$	$I_{R,3}$	$I_{R,4}$	$I_{R,5}$
Original build time (s)	0	0	0	0	0	0
Original nodes	143.0	143.0	143.0	143.0	143.0	143.0
Original value	268.817	282.985	283.819	310.541	325.609	301.185
Original solve time (s)	177.428	673.057	3172.097	17691.455	7439.987	34072.066
`Smallest` build time (s)	11.608	11.546	11.76	12.606	9.227	11.875
`Smallest` nodes	6	9	10	10	16	13
`Smallest` value	325.876	296.471	319.746	332.543	332.692	323.743
`Smallest` solve time (s)	24.116	52.55	45.22	221.103	293.919	313.732
`Realigned` build time (s)	15.626	17.841	17.634	15.852	11.389	12.263
`Realigned` nodes	6	9	10	10	16	13
`Realigned` value	301.722	295.846	310.833	320.893	333.33	323.743
`Realigned` solve time (s)	25.986	50.941	68.361	243.839	563.651	303.307
`On-Ramps` build time (s)	9.281	10.357	9.55	9.867	7.026	7.066
`On-Ramps` nodes	9	11	12	12	18	15
`On-Ramps` value	311.831	308.351	315.408	323.441	329.909	319.588
`On-Ramps` solve time (s)	28.681	63.16	101.626	424.835	1173.704	292.745
`Centrality` build time (s)	0.837	0.953	0.963	0.599	0.646	0.781
`Centrality` nodes	9	11	12	12	18	15
`Centrality` value	311.831	308.351	315.408	323.441	329.909	319.588
`Centrality` solve time (s)	30.098	60.008	101.172	339.176	1217.904	306.564

Table 4.5: RDGS build time, resulting number of nodes, optimal solution value and computation time of I_R instances under reactive sparsification, computed on the four different RDGS networks.

for the simpler k-Median Problem is $1+2/e+\varepsilon$ for metric spaces [28], and seeing how no polynomial-time approximation algorithms exist for MRP unless $P = NP$ [70], we consider an approximation factor of 2.175 to be significant, even if it only

	$I_{1,1}$	$I_{2,2}$	$I_{3,3}$	$I_{4,4}$	$I_{5,5}$	$I_{6,6}$	$I_{7,7}$	$I_{8,8}$
Original build time (s)	0	0	0	0	0	0	0	0
Original nodes	20	20	20	20	100	100	100	100
Original value	2053.7	2654.9	2097.6	1045.8	3457.3	3316.6	3082.1	3460.8
Original solve time (s)	184.809	1.542	2.434	0.574	6369.86	60.273	14597.692	83.612
`Smallest` build time (s)	0.069	0.062	0.046	0.046	3.575	4.201	2.773	2.598
`Smallest` nodes	16	15	13	9	28	23	23	20
`Smallest` value	2053.7	2654.9	2340.7	1224.2	3673.0	3418.6	3261.2	3490.6
`Smallest` solve time (s)	90.987	0.81	0.997	0.44	89.011	31.131	252.592	17.312
`Realigned` build time (s)	0.116	0.122	0.108	0.105	4.808	6.831	3.971	4.028
`Realigned` nodes	16	15	13	9	28	23	23	20
`Realigned` value	2053.7	2654.9	2340.7	1087.3	3662.2	3418.6	3234.2	3613.6
`Realigned` solve time (s)	93.012	0.919	1.109	0.464	137.764	29.527	9829.799	52.998
`On-Ramps` build time (s)	0.028	0.029	0.024	0.024	3.321	2.993	1.846	2.15
`On-Ramps` nodes	16	16	13	9	36	31	25	24
`On-Ramps` value	2053.7	2654.9	2340.7	1227.6	3580.8	3466.0	3273.3	3571.9
`On-Ramps` solve time (s)	91.543	0.81	0.948	0.424	1914.471	62.555	1830.584	23.445
`Centrality` build time (s)	0.017	0.014	0.013	0.01	0.254	0.205	0.352	0.322
`Centrality` nodes	16	16	13	9	37	33	25	24
`Centrality` value	2053.7	2654.9	2340.7	1227.6	3580.8	3467.6	3273.3	3571.9
`Centrality` solve time (s)	91.312	0.755	0.921	0.388	905.363	61.965	1838.241	23.302

Table 4.6: RDGS build time, resulting number of nodes, optimal solution value and computation time of the instances sampled for reactive sparsification, computed on the four different RDGS networks.

holds on this limited class of instances. If there is any doubt about that the MRP data will not realise with these job locations, agent start and end locations, time horizon length or number of agents, we can simply refine this factor for a larger, 'more certain' mandatory subset V', as well as other values of k and ω, using the

same MIJG method.

It is interesting to note that our MIJG framework produces an approximation factor of 1.443 for the unchanged, original network. We would expect a factor 1 here, because if σ^* is an optimal MRP solution, then σ^* is a feasible solution for the unchanged network. It is true that the approach described in Section 4.7 has some inherent slack; namely, it is very pessimistic to assume that the agents will be in their worst allowed configuration at every time step. However, since the projection radius of the graph to itself was confirmed to be 0, we should see that the king and the usurper always have their agents in the same nodes, meaning that this slack does not explain this factor exceeding 1. Instead, we offer two explanations. First, we observed that 1.443 was computed as an initial approximation factor in Algorithm 8, but that computing the next α failed, because solving $M^0(1.443)$ resulted in an out-of-memory error. We thus have to settle for the initial approximation factor, which has a slack of its own. We conclude from this that it may be worthwhile to investigate more scalable alternatives for Algorithm 8. Second, it appears that the network of the MRP instance class I_R has two pairs of nodes for which both nodes have the same coordinates, but different distances to other nodes. Therefore, even if the projection radius is 0, the jester can create some differences by placing the usurper's agents at the 'bad' nodes in the pairs, while the king gets to occupy the 'good' nodes in the pairs.

In Table 4.2, where we applied reactive sparsification on the I_R instances instead of universal sparsification, we see larger variations between the outcomes of the four RDGS methods. This is to be expected, as we now have fewer 'mandatory' nodes steering the solution. We see more time saved than in Table 4.1, but the median quality loss is also worse. This is both likely due to the fact that, as seen in Table 4.5, reactive sparsification for the I_R instances leads to networks with hardly half of the nodes that universal sparsification does. Indeed, the RDGS methods succeed in removing at least 91.6% nodes from the network. Having fewer nodes means there is less to compute, but also more room for error. The time required for each RDGS method to build the sparsified network seems roughly of the same order as in Table 4.1. We remark that we inherit the performance guarantee from universal sparsification, as in theory, the realisations for which we sparsified reactively are among the allowed realisations we are prepared for. In other words: we know that the solution after sparsification is at most α^* times as expensive as the optimal solution over the scenario set Z, so especially after $z \in Z$ is revealed and we apply reactive sparsification, we have a performance guarantee of α^* over the new scenario set $\{z\}$.

Finally, in Table 4.3, we see that reactive sparsification on the instances $I_{1,1}$, $I_{2,2}$, et cetera leads to significantly fewer nodes removed and less time saved than it does on the I_R instances. However, the median quality loss seems smaller and less varied. We remark a large difference between the I_R instances and the instances from the other classes; the I_R instances act on nodes in the real world, where the other instances have nodes in uniformly randomly generated places. We hypothesise that the nodes in I_R are already more 'clustered' in nature,

as they often represent different train stations in the same city, meaning it is possible to remove many more nodes from it while staying R-representative. If fewer nodes can be removed, then it is also not surprising that the saved time and lost quality are both smaller.

A most interesting conclusion would be to see which one of the four RDGS methods works 'best', but at first glance, the results do not appear to point conclusively in one direction. The sparsification time of the `Greedy Centrality` method is surprisingly low, but none of the methods required more than 18 seconds on any instance, not even the `Smallest Realigned` method. Moreover, how much time is saved by using an RDGS method is determined primarily by how much less time is needed to solve the MILP on the sparsified network. Surprisingly, the `Greedy On-Ramps` method appears to find marginally smaller representations than the `Greedy Centrality` method does, but neither is outperformed much by the `Smallest` and `Smallest Realigned` methods.

If we take the ratio of percentual quality loss over percentual speed-up, we would like this ratio to be low. We have computed this ratio for every method and every instance, and have observed the median ratio per method. On basis of this ratio, we can say that `Smallest Realigned` works best for universal sparsification, then `Smallest`, then `Greedy On-Ramps`, then `Greedy Centrality`. The exact same ranking is observed for reactive sparsification as well. This would suggest that `Smallest Realigned` performs best on the benchmarks among these four methods. But all in all, we conclude that all four methods achieve substantial computational speed-up against only marginal loss of solution quality, in both reactive and universal sparsification.

4.10 Conclusions

Though most graph sparsification methods focus on sparsifying enormous graphs for simple queries, we initiated the study of sparsifying moderately sized graphs for NP-hard queries. We introduced the notion of representative, distance-preserving graph sparsifiers and introduced four algorithms by which to obtain ones which are as small yet representative as possible.

Using reactive sparsification on real-life test cases, we have computed MRP solutions in 94.8% percent less time while losing only 6.02% of the solution quality. Using universal sparsification on these same instances, we have computed MRP solutions in 91.2% less time while losing only 3.3% of the solution quality. Using reactive sparsification in a broader range of cases, we have computed MRP solutions in 56.1% less time while losing only 4.06% of the solution quality. The `Smallest Realigned` seems marginally strongest in terms of quality loss over speed-up, but all four methods perform roughly equally well.

Moreover, we have computed an MRP performance guarantee of 2.175 over all possible instances on this sparsified real-world graph. In order to provide this worst-case performance guarantee, we introduced the Mixed Integer Jester Game framework and showed how to apply it for computing this performance

upper bound.

As directions for future research, we propose the following. It could be interesting to perform sparsification and apply the MIJG framework to the other problems listed in Section 4.6, and more deeply investigate which types of problems and data are well suited for sparsification. Furthermore, the `Greedy Centrality` sparsification algorithm is by far fastest in sparsification, yet marginally weakest in the ratio between quality loss and speed-up. If the speed of this method could be hybridised with the effectiveness of other methods, this may yield a stronger RDGS algorithm. In the Mixed Integer Jester Game framework, the usurper is limited to problems that can be written as a Linear Program, rather than a Mixed Integer Linear Program, because we crucially exploit the fact that we can dualise the usurper's problem. The flexibility of this framework will increase if this limitation is overcome. In finding the fixed point of a MIJG, other iterative approaches might exist that scale better than the standard Newton-Raphson method and which give more intermediary improvements. Finally, we believe the MIJG framework may give new computational insights into the field of algorithmic approximation guarantees, and we are excited to see this line of thought further explored.

Chapter 5

The Enriched Median Routing Problem

5.1 Introduction

In this chapter, we will arrive at the final version of MRP that this research project has culminated in. The RDGS algorithms of the previous chapter allow us to heuristically reduce computing times, even for very complicated graph problems. In particular, the `Greedy Centrality` RDGS algorithm will allow us, in this chapter, to generalise MRP to a version that is richly featured enough to be used in practical implementations.

In previous chapters, we saw how the MRP model and proposed solution methods provide new fundamental insight into how to balance the trade-off between urgent and non-urgent tasks properly. However, the application of the results in a real-life pilot setting taught us that the model assumptions were too limiting to be of real use in practice, and that many more limitations that occur in practice should be taken into consideration. For example, in terms of objectives, low emergency response times may be the most important goal, but the distance travelled outside of emergencies is another relevant goal. Also, in terms of constraints, the details of what makes a planning feasible can be numerous: some agents may not be authorised to perform certain tasks; some tasks may have specific time windows; some agents may have to stay close to the base station at all times; some jobs may need several agents to perform; some agents may have scheduled appointments within their shifts, during which they are unable to perform jobs or respond to emergencies. These are just some of many examples of limitations encountered in practice that are not covered by the MRP-model.

Motivated by this, the contribution of this chapter is fourfold. First, on the basis of extensive feedback from our pilot study, we enrich the MRP-model with the inclusion of fifteen extensions to cover the additional objectives and constraints that practical applications may require. Second, we adapt the solution methodology to provide a solution for the E-MRP, demonstrating the flexibil-

ity of the methodology. Third, based on extensive discussions with planners of our partnering railway provider, we propose a Current Practice (CP) model that accurately describes the way in which planners currently schedule tasks (today, they do so without any support from MRP). Fourth, we perform extensive simulation experiments to compare the performance of planning in the CP-model with the performance of E-MRP model in order to assess the gain that can be obtained by using E-MRP. We do so by taking real-life case study data from our partnering railway provider, modelling how they would currently plan with the given data, and comparing that with the improved planning from our heuristic. The results show that the E-MRP solution strongly improves the responsiveness of the current system, even with an increased number of non-urgent tasks. This leads to the conclusion that our E-MRP model provides a powerful means to balance urgent and non-urgent tasks, and is applicable in practice.

The remainder of this chapter is structured as follows. In Section 5.2, we review the related literature, including generalisations in related problems. In Section 5.3, we describe the E-MRP model and solution algorithm. In Section 5.4, we describe the CP-model that describes the way non-urgent task planning is done in current operational practice. In Section 5.5, we elaborate on the simulation environment in which we make comparisons. In Section 5.6, we discuss the numerical results of our comparison. In Section 5.7, we present our conclusions and recommendations.

5.2 Related literature

In the Multi-Period Median Routing Problem, formulated by Kraster et al. [90], the jobs have to be divided over multiple shifts. They proposed doing so with a constructive Median Heuristic. In this heuristic, a compatibility between any two jobs is determined by scheduling one, computing which positions give good coverage when that job is being performed, then seeing if the second job is near one of those reactive positions.

The basic version of the k-Median Problem was discussed in Section 2.2. However, this classical k-Median Problem has been generalised in a number of ways. In the Multi-Capacitated Location Problem by el Amrani et al. [42], we must choose to which 'degree' each facility is opened. Rather than being allowed k medians, the lower degrees take up less of a common facility budget, but they also have less capacity with which to serve clients. They propose a 'Greatest Customer Demand First' heuristic boosted with an initial Branch-and-Cut solution. In the Directional k-Median Problem by Jackson et al. [78], the points lie in a k-dimensional space, and medians can only cover other nodes if they are in the positive direction of the first l dimensions. They propose a polynomial-time algorithm for the 1-dimensional case, which they use as a subroutine in their heuristic for higher dimensions. In the k-Median Problem with Distance Selection by Benati and García [12], the nodes also lie in k-dimensional space, but we select which q dimensions we care about, as well as the k corresponding medians.

This is motivated by clustering on statistical data, where the interesting features are the ones that allow for meaningful clustering. This non-linear problem is linearised in different ways, and the radius formulation performs best. In the Hamiltonian k-Median Problem, we divide the nodes in k directed cycles, rather than k stars pointing at a median. Gouveia et al. [10] solve this problem at competitive speed by combining subtour elimination constraints from the travelling salesman problem with path elimination constraints from location-routing problems and the concept of an 'acting depot'. In the Bi-Criteria k-Median k-Dispersion Problem, we not only minimise the summed distances of nodes to their medians, but we also maximise the smallest distance between any two medians. Colmenar et al. [30] propose a Scatter Search matheuristic to find solutions on or near the Pareto front of these two objectives.

Even more so than K-MED, the basic VRP has been generalised in many ways. VRP-REP lists 50 [154] VRP variants. In the VRP with Time Windows [138], certain clients can only be visited within contiguous time windows. In Orienteering Problems [149], we have bounded time to collect rewards from visited clients, instead of minimising the distance to visit all of them. In the Pollution Routing Problem [11], we may choose to drive more slowly to save fuel, as long as we abide by the time windows. In Distance-Constrained VRPs [93], routes cannot exceed a certain length. In Green VRPs [43], alternative fuel vehicles must visit a specialised refuelling station periodically. In the Carrier-Vehicle Travelling Salesman Problem [53], clients are visited by a vehicle that must stay close to a mobile, but slow, carrier. In Two-Echelon VRPs [33], goods are first brought to satellite stations, and from there to nearby destinations. In Dial-a-Ride Problems [123] and VRPs with Pickup and Delivery [133], goods or persons must be collected from a pickup node and brought to a delivery node. In Consistent VRPs [60], it is important that a client who is visited in multiple time periods is visited as much as possible by the same vehicle and around the same time. In Periodic VRPs [54], some clients must be visited multiple times, but certain combinations of visiting days are not allowed. In Location-Routing Problems [40], we must simultaneously decide where to open depots and how to route over all clients from those depots. Perhaps most importantly, in the Technician Routing and Scheduling Problem [117], the clients must be serviced by a technician with the right skills, tools and spare parts within the right time window. Pillac et al. described a successful metaheuristic for this problem, combining a Regret-based constructive heuristic, an Adaptive Large Neighbourhood Search and post-processing by a Set-Covering-based Binary Program.

While there is other literature that is relevant for the contents of this chapter, such as other publications that touch on both planned jobs and emergency response in one way or another, much of this literature has already been presented in Section 2.2.

5.3 Enriched Median Routing Problem

We will now describe the central problem of this chapter. In Section 5.3.1, we describe the E-MRP and its notation. In Section 5.3.2, we give a formulation of the Mixed Integer Linear Program (MILP) for the E-MRP, and in Section 5.3.3 we propose a fast and scalable heuristic for the E-MRP.

5.3.1 Problem description

The E-MRP is an extended version of MRP, with additional constraints and objectives. As in MRP, we are given a network and a set of agents, tasks (jobs) and discrete time steps. Per time step, agents may hop to an adjacent node or stay where they are. The goal is to decide for each job who will perform it when, and for each agent where they should be throughout the discrete-time horizon. We expand on MRP with the following extensive list of features:

1. The planned travel time is added to the objective function;
2. The makespan is added to the objective function;
3. Penalties for assigning certain jobs to certain agents are added to the objective function;
4. The start and end locations of agents are variable;
5. Agents start and end at heterogeneous times;
6. Some jobs may not be started at certain times;
7. Jobs can require more than one agent;
8. Jobs can require some or all of its agents to have certain qualifications;
9. Some agents are not available for emergency response during a part of their shift;
10. Some agents are not available for processing jobs during a part of their shift;
11. Agents have personal sub-networks they cannot leave;
12. There are mandatory appointments for certain agents to be at a place at a certain time;
13. Jobs may end at a different location than where they start;
14. Aside from preferences, some assignments of jobs to agents are given as hard constraints;
15. Emergency probabilities and response times are time-dependent.

Set	Description
A	The set of agents
J	The set of jobs
V	The set of nodes
T	The set of time steps, $T = \{0, 1, \ldots, \omega\}$
H	The set of shifts in the planning horizon
$A(h)$	The agents in shift $h \in H$, $A(h) \subseteq A$
$T(a)$	The time steps in which agent $a \in A$ is active
$V_P(t)$	At time $t \in T$, the nodes where incidents may occur ($V_P(t) \subseteq V$)
$V(u)$	The nodes within one hop distance of $u \in V$, including u
B	The set of authorizations agents can have
$X^!$	Appointments (a, v, t) that agent a must be at node v at time t

Table 5.1: Notation for the sets in the Enriched Median Routing Problem.

Parameter	Domain	Description
C_{uvt}	$\mathbb{Q}_{\geq 0}$	The response time at time $t \in T$ from $u \in V$ to $v \in V_P(t)$
$C^{\rightarrow}_{uv}$	$\mathbb{Q}_{\geq 0}$	The non-emergency travel time from $u \in V$ to $v \in V(u)$
P_{vt}	$(0, 1]$	Probability that the next emergency is at node $v \in V_P$, time $t \in T$
$Y^{\checkmark}_{at}$	$\{0, 1\}$	Whether agent $a \in A$ is available for emergencies at time $t \in T$
$Z^{\checkmark}_{at}$	$\{0, 1\}$	Whether agent $a \in A$ is available for non-urgent jobs at time $t \in T$
S_{at}	$\{0, 1\}$	Whether agent $a \in A$ can start their shift at node $v \in V$
W_{at}	$\{0, 1\}$	Whether agent $a \in A$ can end their shift at node $v \in V$
V^n_a	$\{0, 1\}$	The current default start and end location of agent $a \in A$
$X^{\checkmark}_{av}$	$\{0, 1\}$	Whether agent $a \in A$ is allowed to visit node $v \in V$
$B^{\checkmark}_{ab}$	$\{0, 1\}$	Whether agent $a \in A$ has authorization $b \in B$
$L^{\triangleright}_j$	V	The start location of job $j \in J$
$L^{\square}_j$	V	The end location of job $j \in J$
R_{jt}	$\{0, 1\}$	Whether job $j \in J$ may be started at time step $t \in T$
Q_j	$\mathbb{Z}_{\geq 0}$	The number of time steps job $j \in J$ takes
$C^{\times}_j$	$\mathbb{Q}_{\geq 0}$	The penalty for not planning job $j \in J$
M_{jb}	$\mathbb{Z}_{\geq 0}$	How many agents with authorization $b \in B$ job $j \in J$ needs
N_{aj}	$\mathbb{Q}_{\geq 0}$	The penalty incurred when assigning job $j \in J$ to agent $a \in A$
$Z^!_{aj}$	$\{0, 1\}$	Whether it is mandatory that agent $a \in A$ is assigned to job $j \in J$
$\phi_{response}$	$\mathbb{Q}_{\geq 0}$	The weight of the response time objective
$\phi_{distance}$	$\mathbb{Q}_{\geq 0}$	The weight of the distance objective
$\phi_{preference}$	$\mathbb{Q}_{\geq 0}$	The weight of the assignment preference objective
$\phi_{makespan}$	$\mathbb{Q}_{\geq 0}$	The weight of the makespan objective
$\phi_{ignoring}$	$\mathbb{Q}_{\geq 0}$	The weight of the job ignoring penalty objective

Table 5.2: Notation for the parameters in the Enriched Median Routing Problem.

Variable	Domain	Description
x_{avt}	$\{0,1\}$	Whether agent $a \in A$ is at $v \in V$ at time $t \in T$
f_{at}	$\mathbb{Q}_{\geq 0}$	Distance travelled by $a \in A$ between time $t \in T$ and $t+1$
y_{uvt}	$\{0,1\}$	Whether a potential emergency at $v \in V_P$, time $t \in T$ will be responded to from $u \in V$
z_{ajt}	$\{0,1\}$	Whether agent $a \in A$ starts job $j \in J$ at time $t \in T$
z'_{aj}	$\{0,1\}$	Whether agent $a \in A$ performs job $j \in J$
z''_{jt}	$\{0,1\}$	Whether job $j \in J$ is started at time $t \in T$
z'''_{j}	$\{0,1\}$	Whether job $j \in J$ is done at all
$\overline{z}$	$[0,\omega]$	The latest completion time among jobs

Table 5.3: Notation for the variables in the Enriched Median Routing Problem.

For the complete definition of E-MRP, we make use of the notation given in Tables 5.1, 5.2 and 5.3. Denote A as the set of agents, J the set of jobs, V the nodes of the network, $V(v)$ the nodes adjacent or equal to $v \in V$, and T the set of discrete time steps. Agents may have heterogeneous working hours, denote $T(a) \subseteq T$ the working hours of agent $a \in A$. There is a set of job authorizations B an agent can have, and we denote $B^{\checkmark}_{ab} = 1$ if agent $a \in A$ has authorization $b \in B$, and 0 otherwise.

We denote $S_{av} = 1$ if node $v \in V$ is an allowed start location for agent $a \in A$, and 0 otherwise. Likewise, W_{av} indicates whether a can end their shift at v. The start time and end time of $a \in A$ are the earliest and latest time steps respectively in $T(a)$. In each time step, each active agent $a \in A$ may stay where they are or move to an adjacent node. However, they have personal sub-networks in which they must stay, and $X^{\checkmark}_{av}$ equals 1 if a is allowed to visit v and 0 otherwise.

Each job must be performed. Once a job $j \in J$ is started, the agents assigned to it must stay at the location of the job for the entire duration Q_j. They must also be available for processing jobs throughout, and we indicate whether or not an agent $a \in A$ is available for job processing at time $t \in T$ by setting $Z^{\checkmark}_{at}$ equal to 1 or 0. Unlike in MRP, the start location $L^{\triangleright}_j \in V$ may be different from the end location $L^{\square}_j \in V$, as some jobs may consist of thoroughly inspecting an 'edge' of the network. For ease of notation, we assume that jobs do not share locations, as we can easily introduce virtual locations. Some jobs have time constraints, meaning j can only be started at time $t \in T$ if R_{jt} equals 1 rather than 0. Note that these 'time windows' are not contiguous per se: it may make sense for certain jobs to be done during the morning peak hour or the afternoon peak hour, but not somewhere in between.

The most impacting difference between MRP and E-MRP, is that a job may require several agents. In fact, $j \in J$ may require that M_{jb} agent with authorization $b \in B$ are assigned to it. We allow an agent with multiple authorizations to count towards the requirement M_{jb} of each of those authorizations: for instance, if a job requires two agents with basic training ($M_{j,basic} = 2$) and one agent with

mechanical training ($M_{j,mechanic} = 1$), the job can be fulfilled by two agents, if they both had basic training and at least one of them had mechanical training.

The decision variables are the following. We must decide for each agent $a \in A$ and each time step $t \in T(a)$ where in the network they are, by setting $x_{avt} = 1$ if a is at node $v \in V$ at time t and 0 otherwise. We must also set z'_{aj} to 1 or 0 to indicate whether agent $a \in A$ is assigned to job $j \in J$, and z''_{jt} to 1 or 0 to indicate whether j is initiated at time $t \in T$. Given the values of these variables, the remaining variables have values that are easy to determine. We set $z_{ajt} = 1$ if $z'_{aj} = z''_{jt} = 1$ and 0 otherwise. If agent $a \in A$ decides to move between time steps $t \in T$ and $t+1$, we denote the travelled distance by $f_{at} \geq 0$. If this movement is from $u \in V$ to $v \in V$, we set f_{at} equal to the travel distance $C^{\rightarrow}_{uv}$. Furthermore, we keep track of the 'makespan' $\overline{z}$: if job $j \in J$ is initiated at time $t \in T$ and needs Q_j time steps to process, then the completion time is $t+Q_j$, and $\overline{z}$ is equal to the latest completion time among the jobs. Finally, at every time step $t \in T$ there are nodes $v \in V_P(t) \subseteq V$ at which an emergency can occur, and we indicate whether it will be responded to from node $u \in V$ by setting y_{uvt} to 1 or 0.

We wish to minimise a sum of four objectives, weighted with normalising factors $\phi_{response}$, $\phi_{distance}$, $\phi_{preference}$ and $\phi_{makespan}$, respectively. First, we want to minimise the expected response time to emergencies in the network. At each time step $t \in T$, we know of each danger-node $v \in V_P(t)$ the probability $P_{v,t}$ of the next emergency occurring there. Once we have decided the positions x_{avt} of the agents throughout time, we know from which nodes $u \in V$ emergency coverage can be given to (v,t). We only count agents who are available for response at that time, as $Y^{\checkmark}_{at} \in \{0,1\}$ indicates whether agent $a \in A$ is available for emergency response at time t. It is always optimal to choose the nearest remaining u for coverage of $v \in V$. So we know the probability P_{vt} for the next emergency to be at (v,t), and we know the corresponding response time C_{uvt}, meaning the expected response time to the next emergency equals $\sum_{t\in T, v\in V_P(t), u\in V} P_{vt} C_{uvt} y_{uvt}$.

Second, we want to minimise the total time agents spend travelling, aside from emergency response. Movement requires fuel, and preventive jobs often cannot be performed while moving. This objective simply equals $\sum_{a\in A, t\in T(a)} f_{at}$.

Third, planners may have their own preferences of assigning specific jobs to specific agents for reasons beyond this model. It may be, for instance, that an agent will soon have an exam for a certain type of job, and should practice that job as much as possible. We therefore define a 'penalty' $N_{ja} \geq 0$ of assigning agent $a \in A$ to job $j \in J$ to discourage unwanted assignments. This final cost component is equal to $\sum_{a\in A, j\in J} N_{ja} z'_{aj}$.

Finally, we want to minimise the makespan $\overline{z}$. If jobs are scheduled near the end of the shift, then any emergency will make it likely that the planned jobs can no longer be performed that day. Additionally, minimising the makespan implicitly means that the workload is divided as 'fairly' over agents as possible.

5.3.2 Mixed Integer Linear Program formulation

We present here a Mixed Integer Linear Program (MILP) formulation for the E-MRP. Technically, it describes a generalisation, as we explain below.

$$\begin{aligned}
\min \quad & \phi_{response} \cdot \left(\textstyle\sum_{t\in T, v\in V_P(t), u\in V} P_{vt} C_{uvt} y_{uvt}\right) + \ \phi_{makespan} \cdot \overline{z} \\
& + \phi_{preference} \cdot \left(\textstyle\sum_{a\in A, j\in J} N_{ja} z'_{aj}\right) + \phi_{distance} \cdot \left(\textstyle\sum_{a\in A, t\in T(a)} f_{at}\right) \\
& + \phi_{ignoring} \cdot \left(\textstyle\sum_{j\in J} C^{\times}_{j} (1 - z'''_{j})\right)
\end{aligned}$$

$$s.t. \sum_{v\in V} S_{av}\ x_{avt^{\triangleright}_{ah}} = 1 \qquad (\forall h \in H)(\forall a \in A(h)) \qquad (5.1)$$

$$\sum_{v\in V} W_{av}\ x_{avt^{\square}_{ah}} = 1 \qquad (\forall h \in H)(\forall a \in A(h)) \qquad (5.2)$$

$$\sum_{v\in V} x_{avt} = 1 \qquad (\forall a \in A)(\forall t \in T(a)) \qquad (5.3)$$

$$x_{avt} \leq \sum_{u\in V_v} (x_{au(t-1)} + \sum_{j\in \vec{J}(u,v)} z_{aj(t-Q_j)}) \qquad \begin{array}{l}(\forall a \in A)(\forall v \in V)\\ (\forall t \in T(a)\backslash\{0\})\end{array} \qquad (5.4)$$

$$\sum_{a\in A} B^{\checkmark}_{ab} z'_{aj} \geq M_{jb} z'''_{j} \qquad (\forall j \in J)(\forall b \in B) \qquad (5.5)$$

$$z_{ajt} \leq z''_{jt} \qquad (\forall a \in A)(\forall j \in J)(\forall t \in T) \qquad (5.6)$$

$$z_{ajt} \leq z'_{aj} \qquad (\forall a \in A)(\forall j \in J)(\forall t \in T) \qquad (5.7)$$

$$z_{ajt} \geq z'_{aj} + z''_{jt} - 1 \qquad (\forall a \in A)(\forall j \in J)(\forall t \in T) \qquad (5.8)$$

$$\sum_{t\in T} z''_{jt} = z'''_{j} \qquad (\forall j \in J) \qquad (5.9)$$

$$\sum_{\tau=t}^{t+Q_j-1} Z^{\checkmark}_{at} x_{aL^{\triangleright}_{j}\tau} \geq Q_j z_{ajt} \qquad (\forall a \in A)(\forall j \in J)(\forall t \in T) \qquad (5.10)$$

$$Z^{\checkmark}_{a(t+Q_j)} x_{aL^{\square}_{j}(t+Q_j)} \geq z_{ajt} \qquad (\forall a \in A)(\forall j \in J)(\forall t \in T) \qquad (5.11)$$

$$\overline{z} \geq (t + Q_j) z''_{jt} \qquad (\forall j \in J)(\forall t \in T) \qquad (5.12)$$

$$\sum_{u\in V} y_{uvt} = 1 \qquad (\forall t \in T)(\forall v \in V_P(t)) \qquad (5.13)$$

$$y_{uvt} \leq \sum_{a\in A: t\in T(a)} Y^{\checkmark}_{at} x_{aut} \qquad \begin{array}{l}(\forall u \in V)(\forall t \in T)\\ (\forall v \in V_P(t))\end{array} \qquad (5.14)$$

$$C^{\rightarrow}_{uv}(x_{aut} + x_{av(t+1)} - 1) \leq f_{at} \qquad \begin{array}{l}(\forall a \in A)(\forall t \in T(a)\backslash\omega)\\ (\forall u \in V)(\forall v \in V(u))\end{array} \qquad (5.15)$$

$$x_{avt}, z_{ajt}, z'_{aj}, z''_{jt}, z'''_{j} \in \{0,1\},\ y_{uvt}, \overline{z}, f_{at} \geq 0$$

As mentioned, this MILP describes a problem more general than E-MRP:

planning is done over several shifts, and allow jobs $j \in J$ to be ignored against a penalty $C_j^{\times}$. We present this more general MILP, because it allows us to describe most of the subroutines in this chapter as a small variation of this MILP.

Let H be the set of shifts. Denote $T(h) \subseteq T$ the time steps in shift $h \in H$. Let $A(h) \subseteq A$ be the set of agents working shift h. For each $a \in A(h)$, denote $t_{ah}^{\triangleright} \in T(h)$ and $t_{ah}^{\square} \in T(h)$ the start and end time of agent a in shift h. Denote $\vec{J}(u,v) := \{j \in J : L^{\triangleright} = u, L^{\square} = v\}$ the jobs that start at u but end at v. Let the binary decision variable z_j''' indicate whether you decide to do the job $j \in J$. With this notation, we obtain the above MILP. We fix $x_{avt} = 1$ for every $(a,v,t) \in X^!$, $x_{avt} = 0$ for every $t \in T$ if $X_{av}^{\checkmark} = 0$, and $z'_{aj} = 1$ if $Z_{aj}^! = 1$.

The objective function of the MILP consists of the four components described in Section 5.3.1, with the added job ignoring penalties $\sum_{j \in J} C_j^{\times}(1 - z_j''')$. Constraints (5.1) and (5.2) indicate that agents must start and end each of their shifts at locations that are allowed for them. Constraint (5.3) states that an agent can be in only one place at a time. Constraint (5.4) states that agents can only move to adjacent nodes. An exception is made when performing jobs that end in different places than they start: if an agent initiates a job $j \in J \in \vec{J}(u,v)$ at time $t \in T$, then we allow that agent to 'teleport' from u to v at time $t + Q_j$. Constraint (5.5) encodes that each job needs a certain number of agents for each authorization. Constraints (5.6), (5.7) and (5.8) set the relation that $(z_{ajt} = 1) \Leftrightarrow (z'_{aj} = z''jt = 1)$. Constraint (5.9) states that if you decide to do a job $j \in J$, it must get exactly one starting time. Constraint (5.10) ensures that agents who start a job stay at its location for the duration, except in its final step, when constraint (5.11) puts them at the job end location. Constraint (5.12) makes $\overline{z}$ behave as the makespan, or the latest completion time among jobs. Constraint (5.13) states that at every time $t \in T$, the danger-nodes $V_P(t)$ need emergency coverage. Constraint (5.14) states, however, that coverage can only be given from nodes with at least one agent on them who is available for emergency response. Finally, constraint (5.15) encodes that if agent $a \in A$ starts moving at time $t \in T$ from node $u \in V$ to node $v \in V(u)$, then the distance f_{at} should equal $C_{uv}^{\rightarrow}$.

We obtain a MILP for E-MRP by observing only one shift, and demanding $z_j''' = 1$ for all $j \in J$. We remark that constraints (5.1) – (5.5), (5.9), (5.13) and (5.14) were already present in some form in MRP, as well as the response time objective. The parameters, variables and indices added to those constraints, together with the new constraints and objectives, constitute the difference between MRP and E-MRP.

5.3.3 An MDSA-inspired heuristic

While it is possible to solve an instance of E-MRP by plugging the MILP into a MILP solver, the required computation times are much too long and unpredictable for this specific application. Whenever an emergency occurs, planners need to have a new planning within minutes, and they cannot afford to wait hours

or days for the optimal one. Therefore, a fast heuristic is needed that provides good solutions within a few minutes. In Chapter 2, `MDSA` was found to be the most effective heuristic for MRP. We recall that, roughly speaking `MDSA` consists of four steps, which are outlined below for later reference.

Step 1: MEDIATE
In this step, the ideal positions for emergency response are determined by solving a k-Median Problem. These medians are matched to agents, such that the distance between the median and the agent's start and end location are minimal in total. Agents are temporarily solely responsible for emergency response in the region belonging to their median.

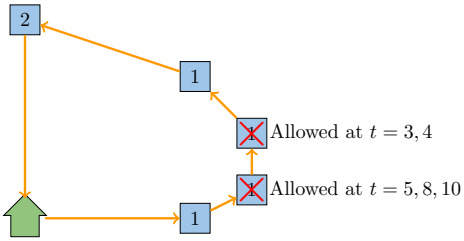
Step 2: DIVIDE
In this step, jobs are assigned as much as possible to the nearest median and its agent.

Step 3: SEQUENCE
In this step, each agent decides in what order to visit the jobs assigned to them. Discrete routes over the network are then drawn with optimal response times to the region of that agent.

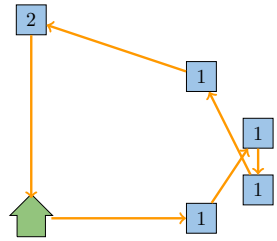
Step 4: AGREE
In this step, it is evaluated if the current routes justify the initial division of emergency nodes over agents, and a small re-optimisation is performed.

Adapting `MDSA` to E-MRP is non-trivial, due to the many new features. To be more specific, the two main hurdles to be taken are the following: (1) it is no longer given that any agent can do any task at any time, and (2) because some tasks may require multiple agents, agents are no longer free to independently sequence their jobs.

An example of the first hurdle is given in Figure 5.1. If we simply compute the fastest route between all jobs assigned to an agent, we can no longer guarantee that this is feasible. For instance, it may be that some jobs j are quite restrictive in their allowed starting times R_{jt}. It no longer suffices to encode with decision variables if one job is followed by another; we must also decide the starting time t and check whether $R_{jt} = 1$. Similarly, appointments $X^!$ must also be fit into the schedule, giving us another reason that we must explicitly track arrival times. These are only two of the many additional features to account for in dividing and sequencing the jobs. The second hurdle, illustrated in Figure 5.2, causes an even deeper issue in the methodology. In `MDSA`, we first divide jobs over agents, then allow them to independently decide in what order to visit these jobs. However, because we now have jobs that require multiple agents, we need to ensure that agents arrive at such jobs at the same time. This makes it very difficult to decouple the routing decisions per agent, especially in conjunction



(a) The route `MDSA` suggests is infeasible; in this case, because some jobs cannot be initiated at all times ($R_{jt} = 0$).



(b) A feasible alternative.

Figure 5.1: The first issue with `MDSA`: the most efficient route may be infeasible, due to the many side constraints of E-MRP.

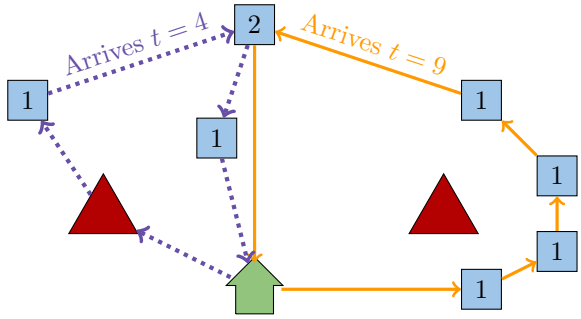


Figure 5.2: The second issue with `MDSA`: if agents draw their own routes, they may arrive at cooperative jobs at completely different times. Due to time limits, we cannot always wait until everyone is present. If $\omega = 13$, then these two routes are feasible in isolation, but infeasible when evaluated jointly.

with the routing challenges in the first hurdle. In conclusion, the arrival of each agent at each key location must be synchronised much more closely, and `MDSA` *as is* falls down. A new algorithm is needed.

Fortunately, by understanding what made `MDSA` successful, we can craft a heuristic in the same spirit. To be more specific, for `MDSA` the quality of the solutions was concluded to be mainly due to the MEDIATE-step (outlined above). Because it is NP-complete to produce a feasible solution for MRP, MILP solvers were required as subroutines, but they terminated very quickly due to the problem being split and heavily compressed.

For E-MRP, we propose the heuristic described in Algorithm 10. We again start with determining good medians. Because finding a feasible solution is NP-complete, we need a MILP solver as a subroutine, but we can greatly speed this

up by compressing the underlying network. We again split the decisions over several steps: the MILP mainly finds a feasible job schedule with reasonable distance to the medians, but we only explicitly optimise emergency response times and travelled distances in later steps.

ALGORITHM 10: High-level overview of the `MDSA`-inspired algorithm for E-MRP.

1: Obtain $|A|$ medians optimal with respect to $\sum_{t \in T} P_{vt} C_{uvt}$
2: Obtain an RDGS $V' \subseteq V$ containing all medians and the start and end locations of agents and jobs
3: Run the MILP, but with nodes V', and $P_{vt} > 0$ only for the medians, and $|H| = 1$, and $z''' = 1$, and $\phi_{distance} = 0$, and an appropriate time-out.
4: Fixing z from the previous step, as well as $X^!$, determine where in V any agent $a \in A$ can be at any time $t \in T$
5: Sort A on how many start locations they can still choose from, then greedily choose start locations minimising response time and travel time to the first goal
6: For $t \in T$, for each active agent, greedily decide their next hop based on response time and travel time

Below we elaborate on the steps in Algorithm 10.

1. Node weights and response times are now time-varying, but for simplicity, we sum these over time. This is equivalent to taking average weights and response times. This allows us to solve one k-Median problem for the entire time horizon, which is much less computationally intensive than solving one for each time step. The underlying assumption is that, though some time steps will have more emergencies and higher travel times, each node will retain more or less the same fraction of the emergency weight, and the increase in response times behaves like a constant multiplication. This is justifiable when emergency probabilities and response times increase mainly due to the morning and afternoon rush hours, and these multiplication factors are uniform over the network. Note that if P_{vt} and C_{uvt} are static enough, then in practice, we can maintain a preprocessed list of medians for the typical numbers of agents in a shift.

2. It is computationally costly to input many nodes into the MILP. Therefore, we find the smallest subgraph that preserves the graph distances between the points of interest, including the medians, using an RDGS algorithm from Chapter 4. If the start and end locations of the agents and jobs are static enough, this step can also be preprocessed. Otherwise, we propose using the `Greedy Centrality` due to its low running time.

3. We run the MILP from Section 5.3.2 for a limited time. Because we use it for an instance of E-MRP, we enforce that every job must be scheduled, and we only observe one shift. We knowingly delay optimising the travelled distance to a later step, trusting that a good makespan will imply an agent's

jobs being close together. This also allows us to skip constraint (15), which is by far the most costly to build.

4. In later steps, we will greedily choose cost-improving hops. However, we need to make sure that we do not make choices that turn out to be infeasible. Therefore, we perform this step to compute which nodes can be visited at which times when abiding by the job schedule z from the previous step and $X^!$. For each agent, we determine from z and $X^!$ their fixed locations at those times. Then, because this set of fixed locations is small and the network adjacency is constant over time, we can make the following computation. For each location of interest for an agent $a \in A$, we use breadth-first search to determine how many hops away it is from each node $v \in V$, remembering that $a \in A$ can only visit nodes $u \in V$ with $X^{\checkmark}_{au} = 1$. Then if a has a fixed location (u_1, t_1) and next has to be at (u_2, t_2), we know for each $v \in V$ that $a \in A$ can be there at times $t_1 + \mathit{hops}(u_1, v) \leq \tau \leq t_2 - \mathit{hops}(v, u_2)$, which may be an empty set of τ. The result of this step is, for each agent $a \in A$ and each time step $t \in T$, a list of nodes $V_z(a, t) \subseteq V$ which that agent can visit at that time.

5. While agents can have multiple start locations to choose from, this selection may be further limited by fixing the job schedule z. We sort A in increasing order of how many start locations they can still choose according to V_z. We initiate the set U of chosen start locations empty. Then, for each agent $a \in A$ in the sorted A, and for each allowed start location u, we determine the summed response cost for $U \cup \{u\}$ as $(C)'_u = \sum_{t \in T, v \in V_P(t)} P_{vt} \min_{u' \in U \cup \{u\}} C_{u'vt}$. If a visits any jobs or appointments, and l is the starting location of the first such visit, we set $(C^{\rightarrow})'_u = C^{\rightarrow}_{ul}$; otherwise, $(C^{\rightarrow})'_u = 0$. We then pick the start location u with minimal $\phi_{response}(C)'_u + \phi_{distance}(C^{\rightarrow})'_u$, set the location of a at their start time to u, add u to U, and move on to the next agent.

6. For each time step $t \in T \backslash \{\omega\}$, and each agent active at that time step and the next, we obtain their current location v and look at all nodes in $V_z(a, t+1)$. Let U be the (possibly empty) set of locations for which it is decided that some response available agent will be there at $t+1$. Then for each $u \in V_z(a, t+1)$, we again determine $(C)'_u = \sum_{t \in T, v \in V_P(t)} P_{vt} \min_{u' \in U \cup \{u\}} C_{u'vt}$, and we set $(C^{\rightarrow})'_u = C^{\rightarrow}_{vu}$. We let the location of a at time $t+1$ be $u := \arg\min_{u \in V_z(a,t+1)} (\phi_{response}(C)'_u + \phi_{distance}(C^{\rightarrow})'_u)$.

In summary, Algorithm 10 yields a schedule for the jobs, and a movement instruction for the agents. By construction, this algorithm will find a feasible solution if it exists, unless the time-out on step 3 is too narrow.

5.4 Current Practice model

To assess the performance improvement of our E-MRP solution with the performance obtained with the current way of planning, it is important to understand how the planning is done in current practice. By making an accurate quantitative model of the current practice, we can computationally investigate the influence that certain decisions have on different metrics. This allows us to investigate a large number of potential scenarios. Furthermore, a model allows us to compare the performance in these scenarios, without having to track how well our proposed solution is abided by in practice. To this end, we have set up extensive interviews with task planners from our partnering railway provider. This has led to a common understanding of the informal steps taken in the current way of planning, which is mainly done manually (without any support from MRP). Throughout, this model will be referred to as the Current Practice (CP) model, and is described below. In the next section, this CP-model will be used as a benchmark to compare the performance of the E-MRP model proposed in this chapter for a realistic use case scenario obtained from our partnering railway provider.

We have observed ten shifts in a single week. A shift consists of several agents, and a shift leader responsible for their coordination. Some agents, including the shift leader, have a role in a shift that requires them to stay at the base of operations. Almost every shift has a designated truck driver, who cannot stray too far from the garage of the heavy emergency-response truck, which is also at the base station.

In the current way of working, agents in our case study make their planning in a rather decentralised manner. Some of the preventive jobs require some paperwork and planning, or collaboration, and these are typically planned and assigned by the shift leader. After that, however, the shift leader asks the agents to propose for themselves which jobs they will do, based on their own expertise and preferences. Agents have often accumulated local knowledge in the area around their home, and prefer to stay in that area. They will typically divide the tasks close to their home over their shifts that week. Once they have performed a sufficient amount of work, they are allowed to await emergencies from their home base. See also Figure 5.3.

As long as this means someone is active in the 'northern part of the region', as well as in the 'middle part' and 'southern part', the expected emergency response time is low enough for the shift leader to be content. Working hours have been designed around that logic, and typically, each shift contains an agent living in the north, the middle and the south. For the ten shifts in our case study, the active agents indeed have home bases in these three areas, and the decentral plans of the agents were not interfered with.

In this way of working, agents may know roughly in what corners of the networks their co-workers are, but they do not know where exactly or what they are doing. Some agents in the late shift may choose to perform a periodic patrol

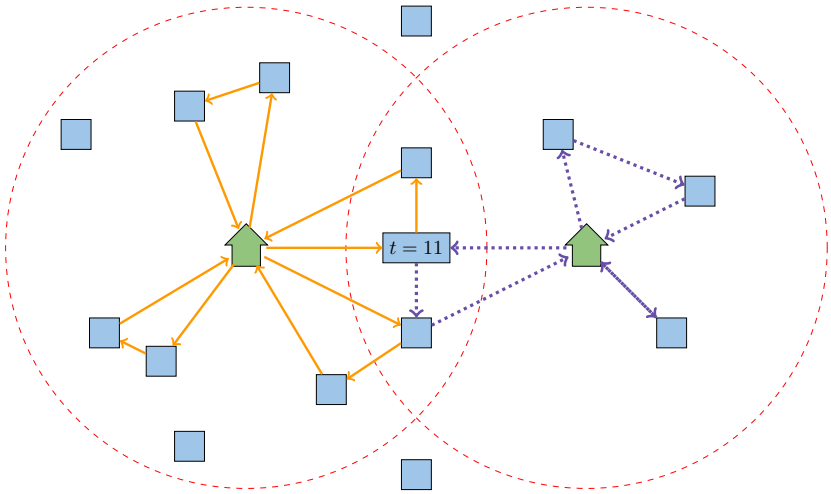


Figure 5.3: A schematic representation of the current practice. Two agents, living at the green house-shaped nodes, determine which jobs are within a given radius of their home base. They disregard jobs outside of that radius. One job in their intersection has to be done by at least two people: a planner decides these two should do it, at time $t = 11$. Apart from that, agents are free to divide jobs over their shifts. The left agent divides eight jobs over his four shifts (solid orange lines), the right agent divides five jobs over his three shifts (purple dotted lines). They are unaware that the same job (just below the cooperative job) is visited by both of them at some point, which is superfluous. The left agent could have instead visited a job that now no one does.

on a piece of the railway, not knowing that a co-worker has already patrolled there the same morning. Moreover, agents in the *same* shift may encounter each other patrolling the same piece of the railway, and continue their patrol together. Two agents staying together is clearly suboptimal with respect to emergency response times. Meanwhile, if some other part of the railway has been neglected for weeks on end, this is untracked and unknown.

The current practice has some practical benefits: it requires very little communication, which makes it robust in the face of emergencies. Furthermore, experience with a given piece of the railway is undoubtedly beneficial to inspecting it effectively. However, now that the communication infrastructure and the size of the organisation are growing, we believe our model and heuristic can help coordinate schedules and decrease response times. We will support this claim in the remainder of this chapter.

We obtain all sets and parameters from the case study data, with one exception. For each agent $a \in A$, from among the variable starting and ending locations, only V_a^n is allowed, the one nearest to their home. If an agent works from the base station or as a truck driver, the base station is taken as the start location. Because the agents must select a subset of jobs to perform, we provide a job 'importance' $C_j^{\times}$ that was also approved by the employee of the organisation. We simulate the current practice as follows, with H the ten shifts in the observed week.

Let $\texttt{MILP}(A', J', V', H', F)$ denote that we call the MILP from Section 5.3.2 with agent subset A', job subset J', node subset V', shift subset H', and a collection F of fixed values for certain variables and parameters. Then Algorithm 11 describes the way plans are made in the CP-model.

In words, Algorithm 11 does the following.

- We compile the jobs $J' \subseteq J$ that the shift leader decides on. These are the jobs that require multiple agents, and the jobs that cannot be done by agents without the shift leader providing some paperwork. We simulate these jobs being planned over the week, by solving a variation of the MILP from Section 5.3.2. That is, we fix z'_{aj} to 0 if job $j \in J$ is not within one hour of the home of agent $a \in A$, measured in $C^{\rightarrow}$. We set the start and end locations of the agents as discussed by fixing $x_{avt} = 1$ if $v \in V$ is the default start and end location of a for that shift, and $t \in T$ is a shift start or end time for a that shift. If a is a driver or has to stay at the base station, we forbid the other locations by setting $x_{avt} = 0$. We demand all jobs J' are assigned, so we set $z'''_j = 1$ for all $j \in J'$, as J' is quite small in this case study. We set $\phi_{response} = \phi_{ignoring} = 0$. As we care mainly about setting start times for those few jobs in J', it suffices to use a simplified node set V'. J' is small enough that we believe a high-quality schedule can be found even when setting a time-out on the MILP. The result is a decision of whom the jobs in J' are assigned to, and in the case of multi-agent jobs, at what time they will be performed.

ALGORITHM 11: Mathematical interpretation of the current practice.

1: Obtain an RDGS, either precomputed (universal) or using Algorithm 7.
2: Encode in collection F_2, for all shifts $h \in H$ and agents $a \in A(h)$ and time steps $t \in T(h)$, that $x_{a,V_a^n,t_{ah}^{\triangleright}} = 1$, $x_{a,V_a^n,t_{ah}^{\square}} = 1$ and that $x_{avt} = 0$ if node v is not accessible for agent a because of their role in the shift h. Also encode for each agent $a \in A$ and each job $j \in J$ that $z'_{aj} = 0$ if j is too far away, that is, $C^{\rightarrow}_{V_a^n,L_j^{\triangleright}} > 60$ or $C^{\rightarrow}_{V_a^n,L_j^{\square}} > 60$.
3: Compile centralised jobs J': jobs that are only done once, and jobs j that must be assigned at least two agents, meaning $\nexists a \in A : (B^{\checkmark}_{ab} \geq M_{jb}\ \forall b \in B)$.
4: The planner plans J'. That is, encode in a new collection F that $z''' = 1$, and $\phi_{response} = 0$ and $\phi_{ignoring} = 0$. Then call $\texttt{MILP}(\{a\}, J', V', H, F \cup F_2)$. Encode in collection F_4 the following: $z'_{aj} = 1$ if $j \in J'$ is assigned to $a \in A$ and 0 otherwise, and $z''_{jt} = 1$ if multi-agent job $j \in J'$ is scheduled at time step $t \in T$.
5: Each agent $a \in A$ divides jobs over their shifts. That is, encode in a new collection F that $\phi_{response} = \phi_{preference} = 0$. Then call $\texttt{MILP}(\{a\}, J(a), V', H, F \cup F_2 \cup F_4)$. For each shift $h \in H$ and each agent $a \in A(h)$, denote $J(a,h)$ the jobs agent $a \in A$ schedules in shift $h \in H$.
6: Agents minimise their makespan in each shift. That is, for each $h \in H$ and $a \in A(h)$, encode in a new collection F that $z'''_j = 1$ for all $j \in J(a,h)$, and $\phi_{response} = \phi_{preference} = \phi_{distance} = \phi_{ignoring} = 0$. Then call $\texttt{MILP}(\{a\}, J(a,h), V', \{h\}, F \cup F_2 \cup F_4)$. Encode in collection F_6 that $z''_{jt} = 1$ if start-time t was chosen for job j.
7: Agents minimise their distance, abiding by this makespan. That is, for each $h \in H$ and $a \in A(h)$, compute the $C^{\rightarrow}$-shortest tour from V_a^n across $J(a,h)$, while not violating $F_2 \cup F_4 \cup F_6$. If this tour is shorter than $T(h)$, let the agents spend the remainder of their shift at V_a^n.

- Each agent $a \in A$ assembles the jobs $J(a) \subseteq J$ within one hour of their start location, and divides these over the week using the MILP. Some values of x_{avt} and z'_{aj} are fixed by the same rules as in the previous step. In addition, the previous step assigned the jobs J' to specific people, and we further fix z'_{aj} accordingly. If those jobs were in J' because they required several people, we also enforce $z''_{jt} = 1$ for the chosen start time t of job $j \in J'$, to ensure that everyone shows up at the same time.

  However, after having those variables fixed, we allow a to decide which jobs to do when. We use the agent set $\{a\}$, the simplified nodes $V' \subseteq V$, and the nearby jobs $J(a) \subseteq J$. We do not constrain z'''_j, and we set $\phi_{response} = \phi_{preference} = 0$. We run the MILP with a time-out. The result is a decision, for each agent, what jobs they will do in each of their shifts.

- Each agent $a \in A$ then determines, for each of their shifts $h \in H$, how to do the jobs $J(a, h)$ they chose for that shift as quickly as possible. We first minimise the makespan. This is done with the MILP, with agents $\{a\}$, jobs $J(a, h)$, simplified nodes V', and shifts $\{h\}$. We fix $z'''_j = 1$, and set $\phi_{response}$ and $\phi_{distance}$ and $\phi_{preference}$ and $\phi_{ignoring}$ to 0. After running this MILP to optimality, we use an adaptation of Dijkstra's algorithm to find the quickest route with respect to $C^{\rightarrow}$. This route starts at the start location, visits the jobs at the times scheduled by the MILP, and goes back to the start location as quickly as possible. The result is a complete solution: for each shift, we know where the agents are at each time step and what jobs they start performing, albeit that some jobs may be performed multiple times.

We emphasise that the CP-model described above describes the informal steps taken by planners in our partnering railways company in today's practice. Simulation experiments show that the response times predicted by the CP-model closely match those observed actually observed in current practice. As we will see in Section 5.6, the CP-model predicts an expected response time of 41.1 minutes. In a recent sample from the partnering company, an average response time of 43.9 minutes was measured over 45 incidents, with a standard deviation of 17.31 minutes. If our hypothesis is that these incidents were sampled from a distribution with mean 41.1, then the measured 43.9 corresponds to a p-value of 0.285, which implies that the hypothesis can not be rejected.

From this, we conclude that the CP-model gives a good description of the current way of working. In the remainder of this chapter, we will use this CP-model to answer what-if questions about the performance under different, both realistic and hypothetical, planning situations.

5.5 Performance metrics, planning scenarios and use case description

In this section, we benchmark the performance of the proposed solution to the E-MRP model with the performance of the CP-model. For this, we have performed extensive simulations for a range of realistic planning scenarios. In Section 5.5.1, we will discuss the performance metrics we wish to improve on. In Section 5.5.2, we will describe the planning scenarios we have simulated to investigate the potential improvements. In Section 5.5.3, we describe the real-life use case that we used for our experiments. The remaining details of our simulation experiment are given in Section 5.5.4.

5.5.1 Performance metrics

In our experiments, we consider the following seven relevant performance metrics:

- **Expected response time:** given the positions x_{avt} of a solution, we know the positions $U(t) := \{u \in V : \sum_{a \in A} Y^{\checkmark}_{at} x_{aut}\}$ from which emergency response can be given. The expected response time then equals $\sum_{t \in T, v \in V_P(t)} \min_{u \in U(t)} P_{vt} C_{uvt}$.
- **Percentage of incidents with response time under χ minutes:** Again given $U(t)$, we define this metric as

$$\sum_{t \in T, v \in V_P(t)} P_{vt} \cdot 1(\exists u \in U(t) : C_{uvt} \leq \chi)$$

- **Weighted unique jobs:** While we have defined the value $C^{\times}_j$ of doing job $j \in J$, this does not account for the fact that a set of decentralised agents may unknowingly do a job multiple times in the same week, which is considered 'useless'. We thus define this metric as $\sum_{j \in J} C^{\times}_j \cdot 1(j \textit{ is scheduled at least once})$.
- **Planned travelling:** Given the positions x_{avt}, this is simply the sum of $C^{\rightarrow}_{uv}$ for every movement from $u \in V$ at time $t \in T \backslash \{\omega\}$ to node $v \in V$ at time $t + 1$.

We do not take computation time into account as a metric, because computation time is only meaningful in a small subset of the scenarios we compare. In our experiments, however, our Algorithm 10-based solver needs only 19.8 seconds per shift, even on modest hardware and without a time-out on Step 3. Computation times, thus, are not a restricting issue in our application.

5.5.2 Planning scenarios

In practice, there are many choices to be made with respect to the restrictions in the planning. To gain insight into the implications of these options, we define a

number of planning scenarios, for which the performance for our use case study is evaluated.

1. **Current practice ("own jobs, own route"):** agents pick their own jobs, schedules and routes, as described in Section 5.4.
2. **Jobless current practice ("no jobs, own route"):** in the later parts of shifts, agents are often already at home in the current practice. To see what kind of emergency response this incurs, we ran the same analysis as in Section 5.4, but with an empty set of jobs. The resulting solutions amount to agents starting at their start location, and staying there until the end of their shift.
3. **New practice ("planned jobs, solver route"):** as part of the case study, jobs were picked by an experienced planner from our industry partner, who had access to the solver running the heuristic from this chapter. The planner picked jobs for all shifts in the case study, and did so without interference from the researchers. These jobs were picked and divided over the shifts so that jobs are not done multiple times, but agents still have jobs somewhat near one of their start locations. In this planning scenario, the job schedules and routes are determined by running the heuristic in Section 5.3.3.
4. **Current jobs with heuristic ("own jobs, solver route"):** when comparing the current practice with the new practice, the quality of the new solutions depends in part on how well the planner chose the jobs. To filter out this effect, in this scenario, we ran the solver with the same set of jobs as in the current practice. The aim of this scenario is to show that, even without changing which jobs are done, the solver can still find job schedules and positions which perform better. We compute this scenario by deciding jobs for shifts as in Section 5.4, but then running the solver for each shift with the jobs thus chosen.
5. **Lower bound response time ("best response"):** to give more context to the results, we computed a lower bound on the expected response time that can be achieved in this case study setting. We did so by running the MILP from Section 5.3.2 for each shift, but with $J = \emptyset$, and $\phi_{distance} = 0$.
6. **Upper bound weighted unique jobs ("best jobs"):** likewise, to upper bound the weighted unique jobs metric, we ran the MILP with $\phi_{response} = \phi_{distance} = \phi_{preference} = \phi_{makespan} = 0$.
7. **Adjusted jobs with solver ("manual jobs, solver route"):** as it turned out, the planner selected significantly fewer jobs for the case study than the simulated agents could handle. This was because at the time, this new technology and way of working were unfamiliar, and not all available agents were taken into account. To correct for this, we manually picked a larger set of jobs to input into the heuristic.

5.5.3 Use case description

In this section, we describe the realistic use case, obtained from our industry partners, which we have used for our simulation experiments. In this case study, we observe a portion of the railway network. We have summarised this as a graph with 294 nodes, which we can simplify to a graph of 126 nodes using the node aggregation technique described in [71]. The nodes represent either railway stations, the 'midway points' of railway sections between the stations, cargo stations or a depot office. Figure 5.4 gives a visualisation of the studied network section in the North-Eastern part of the Netherlands. A fleet of agents

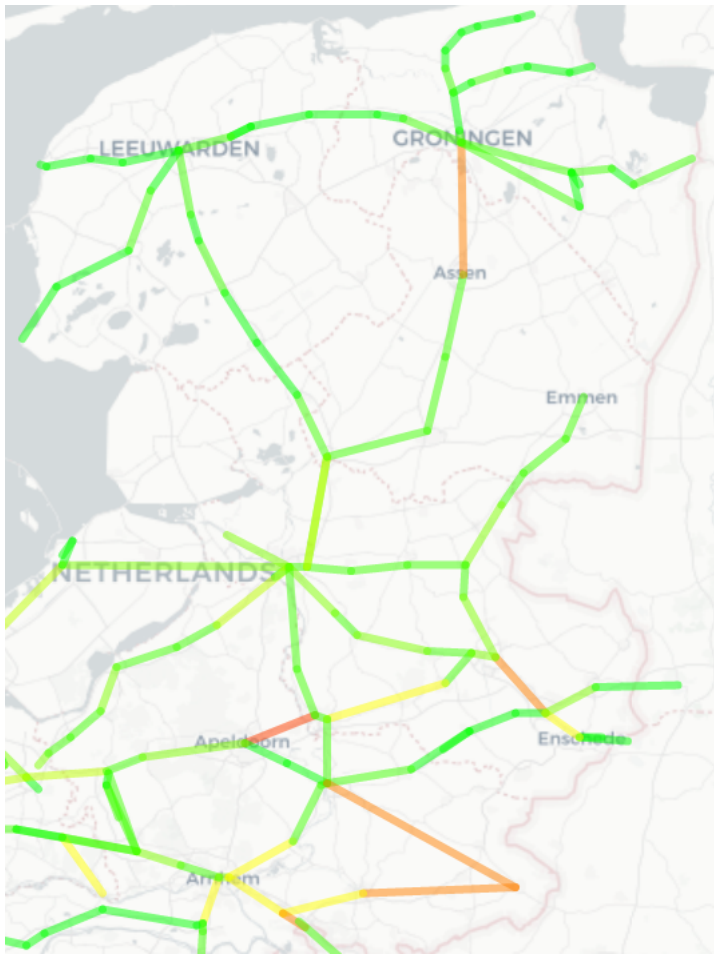


Figure 5.4: The studied section of railway network in the North-Eastern part of the Netherlands. Green edges have low incident rates, while orange and red ones have high incident rates (based on historical incident data).

live in or near this network, and for privacy reasons, their default start and end locations are rounded to the nearest railway node. When using variable start locations, agents have on average 4.11 different locations to choose from. We have a collection of 45 regular tasks on this network: most of them consist of patrolling from one major station to another, but some are facility inspections that should happen exactly once per week. The jobs have different lengths, priorities and locations, and need either one or two agents. The facility inspections require a special training, and may only be performed during daylight hours.

We examine a working week of ten shifts: a morning shift and a late shift, from Monday to Friday. After discretisation, the number of time steps in a shift is either 14 or 12. The number of agents active in a shift lies somewhere between 4 and 10, averaging exactly at 7. Among these, there is always a 'manager' who stays at the depot and is only available for emergency response, and there is often a 'truck driver' who must stay within half an hour of the depot but can perform nearby activities. By design, each shift has at least one agent with a default location in the 'northern' part of the region, and an agent in the 'southern' part.

5.5.4 Implementation details

The case study organisation has access to a solver running this heuristic on a dedicated server. However, for this chapter, all experiments were conducted on a single ThinkPad L470, with an Intel Core i5-7200U CPU processor, 2.50GHz, 8GB RAM.

5.6 Results

In this section, we present the results of our simulation experiments for our case study, and for the scenarios and KPIs listed above. Table 5.4 and Figures 5.5 and 5.6 give an overview of the results. Table 5.5 gives the relative improvements of the KPIs compared to the CP-model.

Overall, the results show significant improvements compared to current practice. To elaborate, if we compare the CP ("Own jobs, Own route") with the E-MRP solution ("Planned jobs, Solver route"), our simulations indicate an improvement of 12.7% in average emergency response times. This number is significant, since taking averages over many response times usually tends to flatten out performance gains. Interestingly, looking at a 60-minute response-time target, we observe that the fraction of late arrivals is reduced by as much as 41.1%. We even see a small reduction in the planned travelling time; however, this is likely due to the planned jobs being significantly fewer than the jobs agent picked for themselves.

If we do not change the jobs of a shift, but only use a solver to redivide the jobs and to determine positions of the agents, we see an even stronger reduction in response time. However, this comes at the cost of 52.4% increased planned travelling outside of jobs and emergencies. Movement is costly and irksome, and

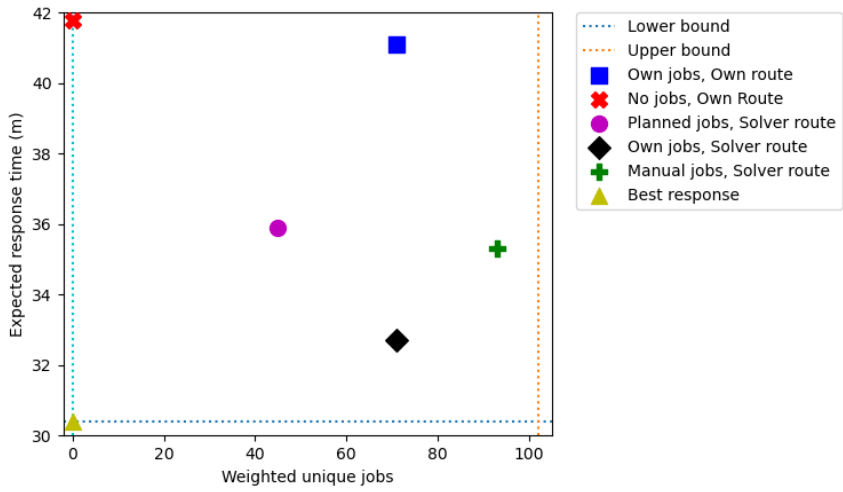


Figure 5.5: Weighted unique jobs versus expected response time in the computed scenarios. On the horizontal axis, high is good; on the vertical axis, low is good.

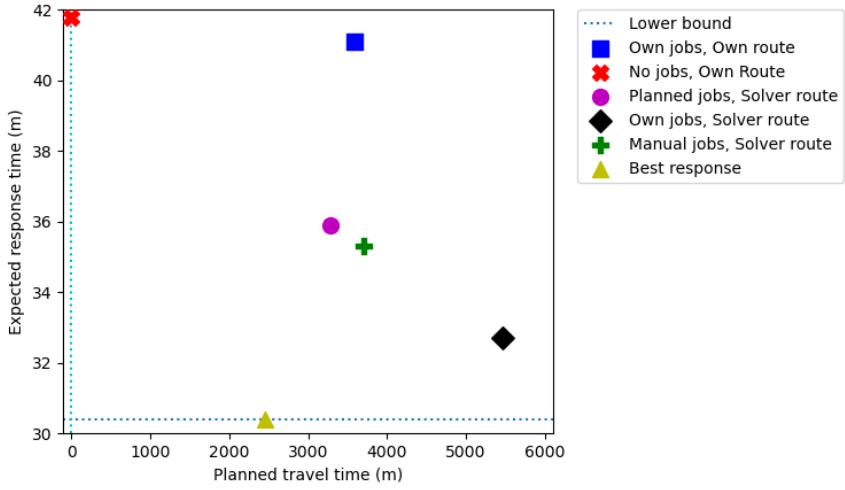


Figure 5.6: Planned travel time versus expected response time in the computed scenarios. On both axes, low is good.

	Own jobs Own route	No jobs Own route	Planned jobs Solver route	Own jobs Solver route	Best C	Manual jobs Solver route
Response (m)	41.1	41.8	35.9	32.7	30.4	35.3
$\leq$ 15 min (%)	6.7	3.4	9.6	10.6	12.8	9.3
$\leq$ 30 min (%)	30.5	29.0	42.1	46.6	53.1	41.4
$\leq$ 45 min (%)	60.3	61.8	72.0	78.2	85.9	73.7
$\leq$ 60 min (%)	83.6	82.6	90.4	94.9	97.2	93.1
Job score	71	0	45	71	0	93
Distance (m)	3580.8	0.0	3284.7	5457.7	2459.6	3697.0

Table 5.4: Computational results of the seven scenarios. The metrics are abbreviations of the metrics listed in Section 5.5.1. By 'Best C', we mean the 'Best response' scenario. The 'Best jobs' scenario is omitted from this table, because the only relevant result is that the weighted unique jobs (or 'job score') in this scenario is 102, which provides an upper bound for the other scenarios.

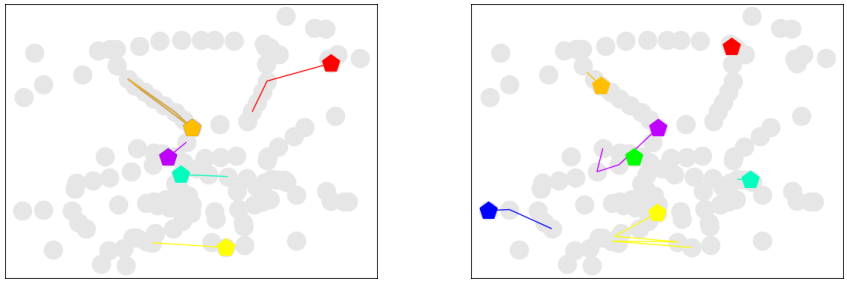
	Planned jobs Solver route	Own jobs Solver route	Manual jobs Solver route
Improvement expected response time (%)	12.7	20.5	14.2
Reduction incidents later than 60 (%)	41.1	68.7	58.1
Improvement weighted unique jobs (%)	-36.6	0	31.0
Improvement planned travelling (%)	8.3	-52.4	-3.3

Table 5.5: Improvements with respect to the current 'Own jobs, Own route' scenario. Only the three scenarios that include jobs, and in which we care about response time, are taken into account.

it is debatable whether a 20.5% improvement in response time is worth a 52.4% increase in travel time.

In the two scenarios where agents choose their own routes, we see expected response times of 41.1 and 41.8 minutes, respectively. In the three scenarios where the solver determines routes across non-empty sets of jobs, we see expected response times of 35.9, 32.7, and 35.3 minutes, respectively. This suggests that the improved response time is somewhat insensitive to the choice of jobs, but is mainly due to the routing being done by a solver. We expect this improvement in response time is mainly due to how 'idle time' is filled. If agents choose their own routes, the latter part of their shifts are spent near their homes, which are distributed somewhat arbitrarily. However, the solver often chooses waiting locations and start locations that are distributed more intelligently with the expected response times in mind. See also Figure 5.7.

The trade-off between improved the expected response time and the movement needed to attain it is perhaps high-lit most by comparing the "No jobs, Own route" and "Best C" scenarios. In the former, we hit the lower bound of 0 minutes on planned movement. In the latter, we hit the lower bound of 30.4 min-



(a) Solution in current practice scenario. Five out of seven agents start near the centre of the network, and four of them stay central.

(b) Solution in same shift in new practice scenario. The seven agents start with a better spread and largely maintain it.

Figure 5.7: For one of the ten shifts, a visual comparison of the solutions in the current practice and the new practice. The grey circles are the 126 nodes of the simplified network. For each agent, a coloured pentagon shows where they start, and the lines in the same colour illustrate their movement over the network. In the new practice, agents are more equally spread over the network.

utes on the expected response time. In total, the latter requires 2459.6 minutes of planned travelling. Over the ten shifts, there are 62 shifts worked by agents, so a rough estimate amounts to an agent moving 39.7 minutes per shift to improve the expected response time to emergencies by 9.1 minutes, and to be 'late' only 2.8% of the times instead of 17.4% of the times.

Alternatively, we can investigate the trade-off by comparing the current situation with the scenario where the same jobs are chosen, but they are scheduled and routed by the solver. While doing the same jobs, agents have to move 30.3 minutes more on average over their shift to improve the expected response time by 8.4 minutes, and to be late only 5.1% of the times instead of 16.4% of the times. Our research is conducted under the notion that response time is much more important than planned travelling time. However, these numerical results could be valuable input for a management discussion concerning this trade-off.

As a final remark, our simulations demonstrate that for our realistic use case, we can simultaneously improve the expected response times by 14.2% *and* the number of weighted unique jobs by 31.0%. We observe this in our final scenario, where we manually correct for the small number of jobs chosen by the case study planner. While these improvements do come at the cost of more planned movements, this increase is only 3.3%. In this scenario, the number of weighted unique jobs is 93, which is only 9% away from the upper bound of 102. The expected response time is 35.3 minutes, which is only 16% away from the lower bound of 30.4. The jobs in this planning scenario were chosen manually, and if this job selection were made methodical, these improvements would likely become

even larger.

5.7 Conclusions

In this chapter, we have generalised the MRP-methodology to incorporate additional objectives and constraints that are required in practice. To quantify the performance improvement that can be realised, we performed an extensive comparison study by simulations, comparing current practice to the results from our optimisation heuristics. We observed that, between the current practice and the new way of working allowed by our heuristic, we could strongly reduce response times and late arrivals in a realistic setting provided by our partnering railway provider. Most importantly, if the jobs are chosen ambitiously enough, we see that we can significantly improve both the response time performance and the (weighted) number of non-urgent tasks at the same time. We believe the reduction in response time is primarily due to agents being positioned in strategic locations during their idle time, rather than the endpoints of their job tours. This improvement can come at the expense of more planned movements outside of emergency response. However, in the most favourable scenario examined, this increase in planned movement is only marginal.

Finally, we address a number of topics for further investigation. Most importantly, we suspect that smartly assigning non-urgent jobs to agents intelligently is crucial to improve performance. In this research, tasks were mostly chosen by hand, which was seen as input to the model. But with an automated selection methodology, a system could smartly plan several shifts ahead independently. This requires further elaboration. Another promising direction is to include the incorporation of planned travel times in such a way that it does not require as many unfavourable constraints. Lastly, graph adjacency is currently independent of time, but this may not be realistic in cases with heavy traffic congestion.

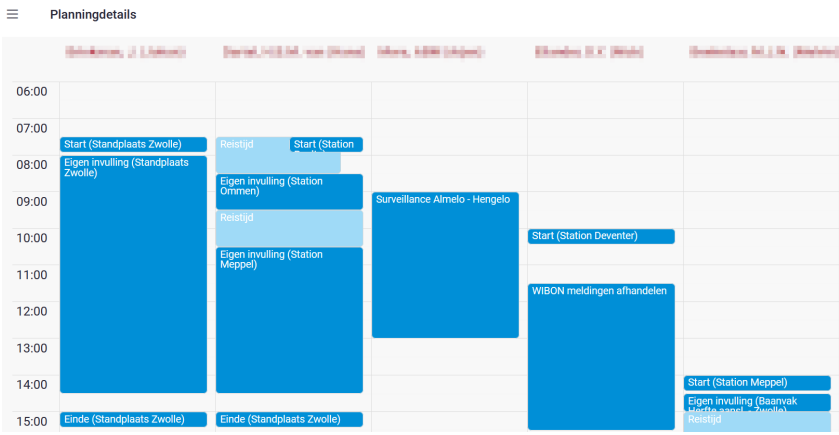
Chapter 6

Real-life Implementation and Next Steps

The ultimate goal of this thesis project was to help improve the operations of emergency response organisations, with the Dutch railway emergency response organisation as a prime example. While an initial prototype was developed for this organisation using the basic model from Chapter 2, it soon became apparent that there were many additional constraints to take into account. Over the course of this research project, these additional features were gradually introduced, eventually culminating in the E-MRP from Chapter 5. Fortunately, these improvements have indeed successfully yielded a solver, based on Algorithm 10 in Chapter 5, that has been taken into operational use by the Dutch railway emergency response organisation [119].

The user application was co-developed by CWI, ProRail BV and Capgemini. It consists of two modules. The *planner module* allows a planner, on a computer, to select agents and jobs for a shift, call the Algorithm 10-based solve server, visualise the result as a Gantt-chart (see Figure 6.1a) and as movements on the map (see Figure 6.1b), make limited manual adjustments to the output, and 'activate' the result. Once activated, every agent can use the *agent module* on his/her smart device to see what jobs and movements are planned for them. Agents can also mark jobs as finished or interrupted, and through integration with an already existing ProRail-app, report remarkable findings and propose new jobs for the planner. The planner can see, when choosing jobs, how often it has been marked as finished this year and how often it should be done this year: in these two ways, agents can provide feedback that the planner can use to choose jobs for subsequent plans.

Using this tool, there is reason to expect that the emergency response organisation will see an increase of approximately 31.0% in the amount of preventive work done and an improvement in emergency response time of approximately 14.2%; the results discussed in Section 5.6 indicate as much. After a period of rigorous usage, it would be valuable to analyse and evaluate the observed changes



(a) Gantt-chart visualisation.

(b) Map visualisation.

Figure 6.1: Example output of the E-MRP-based application used by the case study organisation, both in the Gantt-chart visualisation option and the map visualisation option. Names of employees have been blurred out for anonymity.

numerically.

It is exciting to see that simultaneous planning of emergency response and non-urgent jobs is gradually receiving more attention. At the time of writing, Bos et al. [17] have begun comparing MRP to their own non-discretised model, as well as to the ambulance dispatch model by Van den Berg and Van Essen [150]. Some of the most fundamental assumptions in the MRP model, namely that time is discretised and that we can only prepare one emergency ahead, will

be challenged by this comparison. The results of this comparison are likely to give interesting new insights into the goal of combined planning of emergency response and non-urgent jobs.

While the chapters of this thesis report on successful approaches and promising avenues for future research, it may be important for future researchers to know of four approaches that have not yet been successful.

First, it was expected that Benders decomposition [95] would be a successful approach, because the speed-up that Benders decomposition can create for the k-Median Problem [48] was successfully reproduced, and the approach can be straightforwardly adapted for MRP. However, the adapted procedure unexpectedly created a slow-down rather than a speed-up, even after several modifications, implying that a non-trivial improvement on the Benders decomposition method may be necessary.

Second, a primal-dual algorithm for TkMP can be derived based on the one that exists for the k-Median Problem [81], by interpreting (S, W)-paths of length ω as 'facilities' that are funded by 'clients' $(v, t) \in V \times T$. Though the number of such facilities is exponential, the path that currently collects the most 'crowdfunding' from unserviced clients can be found with a modified shortest path algorithm. The only reason that this procedure does not yield a constant-factor approximation for general metric TkMP, is that we can no longer claim that two clients (v, t) and (v', t') are 'close together' due to them being served by the same 'facility': the 'facility' is the path of an agent, and that agent may be in an entirely different place at t' then they were at t. If this wrinkle could be resolved or circumvented, a constant-factor approximation algorithm for TkMP may be in reach.

Third, we remark that in TkMP, we have to decide for each agent their path and which clients they serve: when one is fixed, the other becomes a trivial min-cost flow problem. Perhaps this problem, and many others, would benefit from an approach that can successfully 'alternate' between these two simple subproblems.

As a fourth and final observation, meta-heuristics like Genetic Algorithms [32], Simulated Annealing [26] and Tabu Search [128] have not yet been successful for MRP, because it is NP-complete to perform a 'neighbourhood operation' which preserves feasibility. That is, while any permutation of cities is trivially a feasible solution for TSP, it is computationally costly to reschedule a job to another agent and check whether this exchange admits a feasible solution. Because meta-heuristics rely on testing very many potential solutions, one would have to perform this NP-complete feasibility testing very many times, which quickly becomes intractable.

Though the above partial results offer interesting challenges moving forward, one topic deserves most attention for future research. At the heart of MRP and comparable models lies the assumption that the non-urgent jobs have already been selected [17]. Given this hard input, the question is how to still make the most

of the response time to potential emergencies. Now that the Median Routing Problem model has been sufficiently enriched to handle practical considerations, users in the railway emergency response organisation face a new bottleneck: how should the jobs be chosen in the first place?

In the case study organisation, there are hundreds of jobs that could potentially be selected as input to the MRP. Some of them are more important than others, in that there are financial consequences if these jobs are not performed before a certain deadline. Other jobs, like patrolling at intersections between the road and the railway, typically have no direct consequence, but neglecting them for too long can increase the probability of emergencies. When selecting jobs, the planner should not only weigh which jobs are currently most important, but the chosen job set should also be 'well spread' enough to admit an MRP solution with low emergency response time. Ideally, the job set should be large enough that every agent can fill up their shift meaningfully, but if the job set is *too* large, a feasible MRP solution no longer exists. Following the discussion in Section 2.3, it is clearly NP-hard to pack as many jobs as possible into a shift without it becoming infeasible. On top of all this, the planner has to take the many practical considerations of Chapter 5 into account, such as authorisations and job time windows and variable agent start locations. Putting all these considerations together, selecting a decent subset from the hundreds of jobs can quickly result in decision paralysis, and planners tend to resort to simple and static rules of thumb. Offering decision support for job selection appears to be the venue where most efficiency can be gained, both in terms of further improving MRP solution quality and in terms of supporting the planners in the case study organisation.

Some initial work has already been put forward in this direction. Kraster et al. have generalised the Median Routing Problem to the Multi-Period Median Routing Problem [90], in which a fixed set of jobs is to be divided over multiple shifts rather than one, so that each shift becomes an instance of MRP. Even with this singular additional feature, case study data suggests that the emergency response time can be further decreased by 3.51%, even with sparse job sets. One could further generalise the job selection process by assigning each job a positive 'reward value', and designing a model to select jobs with the highest reward such that the induced MRP instances still admit a solution with response time above some threshold. Alternatively, one could design a bi-objective optimisation model, in which the goal is to simultaneously minimise response time and penalties for skipping jobs. This latter model is a Lagrangian relaxation of MRP. Though this latter model sounds somewhat more difficult to properly interpret, a large benefit of this bi-objective model is that it is no longer NP-complete for this model to decide whether a feasible solution exists. This opens up many algorithmic venues that were not available to MRP. Remarking that job selection is currently the bottleneck in the decision-making process, and observing from the work of Kraster et al. that further reductions of response time can easily be expected, researching job selection seems like a promising and exciting next step.

Summary

This thesis studies the shift planning of emergency response organisations and how to include plannable, preventive tasks into the schedules of the responders, such that they remain well spread in case of emergencies. Both halves of this planning have individually received significant attention throughout the past decades: that is, much is known about either positioning responders in locations where they will have minimal emergency response times (e.g. the k-Median Problem [35]) or about efficiently visiting plannable jobs with a fleet of agents (e.g. the Vehicle Routing Problem [18]). However, when trying to simultaneously visit plannable jobs and retain a low emergency response time, the literature appears to still be in its infancy. Such a simultaneous planning seems promising, compared to the alternative of having responders wait at a response station: more preventive work can get done, and if tasks are chosen that are spread well, then the spread of agents can be better than if they are bunched together at the same response station.

Forming a planning that both visits plannable jobs and minimises emergency response times turns out to be challenging, both conceptually and computationally. An additional complication is that emergencies will inevitably disturb the planning, meaning rapid updates to the planning have to be made and communicated. This thesis seeks to develop computational decision support for this difficult task. It does so by introducing a new mathematical model that hybridises the k-Median Problem and Vehicle Routing Problem, studying different variants of this problem, and developing various algorithms that offer either optimal solutions, rapidly computable solutions that perform well in practice, or solutions with provable performance guarantees.

In Chapter 2, the *Median Routing Problem* is introduced as a model to compute how to visit plannable jobs in a graph in a fixed-length time horizon using a fleet of emergency response agents, while minimising the expected response time to the next emergency in the graph. This Median Routing Problem forms the backbone of this thesis. In this chapter, the foundational problem and its complexity are explored, and several algorithms are proposed. Not only is this problem found to be NP-hard, it is even NP-complete to decide whether a feasible solution exists. Therefore, aside from an exponential-time algorithm that can compute the optimal solution, heuristics are developed that can be used in practice. Most notably, the `MDSA`-algorithm is found to be an effective heuristic: on

benchmark instances, including case study data, it finds a solution in 2.4 seconds on average, which is on average only 3.2% away from optimal. The strength of `MDSA` lies in decomposing this combined problem into several decision steps per agent, which in isolation are less complex and/or act on much smaller decision spaces.

Because it is NP-complete to decide for the Median Routing Problem whether a feasible solution exists, it makes no sense to develop a polynomial-time approximation algorithm for the general problem. Instead, in Chapter 3, a special version is studied in the form of the *(Uniform) Travelling k-Median Problem*. Constant-factor approximation algorithms are demonstrated for very special cases, namely the cases where the number of agents or the length of the time horizon are disproportionally large compared to the number of nodes in the graph. By eliminating these cases, thus assuming that the number of agents and time-steps are reasonably small, a constant-factor approximation is given for specific graph metrics: namely, those graphs on which we can solve the problem in polynomial time if the agents are allowed to move 'continuously' over the graph instead of 'discretely'. Using a rounding theorem, it is shown that these 'continuous solutions' can be rounded back with bounded loss. For all other metrics, we provide a polynomial-time algorithm with an approximation factor that depends linearly on the graph diameter and sublinearly on the length of the time horizon.

Though Chapters 2 and 3 help us understand the basic Median Routing Problem, there are many more features that come into play in a practical application of this model. In Chapter 5, the *Enriched Median Routing Problem* is posed as an extension of the Median Routing Problem with fifteen additional features. For instance, the Enriched Median Routing Problem can take into account that jobs have time windows, or that some jobs need more than one agent. These additional constraints, especially the latter one, make it much more difficult to perform the decompositions that made `MDSA` successful. But in the same spirit of `MDSA`, a more flexible algorithm is developed that still decomposes some of the decisions into smaller problems. In order to have this simpler problem again act on a small decision space, a subroutine is developed in Chapter 4 that can take an input graph and 'sparsify' it into a graph with much fewer nodes and edges that can still reasonably 'represent' the original graph. In Chapter 4, not only is it demonstrated that this sparsification works well in a sample of practical Median Routing Problem use cases, but a proven upper bound is also given on how much quality loss can occur through sparsification, by introducing and using the Mixed Integer Jester Game framework for explicit worst-case computation.

Finally, in the second half of Chapter 5, the Enriched Median Routing Problem and its `MDSA`-inspired heuristic are compared to the current practice of a case study organisation. It is concluded that, indeed, the results of this thesis can help the case study organisation to simultaneously perform 31.0% more plannable work *and* decrease emergency response times by 14.2%. A solver based on this heuristic has been taken into use [119], and the further implementation of this solver and other outlooks are discussed in Chapter 6. This thesis answers the

question of how to perform plannable tasks with (near-)optimal emergency response time; the most important follow-up topic is how to choose which plannable tasks go into the planning.

Bibliography

[1] A. Agarwal, L. M. Hiot, E. M. Joo, and N. T. Nghia. Rectilinear workspace partitioning for parallel coverage using multiple unmanned aerial vehicles. *Advanced Robotics*, 21(1-2):105–120, 2007.

[2] R. Ahmed, G. Bodwin, F. D. Sahneh, K. Hamm, M. J. L. Jebelli, S. Kobourov, and R. Spence. Graph spanners: A tutorial review. *Computer Science Review*, 37:100253, 2020.

[3] S. Almoustafa, S. Hanafi, and N. Mladenović. New exact method for large asymmetric distance-constrained vehicle routing problem. *European Journal of Operational Research*, 226(3):386–394, 2013.

[4] W. K. Anuar, L. S. Lee, S. Pickl, and H.-V. Seow. Vehicle routing optimisation in humanitarian operations: A survey on modelling and optimisation approaches. *Applied Sciences*, 11(2):667, 2021.

[5] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k-medians and related problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, page 106–113. Association for Computing Machinery, 1998.

[6] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[7] M. Başdere and Ü. Bilge. Operational aircraft maintenance routing problem with remaining time consideration. *European Journal of Operational Research*, 235(1):315–328, 2014.

[8] J. Batson, D. A. Spielman, and N. Srivastava. Twice-Ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

[9] J. Batson, D. A. Spielman, N. Srivastava, and S.-H. Teng. Spectral sparsification of graphs: theory and algorithms. *Communications of the ACM*, 56(8):87–94, 2013.

[10] T. Bektaş, L. Gouveia, and D. Santos. Revisiting the hamiltonian p-median problem: A new formulation on directed graphs and a branch-and-cut algorithm. *European Journal of Operational Research*, 276(1):40–64, 2019.

[11] T. Bektaş and G. Laporte. The pollution-routing problem. *Transportation Research Part B: Methodological*, 45(8):1232–1250, 2011.

[12] S. Benati and S. García. A mixed integer linear model for clustering with variable selection. *Computers & Operations Research*, 43:280–285, 2014.

[13] D. J. Bertsimas and G. Van Ryzin. Stochastic and dynamic vehicle routing in the Euclidean plane with multiple capacitated vehicles. *Operations Research*, 41(1):60–76, 1993.

[14] T. C. Biedl, B. Brejová, and T. Vinař. Simplifying flow networks. In *International Symposium on Mathematical Foundations of Computer Science*, pages 192–201. Springer, 2000.

[15] G. Bodwin. A note on distance-preserving graph sparsification. *arXiv preprint arXiv:2001.07741*, 2020.

[16] A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM*, 39(4):745–763, Oct. 1992.

[17] J. Bos, D. Huizing, R. J. Boucherie, D. Pecin, and R. Spliet. Spread smart, respond fast, 2022. Manuscript in preparation.

[18] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313, 2016.

[19] L. Brotcorne, G. Laporte, and F. Semet. Ambulance location and relocation models. *European Journal of Operational Research*, 147(3):451–463, 2003.

[20] J. Byrka, T. Pensyl, B. Rybicki, A. Srinivasan, and K. Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Transactions on Algorithms*, 13(2), 2017.

[21] J. Caceres-Cruz, P. Arias, D. Guimarans, D. Riera, and A. A. Juan. Rich vehicle routing problem: Survey. *ACM Computing Surveys*, 47(2):1–28, 2014.

[22] F. Camci. Maintenance scheduling of geographically distributed assets with prognostics information. *European Journal of Operational Research*, 245(2):506–516, 2015.

[23] A. M. Caunhye, X. Nie, and S. Pokharel. Optimization models in emergency logistics: A literature review. *Socio-economic Planning Sciences*, 46(1):4–13, 2012.

[24] Centrum Wiskunde & Informatica. Cwi official website, 2021. https://www.cwi.nl. Accessed 30 November 2021.

[25] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002.

[26] F. Chiyoshi and R. D. Galvao. A statistical analysis of simulated annealing applied to the p-median problem. *Annals of Operations Research*, 96(1):61–74, 2000.

[27] P. Christiano, J. A. Kelner, A. Madry, D. A. Spielman, and S.-H. Teng. Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, pages 273–282, 2011.

[28] V. Cohen-Addad, A. Gupta, A. Kumar, E. Lee, and J. Li. Tight FPT approximations for k-median and k-means. *arXiv preprint arXiv:1904.12334*, 2019.

[29] A. M. Cohn and C. Barnhart. Improving crew scheduling by incorporating key maintenance routing decisions. *Operations Research*, 51(3):387–396, 2003.

[30] J. M. Colmenar, A. Hoff, R. Martí, and A. Duarte. Scatter search for the bicriteria p-median p-dispersion problem. *Progress in Artificial Intelligence*, 7(1):31–40, 2018.

[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT press, 2009.

[32] E. S. Correa, M. T. A. Steiner, A. A. Freitas, and C. Carnieri. A genetic algorithm for solving a capacitated p-median problem. *Numerical Algorithms*, 35(2):373–388, 2004.

[33] T. G. Crainic, G. Perboli, S. Mancini, and R. Tadei. Two-echelon vehicle routing problem: a satellite location analysis. *Procedia-Social and Behavioral Sciences*, 2(3):5944–5955, 2010.

[34] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[35] M. S. Daskin and K. L. Maass. The p-median problem. In *Location Science*, pages 21–45. Springer, 2015.

[36] F. De Dinechin and M. Istoan. Hardware implementations of fixed-point atan2. In *2015 IEEE 22nd Symposium on Computer Arithmetic*, pages 34–41. IEEE, 2015.

[37] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[38] J. Ding, J. R. Lee, and Y. Peres. Cover times, blanket times, and majorizing measures. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, pages 61–70, 2011.

[39] L. Doitsidis, S. Weiss, A. Renzaglia, M. W. Achtelik, E. Kosmatopoulos, R. Siegwart, and D. Scaramuzza. Optimal surveillance coverage for teams of micro aerial vehicles in GPS-denied environments using onboard vision. *Autonomous Robots*, 33(1-2):173–188, 2012.

[40] M. Drexl and M. Schneider. A survey of variants and extensions of the location-routing problem. *European Journal of Operational Research*, 241(2):283–308, 2015.

[41] B. Eksioglu, A. V. Vural, and A. Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483, 2009.

[42] M. El Amrani, Y. Benadada, and B. Gendron. Generalization of capacitated p-median location problem: modeling and resolution. In *2016 3rd International Conference on Logistics Operations Management*, pages 1–6. IEEE, 2016.

[43] S. Erdougan and E. Miller-Hooks. A green vehicle routing problem. *Transportation Research Part E: Logistics and Transportation Review*, 48(1):100–114, 2012.

[44] R. Z. Farahani, M. Abedian, and S. Sharahi. Dynamic facility location problem. In *Facility Location*, pages 347–372. Springer, 2009.

[45] R. Z. Farahani, S. Fallah, R. Ruiz, S. Hosseini, and N. Asgari. OR models in urban service facility location: A critical review of applications and future developments. *European Journal of Operational Research*, 276(1):1–27, 2019.

[46] R. Z. Farahani and M. Hekmatfar. *Facility Location: Concepts, Models, Algorithms and Case Studies*. Springer, 2009.

[47] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *Journal of Computer and System Sciences*, 51(2):261–272, 1995.

[48] M. Fischetti, I. Ljubić, and M. Sinnl. Redesigning Benders decomposition for large-scale facility location. *Management Science*, 63(7):2146–2162, 2017.

[49] M. M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956.

[50] J. E. Fontecha, O. O. Guaje, D. Duque, R. Akhavan-Tabatabaei, J. P. Rodríguez, and A. L. Medaglia. Combined maintenance and routing optimization for large-scale sewage cleaning. *Annals of Operations Research*, pages 1–34, 2019.

[51] R. D. Galvão and E. d. R. Santibanez-Gonzalez. A Lagrangean heuristic for the *pk*-median dynamic location problem. *European Journal of Operational Research*, 58(2):250–262, 1992.

[52] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. Freeman, 1979.

[53] E. Garone, R. Naldi, A. Casavola, and E. Frazzoli. Cooperative mission planning for a class of carrier-vehicle systems. In *49th IEEE Conference on Decision and Control*, pages 1354–1359. IEEE, 2010.

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

[55] M. Gendreau, G. Laporte, and F. Semet. A dynamic model and parallel tabu search heuristic for real-time ambulance relocation. *Parallel Computing*, 27(12):1641–1653, 2001.

[56] M. Gendreau, G. Laporte, and F. Semet. The maximal expected coverage relocation problem for emergency vehicles. *Journal of the Operational Research Society*, 57(1):22–28, 2006.

[57] J. C. Goodson, J. W. Ohlmann, and B. W. Thomas. Rollout policies for dynamic solutions to the multivehicle routing problem with stochastic demand and duration limits. *Operations Research*, 61(1):138–154, 2013.

[58] R. Gopalan and K. T. Talluri. The aircraft maintenance routing problem. *Operations Research*, 46(2):260–271, 1998.

[59] S. Gorenstein. Printing press scheduling for multi-edition periodicals. *Management Science*, 16(6):B–373, 1970.

[60] C. Groër, B. Golden, and E. Wasil. The consistent vehicle routing problem. *Manufacturing & Service Operations Management*, 11(4):630–643, 2009.

[61] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228 – 248, 1999.

[62] A. Gupta, M. Pál, R. Ravi, and A. Sinha. Boosted sampling: approximation algorithms for stochastic optimization. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, pages 417–426, 2004.

[63] M. Haouari, S. Shao, and H. D. Sherali. A lifted compact formulation for the daily aircraft maintenance routing problem. *Transportation Science*, 47(4):508–525, 2012.

[64] M. J. Havinga and B. de Jonge. Condition-based maintenance in the cyclic patrolling repairman problem. *International Journal of Production Economics*, 222:107497, 2020.

[65] J. Hochstetler, L. Hochstetler, and S. Fu. An optimal police patrol planning strategy for smart city safety. In *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems*, pages 1256–1263. IEEE, 2016.

[66] J. Hoogeveen. Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10(5):291–295, 1991.

[67] H. Hotelling. Stability in competition. In *The Collected Economics Articles of Harold Hotelling*, pages 50–63. Springer, 1990.

[68] W. V. Huang, R. Batta, and A. Babu. Relocation-promotion problem with Euclidean distance. *European Journal of Operational Research*, 46(1):61–72, 1990.

[69] D. Huizing and G. Schäfer. The traveling k-median problem: approximating optimal network coverage. In *International Workshop on Approximation and Online Algorithms*. Springer (to appear).

[70] D. Huizing, G. Schäfer, R. D. van der Mei, and S. Bhulai. The Median Routing Problem for simultaneous planning of emergency response and non-emergency jobs. *European Journal of Operational Research*, 285(2):712–727, 2020.

[71] D. Huizing, G. Schäfer, R. D. van der Mei, and S. Bhulai. Distance-preserving graph sparsification with bounded loss for network problems, 2021. Submitted.

[72] D. Huizing, R. D. van der Mei, G. Schäfer, and S. Bhulai. The Enriched Median Routing Problem and its usefulness in practice, 2021. Submitted.

[73] S. Ichoua, M. Gendreau, and J.-Y. Potvin. Diversion issues in real-time vehicle dispatching. *Transportation Science*, 34(4):426–438, 2000.

[74] T. Ilhan, S. M. Iravani, and M. S. Daskin. The orienteering problem with stochastic profits. *IIE Transactions*, 40(4):406–421, 2008.

[75] M. Imre, J. Tao, Y. Wang, Z. Zhao, Z. Feng, and C. Wang. Spectrum-preserving sparsification for visualization of big graphs. *Computers & Graphics*, 87:89–102, 2020.

[76] C. A. Irawan, D. Ouelhadj, D. Jones, M. Stålhane, and I. B. Sperstad. Optimisation of maintenance routing and scheduling for offshore wind farms. *European Journal of Operational Research*, 256(1):76–89, 2017.

[77] A. Jackman and M. Beruvides. What emergency responders can learn from the business world. *Harvard Business Review*.

[78] L. E. Jackson, G. N. Rouskas, and M. F. Stallmann. The directional p-median problem: Definition, complexity, and algorithms. *European Journal of Operational Research*, 179(3):1097–1108, 2007.

[79] C. J. Jagtenberg, S. Bhulai, and R. D. van der Mei. An efficient heuristic for real-time ambulance redeployment. *Operations Research for Health Care*, 4:27–35, 2015.

[80] P. Jaillet, J. F. Bard, L. Huang, and M. Dror. Delivery cost approximations for inventory routing problems in a rolling horizon framework. *Transportation Science*, 36(3):292–300, 2002.

[81] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[82] J. W. Joubert. *An integrated and intelligent metaheuristic for constrained vehicle routing*. Doctoral dissertation, University of Pretoria, 2007.

[83] C. Karande, K. Chellapilla, and R. Andersen. Speeding up algorithms on compressed web graphs. *Internet Mathematics*, 6(3):373–398, 2009.

[84] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. II: The p-medians. *SIAM Journal on Applied Mathematics*, 37(3):539–560, 1979.

[85] Y. Kergosien, M. Gendreau, A. Ruiz, and P. Soriano. Managing a fleet of ambulances to respond to emergency and transfer patient transportation demands. In *Proceedings of the International Conference on Health Care Systems Engineering*, pages 303–315. Springer, 2014.

[86] G. Kiechle, K. F. Doerner, M. Gendreau, and R. F. Hartl. Waiting strategies for regular and emergency patient transportation. In *Operations Research Proceedings 2008*, pages 271–276. Springer, 2009.

[87] I. Koutis, G. L. Miller, and R. Peng. A nearly-$m \log n$ time solver for SDD linear systems. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 590–598. IEEE, 2011.

[88] E. Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.

[89] E. Koutsoupias and C. H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995.

[90] D. Kraster. Heuristic approaches for a multi-period job scheduling and emergency coverage problem, 2020. https://www.ubvu.vu.nl/pub/fulltext/scripties/27_2578856_1.pdf.

[91] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[92] S. N. Kumar and R. Panneerselvam. A survey on the vehicle routing problem and its variants. *Intelligent Information Management*, 4(3):66–74, 2012.

[93] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

[94] G. Laporte, M. Desrochers, and Y. Nobert. Two exact algorithms for the distance-constrained vehicle routing problem. *Networks*, 14(1):161–172, 1984.

[95] L. S. Lasdon. *Optimization theory for large systems*. Courier Corporation, 2002.

[96] K. LeFevre and E. Terzi. GraSS: Graph structure summarization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 454–465. SIAM, 2010.

[97] S. Lewandowski. Shortest paths and negative cycle detection in graphs with negative weights. Technical Report No. 2010/05, Stuttgart University, 2010.

[98] X. Li, Z. Zhao, X. Zhu, and T. Wyatt. Covering models and optimization techniques for emergency response facility location and planning: a review. *Mathematical Methods of Operations Research*, 74(3):281–310, 2011.

[99] J.-H. Lin and J. S. Vitter. e-approximations with minimum packing constraint violation. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, page 771–782. Association for Computing Machinery, 1992.

[100] Y. Liu, T. Safavi, A. Dighe, and D. Koutra. Graph summarization methods and applications: a survey. *ACM Computing Surveys*, 51(3):1–34, 2018.

[101] Y. Liu, Y. Yuan, J. Shen, and W. Gao. Emergency response facility location in transportation networks: a literature review. *Journal of Traffic and Transportation Engineering*, 2021.

[102] E. López-Santana, R. Akhavan-Tabatabaei, L. Dieulle, N. Labadie, and A. L. Medaglia. On the combined maintenance and routing optimization problem. *Reliability Engineering & System Safety*, 145:199–214, 2016.

[103] G. Maróti and L. Kroon. Maintenance routing for train units: the transition model. *Transportation Science*, 39(4):518–525, 2005.

[104] Y. Mehmood, N. Barbieri, F. Bonchi, and A. Ukkonen. CSI: Community-level social influence analysis. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 48–63. Springer, 2013.

[105] M. T. Melo, S. Nickel, and F. S. Da Gama. Dynamic multi-commodity capacitated facility location: a mathematical modeling framework for strategic supply chain planning. *Computers & Operations Research*, 33(1):181–208, 2006.

[106] E. Misiołek and D. Z. Chen. Two flow network simplification algorithms. *Information Processing Letters*, 97(5):197–202, 2006.

[107] J. R. Montoya-Torres, J. L. Franco, S. N. Isaza, H. F. Jiménez, and N. Herazo-Padilla. A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering*, 79:115–129, 2015.

[108] A. Mor and M. G. Speranza. Vehicle routing problems over time: a survey. *4OR*, 18(2):129–149, 2020.

[109] V. Nagarajan and R. Ravi. Approximation algorithms for distance constrained vehicle routing problems. *Networks*, 59(2):209–214, 2012.

[110] Y. Nesterov and A. Nemirovskii. *Interior-point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.

[111] S. H. Owen and M. S. Daskin. Strategic facility location: A review. *European Journal of Operational Research*, 111(3):423–447, 1998.

[112] R. Palma-Behnke, C. Benavides, F. Lanas, B. Severino, L. Reyes, J. Llanos, and D. Sáez. A microgrid energy management system based on the rolling horizon strategy. *IEEE Transactions on Smart Grid*, 4(2):996–1006, 2013.

[113] D. Peleg and A. A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

[114] M. Penicka, A. K. Strupchanska, and D. Bjørner. Train maintenance routing. In *FORMS'2003: Symposium on Formal Methods for Railway Operation and Control Systems*, 2003.

[115] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

[116] V. Pillac, C. Guéret, and A. L. Medaglia. An event-driven optimization framework for dynamic vehicle routing. *Decision Support Systems*, 54(1):414–423, 2012.

[117] V. Pillac, C. Gueret, and A. L. Medaglia. A parallel matheuristic for the technician routing and scheduling problem. *Optimization Letters*, 7(7):1525–1535, 2013.

[118] D. R. Plane and T. E. Hendrick. Mathematical programming and the location of fire companies for the Denver fire department. *Operations Research*, 25(4):563–578, 1977.

[119] ProRail BV. Slim alarmeren bij incidenten op het spoor *(in Dutch)*, 2020. https://www.prorail.nl/nieuws/slim-alarmeren-bij-incidenten-op-het-spoor. Accessed 13 October 2021.

[120] ProRail BV. Prorail official website, 2021. https://www.prorail.nl. Accessed 30 November 2021.

[121] ProRail BV. Spoorlopen *(in Dutch)*, 2021. https://www.prorail.nl/veiligheid/spoorlopen. Accessed 11 November 2021.

[122] ProRail BV. Wat wij doen incidentenbestrijding *(in Dutch)*, 2021. https://www.prorail.nl/over-ons/wat-doet-prorail/incidentenbestrijding. Accessed 11 November 2021.

[123] H. N. Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.

[124] S. Raghavan, M. Sahin, and F. S. Salman. The capacitated mobile facility location problem. *European Journal of Operational Research*, 277(2):507–520, 2019.

[125] M. Rashidnejad, S. Ebrahimnejad, and J. Safari. A bi-objective model of preventive maintenance planning in distributed systems considering vehicle routing problem. *Computers & Industrial Engineering*, 120:360–381, 2018.

[126] C. S. ReVelle and R. W. Swain. Central facilities location. *Geographical Analysis*, 2(1):30–42, 1970.

[127] M. Riondato, D. García-Soriano, and F. Bonchi. Graph summarization with quality guarantees. *Data Mining and Knowledge Discovery*, 31(2):314–349, 2017.

[128] E. Rolland, D. A. Schilling, and J. R. Current. An efficient tabu search procedure for the p-median problem. *European Journal of Operational Research*, 96(2):329–342, 1997.

[129] N. Ruan, R. Jin, and Y. Huang. Distance preserving graph simplification. In *2011 IEEE 11th International Conference on Data Mining*, pages 1200–1205. IEEE, 2011.

[130] V. Sadhanala, Y.-X. Wang, and R. Tibshirani. Graph sparsification approaches for Laplacian smoothing. In *Artificial Intelligence and Statistics*, pages 1250–1259. PMLR, 2016.

[131] A. Sadri, F. D. Salim, Y. Ren, M. Zameni, J. Chan, and T. Sellis. Shrink: Distance preserving graph compression. *Information Systems*, 69:180–193, 2017.

[132] A. Sarac, R. Batta, and C. M. Rump. A branch-and-price approach for operational aircraft maintenance routing. *European Journal of Operational Research*, 175(3):1850–1869, 2006.

[133] M. W. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

[134] F. Schalekamp and D. B. Shmoys. Algorithms for the universal and a priori TSP. *Operations Research Letters*, 36(1):1–3, 2008.

[135] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[136] S. Segarra, A. G. Marques, G. Leus, and A. Ribeiro. Aggregation sampling of graph signals in the presence of noise. In *2015 IEEE 6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing*, pages 101–104. IEEE, 2015.

[137] L. Shu, W. Wang, F. Lin, Z. Liu, and J. Zhou. A sweep coverage scheme based on vehicle routing problem. *Indonesian Journal of Electrical Engineering and Computer Science*, 11(4):2029–2036, 2013.

[138] M. M. Solomon. *Vehicle routing and scheduling with time window constraints: models and algorithms (heuristics)*. University of Pennsylvania, 1984.

[139] D. A. Spielman and S.-H. Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014.

[140] K. T. Talluri. The four-day aircraft maintenance routing problem. *Transportation Science*, 32(1):43–53, 1998.

[141] A. Tamir. An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19(2):59–64, 1996.

[142] Y. Tao, C. Sheng, and J. Pei. On k-skip shortest paths. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, pages 421–432, 2011.

[143] M. B. Teitz. Locational strategies for competitive systems. *Journal of Regional Science*, 8(2):135–148, 1968.

[144] T. C. Thayer and S. Carpin. Solving large-scale stochastic orienteering problems with aggregation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2452–2458. IEEE, 2020.

[145] The European Parliament and the Council of European Union. Directive 2012/34/EU. *Official Journal of the European Union*.

[146] H. Toivonen, F. Zhou, A. Hartikainen, and A. Hinkka. Compression of weighted graphs. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 965–973, 2011.

[147] C. Toregas, R. Swain, C. ReVelle, and L. Bergman. The location of emergency service facilities. *Operations Research*, 19(6):1363–1373, 1971.

[148] P. Toth and D. Vigo. *Vehicle Routing: Problems, Methods, and Applications*. SIAM, 2014.

[149] T. Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35(9):797–809, 1984.

[150] P. van den Berg and T. Van Essen. Scheduling non-urgent patient transportation while maximizing emergency coverage. *Transportation Science*, 53(2):319–622, 2019.

[151] M. van Ee and R. Sitters. Routing under uncertainty: the a priori traveling repairman problem. In *International Workshop on Approximation and Online Algorithms*, pages 248–259. Springer, 2014.

[152] P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden. The orienteering problem: a survey. *European Journal of Operational Research*, 209(1):1–10, 2011.

[153] V. Verter. Uncapacitated and capacitated facility location problems. In *Foundations of Location Analysis*, pages 25–37. Springer, 2011.

[154] VRP-REP. Problem variants, 2021. http://www.vrp-rep.org/variants.html. Accessed 7 April 2021.

[155] X. Wang and E. Wasil. On the road to better routes: Five decades of published research on the vehicle routing problem. *Networks*, 77(1):66–87, 2021.

[156] Y. Wang and I. I. Hussein. Cooperative vision-based multi-vehicle dynamic coverage control for underwater applications. In *IEEE International Conference on Control Applications*, pages 82–87. IEEE, 2007.

[157] G. O. Wesolowsky. Dynamic facility location. *Management Science*, 19(11):1241–1248, 1973.

[158] Y. Xiang, R. Jin, D. Fuhry, and F. F. Dragan. Summarizing transactional databases with overlapped hyperrectangles. *Data Mining and Knowledge Discovery*, 23(2):215–251, 2011.

[159] H. Xu, D. Fang, and Y. Jin. Emergency logistics theory, model and method: A review and further research directions. *Advances in Computer Science Research*, 65:188–192, 2018.

[160] R. Zenklusen. A 1.5-approximation for path TSP. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1539–1549. SIAM, 2019.

[161] S. Zhang, J. W. Ohlmann, and B. W. Thomas. A priori orienteering with time windows and stochastic wait times at customers. *European Journal of Operational Research*, 239(1):70–79, 2014.

[162] D. Zhou, J. Huang, and B. Schölkopf. Learning from labeled and unlabeled data on a directed graph. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 1036–1043, 2005.

Appendix A

Subroutines of Chapter 2

A.1 MILP for routing subroutine of WAM

In step 1 of WAM, as described in Algorithm 1 of Section 2.4.2, several medians are identified as points of interest. In step 2, a distance matrix D is obtained. In step 3, the agents are routed over the jobs and these medians in such a way that the total amount of travelling time is minimised, so that the total amount of time that can be spent at these medians is maximised. Aside from the standard condition that all jobs and medians must be visited exactly once, each agent must visit exactly one median. If there were one location where all agents start and end, and the number of medians visited per agent were allowed to be arbitrary, then this would simply be an instance of DVRP. However, these two side constraints merit the explicit description of the MILP used to solve this subroutine.

Define a set $\mathcal{S} = \{s_1, \ldots, s_{|A|}\}$ of virtual nodes, representing the start 'locations' of the agents. Similarly, define $\mathcal{W} = \{w_1, \ldots, w_{|A|}\}$ the virtual end locations of the agents. For any distinct pair $i \in \mathcal{S} \cup \mathcal{M} \cup J$, $j \in \mathcal{M} \cup \mathcal{W} \cup J$, let variable $\zeta_{ij} \in \{0, 1\}$ indicate whether or not someone goes directly to j after visiting i. For each $j \in \mathcal{M} \cup \mathcal{W} \cup \mathcal{J}$, define a variable $f_j \in [0, \omega]$ describing the arrival time at j; for each $j \in \mathcal{S}$, denote $f_j = 0$. For $j \in \mathcal{S} \cup \mathcal{M} \cup \mathcal{W}$, denote 'processing time' $Q_j = 0$. Finally, for each $a \in A$ and each $j \in \mathcal{M} \cup J$, define a variable $z'_{aj} \in \{0, 1\}$ indicating whether or not j is visited by a. For $a_i \in A$, $s \in \mathcal{S}$ and $e \in \mathcal{W}$, denote $z'_{a_i s} = 1$ if $s = s_i$ and 0 otherwise, and $z'_{a_i w} = 1$ if $w = w_i$ and 0 otherwise: in other words, encode that start point s_i and end point w_i are always assigned to agent a_i. Then the following MILP produces the optimal sequences:

$$\min \sum_{i \in \mathcal{S} \cup \mathcal{M} \cup J} \sum_{j \in \mathcal{M} \cup \mathcal{W} \cup J} D_{ij} \zeta_{ij}$$

$$s.t. \sum_{j \in \mathcal{M} \cup \mathcal{W} \cup J} \zeta_{ij} = 1 \qquad (\forall i \in \mathcal{S} \cup \mathcal{M} \cup J) \qquad \text{(A.1)}$$

$$\sum_{i \in \mathcal{S} \cup \mathcal{M} \cup J} \zeta_{ij} = 1 \qquad (\forall j \in \mathcal{M} \cup \mathcal{W} \cup J) \qquad \text{(A.2)}$$

$$f_j \geq f_i - \omega + \zeta_{ij}(\omega + Q_i + D_{ij}) \qquad (\forall i \in \mathcal{S} \cup \mathcal{M} \cup J)$$
$$(\forall j \in \mathcal{M} \cup \mathcal{W} \cup J) \qquad \text{(A.3)}$$

$$\sum_{m \in \mathcal{M}} z'_{am} = 1 \qquad (\forall a \in A) \qquad \text{(A.4)}$$

$$\sum_{a \in A} z'_{aj} = 1 \qquad (\forall j \in \mathcal{S} \cup \mathcal{M} \cup \mathcal{W} \cup J) \qquad \text{(A.5)}$$

$$z'_{aj} \geq z'_{ai} + \zeta_{ij} - 1 \qquad (\forall a \in A)(\forall i \in \mathcal{S} \cup \mathcal{M} \cup J)$$
$$(\forall j \in \mathcal{M} \cup \mathcal{W} \cup J) \qquad \text{(A.6)}$$

$$\zeta_{ij}, z'_{aj} \in \{0, 1\}, \ f_j \in [0, \omega]$$

Here, the objective value equals the total number of time steps spent travelling. Constraint (A.1) states that, aside from end points, each node must be departed from exactly once. Constraint (A.2) states that, aside from start points, each node must be visited exactly once. Constraint (A.3) has two functions: it recursively ensures that agents are at their end points no later than ω, and acts as a subtour elimination constraint. Constraint (A.4) states that each agent must be assigned exactly one median. Constraint (A.5) states that each node must be assigned to exactly one agent. Constraint (A.6) states that if $i \in \mathcal{S} \cup \mathcal{M} \cup J$ is assigned to some agent $a \in A$, and i is followed up by some $j \in \mathcal{M} \cup \mathcal{W} \cup J$, then j is assigned to a as well. These last three rules, combined with the fact that each starting point is assigned to exactly one agent, ensure that each median is assigned to exactly one agent. Furthermore, constraint (A.4) and constraint (A.5) ensure that the route that starts at s_a also ends in the right w_a.

When this MILP is solved with a MILP solver, the values of ζ_{ij} describe the sequences that minimise the time spent travelling, thus creating a maximal amount of time that can be spent waiting at medians.

A.2 Solving path-TSP

In this section, we go into more detail for step 5 of Algorithm 2 in Section 2.4.3. Observe a node set J_a, a start node S_a, an end node W_a and a symmetric distance matrix D_{ij}. For brevity, denote $J^+ = \{S_a\} \cup J_a \cup \{W_a\}$. For each distinct $i \in J^+$, $i \neq W_a$ and $j \in J^+$, $j \neq S_a$, define a variable $\zeta_{ij} \in \{0, 1\}$ describing whether or not j is the next node visited after i. Also define a variable $u_j \in [1, |J^+| - 1]$,

indicating how many nodes have been visited before j. Denote $u_{S_a} = 0$. The path-TSP, to find the shortest path that starts at S_a, ends at W_a and visits all nodes exactly once, can be solved by the following MILP.

$$\begin{aligned}
\min \quad & \sum_{i \in J^+, i \neq W_a} \sum_{j \in J^+, j \neq S_a} D_{ij} \zeta_{ij} \\
s.t. \quad & \sum_{j \in J^+, j \neq S_a} \zeta_{ij} = 1 && (\forall i \in J^+, i \neq W_a) && \text{(A.7)} \\
& \sum_{i \in J^+, i \neq W_a} \zeta_{ij} = 1 && (\forall j \in J^+, j \neq S_a) && \text{(A.8)} \\
& u_j \geq u_i + \zeta_{ij} - 1 && (\forall i \in J^+, i \neq W_a)(\forall j \in J^+, j \neq S_a) && \text{(A.9)} \\
& \zeta_{ij} \in \{0, 1\}, \; u_j \in [1, |J^+| - 1]
\end{aligned}$$

Here, the objective value describes the total amount of distance travelled. Constraint (A.7) states that each node, except the end node, must be departed from exactly once. Constraint (A.8) states that each node, except the start node, must be visited exactly once. Constraint (A.9) acts as a subtour elimination constraint.

After solving this MILP with a MILP solver, reading out ζ_{ij} produces a sequence with minimal time spent travelling.

A.3 Cheapest space-time path for sequence and node costs

The Dynamic Program Algorithm 12 (see Section 2.4.3) takes as input some start point S_a, some end point W_a, some sequence over job set J_a and some cost function on the nodes that may depend on the time step. It returns the cheapest feasible space-time path with respect to these node costs.

A.4 Instance generator

To create an instance of the MRP, the following parameters should be provided:

- The desired number of agents, $|A|$;
- The desired number of jobs, $|J|$;
- The desired number of nodes, $|V|$;
- The desired number of time steps, ω;
- The width of a large square, $SCALE$;

ALGORITHM 12: Optimal execution of job sequence against given node costs

1: Initialise the set of active nodes $\mathcal{N} := \{(depot, 0, j_0)\}$, with j_0 the first job that has to be visited, or $j_0 = W_a$ if the sequence has no jobs;
2: Initialise the reach cost $reach : \mathcal{N} \to \mathbb{R}_{\geq 0}$ as $reach(S_a, 0, j_0) = 0$;
3: Initialise the explored nodes $\mathcal{E} := \emptyset$;
4: Take $(u, t, state) := \arg\min_{\mathcal{N}} reach(u, t, state)$, remove this from $\mathcal{N}$ and add it to $\mathcal{E}$;
5: Observe each 'neighbour' $(v, t+1, state')$ of $(u, t, state)$. $(v, t+1, state')$ is a neighbour of $(u, t, state)$ if $v \in V_u$, and either:
   - $state = j \in \{W_a\} \cup J_a$ and $state' = j$ (the goal does not change);
   - $state = j \in J$, $L_j = v$ and $state' = (j, Q_j)$ (the next job is started);
   - $state = (j, q)$, $q > 1$, $u = v$ and $state' = (j, q-1)$ (the job is processed further);
   - $state = (j, 1)$, $u = v$, j' is the next job if there is one and W_a otherwise, and $state' = j'$ (the job is finished, the next goal retrieved);
   - $state = (j, 1)$, $u = v$, j' is the next job to be visited, $L_{j'} = v$ and $state' = (j', Q_{j'})$ (the job is finished, and the next job happens to be in the same location).
6: For each neighbour $(v, t+1, state')$, if it is not already in $\mathcal{N}$ or $\mathcal{E}$, add it to $\mathcal{N}$ and set $reach(v, t+1, state') = reach(u, t, state) + nodecost_a(v, t+1)$;
7: If the end node $(W_a, \omega, W_a) \notin \mathcal{N}$, go to step 4;
8: Using $reach$, backtrack the cheapest path to (W_a, ω, W_a) from $(S_a, 0, j_0)$.

- The threshold for adjacency, G_CHOP;
- The minimal distance between nodes (for visualisation and realism purposes), MIN_DIST;
- The range of potential processing times for jobs, Q_RANGE;
- The probabilities that a given node has large, medium or small emergency demand,
  $PROB\_LARGE$ and $PROB\_MEDIUM$ and $PROB\_SMALL$;
- The range of how many 'emergency points' nodes with large demand can have,
  $P\_POINTS\_RANGE\_LARGE$;
- The range of how many 'emergency points' nodes with medium demand can have,
  $P\_POINTS\_RANGE\_MEDIUM$;
- The range of how many 'emergency points' nodes with small demand can have,
  $P\_POINTS\_RANGE\_SMALL$;
- The allowed number of $ATTEMPTS$ to generate a feasible instance.

Given these parameters, an instance of MRP is generated using Algorithm 13.

The instances of class I_1 were generated with the following parameters:

$|A| = 3, \quad |J| = 2|A| + 1, \quad |V| = 20,$ $\quad G\_CHOP = 300,$
$\omega = 16, \quad ATTEMPTS = 100,$ $\quad SCALE = 1000,$
$MIN\_DIST = 1,$ $\quad Q\_RANGE = \{0, 1, 2\},$
$P\_POINTS\_RANGE\_LARGE = \{100, \ldots, 300\},$ $\quad PROB\_LARGE = 0.15,$
$P\_POINTS\_RANGE\_MEDIUM = \{10, \ldots, 50\},$ $\quad PROB\_MEDIUM = 0.30,$
$P\_POINTS\_RANGE\_SMALL = \{0, \ldots, 5\},$ $\quad PROB\_SMALL = 0.55.$

The class I_5 instances were generated with identical parameters, except $|A| = 4$ and $|V| = 100$.

As mentioned in Table 2.3, the classes I_1 and I_5 contain instances that are 'productive' and 'sparse'. The 'dense' instances were created from the 'sparse' instances by recomputing the node adjacencies for $G\_CHOP = 600$. The 'unproductive' classes were created from the 'productive' classes by deleting jobs until $|J| = |A| - 1$. We chose to set $|J|$ equal to either $2|A| + 1$ or $|A| - 1$, rather than $2|A|$ or $|A|$, to break symmetry and introduce the additional challenge of assigning unequal amounts of jobs to agents.

ALGORITHM 13: Generate an instance of MRP

1: Set $attempt = 0$;
2: Set $attempt := attempt + 1$. If $attempt > ATTEMPTS$, QUIT;
3: Create $|V| - |J|$ nodes with uniformly random integer X and Y coordinates between 0 and $SCALE$;
4: Create the other $|J|$ nodes with uniformly random integer X and Y coordinates between 0 and $SCALE$. At each of these nodes, place one job, thus setting $L : J \to V$. For each job, pick its processing time uniformly randomly from Q_RANGE;
5: Check if none of the nodes are closer than MIN_DIST to each other in Euclidean norm (truncated to two decimals). If there exist nodes that are too close together, go to Step 2;
6: Determine neighbour sets V_v by checking, for each pair of nodes (u, v), whether or not their Euclidean distance (truncated to two decimals) is under G_CHOP;
7: Check if the graph implied by V_v is connected, by picking an arbitrary node and checking neighbours in a width-first search. This width-first search terminates before finding all nodes if and only if the graph is not connected. If the graph is not connected, go to Step 2;
8: Check if the triangle inequality still holds after truncating Euclidean distances to two decimals, by checking for each $u, w \in V$ whether or not $C_{uw} \leq \min_{v \in V}\{C_{uv} + C_{vw}\}$. If there exists a pair where this inequality does not hold, go to Step 2;
9: Denote *depot* the centremost non-job node, that is, the non-job node with smallest Euclidean distance to $(0.5 \cdot SCALE, 0.5 \cdot SCALE)$;
10: Determine the distance matrix between the job nodes and *depot* using the Floyd-Warshall algorithm [31];
11: Check whether this instance of MRP admits a feasible solution, by checking whether the DVRP over the jobs and *depot* has a feasible solution, for example by solving the MILP describing the DVRP but with objective function 0. If the DVRP is infeasible, go to Step 2;
12: Set C_{uv} equal to the Euclidean distance between each pair of nodes (u, v);
13: For each non-job, non-depot node v, first determine whether v has a large, medium or small emergency demand against probabilities $PROB\_LARGE$, $PROB\_MEDIUM$ and $PROB\_SMALL$. Then, uniformly randomly draw a number of 'emergency points' from $P\_POINTS\_RANGE\_LARGE$ or $P\_POINTS\_RANGE\_MEDIUM$ or $P\_POINTS\_RANGE\_SMALL$.
14: Set P_v for all $v \in V_P$ by normalising until $\sum_{v \in V_P} P_v = 1$;
15: Create $A = \{a_1, \ldots, a_{|A|}\}$, set $S_a = depot$ and $W_a = depot$ and RETURN this instance.