Heuristic approaches for the Periodic Multiple Maintenance Person Problem

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor in Philosophy

Joshua Alcock

June 2023

Abstract

In the UK there is a large number of analytical laboratory companies that perform quality control testing on environmental samples, such as water control or ecological surveys. Each of these companies employs a fleet of Samplers to carry out this work, who may collectively gather more than 100,000 samples annually. This volume of vehicle travel accounts for several hundred thousand tons of CO_2 emissions every year, and a significant carbon footprint. A scheduler is responsible for developing a schedule for this workforce that provides an ordered list of locations for each sampler to visit. A scheduler in each of these industries has a number of challenges in common: 1) the need to visit a large number of locations on a regular basis, based on need (e.g. repair), governmental legislation (quality control), or supply chain logistics; 2) plan daily schedules that result in a subset of locations being visited; 3) comply with work directives that limit the time a technician or employee is traveling and servicing a set of locations in a given work day; and 4) accommodate a fleet of of employees that typically originate from a number of different geographical locations in a given area.

Furthermore, it is the difficulty in planning for the revisiting requirements which, if not managed intelligently, can result in high costs (both in terms of fuel and time) due to a lack of optimality. However, this optimality needs to be traded off with the need for pragmatic and fast solutions that can may need to be re-planned due to uncertainties in the workforce, such as vehicle failures, traffic vagaries, and illness. Thus, the challenge is to investigate a set of problems based on the well known Travelling Salesperson problem, whilst taking into account multiple agents, tour cost limits, and a revisiting requirement. The aim of this thesis is therefore to investigate one of these revisiting requirements, that of having a maximum number of days between visits. We examine the this problem, which we call the Multiple Periodic Maintenance Person People, by systematically breaking the problem down. We first examine the sub problem of scheduling a single maintenance person over multiple days, then we examine scheduling for multiple maintenance people over one day. Finally we bring the work for these sub problems together to schedule for multiple maintenance people over multiple days.

Acknowledgements

I would first like to thank my supervisors Terry Payne and Prudence Wong. Terry was one of the original motivators for me to choose a career in academia, after attending his mobile computing module and seeing a role that I believe I am well suited for. Prudence has often been incredibly patient, carefully steering me through the sometimes tortuous process of learning to think more critically about my work. I would like to thank my family, who have put up with my excited and sometimes frustrated rants about the intricacies of organising fleets of vehicles. I would like to thank my partner Yu Ping who has been endlessly supportive, encouraging me to work hard and change my habits even when the motivation wasn't there.

Thanks goes out to my research colleagues within the department; most notably to Thomas Carroll and Louwe Kuijer. To Tom for kindly explaining what I should expect for each step of the process and maintaining an almost impossibly positive outlook when discussing my work. To Louwe for sometimes giving me hard truths that have helped me improve as a person, as well as the long political and philosophical rants that helped me construct arguments. To the Liverpool Hung Gar Kung Fu Association for allowing me to blow off some steam and keep myself in shape with Lion and Dragon dancing. To Sam Goddard for his insightful discussion about the realities of working in the software development industry, and to the team at Northern Ireland Water for allowing me to shadow some of their employees and pester them with my questions. Finally to all of my former students, who helped ground me when I wondered what I was doing all of this for.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the UK there are a large number of analytical laboratory companies that perform quality control testing on environmental samples; for example: water control, soil analysis, or ecological surveys. Each of these companies may collect more than 100,000 samples annually and this totals to several million samples per year. The collection of these samples is carried out by a fleet of vehicles and this produces a large carbon footprint of several hundred thousand tons of CO_2 emissions every year. Likewise, maintenance companies can be responsible for the repair of a given type of equipment in a designated geographic area, with repair work being carried out by a number of employees who are trained in the servicing of this particular equipment. The type of work varies between companies, with some carrying out safety critical quality control testing, such as the water safety board, and some carrying maintenance that is preventative in nature, for example examining the state of internet infrastructure. It is because of this that the requirements for regularly visiting a location can vary greatly depending on the industry. One form of revisiting requirement is that a certain number of visits must be made within a specific time frame, eg 10 times throughout a month. Another industry may have a set pattern of days that may be visited, such as visiting a location on a Monday and Friday, or Wednesday and Sunday.

A common form of revisiting frequency for quality control companies is that of having a maximum number of days between visits. For example if a location must be revisited within three days of it's last visit which was on Tuesday, then it must be revisited on either: Wednesday, Thursday, or Friday. Thus a challenge is to understand how scheduling techniques can factor in this types of revisiting requirement whilst still achieving the overall

goal of visiting disparate locations within an appropriate time frame. As the visiting and revisiting of these locations by a workforce can account for a significant volume of vehicle travel and therefore of CO_2 production per year, this is a significant problem worthy of academic attention. An additional layer of complexity is added to the problem with the organisation of a workforce. This adds the requirement that each technician only works within a maximum number of hours each day. The technicians are also often based from their homes, which are geographically spread out across the area under the purview of the company. They therefore travel from their home location to the various job sites and back at the end of work, which must be taken into account when scheduling the specific work for them. We call this problem of scheduling for a workforce over a given number of days, where the amount of time each individual can work is constrained by a maximum time limit, each individual is based and works from a home location, and each location must be visited within a maximum number of days from it's last visit, the *Multiple Periodic Maintenance Person Problem (MPMPP)*.

1.1 Motivating Scenario

To gain a deeper insight into the problem challenges, one can create a fictitious scenario that characterises a maintenance company and its typical operation[1]. A laboratory analytics company has a fleet of vehicles, with each vehicle corresponding to a *sampler*, who carry out sampling in an area they are responsible for. The samplers visit a set of locations called *requests* to carry out the water quality control tests at the site and this takes a finite duration depending on the request. A sampler starts and finishes each day at a unique home location; ie it's *depot*, and must visit a laboratory prior to returning to their depot. A sampler has a schedule which is produced ahead of time at the start of the year by a team of laboratory technicians, this process is carried out over the course of a month. Each day within this schedule is referred to as a *tour*. The schedules must adhere to a set of legal requirements relating to the working hours for each sampler and must also visit each request periodically depending on the request's *time window* (which is the maximum number of days since the previous visit that it must be visited within). A request location can also be visited after a dynamic event, such as a pipe breaking in the case of

[1]This characterisation was based on discussions with a UK based water authority and a laboratory sampling organisation that were jointly responsible for carrying out water sampling (i.e. visiting and testing the quality of water) across a specific geographic area in the UK.

water sampling, and are dealt with by the sampler closest to the request. These dynamic requests are only visited once and do not have to be revisited.

1.2 Related Work

The laboratory analytics company typically faces several challenges when scheduling the activities of their different samples, and thus may adopt different solutions to scheduling, where the aim is to find an optimal or sub-optimal order to undertake a set of tasks given additional parameters such as time related cost[68]. Several scheduling problems share similarities with our problem, such as the bus and train scheduling problem[10, 54], the bamboo garden trimming problem[29], the nurse rostering problem[25, 57, 9], and the traveling salesperson problem[24, 31].

The bus and train scheduling problems model having two or more main stations from which a vehicle will depart. This vehicle visits a number of smaller stations, where it stops for a short period of time before departing. A driver for one of these vehicles can only work for a set number of hours per day and must be relieved by another driver at the depot the driver started from. The aim is to reduce the amount of time between visits to a station given a number of drivers available. This problem shares a lot of similarities to ours as a planning horizon and working hours are taken into account so that a driver returns back to the station within the time limit. It does aim to minimise the time between visits to a location, however this does not constrain the maximum number of days between visits and therefore does not provide suitable solutions.

The bamboo garden trimming problem models the challenges faced by landscape gardeners in charge of a set of geographically disparate gardens. Bamboo grows by a constant speed throughout the year, and this speed may vary between plants. Each day a gardener travels between bamboo plants and cuts the plant back to ground level with the aim to minimise the maximum size of the plants. There are both continuous and fixed time variant of this problem. This problem accommodates a form of periodicity, however this is treated as a soft constraint with the aim being to minimise the maximum length of the bamboo instead of to adhere to a maximum number of days between visits.

For the nurse rostering problem there is a set of tasks to be carried out in a hospital that are to be assigned to members of a team of nurses. The aim is to produce a schedule that either maximises the overall patient satisfaction with wait times, or minimises the number of nurses required to complete all tasks. This problem deals with the assignment

of tasks within a time frame for multiple agents which is similar to our problem, however as these tasks do not reoccur there is no mechanism to accommodate the periodicity of our problem.

It is the traveling salesperson family of problems that most closely matches the problem encountered by the laboratory analytics company scenario described above. A salesman wishes to travel to a set of locations to sell their product however they wish to minimise the distance travelled in order to minimise the cost. The aim is to create a schedule referred to as a *tour* where the salesman starts and finishes at a designated point and each of the locations is visited only once. The traveling salesman problem and its variants are often modelled as a network where each vertex represents a location to be visited and each edge represents the shortest travel time between these locations. The triangle inequality holds for all of these graphs as the edge represents the shortest time to reach a location instead of the route to get there. There are several relevant variants of the TSP, however none exactly match the problem presented here. These are:

- Traveling salesman problems (TSP) [24, 59] - As described above with a single salesman aiming to visit all of the locations. This problem usually does not break the schedule into working hours over a set of days and does not revisit the locations. There are a significant number of different mechanisms that have been applied to this problem, and often these mechanisms are used in problems that are an extension of the TSP.

- Orienteering problem (OP) [92] - This models the problem faced in the sport of orienteering, where a single individual travels across a map in a limited amount of time visiting a set of locations. Each of these locations has a reward associated with it, and the goal of the individual is to plan and execute a route that maximises the reward. This problem takes into account the tour cost limit, however does not deal with the periodicity or multiple agents of our problem.

- Multiple traveling salesman problems (MTSP) [3, 56] - A variant of the TSP where there are multiple salesman that collectively visit all of the locations. This problem does account for the multiple agents required by the problem, however it does not deal with either the periodicity of the problem or the tour cost constraints.

- Vehicle routing problems (VRP) [84, 58, 23] - This is an extension of the MTSP where each vehicle has a product capacity and each of the locations has a product

demand. This problem shares the same issue as the MTSP where working hours and a planning horizon are not taken into account.

- Dynamic vehicle routing (DVRP) [74, 100] - A variant of VRP where requests can be added at any time step. These problems attempt to match a task with a schedule that would reduce the overall time for all schedules. These problems often do not modify an existing schedule and only deal with the dynamic requests, and does not account break the schedule into a set of tours over multiple days.

There are several additional constraints that can be added to these problems however not all of these modifiers are relevant to each of the problem types. These modifiers are:

- Planning horizon [15] - This is a number of days that must be planned for and this usually takes the form of a hard time limit on the length of each tour.
- Working hours [34] - This is a constraint that limits the length of a tour within a day. This means that instead of having one large tour that could span over several days, there are several shorter tours. This is different from a planning horizon as the number of hours worked per week can change based off regulations related to that industry. An example would be in the UK trucking industry it is possible to work two additional hours twice a week if a day off occurs within the next three days. Additional changes such as lunch breaks can also be taken into account.
- Multiple depots [12] - In situations where there are multiple salesman this adds that they may not all start from the same depot however they usually must still return to their corresponding starting depot.
- Time windows [26] - This is the period of time that a location can be visited within and this is specified in whatever time unit is used for the amount of time to travel between two locations.
- Periodicity [54] - This is where a location must be revisited regularly and can be considered an extension of time windows.

1.3 Research Aims and Contributions

In this thesis, the challenges associated with the *Multiple Periodic Maintenance Person Problem (MPMPP)* are addressed from various perspectives:

1. *When examining a variant of the MPMPP where only a single agent is examined, what mechanisms can be used in order to solve this problem optimally and heuristically, what are the limitations of these mechanisms?* This can conducted through an examination of both an integer programming approach and a comparative investigation of different heuristics for this problem, which we call the *Periodic Maintenance Person Problem (PMPP)*. This problem is addressed in Chapter 3, with the intention of investigating several questions; the first of which being: *Given the complexity of the problem, is an exact or heuristic algorithm more suitable for the PMPP?*. To address this we discuss the np-hard nature of the problem and formally define the problem in Section 3.2 and then discuss a suitable approach to the problem in Section 3.3. We then investigate the question: *What sort of mechanisms are required to accommodate the visit dependant revisiting requirement?*.

2. *What are the challenges involved when using graph partitioning when scheduling multiple tours in a single day?* In Chapter 4 this is investigated using a comparative empirical study of a number of different graph partitioning mechanisms, adapted to accommodate the constraints of the MTSP. This raises the question: *Can graph partitioning mechanisms be adapted to be suitable as an approach to the Multiple Traveling Salesperson Problem (MTSP)?*. Which is answered in Section 4.2.

3. *Can the use of graph partitioning be combined with heuristic approaches to the Periodic Maintenance Person Problem to find solutions to the Multiple Periodic Maintenance Person Problem (MPMPP)?*. The aim here was to model multiple benevolent cooperating agents determining tours over successive days for a shared set of service locations by adapting the heuristic mechanism and a graph partitioning approach. This problem is addressed in Chapter 5. The resulting empirical analysis was used to answer the question *How can we accommodate multiple agents with a revisiting constraint?*.

1.4 Thesis Outlines

In Chapter 2 we discuss the literature related to this topic. The major family of work examined is that of the Travelling Salesperson Problem (TSP), which is a common scheduling and operations research problem that deals with developing a schedule for a salesperson to visit a set of locations. This salesperson starts from a home location, called a *depot*, and

	Single agent		Multiple agents	
	Without tour cost limit	With tour cost limit	Without tour cost limit	With tour cost limit
Single day	TSP Assignment problem	OP	MTSP	
Multiple days	PTSP Pinwheel problem	PMPP		MPMPP

Table 1.1: A table describing the decomposition of the MPMPP based on the three variables discussed in Section 1.4

visits these locations exactly once before returning back to the depot in what is called a *tour* or path. The aim of this problem is to minimise the total cost of travelling between all of these locations, whether that is measured in time or distance. This can be characterised as a scheduling problem instead of a routing problem as the major concern is the order in which the locations are visited instead of the specific route to travel between any two locations. This problem has a number of natural extensions, such as adding the requirement that a particular location is visited within an allotted time in the day, or modelling a vehicle carrying and delivering a finite amount of a product to each of these locations, with each having it's own demands for it.

In order to examine the MPMPP we take a systematic approach, examining aspects of the problem in order to inform work on the problem as a whole. We identified three major features of the problem, which are: a *tour cost limit*, the number of agents, and the *planning horizon*. A tour cost limit refers to the maximum amount of time that a sampler can spend during a day visiting locations. The number of agents refers to the number of samplers being planned for, and can either be for a single agent or multiple agents. The final variable is the planning horizon, which refers to the number of concurrent days that are being scheduled for, and can either be for a single day or multiple days. We take what we believe is a novel approach to decomposing our problem, by taking a common approach for all of our solutions and examining the effect that these additional features have on the problem and what modifications are required by the solution. We have started with the previously discussed TSP, assuming that the: single agent, without a tour cost limit, and a single day; variant encompasses all of the TSP's features. In Table 1.1 we outline the combinations of these features.

The *assignment problem* describes the problem of scheduling for a number of agents

to complete a series of tasks, with there being a cost that may vary between agents for completing the task [6]. A special case of this problem can exist with a single agent, and with the cost of each task changing to reflect the cost of travelling to that location from the current tour. This can also be viewed as the standard TSP. For the variant of the problem with: a single agent, on a single day, with a tour cost limit, this can be viewed as a homogeneous OP. For the variant of the problem with: A single agent, multiple days, and without a tour cost limit, the problem can be modelled as either a PTSP or Pinwheel problem. For the PTSP, using the maximum number of days revisiting constraint matches with the problem presented by this cell. The *Pinwheel problem* models the problem of having the ability to carry out a single task per day and a series of problems that have a minimum number of days before they must be repeated [11]. The aim of the problem is to create a sequence that can be repeated continuously without breaking this minimum number of days constraint. A special case of this problem could be modelled, with a maximum number of days revisiting constraint added and the minimum number of days constraint being 1. Multiple tasks could be handled on the same day and the cost varies depending on the tasks visited that day. The *Periodic Maintenance Person Problem (PMPP)* is a special case of the MPMPP where only a single agent is being scheduled for.

In Chapter 3 we examine all cells that are related to a single agent, specifically the: TSP, OP, PTSP, and PMPP problems within those cells. An average case empirical analysis is conducted for each of these sub-problems in order to find the modification required of a solution for the TSP. We determine that a heuristic algorithm called *Successive Augmentation* is the most suitable for adaptation to solve these problems. We examine this mechanism's performance for the TSP in Section 3.4 in order to establish a baseline. We then examine the modifications required to separately accommodate the tour cost limit and the multiple days revisiting constraint in sections 3.5 and 3.6 respectively. Finally we combine these modifications together, and discuss what modifications are required to accommodate both of these features together in Section 3.7 for the PMPP.

In Chapter 4 the problem of scheduling a multiple maintenance workers for a single day is examined. A popular extension of the TSP is that of the Multiple Travelling Salesperson Problem (MTSP). This problem adds multiple salespeople, who together visit each of the locations once to create a set of distinct tours. There are a variety of methods that can be taken to this problem, for example using a market based approach trading the ownership of various locations to other salespeople. Another is to model evolutionary processes like that of an ant colony in order to generate a set of tours.

An approach that has received little attention is that of dividing the problem space into a set of distinct sub problems. A MTSP can be modelled as a graph, with each of the locations being modelled as a vertex and the edges between each of these vertices having a weight that is equal to the cost of travelling between these vertices. This graph can therefore be divided into a series of distinct subgraphs, with each of the subgraphs then being treated as a TSP. This problem shares some similarities with Graph Partitioning (GP). GP divides a graph into a series of distinct subgraphs, typically aiming to minimise the total edge cost of all of the edges in between these subgraphs. This means that there is not a direct connection between the two problems, however some methods intended for GP may be applicable for the MTSP with some modifications.

As this problem can be modelled as an MTSP, we explore a number of different graph partitioning approaches and augment the work of Vandermeulen et al. [94], which adopts the use of graph partitioning for the MTSP by using the average length of a subgraph as a proxy for that subgraph's optimal tour cost. We examine if there are other graph partitioning mechanisms that can be adapted for this purpose, with specific references to common geometric approaches to the problem. In Chapter 5 we examine the MPMPP, scheduling for multiple maintenance workers over a multiple day period. This is a combination of the work from Chapter 3 and 4. Finally in Chapter 6 we discuss how successfully these questions have been addressed and what future work could be carried out in this field.

Chapter 2

Related Work

During this thesis the major family of work discussed is the Travelling Salesperson Problem (TSP) [4]. The TSP is one of the most popular combinatorial optimisation problems, which models a situation in which a single salesman must visit a set of cities exactly once each, with the aim of producing the cheapest Hamiltonian circuit or *tour* [78]. There is an additional variant of the TSP that instead aims to find the most expensive tour, which is referred to as the Maximal TSP Maximal tour cost [2]. Due to the variety of real world applications of the the TSP it has received a significant amount of attention. These applications range from the directly applicable scheduling of delivery drivers, to the ordering of embroidery stitching [4]. The TSP is however an NP-hard problem, and therefore a significant amount of attention has been given to developing both the exact and heuristic algorithms.

In this section we discuss the various aspects of the TSP that have been explored in the literature. Work for the TSP can be broadly separated into the two categories of Exact and Heuristic solutions. In Section 2.1 we present an overview of the of the literature related to the exact algorithms for the TSP. In Section 2.2 we discuss the general heuristic approaches taken to the TSP, as well as a common framework that is often used when decomposing heuristics for the TSP family of problems. Each of these heuristics can be split into stages, with each of these sections being classifiable in one of 4 categories [35, 93]. As in this thesis we are interested in generating a set of tours, we provide additional attention to the categories that are relevant to this in Section 2.2.1.

The addition of a tour cost limit to the TSP has been explored in the *Orienteering Problem (OP)* [14], which results in changing the objective of the problem from generating

the shortest Hamiltonian circuit to a combination of vertex selection and the shortest Hamiltonian circuit generation given the resulting subset of vertices. In Section 2.3 we discuss the most popular heuristics for the OP, as identified by Vansteenwegen et al. [95]. The *Periodic Travelling Salesperson Problem (PTSP)* [66] models a variant of the TSP with a revisiting constraint. Specifically, the PTSP models a single agent operating over a set of m "days" within a defined *planning horizon* with the aim of each day visiting a subset of possible locations before returning to a fixed start position. We discuss this further in Section 2.4.

In Section 2.5 we discuss the *Multiple Tour Maximum Collection Problem with Time-Dependant Rewards and a Rolling Horizon Framework (MTMCP-TDR-RHF)* [90]. This is similar to PMPP in that it considers both a tour cost limit constraint and a revisiting frequency, however the later is treated as a soft constraint. In this problem, each location is assigned a reward that varies between location and between days, the aim of the problem is to produce a schedule that maximises the reward while keeping each tour within the tour cost limit. We discuss some additional variants of the TSP in Section 2.6, discussing the Nurse Rostering problem. Finally in Section 2.7 we discuss a popular extension of the TSP that deals with the addition of multiple agents, which is called the Multiple Travelling Salesperson Problem (MTSP).

2.1 Exact approaches to the TSP

The most popular exact approach to the TSP is *linear integer programming* [81], with the most common of those being the *dual simplex method* which is linear algebra based algorithm [96]. Laporte and Nobert [53] produced two algorithms which are referred to as the *assignment-based* solutions, which incorporate a method called *Gomory cuts* [37]. Gomory cuts reduce the problem space and therefore reduce the computational speed required to solve a problem instance, and therefore increases the problem instance sizes that can be solved. Another popular exact technique is *branch and bound* which uses a tree structure to explore the problem space with the initial state of the problem as the root node [55]. From this root vertex, each of the possible selections that can be made are branches. The problem is first explored in a depth first fashion, resulting in an initial solution, and this is referred to as the initial lower bound. Vertices on this tree that were not selected for that initial solution are examined, and if the cost for selecting that vertex would exceed the current lower bound then those vertices are pruned. The tree is then

explored in a bottom-first fashion until there does not exist a vertex that has not either been pruned or explored. Christofides et al. developed two solutions, called *tree based* and *flow based*, that estimate the lower bound for each vertex using row and column reduction, and follow the traditional branch and bound method [18].

A combination of these two methods is called *Lagrangean relaxation with branch and bound* [43]. This approach divides the constraints into hard and soft constraints where the hard constraints are moved to the objective function, with the categorisation of the constraints varying by solution. Ali and Kennington developed a solution that applies Lagrangean relaxation to degree constraints and to each of the tours and has shown close to optimal results for small problem sizes [1]. Gravish and Srikanth [30] developed another algorithm that uses a minimal spanning tree to generate a lower bound and then applies Lagrangean relaxation to improve that the accuracy of that value. This solution had success with solving large scale problems with 500 cities and 10 salesman and is considered to be one of the fastest exact solutions. A final exact approach is *Quasi-assignment* which uses a branch and bound technique to relax the subtour elimination constraint, thus ensuring that the solution can be solved in polynomial time [38]. A subtour elimination constraint examines if there is a cycle in the tour that is disconnected from the main tour, or is a cycle that does not consist of all vertices in the tour. A solution was developed by Gromicho et al. [38] that uses an additive bounding procedure to produce a lower bound which may improve accuracy up to 10% better than the traditional methods.

2.2 Heuristic approaches to the TSP

There exists a common framework that is often used when decomposing TSP heuristic algorithms into a collection of stages [35, 93]. These 4 stages can be characterised as:

1. **Tour construction stage** - where a valid tour is constructed but without any further optimisation [48, 33].

2. **Tour improvement stage** - involving a meta-heuristic to improve an existing tour [77].

3. **Composite stage** - corresponding to a combination of stages 1 and 2 [93].

4. **Tour combination stage** - where multiple tours are generated using the prior stages, and then combined using evolutionary methods [94].

Until recently only stages 1-3 were used as part of this framework, however Vandermeulen et al. [94] observed that evolutionary mechanisms do not fit into the existing stages and proposed the 4th stage. Each TSP heuristic algorithm must have either the tour construction (stage 1) or composite (stage 3) stage to produce the initial tour, which we call *tour generation*. The resulting initial tour can then be used as the input for a tour improvement stage (stage 2) in order to reduce the cost of that tour, and multiple tour improvement stages may be carried out. Multiple tours, generated as the results of stages 1-3 can be combined in stage 4 using a generating mechanism.

2.2.1 Construction stage

Huang and Yu [45] identified three general mechanisms for tour generation, including: *space-partitioning based*, *edge based*, and *node based* mechanisms. Space-partitioning based mechanisms can be decomposed into three steps. The first step is to partition the space covered by the graph into a number of separate segments. The second step involves the production of a Hamiltonian circuit for the subset of vertices in each of the segments. The third and final step is to link all of the Hamiltonian circuits together to form a tour that covers all of the vertices. Two prominent space partitioning based algorithms are used: the *Strip* algorithm [89], and the *Space filling curve* algorithm [69].

The Strip algorithm works by splitting the problem space into k vertical strips of equal width, each of the vertical strips are then split into k horizontal strips of equal height [78]. The strips are combined from the top vertically downwards for each strip. Once the vertical strips each have a Hamiltonian circuit, they are combined left to right until a tour is created. The Space filling curve algorithm divides the space into 4 square segments, each of which are then subdivided into 4 smaller square segments, this subdivision continues until there is only a single vertex in each segment [63]. The process of combining Hamiltonian circuits is then repeated in the reverse order of division.

Edge-based mechanisms work by selecting edges in a tour, or rearrange edges in a structure such as a tree until a tour is made. Common mechanisms in this paradigm include: the *Greedy algorithm* [5], the *Christofides algorithm* [19], and the *Savings algorithm* [20]. The Greedy algorithm starts by selecting the cheapest edge in the graph. The next cheapest edge that does not form a cycle in the tour is selected, and this process is repeated iteratively until all vertices have been visited, the cheapest edge that forms a complete tour is selected. Christofides algorithm is the most common of the three as it has one of

the best known approximation ratios for a heuristic algorithm of 1.5 [19]. The algorithm has has three stages:

1. The construction of the minimum-weight spanning tree and a minimum-weight perfect matching on the vertices of odd degree in the tree.
2. The construction of a Eulerian cycle using the tree.
3. The traversal of the Eulerian cycle, whereby the vertices that have already been visited are skipped.

The Savings algorithm starts with a loop from the depot to each of the vertices. The cost of combining each combination of loops is evaluated, the cheapest combination is then made and the process is repeated until a complete tour is made.

Node based mechanisms can broadly be divided into two categories: *nearest neighbour algorithms* [4], and *successive augmentation* algorithms [48]. The standard nearest neighbour algorithm has the following structure:

1. Select the first vertex based on some metric, append it to a tour with the initial starting vertex.
2. Find the shortest edge from the last added vertex to a currently unvisited vertex, append it to the tour.
3. Repeat step 2 until all vertices have been visited.
4. Append the initial starting node at the end of the tour.

A successive augmentation algorithm is similar to the nearest neighbour algorithm, in that it also starts with a *partial tour* consisting of a non-empty subset of vertices in a Hamiltonian Circuit [45]. Additional vertices are then *selected*, and then added in some location within the tour (a process known as *expansion*). This process is then repeated until all of the vertices are included in the tour. The nearest neighbour algorithm can therefore be characterised as a successive augmentation algorithm, whereby the selection criteria involves determining a new vertex with the shortest distance to the penultimate vertex within the candidate tour. Thus each of the node based mechanisms can be considered to use the mechanism of successive augmentation as a genetic heuristic framework. A comprehensive description of the selection and expansion metrics for each of the procedures

is given in Table 2.1. The approximation ratios, if known, and the time complexities are given in Table 2.2.

The most intuitive approach is to simply start at a location and continuously travel to the next nearest unvisited location, and this is modelled by the *Nearest Neighbour procedure* (AKA next closest city) [5]. This however can lead to 'forgotten locations', as it has been observed that locations skipped over during the tour building process are filled in at the end, often leading to large increases in the tour cost [78]. An extension to the Nearest Neighbour procedure is to add the nearest unvisited vertex to either the start of the tour or the end of the tour, which is called the *Double Ended Nearest Neighbour* (AKA Double Sided Nearest Neighbour) [78, 39]. When reformulating the Double Ended Nearest Neighbour as a successive augmentation procedure, the nearest vertex to the second vertex and the second to last vertex are considered, with the closest of the two selected and inserted at the respective position. Another approach that can be considered an extension of the Nearest Neighbour procedure takes into account the effect on the lower bound of selecting a vertex, called *Dynamic weighting* [70]. This takes inspiration from the Hungarian method, using the cost of the edge to a vertex and the cost increase on the lower bound of selecting that vertex [51].

Rosenkrantz et al. expanded on the Nearest Neighbour procedure by selecting the unvisited vertex that is closest to any vertex in the partial tour, this is called the *Nearest Addition procedure* (AKA closest insertion) [82]. A novel expansion criteria is proposed, whereby the cheapest position on either side of the vertex in the partial tour is selected. For example, given the nearest selection criteria vertex i is selected, vertex j in the partial tour is the vertex that i is closest to. If the vertex before j in the the partial tour is g and the vertex after is k, then a comparison is made between the costs of inserting the vertex between g and j or j and k. The cost of insertion is cost of added the new vertex in-between the vertices in the partial tour and removing the existing edge. More specifically the difference between $cost(g,i)+cost(i,j)-cost(g,j)$, and $cost(j,i)+cost(i,k)-cost(i,k)$, where $cost(i,j)$ refers to the edge cost of travelling between vertices i and j. Rosenkrantz et al. also provided an alternative expansion criteria with the *Nearest Insertion procedure* [82]. Instead of determining which position on either side of the vertex in the partial tour to insert into, each position within the partial tour is considered.

Rosenkrantz et al. identified that selecting the unvisited vertex that farthest away from a vertex in the partial tour produces tours with a low total cost, which is called the *Farthest Insertion procedure* [83]. This is because this method creates a general outline

Procedure name	**Selection metric**	**Expansion metric**
Nearest neighbour [4]	The unvisited vertex with the cheapest edge from the second to last vertex in the partial tour	The second to last position
Double sided nearest neighbour [78]	The unvisited vertex with the cheapest edge from either the second or second to last vertex in the partial tour	At the second or second to last position, depending on the corresponding cheapest edge
Dynamic weighting [70]	The unvisited vertex with the cheapest lower bound when selecting it at the second to last position in the partial tour	The second to last position
Nearest addition [82]	The unvisited vertex that has the cheapest edge from a vertex in the partial tour	At either position on the side of the vertex in the partial tour with the cheapest edge
Nearest insertion [82]	The unvisited vertex that has the cheapest edge from a vertex in the partial tour	At the position in the partial tour that minimises the additional cost
Farthest insertion [83]	The unvisited vertex that has the largest edge from a vertex in the partial tour	At the position in the partial tour that minimises the additional cost
Farthest insertion (MinMax) [78]	The unvisited node where the most expensive edge to a partial tour node is minimal	At the position in the partial tour that minimises the additional cost
Farthest insertion (MaxMin) [78]	The unvisited node where the cheapest edge to a partial tour node is maximal	At the position in the partial tour that minimises the additional cost
Cheapest insertion [49]	The unvisited vertex which has the minimal additional cost at any point in the partial tour	At the position in the partial tour that minimises the additional cost
Cheapest Ratio	The unvisited vertex which has the minimal additional cost at any point in the partial tour	At the position in the partial tour with the minimum ratio between the cost of the additional edges over the replaced edge
Largest insertion [93]	The unvisited vertex with the most expensive additional cost	At the position in the partial tour that minimises the additional cost
Difference insertion [76]	The unvisited vertex with the largest difference between the cheapest and second cheapest insertions	At the position in the partial tour that minimises the additional cost
Loss [98]	The unvisited vertex with the largest different between the additional edge cost of inserting it at the cheapest and second cheapest positions	At the position in the partial tour that minimises the additional cost
Smallest sum insertion [78]	The unvisited vertex with smallest sum of distances to the partial tour vertices, equivalent of choosing the node with the minimal average distance to the tour nodes	At the position in the partial tour that minimises the additional cost
Largest sum insertion [78]	The unvisited vertex with largest sum of distances to the partial tour vertices, equivalent of choosing the node with the maximal average distance to the tour nodes	At the position in the partial tour that minimises the additional cost
Random insertion [82]	A random unvisited vertex	At the position in the partial tour that minimises the additional cost
Greatest angle insertion [62]	The unvisited vertex where the angle between the two edges added to insert the vertex in the partial tour is maximal	At the corresponding position from selection
Most eccentric ellipse [62]	An ellipse is drawn around each pair of vertices, using the vertices as the foci. The unvisited vertex that is the first to encounter the edge of any one of the ellipses	At the position between the two vertices corresponding to the ellipse

Table 2.1: A description of the different selection and expansion criteria of the various successive augmentation procedures

Procedure name	Approximation ratio	Time complexity
Nearest neighbour [4]	1/2Logn + 1/2 [83]	$O(n^2)$ [4]
Double sided nearest neighbour [78]	Unknown	$O(n^2)$ [78]
Dynamic weighting [70]	Unknown	$O(n2n^2)$ [70]
Nearest addition [83]	2 [82]	$O(n^2)$ [82]
Nearest insertion [83]	2 [82]	$O(n^2)$ [82]
Farthest insertion [83]	2 [35]	$O(n^2)$ [83]
Farthest Insertion (MinMax) [78]	Unknown	$O(n^2)$ [78]
Farthest Insertion (MaxMin) [78]	Unknown	$O(n^2)$ [78]
Cheapest insertion [49]	2 [82]	$O(n^2log_n)$ [82]
Cheapest ratio [35]	Unknown	$O(n^2log_n)$ [35]
Largest insertion [93]	Unknown	$O(n^2log_n)$ [93]
Difference insertion [76]	Unknown	$O(n^2)$ [46]
Loss [98]	Unknown	$O(n^2)$ [98]
Smallest sum insertion [78]	Unknown	$O(n^2)$ [78]
Largest sum insertion [78]	Unknown	$O(n^2)$ [78]
Random Insertion [83]	2 ln(n) + 0.16 [82]	$O(n^2)$ [82]
Greatest angle insertion [62]	Unknown	$O(n^2)$ [62]
Most eccentric ellipse [62]	Unknown	$O(n^2)$ [62]

Table 2.2: A description of the approximation ratio and time complexities for the various successive augmentation procedures

of a tour and then fills in the remaining nodes at more optimal positions due to having more information about the final tour. The selection criteria uses the edge cost in the same way as the Nearest Insertion selection criteria. Reinelt expanded on the Farthest Insertion procedure by selecting the unvisited vertex with the largest minimum edge to a vertex in the partial tour with the *Farthest Insertion procedure (MaxMin)* [78]. Which is to say each unvisited vertex has an associated cost, which is the minimum edge cost to any vertex in the partial tour. The opposite is also proposed with by selecting the unvisited vertex with the smallest maximum edge to a vertex in the partial tour with the *Farthest Insertion procedure (MinMax)* [78].

An extension of inserting a vertex at the position that minimises the additional cost, is to use that as a selection metric as in the *Cheapest Insertion procedure* [49]. This is the most common successive augmentation heuristic due to it's ease of implementation as the selection and expansion criteria are the same. The expansion criteria however can be modified, Golden et al. identified that the ratio between the additional edges and the replaced edge can be used as an expansion metric [35]. This was discussed as a procedure

that uses a convex hull as an initial partial tour and the procedure itself was not named, so we have called it the *Cheapest Ratio procedure*. The *Largest Insertion procedure* was described by Ursani and Corne as a combination of the Farthest Insertion and Cheapest Insertion procedure, by selecting the vertex with the most expensive additional cost and inserting it at the position that minimises the additional cost [93].

The successive augmentation heuristic selects an unvisited vertex that is locally optimal given some selection metric. Raymond attempted to avoid that issue by taking inspiration from chess and using a selection metric that plans ahead, with the *Difference Insertion procedure* [76]. This is done by selecting the unvisited vertex with the greatest difference between best and second best selection criteria costs. Webb modified that approach by only considering the difference between the added edges between the cheapest and second cheapest positions with the *Loss procedure* [98]. Reinelt identified that multiple edges can be used instead of a single edge as the selection metric *Smallest Sum Insertion procedure* and *Largest Sum Insertion procedure* [78]. The former takes a similar approach to Nearest Insertion by selecting the unvisited vertex with the smallest total edge cost to all vertices in the partial tour, with the later taking the opposite approach. This is the equivalent to the average distance of each unvisited vertex to all vertices in the partial tour.

Rosenkrantz et al. proposed an additional procedure, called the *Random Insertion procedure* AKA (Arbitrary Insertion) [82]. This procedure selects a random vertex that is not currently in the partial tour and insert it at a position that minimises the additional time. Norback and Love propose two procedures based on geometric methods for solving the TSP. The first is *Greatest Angle Insertion*, which selects the vertex where the angle between the additional edges of two consecutive vertices in the partial tour is greatest. The second is *Most eccentric ellipse*, which creates an ellipse around each consecutive pair of vertices in the partial tour using the vertices as foci. The ellipse is then expanded until an unvisited vertex bisects with the edge of the ellipse, this vertex is then selected for insertion inbetween the vertices that were the foci for the ellipse.

The initial partial tour criteria is not described by a successive augmentation procedure, however as demonstrated by: Hock [46], Ursani & Corne [93], and Huang & Yu [45], there can be a measurable effect between different criteria. The different criteria documented in the literature are:

- Specific initial vertex with a self loop - Often vertices can be ordered in a specific way, such as having the depot as the first vertex. In this situation this initial partial

tour can be used as the starting point for the tour [5].

- Arbitrary initial vertex with a self loop - If the order of the vertices is not important, then the selection of a random vertex is often carried out [78] [83].
- An arbitrary pair of vertices - This produces two edges that may be modified, and therefore the insertion metric can select different positions [33].
- An arbitrary vertex with a second vertex using a given metric - A pair of initial vertices produces this way aim to have the benefit of that metric, such as using farthest Insertion as it tends to produce good quality tours [64].
- A pair of vertices with the cheapest 2 edge cycle cost - This metric intends to find a partial tour with two vertices that has the smallest additional cost [42].
- Convex hull - This metric was inspired by the observation by Eilon et al. that the order of vertices in a convex hull is similar to that of the optimal tour for a euclidean problem [35].

It was identified by Hock that the different initial partial tour criteria do not have a noticeable effect on the final tour cost, with the exception of the Convex Hull metric [46]. Golden et al. identified that this is likely due to the similarity between the order of vertices in the convex hull, and the order of vertices in the optimal TSP tour [35]. This however only holds true for TSP problems where euclidean distances are used for the edge weights.

2.3 Orienteering problem

The Orienteering problem (OP) can be viewed as a combination of the Knapsack problem and the TSP [95]. Also known as the selective TSP, the maximum collection problem, or the bank robber problem, the OP models the problem of an agent starting and ending at some fixed location, and visiting a subset of possible locations within a fixed time limit [14]. As each location has a reward associated with it, the objective function is to maximise the reward whilst returning to the initial position within the time limit. By assuming homogeneous rewards, the objective function effectively becomes one of maximising the number of locations visited within a time limit. Five state of the art heuristic approaches are identified by Vansteenwegen et al. [95].

Tsiligirides [92] describes two algorithms that solve the OP, the first being the *S-algorithm* which is stochastic and the second being the *D-algorithm* which is deterministic. The S-algorithm starts with a partial tour of the depot node with a self loop, then a Monte Carlo mechanism is employed to determine which unvisited vertex is inserted into the partial tour. Each selection is then made based using a probability generated from the ratio between the reward over the cost of travelling to that vertex. The selected vertex is inserted into the second to last position, and this process is repeated until a vertex there does not exist a selection that would not bring the tour cost over the tour cost limit. The D-algorithm is the same process, however only the vertex with the largest ratio is selected. These two algorithms can be described as a modified Nearest Neighbour procedure.

Golden et al. [36] proposes a centre of gravity based algorithm. A modified *Cheapest Insertion Procedure* is used to produce an initial tour. Three factors are used for selection: the reward, distance from the centre of gravity, and distances from two ellipse points called foci. An equation uses these factors to generate a cost for insertion, with the minimal cost vertex being used for selection. The selected vertex is inserted at the position in the partial tour with the smallest additional cost. This is repeated until a further insertion would increase the tour cost over the tour cost limit.

Ramesh and Brown [75] propose a four step heuristic. A tour is generated using a modified cheapest insertion procedure, with the selection criteria being the ratio between the reward over the additional cost for each vertex. The 2-opt and 3-opt algorithms are used to reduce the cost of the tour. The same modified cheapest insertion procedure is then ran on the resulting tour to determine if additional vertices can be added to the tour. The process repeats from the second step until the tour is not modified.

Chao et al. [14] proposes a five step heuristic, the tour is generated using a modified cheapest insertion. An ellipse is drawn around the centre of the problem, and it's radius is reduced until it contains only two vertices, which are used as the initial partial tour. The ellipse is then expanded, with each vertex being added at the position in the partial tour with the minimum additional cost. This procedure is carried out until all vertices have been added to the tour, ignoring both the reward and total cost. Vertices with the largest additional tour cost are pruned from the tour until the cost is less than the tour cost limit. A two point exchange procedure is carried out and the 2-opt algorithm. A specified number of vertices are removed from the tour that have a high cost to reward ratio, and the algorithm is repeated from the second step. This procedure repeats until two iterations procedure the same tour and remove the same vertices. The tour without

the removed vertices is used as the final tour.

Gendreau et al. [32] presents a Tabu search based heuristic where the initial tour is generated using the Ramesh and Brown's cheapest insertion procedure [75]. The Tabu search is then applied to the resulting tour, relaxing the tour cost limit in order to find potentially cheaper tours. For each of the algorithms examined, the tours are generated using a successive augmentation procedure, with a majority of those procedures a modified cheapest insertion procedure, with the exception being the nearest neighbour procedure of Tsiligirides [92]. While the OP maps reasonably well to our problem, as it accounts for the tour cost limit, it does not however account for the periodicity of the MPP.

2.4 Periodic Travelling Salesperson problem

The Periodic Travelling Salesperson Problem (PTSP) models a single salesperson who, over an m day planning horizon, travels each day from a depot to visit at least one of the n locations before returning to the depot [66]. Two different types of revisiting frequency are considered in the literature: *Visit cardinality* and *Fixed visit schedule*. For the visit cardinality form of frequency, each location should be visited a fixed number of times within the planning horizon, with each visit occurring on different days (e.g. visit twice in a every ten day period). Each of the visits must be on different days, for example if location i must be visited 3 times within a planning horizon of 5 days, then it may be visited on any subset of 3 days. This could for example be the: 1st day, 3rd day, and 5th day; or on the: 3rd day, 4th day, and 5th day. In contrast the fixed visit schedule mandates specific days in which a location should be visited. For example location i may have two visit combinations: the 1st day, 2nd day, and 3rd day; or the 2nd day, 4th day, and the 5th day. The visit combination must be used in its entirety, and visiting on a subset of a combination is not permitted. Each day must have an associated tour, with a tour being a Hamiltonian circuit that visits that day's subset of locations, starting and ending at the depot. The goal is to produce a set of tours, called a schedule, which minimises the total distance travelled while still adhering to the revisiting constraint.

The first type of revisiting frequency is the the default when discussing the PTSP, and was first outlined by Christofides and Beasley [17]. Their algorithm has three phases, with the first phase creating a set of tours by using a modified cheapest insertion procedure to generate an initial set of tours. This procedure start with m partial tours, with each having an instance of the depot and a self-loop. A pool of vertices is generated, which

includes multiple instances of each vertex, given the number of times that vertex must be visited. The cost of inserting each vertex at each position is evaluated across each of the tours. The second phase uses the 2-opt procedure to reduce the cost of the tours, and the third phase exchanges locations between tours as long as the total distance for the two effected tours is reduced. Chao et al. presents an algorithm that starts integer programming based solution, then uses a novel location moving procedure and 2-opt to reduce the total tour cost [13]. Cordeau et al. developed an algorithm that starts with an initial solution, generated using the GENI tour generation procedure, then improves that initial solution with a modified tabu search [22]. Paletta proposed an algorithm that starts with the same modified cheapest insertion procedure as Christofides and Beasley [17], and then uses a novel location exchange procedure to ensure no tour is empty and reduce the total cost of the tours. Hemmelmayr et al. presents an algorithm that generates the initial set of tour using a modified savings algorithm. The CROSS variable neighbourhood search procedure is then used to exchange locations between tours, which is then improved upon using the 3-opt procedure to reduce tour costs [44].

The second form of revisiting frequency is referred to as the Multi Period travelling salesperson problem (MPTSP) [65]. Paletta proposed two algorithms, the first used a modified cheapest insertion procedure where the cost for each potential insertion is the ratio between the additional edge weight cost and the number of days from the previous visit [65]. The second algorithm uses a similar concept but instead uses the ratio between farthest insertion's selection criteria cost and the number of days from the previous visit. Polacek et al. describe a solution that uses a modified nearest neighbour procedure to generate the initial tours that are not necessarily feasible. A variable neighbourhood search is then used to improve the total cost of all tours [71]. A majority of these heuristics use a successive augmentation procedure to generate the initial set of tours, with a majority of those being the cheapest insertion or nearest neighbour procedures.

2.5 Multiple Tour Maximum Collection Problem with Time-Dependant Rewards and a Rolling Horizon Framework

The *Multiple Tour Maximum Collection Problem with Time-Dependant Rewards and a Rolling Horizon Framework (MTMCP-TDR-RHF)*, is an extension of the OP with soft maximum number of days revisiting constraint [90]. A heterogeneous reward is associated

with each location that varies both with respect to the locations and day visited, with an objective function to maximise the total reward of the schedule while satisfying the tour cost limit constraint for each tour. This is similar to our problem, in that the MTMCP-TDR-RHF has a tour cost limit for multiple tours over a planning horizon, however the revisiting constraint for each location while similar is treated as a soft constraint.

Tang et al. presents a heuristic algorithm for a simpler problem, the MTMCP-TDR, which is an extension of the OP whereby each location must be visited exactly once over a planning horizon. In a similar way to the MTMCP-TDR-RHF, the reward for visiting each location varies between locations and varies between days for a location. The authors provide a description of a modified randomised cheapest insertion procedure with a novel tabu and neighbourhood search, and a description of how to extend this algorithm to accommodate the additional constraints of the MTMCP-TDR-RHF. The description given however, does not provide salient details that are required to replicate these modifications.

The first portion of the algorithm is the modified randomised cheapest insertion procedure which populates the initial set of tours. For each of the m days, ψ candidate tours are generated, each populated with the depot vertex and a random vertex. The same vertex can be used for different candidate tours. For each of these tours, the additional edge cost of inserting each vertex not currently in the tour is evaluated. The cost of each insertion is then assigned as the ratio between the additional edge cost and the reward for visiting that vertex on the day associated with the candidate tour. The insertion with the best ratio is selected and the insertion is made. This process repeats until there does not exist an insertion that can be made without exceeding the tour cost limit.

The candidate tour with the highest total reward is then selected, with the total edge weight used as the tie breaking condition. This candidate tour is added to the initial tours for it's respective day. Every remaining candidate tour that contains the any of the vertices from this newly selected candidate tour is removed from the candidate tour pool, and a new tour is generated in it's place until there are ψ candidate tours for each of the remaining days. The costs are then reevaluated and the selection process is repeated until a tour is selected for each of the m days.

A neighbourhood search procedure is then applied to each tour in the initial solution, relaxing the tour cost limit constraint and generating α candidate tours for each day. These new candidate tours may not include the same vertices that had previously been selected, and can instead duplicate vertices from other tours. A tabu search procedure is applied to each of the candidate tours, which aims to bring each cost of that tour within the tour cost

limit. After β iterations the tabu search, if the cost of the tour is above the tour cost limit, then a new candidate tour is generated and the tabu search repeated until there are α valid candidate tours for each day. The candidate tour with the highest reward is selected and added to the final solution for the associated day. In a similar way to the first half of the algorithm, the tours that share vertices are removed and α candidate tours are generated for each of the remaining days. This process is repeated until a complete solution is found.

A brief description of how this procedure can be adapted for the MTMCP-TDR-RHF is provided by the authors; however crucial parts of the description are missing. In this adapted algorithm, the reward mechanism is used to encourage gaps between visits to a location. This is done by increasing the reward for days after a location's last selection in order to encourage larger gaps between selections. This means when a vertex is added to a tour, the reward for that vertex in the subsequent days is updated; however the equation used to generate the rewards is not defined. The reward could increase linearly for a location, e.g. if the time window for location i is 5, then the rewards could be: 1, 2, 3, 4, and 5. Alternatively, an exponential equation could be used. The way in which the reward should be handled for days outside of the time window is not specified, for example the reward for location i on day 6.

The modifications for the initial cheapest insertion candidate tour process are also not specified. A vertex may be present in a tour, only if it exists in an earlier tour within it's maximum revisit time window or if it is within it's maximum revisit time window from the start of the planning horizon. How this is accounted for by the candidate selection process is also not discussed. It also may not be possible, given a set of selections on certain days, to build a valid schedule and how that is determined in advance or a backtracking mechanism is not discussed. The modifications to the neighbourhood and tabu search procedures is also not specified and the same issue applies for this portion of the algorithm. Tang et al.'s algorithm for the MTMCP-TDR-RHF is therefore not replicable and a suitable solution for the PMPP does not exist in the literature.

2.6 Additional Travelling Salesperson Problem variants

There exists a variety of additional variants of the TSP, usually related to a specific real world problem and typically relate to scheduling for a pre-existing set of resources. An example of this is the bus or train scheduling problems, where there are two or more major stations from which a vehicle will depart and visit a number of smaller stations where they

will stop for a short period of time [10]. A driver for one of these vehicles can only work for a set number of hours per day and must be relieved by another driver at the depot the driver started from. The aim is to reduce the amount of time between visits to a station given a number of drivers available [54]. This problem accommodates the working hour nature of the PMPP, however does not accounting for the location scheduling as the routes for these vehicles are predetermined. Another example is the nurse rostering problem, where there is a group of tasks in a hospital that need completing and a set of nurses that need to be assigned tasks [25]. The aim is to produce a schedule that either maximises the overall patient satisfaction with wait times or minimises the number of nurses required to complete all tasks [57]. This problem accommodates the location scheduling, however the periodic nature of tasks are not.

2.7 Multiple Travelling Salesperson Problem

The multiple travelling salesperson problem (MTSP) extends the travelling salesperson problem (TSP), and aims to produce the cheapest set of distinct Hamiltonian circuits AKA *tours*, one for each of the salespeople, that visits each vertex exactly once. There are three elements of each MTSP configuration that determine exact what form of MTSP is being carried out. The three elements are: open or closed path, single or multiple depots, and MinSum or MinMax. The open or closed path describes if the agent must return to their corresponding depot. For the closed path MTSP, each agent leaves their depot and at the end of the tour returns to that depot. The open path MTSP by contrast has each agent leave their corresponding depot but the tour ends on a non-depot location instead of returning back to their depot. A vast majority of MTSP problems in the literature are closed path, with only a small minority addressing open path.

The single or multiple depots element describes if all of the agents start at the same location or from multiple disparate locations. For a single depot problem, each agent starts at the designated location. For a majority of multiple depot problems, each agent is assigned to a unique location as it's depot, however this is not necessarily the case a depot may be shared by two or more agents. The final element is the objective function, for which there are two popular variants: (1) MinSum, which minimises the total cost of all tours; and (2) MinMax, which minimises the cost of the longest tour. The MinSum MTSP is used in situations where the total cost of travel is the primary concern, whereas MinMax MTSP is used when minimising mission time is required.

The MTSP can be demonstrated as NP-hard, each agent visits a set of locations in the equivalent of a TSP tour, as the TSP is NP-hard the MTSP is therefore also NP-hard [99]. This NP-hardness has encouraged the development of a wide range of heuristic algorithms, for which Tolga Bektas identified five categories [3]. These are: Simple heuristics, Simulated annealing, Tabu search, Genetic algorithms, and Neural networks. For the simple heuristics, Robert Russell presented an extension to the Lin-Kernighan TSP heuristic, which uses the 2-opt and 3-opt [84]. This is for the single depot MinSum variant of the problem for symmetric graphs. Potvin et alṗresented a heuristic that works in a similar way with a k-exchange procedure for the same problem [72].

Simulated annealing is another mechanism which has been used, Song et alṗroposed an algorithm called Extended Simulated Annealing, replicating a thermodynamic system [88]. Ryan et alṗroposed an algorithm that generates an initial solution using linear integer programming, a tabu search procedure is then applied to the resulting set of tours [85]. Several genetic algorithms have been proposed, Zhang et alintroduced a mechanism that uses binary string encoding to represent which vertices are visited by each agent [104]. Yu et alàpplied a genetic mechanism, with both random and greedy initial tours for the MinSum and MinMax objective functions [103]. Tang et alàpplied a genetic mechanism to the MTSP modelled as a TSP with dummy nodes [91]. Neural network approaches to the MTSP have also been developed by Wacholder et al. and Modares et al. [97, 61]. Wacholder et alṗresent a MTSP transformed to a TSP with dummy nodes, using the Basic Differential Multiplier method. Modares et al's approach uses a Self Organising Feature Map for the MinMax single depot variant of the problem.

A recent approach that uses a graph partitioning mechanism has been introduced by Vandermuelen et al. [94]. *Graph partitioning* is the term describing the process of separating a graph into several distinct subgraph. The objective function is often to minimise the total cost of edges removed to split the graph into the subgraphs. This does not directly correspond to the MTSP, as a graph partitioned in a to minimise for the MinSum or MinMax objectives. As determining the cost of a given graph is an NP-hard problem, a method for applying graph partitioning is not immediately relevant. The authors present empirical testing that finds for euclidean TSP there is a strong correlation between the average length of a graph and the optimal tour cost. This is used for a variant of the MTSP called the Average Hamiltonian Partition Problem (AHPP), whereby the graph is partitioned for either the MinSum or MinMax objective function for the subgraphs. For MinMax the graph is partitioned into a set of m subgraphs with each subgraph corresponding to an

agent, and then each subgraph is then solved as a TSP.

The approach starts with a randomly generated partition with a depot in each of the subgraphs. Non-depot vertices are transferred and swapped between each pair of subgraphs if it minimises the maximum average length of the pair. This process is repeated until there are no more swaps and transfers between pairs of subgraphs that reduces the maximum average length. Once this process is complete, outlying vertices are then transferred if they contribute disproportionate to the average length. If vertices have been moved, then the swap and transfer procedure is repeated, followed by the outlying transfer. If these procedure don't change the subgraphs then the algorithm terminates. An exact TSP algorithm can then be applied to each of the subgraphs to produce a solution for the MTSP.

Chapter 3

Periodic Maintenance Person Problem

3.1 Introduction

Maintenance and inspection companies typically carry out regular visits to different clients for the purposes of repair (servicing) or quality assurance, with each client being located at a (potentially large) number of geographically disparate *service locations*. Visiting these service locations accounts for a significant volume of vehicle travel per year, but is required to ensure that equipment at these locations can continue to be used safely. Furthermore, there are often legal regulations that stipulate how regularly such visits should be carried out, as well as legislation on working hours that should not be exceeded per day (i.e. the combined time spent at each visit location within a day as well as the time spent travelling between visit locations and the depot). Therefore, scheduling mechanisms are required to determine what service locations are visited on what days, and in what order (bearing in mind that any schedule will start and end at some initial home location), such that the schedules satisfy a set of requirements. The first of these requirements, known as the *fixed visit schedule* ($\mathcal{R}_1$) represents the need to visit locations on specific days (e.g. every second and seventh day of some time period). Another is the *visit cardinality* ($\mathcal{R}_2$), which ensures that a location is visited a fixed number of times over some time period (e.g. a location is visited twice in every ten day period). A third represents the requirement of visiting a location within a fixed number of days since the previous visit (e.g. return within four days of the previous visit). This requirement, i.e. that of having

a visit dependant time window, or *revisiting requirement* ($\mathcal{R}_3$), is typically encountered in scenarios whereby some form of quality control is required; for example ensuring that Water Treatment plants are regularly checked to ensure that there is no build-up of contaminants. Thus, in this chapter, the *Periodic Maintenance Person Problem (PMPP)* is characterised as the problem of scheduling a sequence of *tours* for a single technician over a set period of days (i.e a *planning horizon* - $\mathcal{R}_4$), whereby each tour corresponds to a different day in the planning horizon. Each tour defines a schedule of visits to different locations that start and end at a predefined location, or *depot*, such that the technician can visit each of the locations and return to the depot within a specific time frame each day (as specified by the *tour cost limit* - $\mathcal{R}_5$), subject to satisfying the *revisiting requirement* ($\mathcal{R}_3$) whilst minimising the total time cost of the set of tours.

The PMPP is similar to the family of logistics problems based on the *Travelling Salesperson Problem (TSP)*. In its simplest form, the TSP models a single individual travelling to a set of geographically disparate locations before returning to their original location, whereby each location is visited exactly once, and the total travelling cost is minimised [4]. As discussed in Chapter 2 the TSP itself has been studied extensively, resulting in a variety of different algorithmic solutions with known complexities, as well as heuristic solutions that can determine approximate solutions with known bounds [35]. A number of variations on the TSP have also been explored [47], that vary the number of individuals, impose time constraints or revisiting periodicity, etc. The *Orienteering Problem (OP)* is an extension of the TSP which ensures that tours satisfy the *tour cost limit* ($\mathcal{R}_5$) requirement. Each non-depot location has an associated reward, with the solution aiming to maximise the reward for the tour while keeping within the tour cost limit [95]. However, this solution does not address the revisiting requirement ($\mathcal{R}_3$). The Periodic Travelling Salesperson Problem, or PTSP [66], is another variant that extends the traditional TSP problem by modeling an individual visiting a set of locations over a finite number of days; i.e. within the *planning horizon* ($\mathcal{R}_4$). This problem typically addresses problems that include the *visit cardinality* ($\mathcal{R}_2$) and *fixed visit schedule* ($\mathcal{R}_1$) requirements. Whilst there are some similarities with these problems and the use of a *time window*, they are not directly analogous and a mechanism designed for visit cardinalities or fixed visit schedules can't be used without adaptation for problems with a time window. The *Multiple Tour Maximum Collection Problem with Time-Dependant Rewards and a Rolling Horizon Framework (MTMCP-TDR-RHF)*, is similar to PMPP in that it considers both a tour cost limit requirement ($\mathcal{R}_5$) and a revisiting requirement ($\mathcal{R}_3$) [90]. However, the notion

$\mathcal{R}_1$	*Fixed Visit Schedule*	Locations should be visited on specific days
$\mathcal{R}_2$	*Visit Cardinality*	A location should be visited a fixed number of times over some time period
$\mathcal{R}_3$	*Revisiting Requirement*	A location should be revisited within a fixed number of days since the previous visit
$\mathcal{R}_4$	*Planning Horizon*	A set period of days over which a tour is planned
$\mathcal{R}_5$	*Tour Cost Limit*	A tour starts and ends at some depot and visits a subset of service locations within a specific time frame

Table 3.1: The different tour requirements used by the Maintenance Person Problem

of revisiting is only treated as a soft constraint in this work.

Thus, in this chapter, a heuristic algorithm for the PMPP is presented, that determines TSP-based tours associated with each day in the planning horizon ($\mathcal{R}_4$), whilst satisfying the tour cost limit requirement ($\mathcal{R}_5$) and revisiting requirement ($\mathcal{R}_3$). We examine how the *Successive Augmentation mechanism* (i.e. iteratively adding viable vertices to a seed tour to form a complete tour) can be augmented to accommodate: 1) either the tour cost limit ($\mathcal{R}_5$) or the revisiting requirement ($\mathcal{R}_3$); and 2) both of those requirements combined. We then perform a comparative evaluation of the resulting approach across a number of different successive augmentation procedures given a number of test cases.

The chapter is structured as follows; we first present a formalisation of the PMPP in Section 3.2. In Section 3.3 we discuss why a heuristic algorithm is suitable for the PMPP, and the qualities required in this algorithm. As the PMPP is an extension of the TSP, we chose to extend a mechanism used for the TSP in order to solve the PMPP and identify the most suitable mechanism as *Successive Augmentation*. We take a systematic approach to identifying the modifications required, by first examining the baseline performance using empirical testing in Section 3.4. We then examine the modifications required when adding a tour cost limit ($\mathcal{R}_5$) in Section 3.5, and adding a revisiting requirement ($\mathcal{R}_3$) in Section 3.6, and similarly empirically evaluate the performance. We combine these modifications and discuss additional modifications required to accommodate both features ($\mathcal{R}_3$, $\mathcal{R}_5$) in Section 3.7, with a final set of empirical evaluation. Finally in Section 3.8 we discuss how the resulting algorithm performs and consider future areas of research.

3.2 Formalisation of the PMPP

The problem is modelled using a complete directed graph $G = \langle V, E, w \rangle, w : E \to \mathbb{R}_+, V = L \cup D$ where the vertices V comprise a set of n geographic service locations of which a subset will be visited in a tour π (defined below), and the singleton D represents the *depot* where each tour both starts and ends. An edge $(i, j) \in E, i \neq j$ connects two vertices, such that $t_{i,j} = w(i, j)$ represents the shortest *travel time* between two locations. We assume that the triangle inequality holds, where $t_{i,j} \leq t_{i,p} + t_{p,j}, i, j, p \in V$. Furthermore, each service location $i \in L$ has a service time s_i which represents the amount of time an agent must spend at that service location. Thus, a *tour* $\pi = \{v_d, \ldots, v_i, v_{i+1}, \ldots, v_d\}, v_d \in D, v_i, v_{i+1} \in L$ is a path traversing a subset of service locations that starts and ends at the depot and minimises the time taken to leave and return to the depot such that the time taken to complete a tour π in one day is less than or equal to the tour time limit Q (i.e. the maximum amount of time each day that a tour may take). The planning horizon $H \subset \mathbb{Z}^+$ is the strictly increasing set of m days over which a sequence of tours are planned (i.e. before any tour is repeated) where each element $[1 \leq k \leq m]$ corresponds to a day where the tour π_k takes place.

The *time window* $r_i \in \mathbb{Z}^+, i \in L$ represents the maximum number of days between visits to a service location i, and prior to planning, we assume that each location has been visited the day before the first day in the planning horizon. Visits at service locations within the planning horizon are represented as a set of binary values $Z = \{z_{i,k}\}, i \in V, k \in H, z_{i,k} \in \{0, 1\}$, where $z_{i,k} = 1$ if i is visited on day k. Likewise, the roster of tours across the planning horizon is represented as a set of binary values $Y = \{y_k\}, k \in H, y_k \in \{0, 1\}$, where $y_k = 1$ if an agent carries out a tour on day k and 0 otherwise. Finally, the edges within a tour $\pi_k, k \in H$ are represented as a set of binary values $X = \{x_{i,j,k}\}, i, j \in V, x_{i,j,k} \in \{0, 1\}$ where $x_{i,j,k} = 1$ if $(i, j) \in \pi_k$ and 0 otherwise.

We define the problem as an integer program:

$$\min \sum_{i,j \in V} \sum_{k \in H} (t_{i,j} + s_j) x_{i,j,k} + \sum_{i \in L} \sum_{k \in H} t_{i,(n+1)} x_{i,(n+1),k} \tag{3.1}$$

subject to the following constraints:

$$\sum_{j \in L} x_{(n+1),j,k} = \sum_{i \in L} x_{i,(n+1),k} = y_k \qquad k \in H \tag{3.2}$$

$$\sum_{j \in V \setminus \{i\}} x_{j,i,k} = \sum_{j \in V \setminus \{i\}} x_{i,j,k} = z_{i,k} \quad i \in V; k \in H \tag{3.3}$$

$$\sum_{i \in V} \sum_{j \in L} (t_{i,j} + s_j) x_{i,j,k} + \sum_{i \in L} t_{i,(n+1)} x_{i,(n+1),k} \leq Q \quad k \in H \tag{3.4}$$

$$\sum_{p=k}^{k+r_i-1} \sum_{j \in V} x_{i,j,p} \geq 1 \quad i \in L; k \in H; k + r_i - 1 \leq m \tag{3.5}$$

Subtour elimination constraint

$$\sum_{i \in U} \sum_{j \in U} x_{i,j,k} < |U| \quad k \in H; O \in Z; p \in V; z_{p,k} = 1; U \subset O; |U| \geq 2 \tag{3.6}$$

$$x_{i,j,k} \in \{0, 1\} \quad i, j \in V; k \in H \tag{3.7}$$

$$y_k \in \{0, 1\} \quad k \in H \tag{3.8}$$

$$z_{i,k} \in \{0, 1\} \quad i \in V; k \in H \tag{3.9}$$

The objective function (Equation 3.1) minimises the total time cost for all of the tours, subject to constraints (3.2 - 3.9). Each of the constraints is summarised below, and a fuller description of each is given later in the chapter as part of a worked example. Constraint (3.2) states that if an agent leaves the depot then they must return to the depot, and also that a depot cannot be visited in the middle of a tour. Constraint (3.3) states that any service location that is visited must not be the final location within the tour (this complements constraint 3.2), whereas constraint 3.4 ensures that the total time taken for a tour should not exceed the time limit, where the total time taken is the sum of the total travel time and the total service time. Finally, constraint (3.5) ensures that each service location is revisited within the maximum time window that is defined for that location (note that an assumption is made that the location was visited the day before the planning horizon starts). The subtour elimination constraint (SEC) used for the tours is stated by constraint (3.6), which states a lower limit for the number of edges within any proper subset of a potential tour, such that there are at least two vertices corresponding to the start and end vertex (i.e. the depot). If the number of edges within a proper subset is equal to the number of vertices, then there is a cycle that does not include all vertices invalidates the tour. As this method of avoiding subtours is computationally expensive, it is often omitted from a formalisation and the resulting set of tours can be checked to

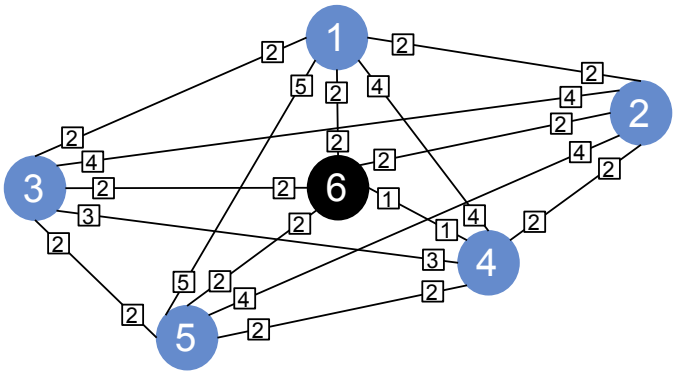


Figure 3.1: A visualisation of the example graph, where service locations are represented as the vertices in the graph $\{1, 2, 3, 4, 5\}$ and the vertex $\{6\}$ represents the depot vertex. Each edge is labeled with the time taken to travel between each vertex.

verify that the constraints are not violated. If a violation is detected, the integer program is re-executed with an additional constraint that specifically prohibits the invalid subtour [67]. The integrality requirements are given by constraints (3.7), (3.8) and (3.9).

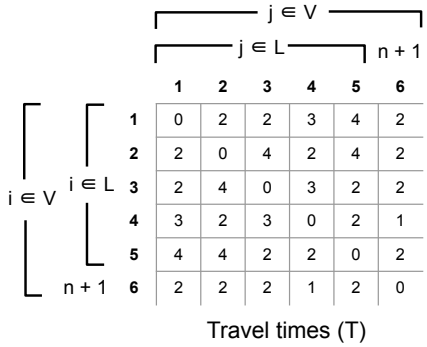
The intuition behind this integer program can be illustrated through a worked example of the periodic maintenance person problem (PMPP). We first define our problem using the terms given in the formalisation, and then provide two tours, both of which commence on day one of the time window (i.e. are scheduled for the 1st day), and therefore represent partial solutions. The first of these solutions represents a valid tour for day 1 which satisfies all of the constraints (Figure 3.3), whereas the second solution illustrates a case where the constraints 3.2 - 3.6 are violated, resulting in an invalid tour (Figure 3.4). The rationale for using a single example to illustrate the violation of all constraints (as opposed to multiple examples for each constraint) is one of expediency; the violation of any single constraint

Service location $i \in L$	**Service time** s_i	**Time window** r_i
1	3	1
2	2	2
3	2	2
4	3	1
5	1	2

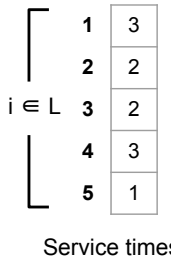
Table 3.2: The time required to service each of the locations, and their respective time windows

is still sufficient to result in an invalid tour. The use of the objective function as given in Equation 3.1 will then be demonstrated only for the valid tour.

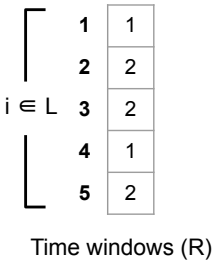
Assume that we have a scenario with $n = 5$ service locations (denoted $L = \{1, 2, 3, 4, 5\}$) that should be visited over a period of $m = 3$ days (such that the planning horizon is given as $H = \{1, 2, 3\}$). In addition, we have a single depot represented by the singleton vertex $D = \{6\}$ such that the total number of locations for the problem are given by the union of two sets $V = L \cup D = \{1, 2, 3, 4, 5, 6\}$. Furthermore, the time cost associated with travelling between any pair of vertices $i, j \in V$ may differ depending on the direction (i.e. the edge $t_{3,5} \neq t_{5,3}$). This can therefore be modelled as a complete directed graph with 6 vertices (i.e. 5 service locations and 1 depot), as visualised in Figure 3.1, where the labels of the edges represent the travel time cost associated with moving from a source vertex (adjacent to the labelled edge) to the destination vertex. Each of the vertices in the graph also has an associated *service time* s_i corresponding to the time that an agent has to remain at that given service location before moving on (to complete some task) and an associated *time window* r_i corresponding to the maximum number of days that could elapse before the agent should return to that service location to repeat its task. Table 3.2 lists the different service times s_i and time windows r_i for each $i \in L$.



(a) The travel times $t_{i,j}$ for each pair of service locations $i, j \in V$



(b) The service times s_i for each location $i \in L$

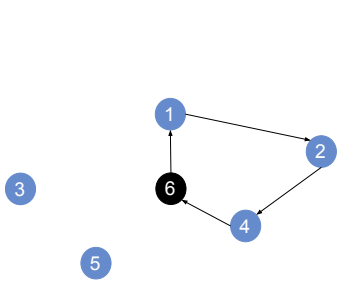


(c) The time window r_i for each location $i \in L$

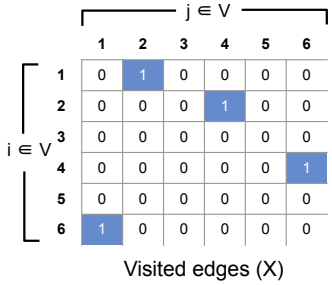
Figure 3.2: The matrix in Figure 3.2a represents the travel times $t_{i,j}$ between each pair of vertices in the example, whereas the vectors in Figures 3.2b and 3.2c represent the service times and time windows respectively.

To better illustrate the use of the objective function (Equation 3.1), the travel times, service times and time windows can also be represented pictorially, as illustrated in Figure

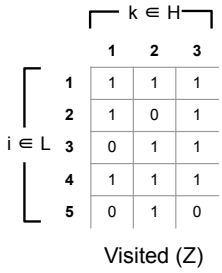
3.2. The travel times $t_{i,j}$ for each pair of service locations $i, j \in V$ are given as a matrix (illustrated in Figure 3.2a), with source locations i given as rows and destination locations j given as columns. Thus, the travel time $t_{i,j}$ between location $i = 3$ and location $j = 5$ is given by $t_{3,5} = 1$. The service time for each service location $i \in L$ is given as a vector in Figure 3.2b; for example the service time s_i for location $i = 3$ is given by $s_i = 2$. Likewise, the time window r_i for each service location $i \in L$ is given in Figure 3.2c; for example, the time window r_i for location $i = 3$ is $r_3 = 2$. Finally, as a tour is scheduled for each day, the roster is given as $Y = \{1, 1, 1\}$, and the total time taken for each tour should be less than or equal to the tour time limit $Q = 20$.



(a) A valid tour for day 1



(b) The visited edges X for valid tour 3.3a



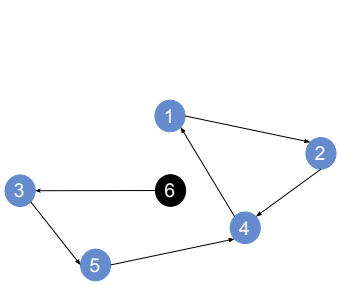
(c) The visited service locations Z for valid tour 3.3a

Figure 3.3: A collection of all figures relating to a **valid** tour for the example problem

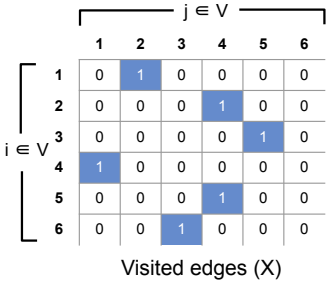
Figure 3.3a illustrates the resulting valid tour for day 1 of the planning horizon, where each vertex is visited in a strict order as given by the edge connectivity matrix in Figure 3.3b. In this solution, the tour for day 1 is defined as $\pi_1 = \{6, 1, 2, 4, 6\}$ with a total cost (including travel time and service times) of 15. Figure 3.3c shows the visit matrix, Z, where each column represents the vertices visited that day (given as a binary value where the value 1 states that a vertex in a given row is visited). Thus, the first column represents the vertices visited on day 1 as illustrated in Figure 3.3a[1]. This contrasts with the solution for an invalid tour (see Figure 3.4), which visits every service location on day 1 and includes a tour cycle without returning to the home depot, as well as having a minimum total cost (including travel time and service times) of 23, which exceeds the tour time limit $Q = 20$.

On the basis of this example, we can now examine the constraints of Equation 3.1 in

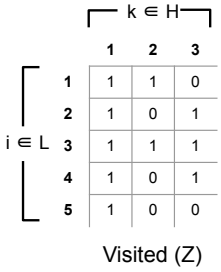
[1]Note that Figure 3.3c presents the visit schedule for the vertices for each of the days within the planning horizon $H = \{1, 2, 3\}$, and reflects the time window constraints for each location. For example, r_1 for service location $i = 1$ (Figure 3.2c) has a max time window of 1 and thus this location should be visited every day.



(a) An invalid tour for day 1



(b) The visited edges X for the invalid tour 3.4a



(c) The visited service locations Z for the invalid tour 3.4a

Figure 3.4: A collection of all figures relating to an **invalid** tour for the example problem

more depth. Recall that constraint 3.2 ensures that if an agent leaves the depot to begin a tour, then it must return to that same depot only at the end of the tour (i.e. the depot should not be visited at any point in the middle of a tour).

$$\sum_{j \in L} x_{(n+1),j,k} = \sum_{i \in L} x_{i,(n+1),k} = y_k \qquad k \in H \tag{3.2}$$

If a tour is scheduled for some day k, then this is represented by the corresponding roster value $y_k = 1$; otherwise $y_k = 0$ (i.e. no tour is scheduled for that day). Furthermore, the depot vertex should have at most 1 outgoing and incoming edge. The constraint has an equality between each of the three terms: the first, $\sum_{j \in L} x_{(n+1),j,1}$, returns the outgoing degree of the depot vertex; and the second, $\sum_{i \in L} x_{i,(n+1),1}$, returns the incoming degree. The third term ensures that if there is an equal number of incoming and outgoing edges to the depot, then this is reflected in the roster for that day.

For the valid tour there is only $\sum_{j \in L} x_{6,j,1} = 1$ outgoing degree and $\sum_{i \in L} x_{i,6,1} = 1$ incoming degree, both of which are equal to the roster value $y_1 = 1$. Therefore this constraint is held. In contrast, for the invalid tour, there is $\sum_{j \in L} x_{6,j,1} = 1$ outgoing degree, but $\sum_{i \in L} x_{i,6,1} = 0$ incoming degree. As these values are not equal to each other and both not equal to the roster value $y_1 = 1$, the constraint is violated and thus the resulting tour is invalid.

The next constraint (3.3) states that if a service location is visited within a tour then

it should only have a single incoming and a single outgoing edge.

$$\sum_{j \in V \setminus \{i\}} x_{j,i,k} = \sum_{j \in V \setminus \{i\}} x_{i,j,k} = z_{i,k} \quad i \in V; k \in H \tag{3.3}$$

This ensures that a tour does not terminate at a service location (as each tour *must* return to the depot) and that each service location is visited at most once during a tour. If a service location i is visited on day k then this is represented by $z_{i,k} = 1$; with the term $z_{i,k} = 0$ used to denote no visit to i on day k. Here, we have an equality between the two terms: the first term $\sum_{j \in V \setminus \{i\}} x_{j,i,k}$ returns the incoming degree whereas the second term $\sum_{j \in V \setminus \{i\}} x_{i,j,k}$ returns the outgoing degree for each service location i.

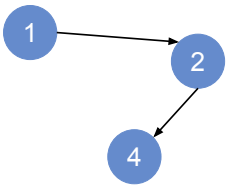
To illustrate this, observe service location $i = 4$ for day $k = 1$, for both the valid and invalid tour scenarios. For the valid tour, the incoming degree $\sum_{j \in V \setminus \{4\}} x_{j,4,1} = 1$ and outgoing degree $\sum_{j \in V \setminus \{4\}} x_{4,j,1} = 1$. Both of these degrees are equal to the visited value for that day, i.e. $z_{4,1} = 1$, and thus the constraint is satisfied. However, this is not the case for the invalid tour, where the incoming degree $\sum_{j \in V \setminus \{4\}} x_{j,4,1} = 2$ and the outgoing degree $\sum_{j \in V \setminus \{4\}} x_{4,j,k} = 1$, which violates the equality constraint.

The need to ensure that the total time taken for each tour does not exceed the tour time limit is represented by constraint 3.4. There are two terms in this constraint, the first corresponds to the total time taken travelling to the service locations and servicing those locations, whereas the second returns the time taken travelling from the last service location to the depot. The total of these two summations are checked to determine if the total time taken is less than the tour limit Q.

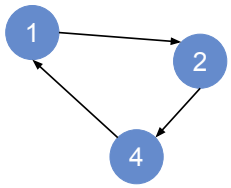
$$\sum_{i \in V} \sum_{j \in L} (t_{i,j} + s_j) x_{i,j,k} + \sum_{i \in L} t_{i,(n+1)} x_{i,(n+1),k} \leq Q \quad k \in H \tag{3.4}$$

For the valid tour on day $k = 1$, the time taken in travelling to the four service locations $\{1, 2, 4\}$ starting from the dept $\{6\}$ is given by $\sum_{i \in V} \sum_{j \in L} (t_{i,j} + s_j) x_{i,j,1} = 14$, and the time taken returning to the depot is given by $\sum_{i \in L} t_{i,6} x_{i,6,1} = 1$. As the total time taken $14 + 1 \leq 20$, this satisfies this constraint. However, for the invalid tour on day $k = 1$, the time taken travelling to service locations $\sum_{i \in V} \sum_{j \in L} (t_{i,j} + s_j) x_{i,j,1} = 24$ and the time taken returning to the depot $\sum_{i \in L} t_{i,6} x_{i,6,1} = 0$. As a result, the constraint is violated as the inequality does not hold; i.e. $24 + 0 \leq 20$ is false.

The next constraint (3.5) ensures that each service location is revisited within its time



(a) A visualisation of the subset for the valid tour given in Figure 3.3a



(b) A visualisation of the subset for the invalid tour given in Figure 3.4a

Figure 3.5: The subset $U = \{1, 2, 4\}$ of vertices for the valid (a) and invalid (b) tour, to illustrate constraint 3.6.

window:

$$\sum_{p=k}^{k+r_i-1} \sum_{j \in V} x_{i,j,p} \geq 1 \qquad i \in L; k \in H; k + r_i - 1 \leq m \tag{3.5}$$

Each service location i has an associated time window r_i (Figure 3.2c). For each service location, and for each day k, there is a time period extending r_i days into the future. Each of those time periods represents the maximum revisit period for that service location starting from day k. Within that time period there should be at least 1 visit to that service location. The time period should cover from day k to day $k + r_i - 1$.

For example, service location $i = 5$ has the time window $r_5 = 2$, meaning that this location should be visited at least every other day, if not on adjacent days. We can examine how this is used for the valid and invalid tour by looking at the visited Z values (Figures 3.3c and 3.4c respectively). There are 2 time periods that are covered; from $k = 1$ to $k = 2$, and from $k = 2$ to $k = 3$. For the valid tour, $\sum_{p=1}^{1+2-1} \sum_{j \in V} x_{5,j,p} = 1$ and $\sum_{p=2}^{2+2-1} \sum_{j \in V} x_{5,j,p} = 1$. In both cases, these satisfy the constraint (i.e. that the sum of the values is greater or equal to 1). However, for the invalid tour, we find that $\sum_{p=1}^{1+2-1} \sum_{j \in V} x_{5,j,p} = 1$ and $\sum_{p=2}^{2+2-1} \sum_{j \in V} x_{5,j,p} = 0$. Although the constraint for day 1 (i.e. the first time period) is satisfied, it is violated for day 2 (the second time period) where we have the expression $0 \geq 1$ which is false.

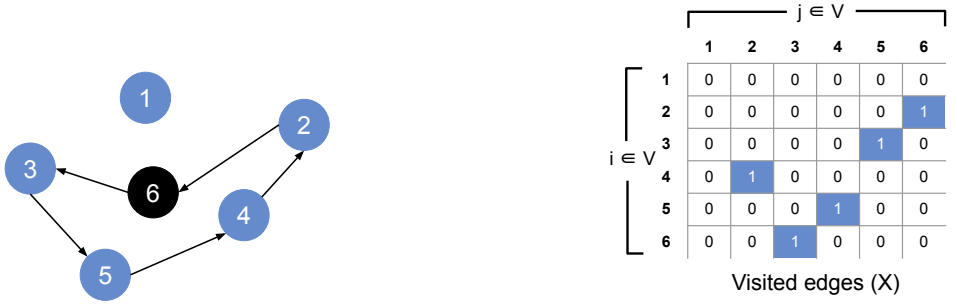
Constraint 3.6 ensures that there are no cycles present in any non-empty proper subset of the tour and therefore no subtours:

$$\sum_{i \in U} \sum_{j \in U} x_{i,j,k} < |U| \qquad k \in H; O \in Z; p \in V; z_{p,k} = 1; U \subset O; |U| \geq 2 \tag{3.6}$$

A cycle has the same number of edges and vertices. For each non-empty proper subset

U of the tour, the total number of edges $\sum_{i \in U} \sum_{j \in U} x_{i,j,k}$ is checked to determine if it is less than the number of vertices $|U|$. For this example we will use the subset $U = \{1, 2, 4\}$ for both the valid and invalid tours, with the number of vertices $|\{1, 2, 4\}| = 3$. This subset for the valid tour is visualised in Figure 3.5a; the corresponding subset for the invalid tour appears in Figure 3.5b. For the valid tour on day $k = 1$, the number of edges $\sum_{i \in U} \sum_{j \in U} x_{i,j,1} = 2$ and therefore as $2 < 3$ the constraint is satisfied. For the invalid tour on day $k = 1$, the number of edges $\sum_{i \in U} \sum_{j \in U} x_{i,j,1} = 3$. This violates the constraint as the expression $3 < 3$ is clearly false.

As there can be an exponential number of subsets for a given graph, it can be computationally expensive to implement this constraint. For this reason, *subtour relaxation* [67] is typically used as a method for reducing the impact of this constraint. The process works by solving the linear integer programming problem without any SECs, where each non-empty proper subset of the resulting solution is checked to determine if it violates the SEC. For each subset that violates the SEC the corresponding SEC is added and a new solution is generated, this process is repeated until the solution does not violate any SEC.



(a) The graph representation of a valid tour

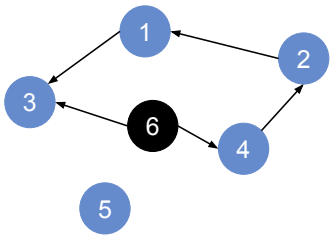
(b) The visited edges X for the valid tour

Figure 3.6: The valid tour for day $k = 2$ given the example problem from Figure 3.1

The objective function (Equation 3.1) returns the total time taken for all tours. It

Day(k)	$\sum_{i,j \in V} \sum_{k \in H} (t_{i,j} + s_j) x_{i,j,k}$	$\sum_{k \in H} t_{i,(n+1)} x_{i,(n+1),k}$	**Total tour time**
1	14	1	15
2	16	1	17
3	17	2	19

Table 3.3: A table that shows the total time cost for the valid tours given in Figures 3.3a, 3.6a, and 3.7a



(a) The graph representation of a valid tour

j ∈ V

i ∈ V	1	2	3	4	5	6
1	0	0	1	0	0	0
2	1	0	0	0	0	0
3	0	0	0	0	0	1
4	0	1	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	1	0	0

Visited edges (X)

(b) The visited edges X for the valid tour

Figure 3.7: The valid tour for day $k = 3$ given the example problem from Figure 3.1

works in a similar way to constraint 3.4 in that the final result is the sum of two terms, where the first term calculates the time travelling to and servicing the service locations $\sum_{i,j \in V} \sum_{k \in H} (t_{i,j} + s_j) x_{i,j,k}$, and the second determines the time taken to return to the depot $\sum_{k \in H} t_{i,(n+1)} x_{i,(n+1),k}$. The valid tour corresponds with the first day $k = 1$ of a valid solution, as illustrated in Figures 3.6 and 3.7. Figure 3.6b is a representation of visited edges X for day $k = 2$, with a visualisation given in Figure 3.6a. Figure 3.7b is a representation of visited edges X for day $k = 3$, with a visualisation given in Figure 3.7a. Table 3.3 shows the total time each tour takes with the total for all tours being $15 + 17 + 19 = 51$.

3.3 Tour generation mechanism for the PMPP

As the TSP is an NP-hard problem, extensions of the TSP (such as the Orienteering problem, Periodic Travelling Salesperson Problem) tend also to be NP-hard [47, 99]. In the special case of the PMPP where only a single day is considered, and each vertex must be visited, then the PMPP can be viewed as a traditional Travelling Salesperson Problem (TSP). As the TSP is NP-hard, our problem is therefore also NP-hard, and thus it is desirable to find a heuristic algorithm that can provide practical solutions for this set of problems. One method for developing a suitable algorithm for the PMPP is to modify a mechanism for the TSP to accommodate the additional features of the PMPP. As described in Chapter 2, a TSP heuristic algorithm can consist of a number of stages but requires a tour generation stage, which is responsible for generating an initial tour that may be used as a solution, or subsequently improved. Although a number of tour construction mechanisms

exist for the TSP, they are not directly suitable for use in providing solutions for the PMPP, as they do not take into account the two additional constraints that characterise the PMPP. These two characteristics are: 1) the cost limit for each tour ($\mathcal{R}_5$); and 2) the revisiting constraint for each of the service locations ($\mathcal{R}_3$).

It is not always possible to simply add a tour cost limit ($\mathcal{R}_5$) to a TSP, due to the fact that if an optimal tour has a cost greater than this limit, then the constraint is violated. For such scenarios, a desirable solution is one that selects a subset of vertices that can maximise some objective function, such as maximising the number of locations visited. However, for almost all tour generation mechanisms, an appropriate subset of vertices needs to be identified in order to determine a valid tour, such that the resulting solution does not exceed the tour cost limit. A possible solution to this is the *successive augmentation mechanism* whereby an initial tour is generated consisting of an initial vertex, before a new vertex is greedily added to the tour to improve the solution. This mechanism can be modified to terminate once the addition of any further vertices would result in the violation of the tour cost limit, and thus makes the modified successive augmentation mechanism suitable for the TSP with a tour cost limit ($\mathcal{R}_5$).

The inclusion of the visit dependant revisiting constraint ($\mathcal{R}_3$) to the TSP is also problematic due to the fact that the presence of vertices in earlier tours imposes constraints on the selection of vertices for latter tours. For example, if service location i has a time window of 3 ($r_i = 3$), and that service location is only visited on day 1, then it needs to be visited again within the following 3 days (i.e. on day 2, 3, or 4). The vertices that can be added to each of the days of a tour can be referred to as the **potential vertex insertions**. A possible method for adding such vertices to a partially computed tour is to augment the tour by updating the potential vertex insertions after each vertex is selected for a given tour. The successive augmentation mechanism can therefore also be used to determine which of the potential vertex insertions should be selected for a tour that will satisfy this constraint.

Using the successive augmentation mechanism poses a number of challenges that need to be resolved for it to be used:

1. *Initial partial tour* - What vertex or vertices are used for the initial partial tour?

2. *Selection* - Which unvisited vertex should be selected for insertion into the partial tour?

3. *Expansion* - Between which two consecutive vertices in the partial tour should the selected vertex be inserted?

A selection metric paired with an expansion metric is typically called a *successive augmentation procedure*, such that different *successive augmentation procedures* may be identified, with each consisting of a pairing of some selection criteria with an *expansion criteria*. In our investigation, we explore the use of eight procedures commonly discussed in the literature [76, 93, 45]. Each procedure, as well as their selection and expansion criteria, is listed in Table 3.4.

The name for each procedure is generally given as a concatenation of the selection criteria and the expansion criteria. For example the *Nearest* selection criteria identifies the unvisited vertex that has the lowest-cost edge from a vertex in the partial tour, whereas the *Insertion* expansion criteria inserts the selected vertex at the position between two consecutive vertices that minimises the additional tour cost. Thus, the name used for this combination is the *Nearest Insertion* procedure. We consider three possible expansion criteria, each of which selects a different insertion position, which corresponds to an edge between two vertices:

- *Addition* - If the selection criteria identifies a vertex in the partial tour, then the position in the partial tour before or after the selected partial tour vertex that minimises the additional tour cost is selected.

- *Insertion* - The position between two consecutive vertices that minimises the additional tour cost is selected.

- *Neighbour* - The position that is between the second to last vertex in the partial tour and the last vertex is selected.

For the selection criteria, there are six separate unique metrics studied. The *largest sum* selects a vertex with maximal sum of distances to all vertices in the partial tour, whereas the *smallest sum* selects the vertex with the minimal sum of distances. The *nearest* simply selects the vertex with the smallest (minimal) cost to one of the vertices in the partial tour, whereas the *farthest* selects the vertex with the largest (maximal) cost to one of the vertices in the partial tour. The *cheapest* selects the vertex that has the minimal additional cost at any point in the partial tour, and the *difference* selects the vertex with the largest difference between the cheapest and second cheapest insertions.

Procedure name	**Selection metric**	**Approximation ratio**	**Time complexity**
Nearest addition [82]	The unvisited vertex that has the cheapest edge from a vertex in the partial tour	2	$O(n^2)$
Nearest neighbour [4]	The unvisited vertex with the cheapest edge from the second to last vertex in the partial tour	$\frac{1}{2}\log n + \frac{1}{2}$	$O(n^2)$
Nearest insertion [82]	The unvisited vertex that has the cheapest edge from a vertex in the partial tour	2	$O(n^2)$
Smallest sum insertion [78]	The unvisited vertex with smallest sum of distances to the partial tour vertices, equivalent to choosing the node with the minimal average distance to the partial tour vertices	Unknown	$O(n^2)$
Largest sum insertion [78]	The unvisited vertex with largest sum of distances to the partial tour vertices, equivalent to choosing the vertex with the maximal average distance to the partial tour vertices	Unknown	$O(n^2)$
Farthest insertion [83]	The unvisited vertex with the most expensive edge to any vertex in the partial tour	2	$O(n^2)$
Cheapest insertion [49]	The unvisited vertex which has the minimal additional cost at any point in the partial tour	2	$O(n^2 \log n)$
Difference insertion [76]	The unvisited vertex with the largest difference between the cheapest and second cheapest insertions	Unknown	$O(n^2)$

Table 3.4: A description of a number of different Successive Augmentation Procedures

In order to examine what modifications would need to be made to the successive augmentation mechanism, we take a structured approach. We first examine how the mechanism currently performs for the TSP to establish a baseline for comparison. We do this through empirical evaluation in Section 3.4. Then we examine what modifications are required to deal with the TSP with each of the additional features separately, i.e. the TSP with a tour cost limit ($\mathcal{R}_5$) and the TSP with the revisiting constraint ($\mathcal{R}_3$). The problem that is closest to the TSP with a tour cost limit ($\mathcal{R}_5$) is referred to as the *Orienteering Problem (OP)*, and a discussion of the modification and the results of the empirical testing is given in Section 3.5. In a similar way, a TSP with a revisiting constraint ($\mathcal{R}_3$) is referred to as the *Periodic Travelling Salesperson Problem (PTSP)* and it's modification and results are presented in Section 3.6. Finally in Section 3.7 we present our proposed mechanism to solve the PMPP which, takes the modifications from Sections 3.5 and 3.6 and adds additional modifications to deal with the combination of both additional features.

3.4 A comparison of successive augmentation procedures for the Travelling salesperson problem

In order to evaluate the effectiveness of modifications to the successive augmentation mechanism, a baseline of performance is first required, which we establish with an average case empirical analysis of the selected successive augmentation procedures. These procedures were presented in Section 3.3, and further highlighted in Table 3.4. As the successive augmentation mechanism requires a selection and expansion criteria, grouped in the form of a successive augmentation procedure, the similarities and differences between these criteria can be used in order to identify why particular criteria outperform others in the context of the TSP. In this section we present the results of executing these procedures on two problem sets. The first of these is a synthetic problem set of complete euclidean graphs where the x and y coordinate of each vertex are random values. These graphs range from a size of 25 to 100 vertices, and for each problem size there are 1000 different problem instances. The graph size was varied in order to examine the performance scalability of the various criteria. The second problem set is based on real world topologies and are a subset of the TSPLIB problem library [79]. The subset chosen was undirected complete euclidean graphs with an edge weight type referred to in the library as 'EUC_2D', as it most closely matches our problem. The real world graphs vary in their number of vertices

from 51 to 18512. The optimal tour cost for the synthetic problem set was found, and the optimal tour cost for the real world data set is known and this information was used in order to evaluate the performance of the procedures in terms of the optimal tour cost.

This section is structured as fellows, first we provide a detailed description of the successive augmentation mechanisms in Section 3.4.1, with a discussion of how the mechanism as well as the various selection and expansion criteria may be implemented. A full description of the experimental methodology adopted by this study is given in Section 3.4.2, with subsections covering aspects of the methodology in greater detail. The explanation and justification of the graph structure used for both of the problem sets is discussed in Section 3.4.2, with a description of the graph sizes for both the artificial and real world problem sets is given in Section 3.4.2. The reasoning for the selection of a measuring metric called percentage tour cost increase is described in Section 3.4.2. Lastly for the experimental methodology in Section 3.4.2 we describe the technical setup, detailing the equipment and languages used. Finally we present the results of the empirical analysis and a discussion of those results in Section 3.4.3.

3.4.1 Description of the successive augmentation mechanism

The successive augmentation mechanism is a polynomial heuristic algorithm for the TSP. We provide an algorithmic description of this mechanism in Algorithm 1. The mechanism iteratively selects a currently unvisited vertex using some selection criteria and then inserts it at some position within a partial TSP tour given some expansion criteria. The algorithm takes in a set of all locations that must be visited and the depot location. For example, given a set of vertices $V = \{1, 2, 3, 4, 5\}$, then the last depot is specified as the depot $D = \{5\}$ and the remaining locations are locations to visit $L = \{1, 2, 3, 4\}$. On line 1, the algorithm instantiates an empty partial tour. A partial tour, starts at a depot location and visits a subset of non-depot vertices before returning to the depot. An empty partial tour therefore consists of the depot vertex and a self loop, for example $P = \{5, 5\}$. Other initial tours can be used, such as a convex hull of the problem instance graph, however we have not chosen that variant.

Between line 2 and 6, there is the main loop for the algorithm which encompasses the selection and expansion of the tour. This loop terminates once there no longer are vertices that need to be added to the tour, and therefore the tour visits all locations. On line 3, the vertex is selected using the selection criteria. Each selection criteria will have

a different implementation, however they all will iterate through each potential selection and determine the associated score. A situation may occur where two or more potential selections have the same score, and therefore a tie breaking condition is required in order to determine which of those are chosen. For our implementation we have chosen the earliest position in the set of vertices as the tie breaking condition, as it is unique to each vertex. For example if the selection of vertex 2 and 4 had the same score, then vertex 2 would be chosen as it has an earlier position in the set of vertices. Line 4 completes the selection process by removing the selected vertex from the list of locations yet to visit.

Line 5 completes the iteration, by selecting the position within the existing partial tour where the selected vertex will be inserted. The potential positions are after the first depot instance and before the second depot instance, in this way a tour can always be traversed at each iteration. The expansion criteria similarly assigns a score to each potential expansion and a tie breaking condition may also be required for this. For our implementation we have chosen to use the earlier position within the partial tour as the unique attribute. For example if the position 3 and 7 have the same expansion score, then position 3 would be selected.

Algorithm 1 Successive Augmentation Mechanism

Require: L, D
1: $P \leftarrow \{D, D\}$
2: **while** $L \neq \{\}$ **do**
3: Select vertex v from L using selection criteria
4: Remove v from L
5: Insert v into P using expansion criteria
6: **end while**

3.4.2 Experimental Methodology

In this section we describe the reasoning behind the experimental and implementation decisions for the average case empirical analysis of successive augmentation procedures for the TSP. The structure of the graphs used for the testing is described in Section 3.4.2. The problem sizes for the artificial problem set, and the real world problem instances are described in Section 3.4.2. Section 3.4.2 describes the metric used in order to evaluate the performance of the procedures. The metric used compares the tour cost for each problem instance against the optimal tour cost for that problem instance, therefore the optimal

tour cost must be found. The description of the optimal solver used and the reasoning for that is given in Section 3.4.2. Finally the technical details are given in Section 3.4.2.

Graph structure

The average case empirical analysis was carried out on two problem sets of complete euclidean undirected graphs, with the last vertex of each graph being designated as the depot. A valid tour in the context of the TSP is a Hamiltonian path; i.e. a path between two vertices on a given graph that visits each vertex exactly once. If a graph is incomplete, in that there exists a pair of vertices that are not directly connected with an edge, then when generating a Hamiltonian path a situation may occur whereby it is not possible to travel to an unvisited vertex without first travelling through a previously visited vertex. Although a backtracking mechanism could be used, where the last decision is reverted and another option selected to avoid this situation, a problem occurs in that the successive augmentation mechanism does not have a backtracking mechanism. Given a graph any permutation of the vertices are possible as output using the successive augmentation mechanism [78]. Given an incomplete graph has at least one permutation that would not produce a valid tour; complete graphs were instead chosen [40].

Although the original problem characterisation was given for a directed graph, for this set of evaluations, an undirected graph was chosen as a majority of the selected procedures were intended for the symmetric TSP, which in the context of the TSP refers to a undirected graph [78]. For an undirected graph, the cost of travelling from vertex i to vertex j ($t_{i,j}$) is the same as travelling from vertex j to i ($t_{j,i}$) [77]. The weight for each of the edges corresponds to the euclidean distances between two vertices, the triangle inequality can therefore be said to hold. This is because the cost of travelling from vertex i to vertex j ($t_{i,i}$) will be less than or equal to the cost of travelling from i to k and then to j.

A program called Concorde was used to obtain the optimal tour cost of each problem instance, as discussed more detail in Section 3.4.2, and as this solver uses a particular variant of the euclidean distance equation, this was used for the edge weights for the analysis [21]. This particular edge weight type is called 'EUC_2D' by Concorde, and returns an integer distance between two coordinates [79]. This edge weight type was selected as it had the most suitable selection of real world problem instances, as discussed further in Section 3.4.2. This variant of the euclidean equation uses the 'bankers rounding' method, whereby a value of 0.5 is rounded up and all others values are rounded to the nearest integer. For

example, a value of 1.3 becomes 1, a value of 1.5 becomes 2, and a value of 1.7 becomes 2. If we represent the X coordinate of the first point with a and the Y coordinate as b, then if we represent the X coordinate and Y coordinate of the second point as c and d respectively, then we can describe this variant of the euclidean equation as Equation 3.10.

$$round((c-a)^2 + (d-b)^2) \tag{3.10}$$

For the artificial problem set, each of the vertices has an X and Y coordinate, with both of these values being a uniform random real number in the range of 0 to 100. This range is arbitrary, and in previous empirical analyses various spacial sizes have been used such as 0 to 100 by Gendreau et al. and 0 and 1 by Norbeck and Love [32] [62]. The range used by Gendreau et al. was selected arbitrarily and is not linked to the range of vertices used. As the coordinates for the synthetic problem set use arbitrary distance units, the euclidean distance for this problem set also represent arbitrary distance units. This is also true for the real world problem set, with each individual problem instance representing either: miles, meters, etc and the units of the euclidean distance also representing that.

Problem instance sizes

For both of the problem sets the number of vertices is varied, in order to determine the effect of this variable on the performance of the various procedures. The synthetic problem set consists of 16 different graph sizes, starting with 25 vertices and increasing in increments of 5 vertices up to 100 vertices. 100 vertices was chosen as the maximum graph size as finding the optimal tour cost using the exact TSP solver Concorde has a time cost associated with it [21]. Given the equipment available this graph size was chosen, with a round value being used in order to present a range that divides evenly given an increment of 5. The minimum size of 25 was an arbitrary selection as it is a multiple of 5.

The most popular of the problem set libraries for the TSP is TSPLIB, with a variety of sub-libraries for variants of the TSP such as the Capacitated vehicle routing problem. Symmetric and asymmetric in the context of the TSP refer to undirected and directed graphs respectively [78]. As undirected complete euclidean graphs are the most suitable for this empirical analysis, the symmetric sub-library of problems was selected which has a total of 109 problem instances, that vary in size between 14 and 85900 vertices. These problem instances can be further grouped into set of problem instances based on their edge weight type, for which each problem instance may only have one type. These edge weight

types describe the method by which the edge weight should be generated for the problem instance.

Two criteria were identified as desirable for a real world problem set, which are that the edge weight type for the problem is the euclidean distance, and that the number of problem instances was of a sufficient size, which was arbitrarily determined to be 16 or more vertices as that is the size of the synthetic data set. The first group of problem instances is that with the edge weight type 'GEO', and refers to an equation that takes into account the curvature of the earth in order to accurately give the euclidean distance between two points. This edge weight type may differ significantly from the euclidean distance on an assumed flat plane, and only has 14 problem instances and was therefore deemed unfit for our purposes. The second group of problem instances has the edge weight type 'ATT', which refers to a special distance function that is used for 2 problem instances, and is therefore has a problem size too small for our purposes.

The 'EXPLICIT' edge weight type is the third group, and specifies the distance between each pair of vertices in the form of a table. These problems are not necessarily representative of an undirected graph, and the distance between two vertices may be different between the two directions. There are 11 problem instances and a majority have between 17 and 58 vertices. As this problem group is not symmetric and have a comparatively small number of vertices, this edge weight type was therefore not selected. The fourth group uses the 'CEIL_2D' edge weight type, which is a variant of the euclidean distance that returns an integer ceiling value for the distance. There is only however 4 instances that use this edge weight type which is not a significant enough number of problem instances for our purposes. The final edge weight type is 'EUC_2D', which is a variant of the euclidean distance equation that returns an integer value using banker's rounding. There are 78 problem instances that use this edge weight type, and we believe that this is a significant sample size and is therefore chosen. The euclidean distance method for the synthetic data set was modified after this selection in order to ensure both data sets are as similar as possible. A list of these instances is provided in Table A.1, which can be found in Appendix A.

Metric for comparison

In order to evaluate the average case performance of the selected successive augmentation procedures, the resulting tour costs must be compared using a comparison metric. As the

effect of varying the number of vertices is the variable being examined for the various procedures, the average performance of each procedure for each graph size should be found. There are three methods in the literature for comparing the tour cost performance of successive augmentation procedures. The first is to compare the tour costs directly, which simply requires the result of executing the procedure on a problem instance and does not require additional values such as the optimal tour cost. The tour cost between different problem instances can vary significantly, for example consider two problem instances with the same number of vertices. For the first instance all of the vertices are tightly clustered together and for the second they are spread out significantly far apart. The first instance would in general result in shorter tours than the second instance and therefore comparing the tour costs for these instances would not be directly analogous. This method is however suitable for comparing single problem instances, for example using a TSP problem instance library. This method is also suitable for use when computing the optimal tour cost would be too computationally expensive and therefore comparing this value would be the only suitable metric. An example of use of this method can be found in the work of T. C. Raymond, who examined the tour cost performance for their proposed method and presented the: worst, average, and best tour costs. Another example can be found in the work of of Gendreau et al. , who carried out an average case empirical analysis on their proposed successive augmentation procedure for problem sizes of 500 vertices which given their resources was too computationally complex to compute [32].

The second method is an extension of the first method, where the tour cost is instead expressed in relation to the optimal tour cost for a problem instance. This has the advantage of normalising the tour cost across difference problem instances and different problem sizes. For example, if a problem instance has an optimal tour cost of 100 and a heuristic algorithm tour cost of 120, then the difference is 20. For another problem instance, if the optimal tour cost is 1000 and the heuristic algorithm tour cost is 1200, then the difference is 200. In both of these examples, the heuristic tour cost is 1.2 times that of the optimal tour cost, therefore by presenting the difference as an increase over the optimal tour cost as a ratio of the optimal tour cost the problem sizes can be normalised. This does however require that the optimal tour cost is known, as finding this value is NP-hard and a successive augmentation mechanism runs in polynomial time then there will be a problem size that will be too computationally expensive to compute in a reasonable time [47, 99]. This method has been applied by Huang and Yu in their survey of tour construction mechanisms for the TSP, where this was the method used to compare the performance of the various

mechanisms. [45]

The final method is an extension of the second method, whereby the value is not expressed as a result but is instead given as a percentage over the optimal tour cost. For example, the ratio of 1.2 would be expressed as 20%, as in the heuristic tour cost is an additional 20% of the optimal tour cost. This is is expressing the same result as a ratio, with the same advantages and disadvantages, however it is subjectively more readable. This method has been used by a variety of authors, and is referred to by several different names. Glover et al. referred to this method as 'Average excess over optimal' [33], with Brest and Zerovnik instead calling this method 'Percent from optimal' [7]. Xiang et al. calls this method 'Percent excess' [101], finally with Ursani and Corne referring to it as 'Percentage difference from the optimal value' [93].

Optimal solver

As discussed in Section 3.4.2, several existing surveys of tour construction mechanisms have used the Percentage Tour Cost Increase (PTCI) as a comparison metric, and this is typically used on problem instances from TSPLIB where the optimal tour cost is known. For the artificial data set we have chosen to use this same metric in order to allow for comparisons to be more directly applicable between each problem instance in the real world problem set and the artificial problem set. In order to determine the optimal tour cost, an exact TSP algorithm must be used for which there are a wide variety in the literature [99]. A popular existing solution for this is the state-of-the-art exact TSP solver *Concorde* [21]. This is a freely available solver, and uses a Chained Lin Kernighan algorithm to generate an initial solution, which is then improved using the 3-opt procedure [87]. From this initial solution, the remaining problem space is explored using a branch and bound mechanism [48]. The Concorde solver takes in a '.tsp' file and outputs the result into the terminal. The terminal output was piped into a text file in order to store the results.

Technical setup

Two different machines were used for the empirical evaluation. The first machine used was a Windows 10 laptop with an Intel Core i7-8750H CPU @ 2.20GHz with 16GB of RAM. This machine was used in order to generate the graphs and execute the test frame for the empirical evaluation. The operating system was upgraded to Windows 11 in between the graph generation and empirical evaluation. The second machine used was an Ubuntu 20.04

Procedure name	Summary PTCI (%)	Standard deviation
Difference insertion	4.069	0.892
Largest sum insertion	8.499	2.997
Farthest insertion	8.696	2.877
Cheapest insertion	16.213	1.614
Nearest insertion	19.290	1.917
Smallest sum insertion	20.725	3.155
Nearest neighbour	23.308	1.678
Nearest addition	25.258	2.646

Table 3.5: A table showing the mean tour cost and standard deviation of the successive augmentation procedures, ranked from the smallest *Percentage Tour Cost Increase (PTCI)* to the largest for the artificial problem set

laptop with an Intel Celeron Dual-core T3300 CPU @ 2.00GHz with 2GB of RAM. The optimal tour costs were found using the Concorde TSP solver, and this was executed on the second machine.

The graph generation code was written using Python 3.7, with the co-ordinates for each vertex generated using a random number generator. The seed was generated from the current time (measured in milliseconds) when the code was executed. The graphs were saved in the '.tsp' file format [80], with the co-ordinates for the graph specified instead of the distance matrix and the edge type being specified as 'EUC_2D'. The test frame was also written in Python 3.7, and the tour and tour cost were saved for each of the procedures as a '.dat' file for each of the problem instances. A test frame for Concorde was written in Bash, and passes the '.tsp' file for each

3.4.3 Results and discussion

The results of the average case empirical analysis for the artificial problem set is presented in Figure 3.8, and the performance of each procedure is summarised in Table 3.5. Each point in Figure 3.8 represents the mean *Percentage Tour Cost Increase (PTCI)* across 1000 problem instances for the corresponding graph size. Table 3.5 presents the mean of these points for each of the procedures, which we will call the *summary PTCI*, and also presents the standard deviation of those points. The results for the real world problem set is presented in Figure 3.9, with the performance of each procedure summarised in Table 3.6. The real world problem set examines the performance for a single problem instance, and

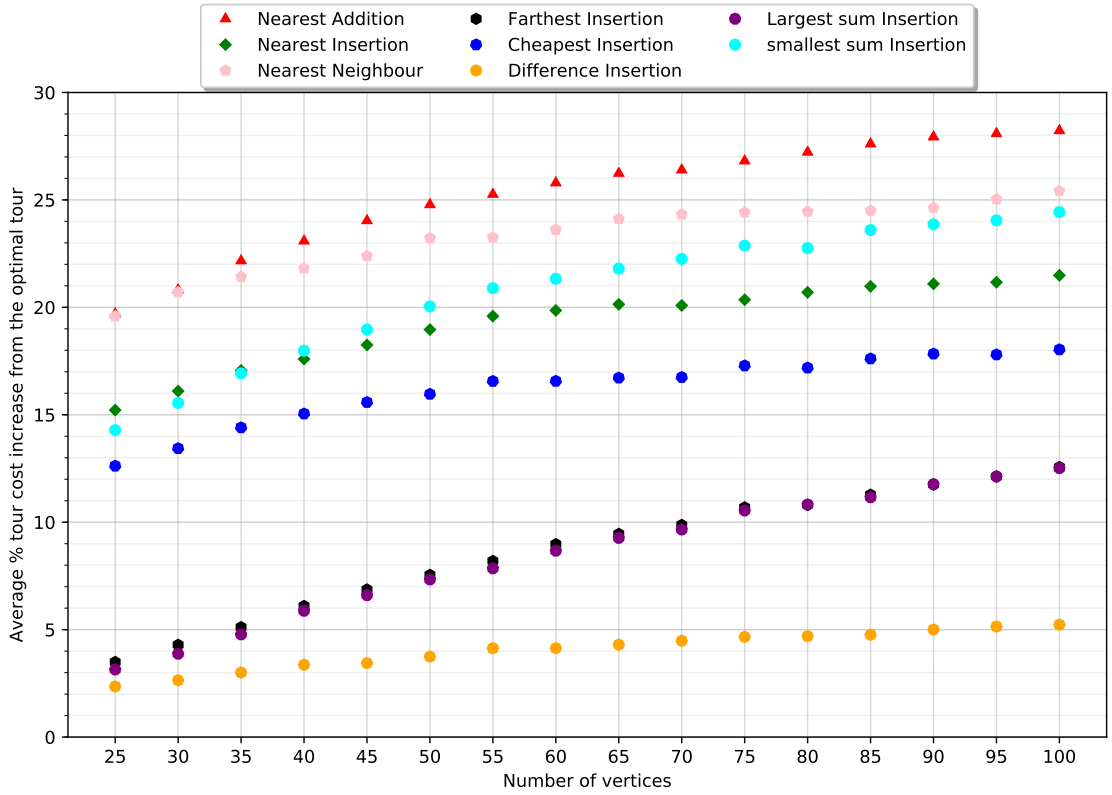


Figure 3.8: A mean *Percentage Tour Cost Increase (PTCI)* comparison of the selected successive augmentation procedures for the artificial problem set

Procedure name	Summary PTCI (%)	Standard deviation
Difference insertion	7.477	4.588
Farthest insertion	18.060	7.636
Cheapest insertion	18.302	4.712
Largest sum insertion	18.576	7.885
Nearest insertion	22.173	4.526
Nearest neighbour	25.179	5.537
Smallest sum insertion	26.748	5.688
Nearest addition	32.087	6.917

Table 3.6: A table showing the mean tour cost and standard deviation of the successive augmentation procedures, ranked from the smallest *Percentage Tour Cost Increase (PTCI)* to the largest for the real world problem set

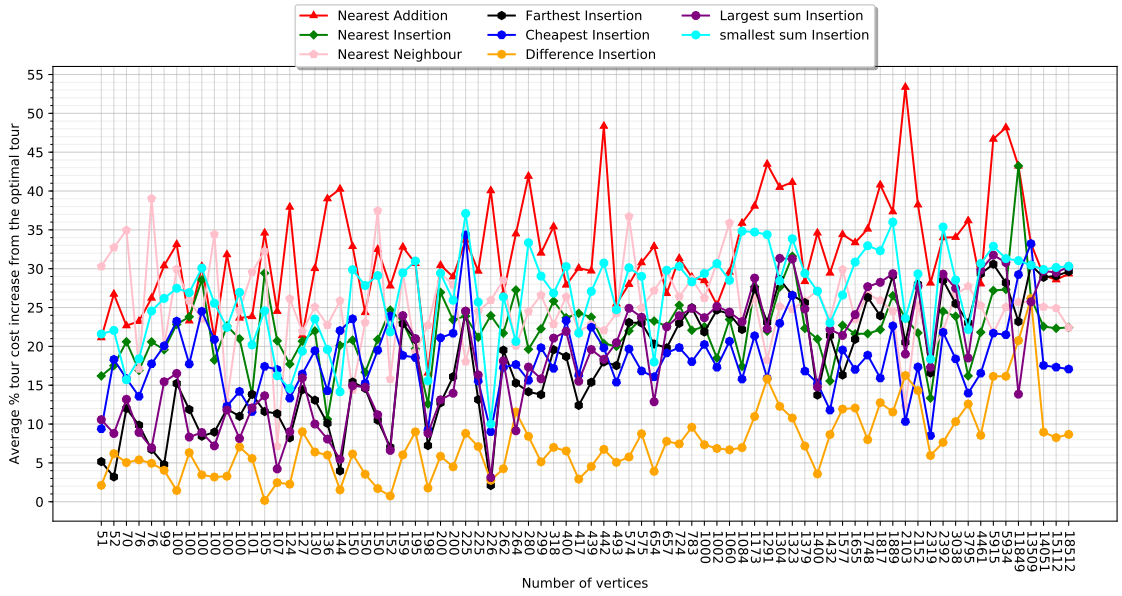


Figure 3.9: A mean *Percentage Tour Cost Increase (PTCI)* comparison of the selected successive augmentation procedures for the real world problem set

therefore each point in Figure 3.9 is not representative of the mean PTCI for that graph size, as in Figure 3.8. The results are therefore more effected by the problem instance's topology and more variance between the performance of graph sizes. Table 3.6 presents the results in a similar way to Table 3.5, with a summary PTCI and standard deviation. We present the results in four groups, based on the similarities and differences between the procedures as described in Section 3.3. These four groups are presented in order of their best performing procedure, as: Difference insertion and Cheapest Insertion; Largest Sum Insertion and Farthest Insertion; Nearest Insertion and Smallest Sum Insertion; and finally Nearest Neighbour, Nearest addition, and Nearest Insertion. With Nearest Insertion being discussed twice, the second time only to compare the performance of the expansion criteria.

The best performing procedure can be classified in several different ways, however the most evident is that with the smallest summary PTCI across the problem sets, which is Difference Insertion. This procedure performs best for both the artificial and real world problem set, with a summary PTCI of 4.069% for the artificial problem set and 7.477% for the real world problem set. For the artificial problem set, the summary standard deviation was also the smallest of all the procedures tested with 0.892. Note that all of the summary

standard deviation for all procedures are based on results for a single problem instance for each graph size, and therefore cannot be compared directly. The summary standard deviation for the real world problem set is the second best, being 0.062 larger than the best performing procedure. As discussed in Section 3.3 Difference Insertion is an extension of Cheapest Insertion as it uses the difference between the smallest and second smallest insertion cost as the selection criteria.

The Cheapest Insertion procedure was the fourth best performing procedure for the artificial problem set with a summary PTCI of 16.213%, and 18.302% for the real world problem set. The summary standard deviation for the artificial problem set is 1.614, and is 4.712 for the real world problem set. Difference Insertion therefore performs approximately 12% better than Cheapest Insertion for the artificial problem set and approximately 11% better for the real world problem set, in terms of summary PTCI. As the expansion criteria is the same for both, we can conclude that the extended selection criteria of Difference Insertion accounts for the improvement in performance. Difference Insertion also has a smaller standard deviation, with it being almost half that of Cheapest Insertion for the artificial problem set and marginally smaller for the real world problem set, which shows that it consistently produces tours with a smaller cost.

The second best performing procedure by summary PTCI is Largest Sum Insertion, with 8.499% for the artificial problem set, and is the fourth best performing procedure with 18.576% for the real world problem set. The summary standard deviation of Largest Sum Insertion for the artificial problem set is 2.997, and 7.885 for the real world problem set. Rosenkrantz et al determined that selection metrics that select an unvisited vertex that is farthest away tended to generate good quality tours, due to a process of generating an outline of a tour first and then filling in the remaining vertices given more information and therefore being closer to optimal [83]. This strategy is also used by the Farthest Insertion procedure, which was the third best performing procedure with the artificial problem set with a summary PTCI of 8.696% and was the second best performing procedure for the real world problem set with 18.060%. The summary standard deviation for the artificial problem set is 2.877, and 7.636 for the real world problem set.

The performance of both of these procedures is similar for both problem sets, with Largest Sum Insertion outperforming Farthest Insertion by only 0.197% for the artificial problem set, and conversely Farthest Insertion outperforming Largest Sum Insertion by 0.57%. Farthest Insertion has a marginally smaller standard deviation, with a difference of 0.12 for the artificial problem set and 0.249 for the real world problem set. As Largest

Sum Insertion performs better than Farthest Insertion for the artificial problem set and performs worse for the real world problem set, this indicates that a selection criteria that uses multiple edges using this strategy performs better for graphs without the clustering seen in some real world problems instances. This strategy performs better than the cost based procedure Cheapest Insertion for the artificial problem set and similarly for the real world problem set, and outperforms the closest selection strategy of the rest of the procedures.

Nearest Insertion is the fifth best performing procedure for both problem sets, with a summary PTCI of 19.29% for the artificial problem set and 22.173% for the real world problem set. The summary standard deviation of the artificial problem set is 1.197 and for the real world problem set is 4.526. This procedure uses a strategy of identifying the unvisited vertex that is closest to the partial tour and then inserts that vertex at the position in the tour with the smallest additional cost. This particular strategy is also shared by Smallest Sum Insertion, which is the sixth best performing procedure for the artificial problem set with a summary PTCI of 20.725%, and was the seventh best performing problem set with 26.748%. The standard deviations for the artificial problem set is 3.155, and for the real world problem set is 5.688. Nearest Insertion outperforms Smallest Sum Insertion for both problem sets, and has a significantly smaller standard deviation for the artificial problem set, with it being approximately half of Smallest Sum Insertions. In comparison with Farthest Insertion and Largest Sum Insertion, the use of multiple edges has a negative effect on performance for this strategy.

This final group of procedures use the same selection criteria of identifying the unvisited vertex that is closest to the partial tours but have a different expansion criteria. The Nearest Neighbour procedure is the seventh best performing procedure for the artificial problem set with a summary PTCI of 23.308%. It was however the sixth best performing for the real world problem set with 25.179%. The standard deviation for the artificial problem set is 1.678, and for the real world problem set is 5.537. The worst performing procedure procedure across both problem sets is Nearest Addition, with a summary PTCI of 25.258% for the artificial problem set and 32.087% for the real world problem set. The final procedure in this group, Nearest Insertion, was discussed in detail in the third group. For both the artificial and the real world problem set the stratification of performance is consistent between these procedures, with the insertion expansion criteria outperforming the neighbour criteria, which in turn outperforms the addition criteria. The Neighbour expansion criteria necessitates the modification of the selection criteria to only take into

account the edge from the second to last position in the tour and therefore greedily selects the nearest unvisited vertices from the last selection. In this way it aims to minimise the edge travelling to that vertex and away from that vertex, which appears to be a more successful strategy than selecting the cheapest insertion cost from either side of the selection.

3.5 A comparison of successive augmentation procedures for the Orienteering problem

In this section we present a modification to the successive augmentation mechanism to accommodate a tour cost limit ($\mathcal{R}_5$) for the TSP, and an average case empirical analysis for this mechanism. In Section 3.4, we presented the average case empirical analysis for the standard successive augmentation mechanism, and this study will be directly comparing the performance of the selection and expansion criteria against the baseline of this previous study. As discussed in the introduction for Section 3.3, the TSP with a tour cost limit ($\mathcal{R}_5$) may be formulated as several different existing problems. We have chosen to formulate this problem as an Orienteering problem with homogeneous rewards. The procedures for this empirical analysis are presented in Section 3.3, and further highlighted in Table 3.4. We present the results of executing these procedures using the modified mechanism on the same two problem sets from Section 3.4. Using the optimal tour cost for each problem instance as the tour cost limit, the number of vertices visited by each tour is used as the comparison metric. We independently examine two variables, the first being the number of vertices in order to examine the scalability of the procedures with this modified mechanism. The second variable is the tour cost limit, which we vary from 10% of the optimal tour cost to 100% of the optimal tour cost in order to examine how the cost of each procedure increases as the number of possible vertices is reduced.

This section is structured as follows, first we describe the modifications to the successive augmentation mechanism in Section 3.5.1, with a discussion of the justification for these modifications. A description of the experimental methodology for this study is provided in Section 3.5.2. As the same problem instances are used in this study as in the study from Section 3.4, the justification for the type of graph is given in Section 3.4.2, and the reasoning for the selection of the problem instances is described in Section 3.4.2. A different metric for comparison was used however, and this is described in Section 3.5.2.

The motivation for choosing the range of tour cost limits is examined in Section 3.5.2. As the same problem instances are used in this study as the former, the same optimal tour costs are used, and was found using the same method outlined in Section 3.4.2. The technical set up is similar to that described for the previous study, however the differences are outlined in Section 3.5.2. Finally we present the results of the empirical analysis, and a discussion of those results in Section 3.5.3.

3.5.1 Modifications to the successive augmentation mechanism for the tour cost limit

In Section 3.4.1 a description of the successive augmentation mechanism was provided with an algorithm description in Algorithm 1. In previous work in the literature, modifications have been proposed to the mechanism, however they have been discussed specifically in the context of a single successive augmentation procedure, typically cheapest insertion. In this section we describe the selected modification from the literature and describe this mechanism in general non-procedure specific manner, with the algorithm itself described in Algorithm 2.

An example of a variant of the TSP with a tour cost limit is the Orienteering problem. As discussed previously in Chapter 2, a majority of heuristic algorithms for the OP use a successive augmentation procedure for the tour generation stage. For the five heuristics described by Vansteenwegen et al. [95], all use a modified version of a successive augmentation mechanism. Tsiligirides [92] proposed a modification to the successive augmentation mechanism where a monte carlo method was employed. Each vertex at each position was assigned a probability of selection based on it's additional cost until no further vertices may be added to the partial tour. Golden et al. [36] proposed a modification whereby the mechanism would insert the most suitable vertex, given the selection criteria, into the partial tour until no further vertices could be inserted. This was described by Golden et al. in relation to the cheapest insertion procedure. Ramesh and Brown [75], Chao et al. [14], and Gendreau et al. [32] all use Golden et al.'s mechanism for the cheapest insertion procedure.

The algorithm takes in a set of all locations that must be visited and the depot location. For example, given a set of vertices $V = \{1, 2, 3, 4, 5\}$, then the last depot is specified as the depot $D = \{5\}$ and the remaining locations are locations to visit $L = \{1, 2, 3, 4\}$. On line 1, the algorithm instantiates an empty partial tour. A partial tour, starts at a depot location

Algorithm 2 Modified Successive Augmentation Mechanism for a Tour Cost Limit

Require: L, D
1: $P \leftarrow \{D, D\}$
2: $finished \leftarrow False$
3: **while** $finished \neq True$ **do**
4: $bestCost = Null$
5: $bestV = Null$
6: **for** each v in L **do**
7: **if** Selection criteria cost of v is better than *bestCost* **then**
8: *additionalCost* $\leftarrow$ Expansion cost of v given the expansion criteria
9: **if** cost(P) + *additionalCost* $< Q$ **then**
10: *bestCost* $\leftarrow$ selection criteria cost of v
11: *bestV* $\leftarrow v$
12: **end if**
13: **end if**
14: **end for**
15: **if** $bestV \neq Null$ **then**
16: Insert v into P using expansion criteria
17: Remove v from L
18: **else**
19: $finished \leftarrow True$
20: **end if**
21: **end while**

and visits a subset of non-depot vertices before returning to the depot. An empty partial tour therefore consists of the depot vertex and a self loop, for example $P = \{5, 5\}$. Other initial tours can be used, such as a convex hull of the problem instance graph, however we have not chosen that variant.

On line 2, a variable $finished$ is instantiated with the flag $False$. This variable is used in order to determine when vertices can no longer be added to the partial tour, and therefore that process is finished. Between lines 3 and 20, there is the main loop for the algorithm which encompasses the selection and expansion of the tour. This loop terminates when the previously discussed $finished$ flag is set to true. Line 4 instantiates the *bestCost* local variable, which holds the current best cost for the selection criteria. A selection criteria evaluates a potential vertex and assigns a cost based on the specific criteria, with the best performing of these being assigned to this variable. Line 5 instantiates the *bestV* local variable, which holds the vertex with the best cost given the selection criteria.

Between lines 6 and 14, there is a loop that iterates through each of the currently unassigned vertices from the set L. Each of these vertices is assigned to the local variable v. Between lines 7 and 13, there is an if statement that determines if the selection criteria evaluates vertex v to be a better selection than the existing best cost. A selection criteria may minimise or maximise a metric, which is why this statement determines the best instead of the least or largest of a selection criteria. Line 8 occurs if a vertex v has the currently best known selection criteria cost, and assigns the expansion cost of that vertex to the local variable *additionalCost*, given the expansion criteria. Between lines 9 and 12, there is an if statement that determines if the vertex can be added to the tour without exceeding the tour cost limit. $cost(P)$ refers to the current cost of the partial tour at the point where the vertex v may be added, and this in combination with *additionalCost* represent the cost of the tour is vertex v is added to the partial tour. Line 10 occurs if the vertex being added to the partial tour would not violate the tour cost limit, and the new *bestCost* is found. Line 11 similarly occurs in the same situation and the *bestV* variable is assigned the vertex with the best selection criteria cost that does not violate the tour cost limit. Between lines 15 and 20

3.5.2 Experimental Methodology

In this section we explain our reasoning behind the experimental and implementation decisions of the average case empirical analysis discussed in Section 3.5. The same graph

structure and problem instances from the empirical analysis shown in Section 3.4 are used in this empirical analysis, and a discussion of the graph structure is provided in Section 3.4.2, with a description of the problem instances given in Section 3.4.2. A description of why the specific tour cost limits were chosen is given in Section 3.5.2 As the same problem instances are used, the same optimal tour cost values can be used, and therefore was found previously with the optimal solver described in Section 3.4.2. These same problem instances were used in order to allow for a direct comparison between the TSP empirical analysis and this empirical analysis. Section 3.5.2 describes the reasoning for using the selected comparison metric. Finally the technical details that differ from the previous empirical analysis is given in Section 3.4.2.

Metric for comparison

In Section 3.4.2 the Percentage Tour Cost Increase (PTCI) metric was selected as the most suitable for evaluating the effectiveness of a tour in the context of the TSP. When a tour limit is introduced, this metric is no longer suitable for evaluating performance as two tours for the same problem instance are capped by this limit. The opposite of the optimal tour cost is the *maximal tour cost*, which is the tour that visits each of the locations exactly once, starting and ending at the home depot, where the cost is maximised [2]. There are three options for how the tour limit can relate to the maximal and optimal tour costs.

1. If the tour cost limit is higher than the maximal tour cost, then it is not possible that the tour cost limit will cap the length of a tour.

2. If the tour cost limit is lower than the maximal tour cost but higher than the optimal tour cost, then a situation may occur where only a subset of the vertices can be visited.

3. If the tour cost limit is lower than the optimal tour cost then it is guaranteed that only a subset of vertices may be visited.

The first option can be reduced to the standard TSP, and therefore the PTCI would be the most suitable metric. For the second option, the closer the tour cost limit is to the maximal tour cost, the more likely that the resulting tour will visit all of the vertices, however the closer the tour cost limit is to the optimal tour cost, the more likely that only a subset of vertices will be visited in the resulting tour. For example, if the maximal tour

cost for a problem instance is 80, and the tour cost limit is 40 then both tour A and B may both have a tour cost of 40. The use of any metric discussed in Section 3.4.2 would therefore not be suitable as they are comparing tour costs that may not have a difference. The two tours may differ only in the number of vertices visited, so for the example given before, tour A visits 20 vertices and tour B visits 15 vertices. In this case tour A would be the better tour as more vertices are visited within the tour, and the number of vertices can be used as a comparison metric. Option 3 guarantees that a subset of vertices are visited and in this case the number of vertices would be the most suitable comparison metric. As the goal of this empirical analysis is to examine the effect of the modification of the successive augmentation mechanism, using this option ensures that the modified mechanism is examined.

Tour cost limits examined

Different procedures use different strategies in order to build a tour, and by varying the tour cost limit, the performance during different portions of the tour building process can be examined. There are three potential ranges for a tour cost limit in relation to the maximum tour cost and optimal tour cost, as detailed in Section 3.5.2. When the tour cost limit is less than the optimal tour cost, the mechanism for determining which vertices will be visited is always engaged during the tour building process, and therefore this is the range chosen. the specific range value chosen was between 10% of the optimal tour cost to 100% of the optimal tour cost, increasing in intervals of 10%, giving a total of 10 different tour cost limit percentages. A small scale experiment was carried out in order to determine which interval would be appropriate for showing a noticeable different in performance. This experiment used 10 problem instances for each graph size and aimed to find which interval showed a minimum change of 3% number of vertices visited. It was found that 10% showed the appropriate amount of change. As 10% was the smallest possible value given this interval, it was chosen as the starting value, and 100% was chosen as it is the highest value it could be in comparison with the optimal tour cost. For 100% there is a possibility that the subset determining mechanism will not be required, however in practise this did not occur.

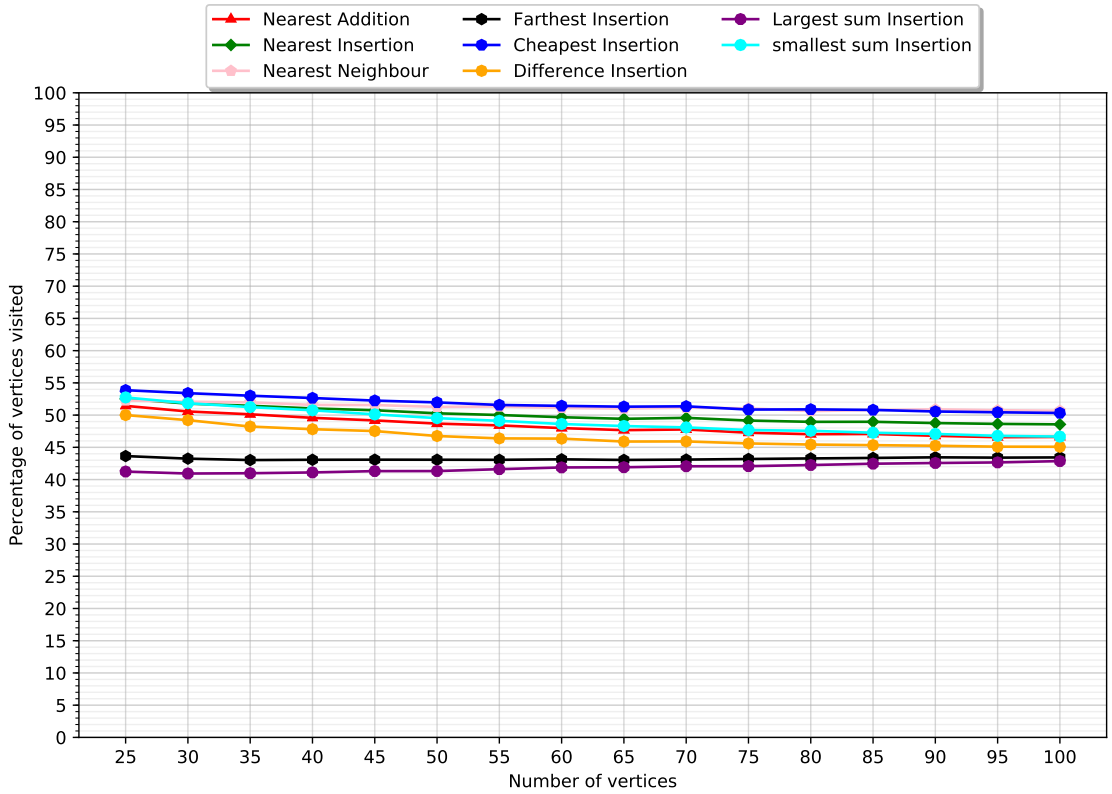


Figure 3.10: A percentage of vertices visited comparison of the selected successive augmentation procedures for the artificial problem set, varying the number of vertices

Technical setup

A portion of the technical work was carried out during the TSP average case empirical analysis, and that technical work was detailed in Section 3.4.2. The graphs and their associated optimal tour costs were generated in this previous empirical analysis. The test frame was written using Python 3.7, and took in the graphs in the '.tsp' file format and generated a tour for each procedure, for each graph, for each tour cost limit. The resulting tours and tour costs were saved using the '.dat' format.

3.5.3 Results and discussion

In this section we present the results of the average case empirical analysis for the modified successive augmentation procedure with a tour cost limit constraint. Two distinct problem

Procedure name	Summary percentage vertices visited (%)	Standard deviation
Cheapest insertion	51.669	1.092
Nearest neighbour	51.261	0.510
Nearest insertion	49.971	1.219
Smallest sum insertion	48.961	1.897
Nearest addition	48.295	1.503
Difference insertion	46.619	1.523
Farthest insertion	43.225	0.185
Largest sum insertion	41.827	0.631

Table 3.7: A table showing the summary percentage of vertices visited and standard deviation of the successive augmentation procedures, varying the number of vertices, ranked from the largest percentage of vertices visited to the smallest for the artificial problem set

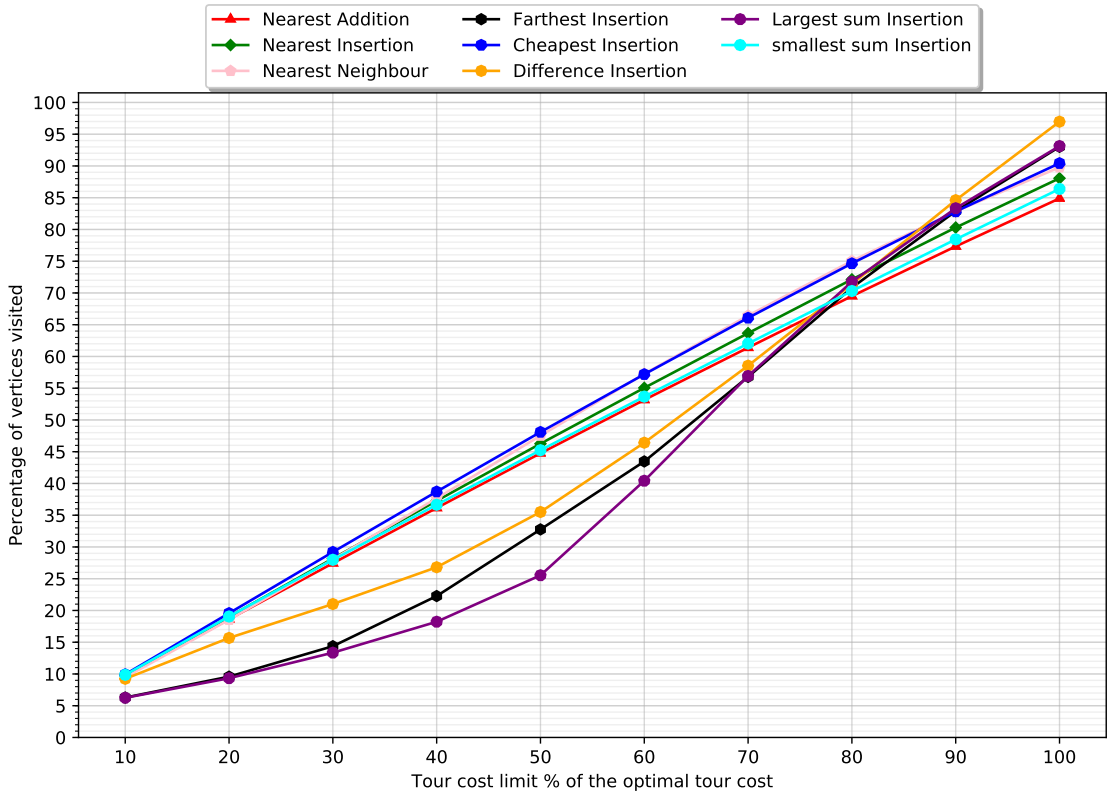


Figure 3.11: A percentage of vertices visited comparison of the selected successive augmentation procedures for the artificial problem set, varying the tour cost limit

Procedure name	Summary percentage vertices visited (%)	Standard deviation
Cheapest insertion	51.669	27.277
Nearest neighbour	51.261	27.628
Nearest insertion	49.971	26.456
Smallest sum insertion	48.961	25.707
Nearest addition	48.295	25.368
Difference insertion	46.619	30.299
Farthest insertion	43.225	31.404
Largest sum insertion	41.827	32.381

Table 3.8: A table showing the summary percentage of vertices visited and standard deviation of the successive augmentation procedures, varying the tour cost limit, ranked from the largest percentage of vertices visited to the smallest for the artificial problem set

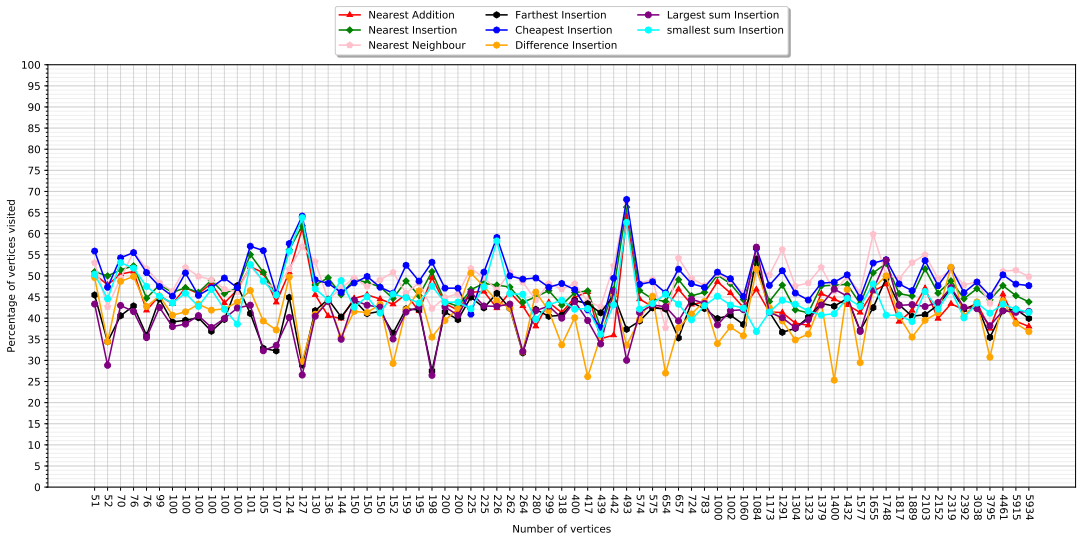


Figure 3.12: A percentage of vertices visited comparison of the selected successive augmentation procedures for the real world problem set, varying the number of vertices

Procedure name	Summary percentage vertices visited (%)	Standard deviation
Cheapest insertion	49.497	4.737
Nearest neighbour	49.303	4.246
Nearest insertion	47.559	4.217
Smallest sum insertion	44.798	4.970
Nearest addition	44.573	4.761
Farthest insertion	40.894	4.318
Largest sum insertion	40.878	5.073
Difference insertion	40.549	6.174

Table 3.9: A table showing the summary percentage of vertices visited and standard deviation of the successive augmentation procedures, varying the number of vertices, ranked from the largest percentage of vertices visited to the smallest for the real world problem set

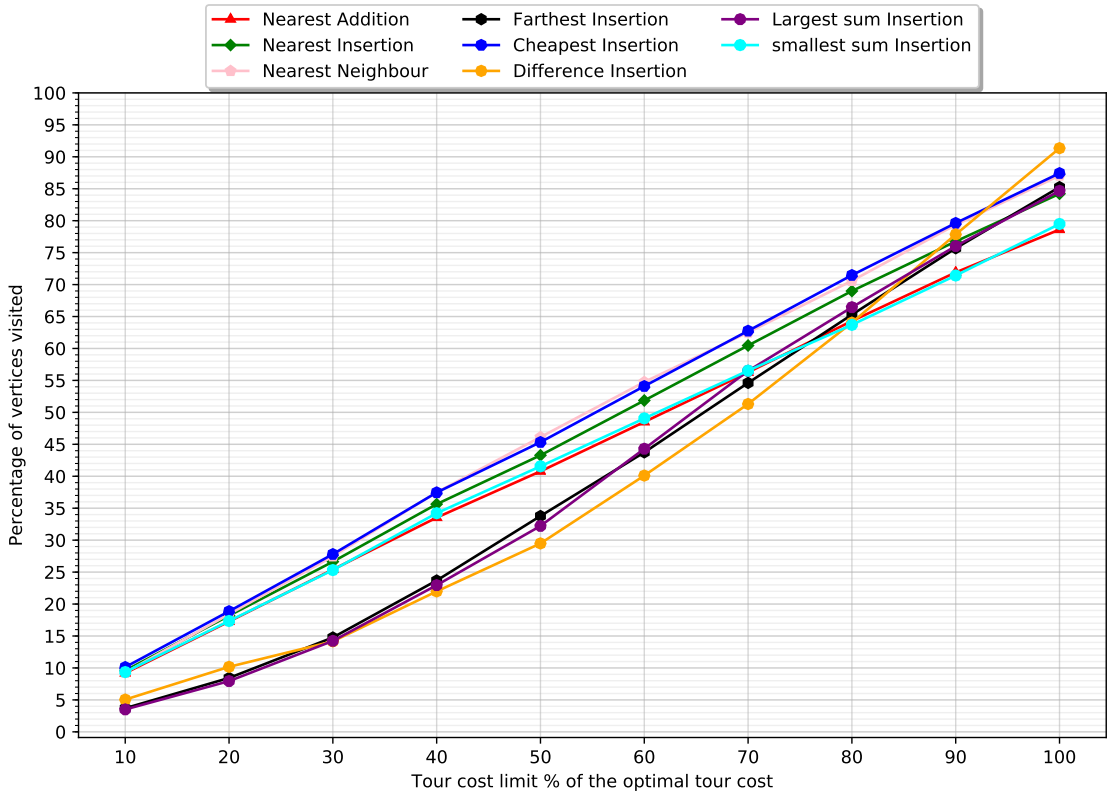


Figure 3.13: A percentage of vertices visited comparison of the selected successive augmentation procedures for the real world problem set, varying the tour cost limit

Procedure name	Summary percentage vertices visited (%)	Standard deviation
Cheapest insertion	49.497	26.135
Nearest neighbour	49.303	26.181
Nearest insertion	47.559	25.263
Smallest sum insertion	44.798	23.420
Nearest addition	44.573	23.447
Farthest insertion	40.894	28.744
Largest sum insertion	40.878	29.154
Difference insertion	40.549	29.787

Table 3.10: A table showing the summary percentage of vertices visited and standard deviation of the successive augmentation procedures, varying the tour cost limit, ranked from the largest percentage of vertices visited to the smallest for the real world problem set

sets were used, the artificial and real world. For both of these problem sets, the number of vertices and the tour cost limit percentage was varied independently so their effect on the procedure can be examined. The results for the artificial problem set varying the number of vertices is presented in Figure 3.10, and the performance of each procedure is summarised in Table 3.7. For each graph size and tour cost limit combination there are 1000 problem instances, and a mean number of vertices visited is found for each combination, which we call the *combination summary*. Each point in Figure 3.10 represents the mean value for each combination summary with the same number of vertices, and therefore summarises the performance for that graph size. In Table 3.7 two values are presented for each of the procedures. The first is the *summary percentage vertices visited* which represents the mean value for all points in Figure 3.10 for a procedure. The second is the standard deviation across the points in Figure 3.10. The results for the artificial problem set varying the tour cost limit percentage is presented in Figure 3.11, and the performance for each procedure is then summarised in Table 3.8. Each point in Figure 3.11 represents the mean value for each combination summary with the same tour cost limit. Table 3.8, in a similar way to Table 3.7, summarises the performance for each procedure with both the summary percentage vertices visited and the standard deviation for Figure 3.11. The summary percentage visited is the same between Table 3.7 and Table 3.8 for each procedure, and the information is repeated for the readability of the tables.

For the real world problem set, the results for varying the number of vertices is presented in Figure 3.12, and the performance of each procedure is summarised in Table 3.9. As there

is only a single instance for each vertex size, there is a single tour for each tour cost limit. Each point in Figure 3.12 represents the mean value for the number of vertices visited for that graph size across the tour cost limits. Table 3.9 presents the summary number of vertices visited and standard deviation for each of the procedures in Figure 3.12. For the real world problem set, the results for varying the tour cost limit is presented in Figure 3.13, and the performance of each procedure is summarised in Table 3.10. Each point in Figure 3.13 represents the mean value for that tour cost limit for each of the graph sizes in the problem set. In Table 3.10, presents the summary number of vertices visited and standard deviation for each of the procedures in Figure 3.13. In a similar way to the tables for the artificial problem set, Table 3.9 and Table 3.10 have the same summary percentage vertices visited for each procedure, repeated for readability.

We present the results in four groups, based on the similarities and differences between the procedures as described in Section 3.3. These four groups are presented in order of their best performing procedure, as: Difference insertion and Cheapest Insertion; Nearest Neighbour, Nearest addition, and Nearest Insertion; Nearest Insertion and Smallest Sum Insertion; and finally Largest Sum Insertion and Farthest Insertion. With Nearest Insertion being discussed twice, the first time only to compare the performance of the expansion criteria.

The best performing procedure in terms of having the largest number of vertices visited for both the artificial and real world problem sets is the Cheapest Insertion procedure. The the artificial problem set the summary number of vertices visited is 51.669%, and is 49.497% for the real world problem set. The performance of all procedures, when varying the number of vertices for the artificial problem set, is consistent across the number of vertices, which can be seen by the small standard deviation of 1.092. When varying the tour cost limit, the performance of the Cheapest Insertion procedure has an average increase of 8.049% for each increase of 10% to the tour cost limit. The performance of the procedures while varying the number of vertices for the real world problem set varies significant between problem instances, which can be accounted for by the different topologies of the graphs. This results in a standard deviation of 4.737. The performance of cheapest insertion when varying the tour cost limit for the real world problem set is also consistent throughout the tour cost limits, with an average increase of 7.730% for each increase of 10% to the tour cost limit.

The difference insertion procedure was the 6th best performing procedure in terms of summary number of vertices visited for the artificial problem set, with 46.619%, and

the 8th best performing procedure for the real world problem set using the same metric, with 40.549%. As previously discussed, the performance when varying the number of vertices for the artificial problem set is consistent across the number of vertices, with a standard deviation of 1.523. The performance of the difference insertion procedure under performs on average in comparison with cheapest insertion due to it's performance with smaller tour cost limits as a percentage of the optimal. For a tour cost limit of 10%, different insertion performs 0.698% worse than cheapest insertion with regards to the combination summary. The performance of difference insertion becomes progressively worse by comparison to cheapest insertion until a tour cost limit of 50%, where it is 12.587% worse. From that point on-wards the performance is still worse, but the gap in performance closes until a tour cost limit of 90% where the performance improves so difference insertion outperforms cheapest insertion by 1.785%. This trend continues with a tour cost limit of 100% having an average performance improvement of 6.549% between different insertion and cheapest insertion. This indicates that the difference insertion procedure initially creates a more expensive tour, then it fills in the remaining vertices in more optimal positions than cheapest insertion, resulting in a cheaper tour. For the real world problem set, the summary percentage vertices visited is 40.549%, which is approximately 9% worse than using cheapest insertion. The standard deviation for varying the number of vertices is 6.174, which is the highest of any of the procedures examined for the real world problem set. For varying the tour cost limit the procedure follows a similar pattern to the artificial problem set, starting with a 5.078% worse performance with a tour cost limit of 10% when compared to cheapest insertion. The performance decreases until a tour cost limit of 50% where the gap is 15.947%. This then improves until a tour cost limit of 100% where the difference insertion produces a 3.915% improvement on cheapest insertion.

The second best performing procedure for both of the problem sets was nearest neighbour, with a summary percentage vertices visited of 51.261% for the artificial problem set, and 49.303% for the real world problem set. When varying the number of vertices for the artificial problem set, the performance of the procedure was consistent and has a standard deviation of 0.510, which is the second lowest. When varying the tour cost limit, the procedure has an average increase of performance of 8.032% for each increase of 10% to the tour cost limit. For the real world problem set, varying the number of vertices, the standard deviation was the second smallest with 4.246. For varying the tour cost limit, the increase was similarly linear with an average increase of 7.778% for every increase of 10% to the tour cost limit.

The third best performing procedure for both the artificial and real world problem sets was Nearest Insertion, with a summary percentage vertices visited of 49.971% and 47.559% respectively. When varying the number of vertices for the artificial problem set, the performance of the procedure is consistent across the various graph sizes, with a standard deviation of 1.219, which is the 5th best performing. When varying the tour cost limit, the performance increases consistently in a linear manner. There is an average increase in the mean number of vertices visited of 7.828% for a 10% increase in the tour cost limit. For the real world problem set, the standard deviation in performance is 4.217 when varying the number of vertices, which is the lowest for this problem set. When varying the tour cost limit, the performance is linear with an increase of 7.463% for every increase of 10% to the tour cost limit.

Nearest addition is the fifth best performing procedure for both the artificial and real world problem set. The procedure has a summary percentage vertices visited of 48.295% for the artificial problem set and 44.573% for the real world problem set. For the artificial problem set, the varying of the number of vertices resulted in a standard deviation of 1.503, which is the 6th best performing. When varying the tour cost limit, each increase of 10% leads to an average increase of 7.519% to the mean percentage number of vertices visited. For the real world problem set, varying the number of vertices has a standard deviation of 4.761, which is the 5th best performing by this metric. Varying the tour cost limit resulted in a mean increase of 6.942$ for an increase of 10% to the tour cost limit.

The third group examined is the formerly discussed nearest insertion, and smallest sum insertion. Smallest sum insertion was the 4th best performing procedure for both problem sets, with a summary percentage vertices visited of 48.971% for the artificial problem set, and 44.798% for the real world problem set. Varying the number of vertices for the artificial problem set resulted in a standard deviation of 1.897, which was the worst of the procedures examined. Varying the tour cost limit, the performance is in line with Nearest Insertion with an increase of 10% to the tour cost limit resulting in a mean increase of 7.652% to the mean percentage number of vertices visited. Varying the number of vertices for the real world problem set resulted in a standard deviation of 4.970, which was the 6th best performing. Varying the tour cost limit resulted in a mean increase of 7.013% to the mean percentage number of vertices visited for each increase of 10% to the tour cost limit.

The worst performing group of procedures are Largest Sum Insertion and Farthest Insertion. Farthest Insertion was the 7th best performing procedure for the artificial problem set with a summary percentage vertices visited of 43.225%, and was the 6th best

performing procedure for the real world problem set with a summary percentage vertices visited of 40.894%. For the artificial problem set when varying the number of vertices, the standard deviation was 0.185 which is the best standard deviation for this problem set. When varying the tour cost limit, the cost does not increase linearly across the different increments of 10% to the tour cost. As both Farthest Insertion and Largest Sum Insertion have a non-linear increase in performance we will compare their performance against the average performance of the linearly increasing procedures. These procedures are: Nearest Addition, Nearest Insertion, Nearest Neighbour, Cheapest Insertion, and Smallest Sum Insertion. For a tour cost limit of 10%, Farthest insertion has a combination summary that is 3.468% worse than the average combination summary for the linearly increasing procedures. The difference in performance increases until a tour cost of 40% where farthest insertion performs 15.017% worse than the average combination summary. The performance then improves until a tour cost limit of 90%, where farthest insertion outperforms the average combination summary by 2.536%. Finally with a tour cost limit of 100% farthest insertion outperforms the average by 5.108%. For the real world problem set, the summary percentage vertices visited is 40.894%, which is the 6th best performing procedure. For varying the number of vertices, the standard deviation is 4.318 which is the 3rd smallest for this problem set. When varying the tour cost limit, farthest insertion follows a similar non-linearly increasing pattern as for the artificial problem set, starting with a decreasing in performance of 5.808% against the average combination summary of the linearly increasing procedures, for the tour cost limit of 10%. The gap in performance increases until a tour cost limit of 40%, where farthest insertion is 15.017% worse than the average linearly increasing procedures. The performance then improves, with the gap closing until a tour cost limit of 90%, where farthest insertion only under performs by 0.124%. Finally at a tour cost limit of 100% the performance of farthest insertion is 1.893% better.

3.6 A comparison of successive augmentation procedures for the Periodic travelling salesperson problem

In Section 3.1, the revisiting requirement ($\mathcal{R}_3$) was introduced as the need to revisit some location within a fixed number of days since being previously visited. This type of requirement is particularly important in many maintenance or service scenarios, such as those where a location needs to be regularly checked, for example with the Water Treat-

ment Plant scenario, to ensure that there is no build-up of contaminants at such locations. Thus, the addition of a revisiting requirement ($\mathcal{R}_3$) to the TSP changes the problem to that of having multiple tours (where each tour is associated with each day in the planning horizon) where a different subset of vertices are visited on each of the days such that a location is re-visited within its time window (Figure 3.2c). This differs from the notion of visiting all of the vertices each day in the planning horizon as that would simply be equivalent to that of running the same TSP problem instance multiple times; rather this problem also focuses on the selection of a subset of vertices each day to ensure that the time window of each vertex is respected.

The problem itself is very similar to the Periodic Travelling Salesperson Problem (PTSP), as discussed in Section 2.4, in that traditional solutions to the PTSP aim to produce sets of tours, each associated with a day in the planning horizon, that minimise the total cost of all tours while revisiting the non-depot vertices given some set of requirements. However, the majority of the existing solutions achieve this by assuming a *fixed visit schedule* ($\mathcal{R}_1$) or *visit cardinality* ($\mathcal{R}_2$). As the focus of this work is to specifically model the revisiting requirement ($\mathcal{R}_3$), we therefore refer to this specific variant of the PTSP as the *Periodic Travelling Salesperson Problem with Visit Dependant Time Windows (PTSP-VDTW)*. The standard successive augmentation mechanism is intended for use with the Travelling Salesperson Problem (TSP), and iteratively adds a currently unvisited vertex to a partial tour until all vertices are present in that tour.

In order to accommodate the PTSP-VDTW, three modifications are required:

1. A mechanism for creating a partial tour that is associated with each day in the planning horizon;

2. A mechanism that determines which insertions are viable for each of the days in the horizon; and

3. A means for terminating execution when the schedule has satisfied the revisiting constraint for all non-depot locations.

In Section 3.6.1 we discuss the modifications required to the mechanism and selection metric, as well as the additional data structure required to accommodate the changes. In Section 3.6.2 we discuss the experiment settings for the empirical evaluation, and present the results in 3.6.3.

The construction of multiple tours also introduces an additional criteria that the procedures should take into account for selection. We have identified two factors, the cost of the tours and the time window for each location, that can be used to can be used augment existing procedures. For example, the tour with the cheapest tour cost can be selected and then a procedure such as cheapest insertion used. We have therefore identified four variants for each of the procedures: 1) cheapest tour; 2) most expensive tour; 3) smallest time window; and 4) largest time window. We discuss the effect of augmenting each procedure with these new variants in Section 3.6.4

3.6.1 Modifications to the successive augmentation mechanism for the PTSP-VDTW

There are two major modifications required for the successive augmentation procedure, and an additional data structure required for keeping track of which vertices may be inserted on each of the days. The additional data structure required is called the *potential visits*, which stores and represents which vertices can be inserted into each of the tours. In Section 3.6.1 we discuss the reasoning behind this data structure, and how it can be generated and updated. Section 3.6.1 then describes how the mechanism can be modified. Finally in Section 3.6.1 an additional modification is mentioned, which is the tie breaking conditions and extensions to the selection criteria.

Potential visits data structure

One approach for ensuring that a schedule satisfies the revisiting requirement is to ensure that a vertex is only inserted into a tour if it is within the time window of it's previous visit. For example, if location i is present in the tour for day 4 and has a time window of 2 days, then it would be available for insertion during days 5 and 6. Thus, it is desirable to keep track of candidate vertices that are available for each day, which we refer to as *potential visits*. This can be achieved by storing tuples as a list with there being m lists, each associated with a day in the planning horizon and holding the potential vertices for that day. The initial state of the potential vertex insertions is that an instance of each is vertex present in each list that corresponds to a day that is less than or equal to the vertex's time window. For example, if the planning horizon is 4 days, and the time window for vertex i is 2 and the time window for j is 1, then both i and j are present in the list for day 1, only i is present in the list for day 2, with day 3 and 4 being empty.

Vertex coverage refers to the latest day in the planning horizon that a vertex may be inserted. For vertex i in the previous example, the vertex coverage would be 2. The list of candidate tuples is then updated once a new selection has been made to reflect the change in which vertices may be added to each tour. If vertex i is selected and inserted into the tour for day 2 then the lists would be updated, and as a consequence, vertex i could now be viably visited on either day 3 or day 4. Thus, i has a coverage of 4. As the triangle inequality holds for each of the edges in the graph, adding a vertex into the prior tour (for day 1) provides no benefit in terms of extending the vertex coverage and only increases the resulting cost of that tour. We therefore only permit insertions that would result in an increase in the coverage; and when updating the potential vertex insertions, all visits before the current selection are removed. If a vertex's coverage extends beyond the planning horizon, then it can be considered planned for.

Algorithm 3 Generating potential visits

Require: *pvi*, r
1: **for** each v in L **do**
2: **for** each h in $r[v]$ **do**
3: $pvi[\ v\ ]\ [\ h\ ] \leftarrow True$
4: **end for**
5: **end for**

Algorithm 3 describes the algorithm that generates the initial potential visits data structure, which we refer to as *pvi*. The algorithm takes in an empty *pvi* which consists of a list with the same length as the number of service locations, and each element of that list is a list with the same length as the number of days in the planning horizon. Each element of *pvi* initially starts as false. The loop starting on line 1 and ending at line 5 iterates through each service location. The loop starting on line 2 and ending on line 4 takes the current vertex being examined v and then iterates the number of times equivalent to the time window for v. Line 3 assigns the positions in the potential visits data structure to $True$ in order to represent that a visit on that day is possible for that vertex. The updating procedure is similar to the generation procedure. All elements of the sub-list corresponding to the vertex's position in the set is set to $False$. The elements following the last visit are then set to $True$. If a vertex is totally planned for, this entire sub list is designated as $False$.

Successive augmentation modifications

There are two major modifications that are required for the successive augmentation mechanism to produce sets of tours for the Periodic Travelling Salesperson Problem with Visit Dependant Time Windows (PTSP-VDTW). The first modification required is that a different termination criteria is required. Each iteration, the partial tour of the successive augmentation mechanism is re-examined in order to determine if the termination criteria is met. For the PTSP, the mechanism should terminate if each vertex is present within it's time window of the end of the planning horizon. If a vertex has vertex coverage that extends beyond the planning horizon, then it may be visited on one of these days, and therefore planning for it should cease. If all vertices reach this condition, and their coverage extends beyond the planning horizon, then the termination criteria is met.

The second modification is that the successive augmentation mechanism must iterate through the potential visits and determine which must be inserted into the relevant tours within the planning horizon. Algorithm 4 presents a pseudocode description of the modified successive augmentation mechanism. Line 1 instantiates a list S which has the same number of elements as the number of service location. The loop between lines 2 and 4 iterates through each element of S and assigns a sub list with two elements of the depot. This process results in a completed empty schedule S, which is the equivalent of the partial tour P for the standard successive augmentation mechanism.

The loop spanning from lines 5 to 22 is the main loop of the modified successive augmentation mechanism, and uses a method called $check_completely_scheduled$, that uses S as a parameter to determine if the mechanism can terminate. Lines 6, 7, and 8 instantiate the values $bestCost$, $bestV$, and $bestH$ respectively. $bestCost$ holds the information about the cost of the currently best found vertex, with $bestV$ holding that vertex, and $bestH$ holding which partial tour that vertex should be inserted into. The loop between lines 9 and 19 iterates through each position within the planning horizon in reverse. This iteration order is chosen in order to preference later days for a tie breaking condition as they are considered first. The loop between lines 10 and 18 iterates through each element in the service locations set. The if statement between lines 11 and 17 determines if the currently examined vertex, on that day within the planning horizon is possible by checking the potential visit data structure. If it may be visited, then the if statement between lines 12 and 16 is entered, which checks if the selection criteria cost of that vertex is better than the current cost. If that is true then the relevant variables on lines 13, 14, and 15 are

modified and the loops continue. Line 20 inserts that vertex at the correct position given the expansion criteria, and then on line 21 the potential visits are updated.

Algorithm 4 Modified Successive Augmentation Mechanism for a Revisiting constraint

Require: L, D
1: $S \leftarrow \{0, ..., n\}$
2: **for** each s in S **do**
3: $s \leftarrow \{D, D\}$
4: **end for**
5: **while** $check_completely_scheduled(S)$ **do**
6: $bestCost = Null$
7: $bestV = Null$
8: $bestH = Null$
9: **for** each h in H, from H to 1 **do**
10: **for** each v in L **do**
11: **if** $pvi[\ v\ ]\ [\ h\ ]$ **then**
12: **if** Selection criteria cost of v is better than $bestCost$ **then**
13: $bestCost \leftarrow$ selection criteria cost of v
14: $bestV \leftarrow v$
15: $bestH \leftarrow h$
16: **end if**
17: **end if**
18: **end for**
19: **end for**
20: Insert v into P using expansion criteria
21: Update pvi
22: **end while**

Selection criteria and tie breaking conditions

Instead of having a single initial partial tour associated with a single day, there are multiple initial partial tours with one associated with each of the days in the planning horizon. The selection metric therefore applies to each of the partial tours. For a TSP tour using a successive augmentation mechanism, a situation may occur where two candidate insertion positions may be identified with the same cost. In such a situation, a conflict resolution policy or rule is required as a means of breaking the tie; i.e. if the two vertices in this tie are different, then a rule can be used to the order in which the vertices are evaluated. For example, if inserting vertex 1 and vertex 2 would have the same cost, and the rule of

selecting the later vertex to resolve ties had been adopted, then vertex 2 will be chosen. This is because the order in which vertices are considered for insertion is unique and this will always resolve this tie breaking condition.

The insertion position within a partial tour can also raise the same issue, with there being either a preference for earlier or later within a tour. For example, inserting vertex i at either position 1 or position 5 in the tour, and if the preference is for the later position, then position 5 will be chosen. With the addition of multiple tours, a further condition must also be considered; that of using either an earlier or later preference of tours. In this situation a preference of the later tour would be a more logical choice, as it increases the vertex coverage to a greater degree than the earlier tour preference. We therefore will only examine the later tour preference.

3.6.2 Experimental Methodology

An average case empirical evaluation was conducted over a set of small scale problem instances in order to evaluate if the modified successive augmentation mechanism produces schedules with costs that are within a reasonable percentage of the optimal schedule cost. A small scale problem set was selected due to the NP-hardness of the problem, the cost of determining the optimal schedule cost of a problem instance can have a significant time cost associated with it. A real world problem set is then examined in order to determine the applicability of the mechanism for practical purposes. In Section 3.6.2 the graph structure of these problem instances is discussed, and the problem instance sizes are covered in Section 3.6.2. The resulting tours for the small scale problem set are compared against the optimal tour cost, with the metric of comparrison being percentage schedule cost increase, which is discussed in Section 3.6.2. Finally the way the optimal tour cost is found, as well as the other technical details are discussed in section 3.6.2.

Graph structure

The graph structures used for the PTSP do not differ greatly from that described in Section 3.4.2, as the graphs are complete undirected graphs. The same edge weight is used even though Concorde is not used to determine the optimal schedule cost for this problem set. This is in order to allow results to be more directly comparable between these empirical evaluations. Each of the non-depot vertices has a time window, which is a random integer between 1 and the floor value of half of the planning horizon. This range was chosen in

order to ensure that each vertex is visited at least twice within the planning horizon. For example if the planning horizon is 5 days, then the time window can be either 1 or 2.

Problem instance sizes

For the optimal problem set, as the optimal tour cost must be computed and there is a significant computational cost associated with this, only a smaller size of graphs could be used. Therefore a graph size of 6 vertices was chosen, with each the first vertex being designated as the depot this results in 5 service locations. For this graph size, a maximum number of days of that can be computed is 6 days within a reasonable computation time, which we have determined to be approximately 4 hours. The range was therefore determined to be between 4 and 6 days, as 3 days only allows for each time window to be 1 given our twice revisiting constraint. In order to minimise the effect of any one randomly generated graph, 1000 graphs were generated for the 4 and 5 day problems, with 700 being generated for the 6 day problem. For the real world problem set, instances from TSPLIB identified in Section 3.4.2 were selected, with problem instances used ranging in a graph size from 51 to 1000 vertices. This size was chosen as the time complexity for the largest graph size has a computational complexity associated with it of around 4 hours.

Metric

As discussed in detail in Section 3.4.2, the use of the percentage tour cost increase (PTCI) metric has several benefits when comparing the performance of an approach in generating tours where the optimal tour cost is known. This normalises for different graph sizes, the variation in tour cost that may occur between multiple graphs of the same size. As the PTSP has multiple tours, the use of PTCI on it's own would not be suitable as a comparison metric. We therefore use an extension of this metric, called percentage schedule cost increase (PSCI). This takes the total cost of all tours in the schedule, and expresses this as a percentage of the optimal schedule cost. For example if the tour costs for a problem instance is 40, 60, and 50, and the optimal tour costs for a problem instance is 30, 50, and 20, then the total schedule cost is 150 and the optimal schedule cost is 100. The cost is not shown for each of the individual tours, but the total schedule and therefore the PSCI is 50%. For the real world problem set the raw schedule cost is used as a metric as no optimal tour cost can be found.

Technical implementation

The graphs were generated using python 3 on a windows 10 laptop with Intel Core i7-8750H CPU @ 2.20GHz, with the co-ordinates for each vertex generated using a random number generator, who's seed was generated from the current time (measured in milliseconds) when the evaluation was conducted. The method used to find the optimal schedule cost was to implement a modified integer programming formalisation presented in Section 3.2. This modification was to drop the tour cost limit given in Constraint 3.4, to correctly match the problem presented by the PTSP-VDTW. The solution to each of the small scale problem instances was found using the modified Integer Programming formulation, which was implemented using pyomo library for python 3, and was executed on the same Windows 10 laptop.

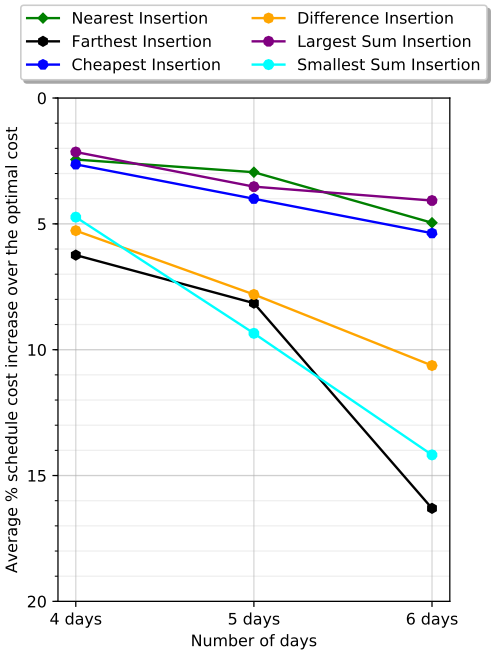
3.6.3 Results

The successive augmentation procedures used within this study can be divided into three separate groups on the basis of their overall performance:

- The first group corresponds to the best performing approaches, which include Largest Sum Insertion, Nearest Insertion, and Cheapest Insertion;
- The performance of the second group is not as good, and include Difference Insertion, Smallest Sum Insertion, and Farthest Insertion;
- The poorest performing approaches include Nearest Addition and Nearest Neighbour.

The results of each of the procedures are presented in Figure 3.14, with a visualisation provided in Figure 3.14a, excluding the results for the Nearest Addition and Nearest Neighbour procedures as their performance is far worse than the others and would skew the visualisation. The best performing procedure is Largest Sum Insertion with a mean PSCI of 3.247%, It's performance is also consistent across the different problem sizes with a standard deviation of 0.993. In the same way as discussed in the empirical analysis of Sections 3.4 and 3.5, largest sum insertion generates tours by first generating an outline by identifying vertices that are furthest away from the existing partial tour and then fills in the remaining vertices.

The second best performing procedure is Nearest Insertion, with a mean PSCI of 3.449%. The performance of this procedure is also consistent across the problem sizes



(a) A plot of the average percentage increase in scheduling cost for the top performing successive augmentation procedures with different planning horizons

Procedure name	Mean PSCI	Standard deviation
Largest sum insertion	3.247	0.993
Nearest insertion	3.449	1.329
Cheapest insertion	4.003	1.367
Difference insertion	7.901	2.681
Smallest sum insertion	9.421	4.724
Farthest insertion	10.232	5.345
Nearest addition	67.307	1.454
Nearest neighbour	74.382	1.853

(b) The mean *Percentage Schedule Cost Increase* (PSCI) for each of the successive augmentation procedures across different planning horizons investigated

Figure 3.14: The performance of different successive augmentation procedures when used with the PTSP-VDTW with different planning horizons

with a standard deviation of 1.329. This procedure has a different strategy as it aims to create a small tour by identifying vertices that are nearby, which may indicate that several different strategies may be successful for small scale problem instances. Cheapest insertion is the third best performing procedure, with a mean PSCI of 4.003%. The performance, as with the prior two procedures, is stable when increasing the number of days, with a standard deviation of 1.367. This procedure also aims to create small tours using the cost as a selection criteria, and this would indicate that this strategy performs well regardless of the problem size.

The difference insertion expands on the Cheapest Insertion procedure, and performs worse with a mean PSCI of 7.901. The performance of the second group is more effected by an increase in the number of vertices, this procedure starts with a mean PSCI of 5.269%

Procedure name	Summary mean schedule cost	Standard deviation
Difference insertion	4001908.031	25564997.090
Farthest insertion	4174344.754	26798454.798
Largest sum insertion	4259375.301	27248037.388
Cheapest insertion	4351373.214	27814234.979
Smallest sum insertion	4377738.803	27980222.878
Nearest insertion	4393209.427	28060828.773
Nearest addition	4680802.728	29882211.782
Nearest neighbour	6793028.939	43660151.508

Table 3.11: A table showing the summary mean schedule cost and standard deviation of the successive augmentation procedures, varying the number of vertices, ranked from the smallest summary mean schedule cost to the largest for the real world problem set

for 4 days, however this increases to 10.629% with 6 days. The fifth best performing procedure is Smallest Sum Insertion, with a mean PSCI of 9.421%. In a similar pattern this procedure starts with a mean PSCI of 4.733% for 4 days, increasing to 14.18% for 6 days. The last procedure in this group is Farthest Insertion with a mean PSCI of 10.232%. This procedure also follows a similar pattern with a mean PSCI of 6.241% for 4 days, and 16.304% for 6 days.

The final group contains the two procedures that do not use the insertion expansion criteria. The first procedure in this final group is Nearest Addition, with a mean PSCI of 67.307%. The performance of this approach is consistent when increasing the number of days with a standard deviation of 1.454. It is clear from this that the addition expansion criteria is less effective than the insertion expansion criteria. Finally the worst performing of all of the procedures examined is Nearest Neighbour, with a mean PSCI of 74.382%. This also has a consistent performance when increasing the number of days with a standard deviation of 1.853. The neighbour expansion criteria performed far worse than the other two expansion criterias examined.

In Figures 3.15 and 3.16 we present the results of the empirical evaluation, using the successive augmentation procedures with the modified successive augmentation mechanism on the real world problem set. The results for this problem set follows a similar pattern as that observed in Section 3.4.3, with procedures that generate outlines and then fill in vertices performing the best such as Difference Insertion, Farthest Insertion, and Largest Sum Insertion. The next next best performing group are those procedures that identify ver-

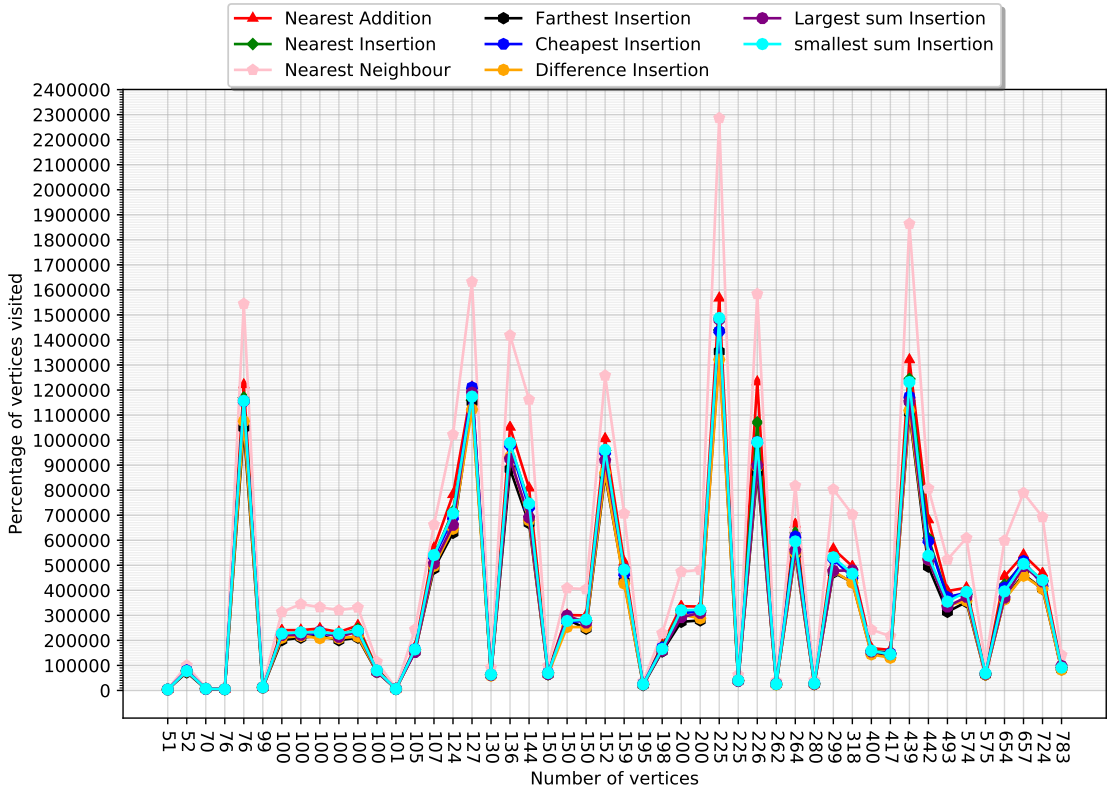


Figure 3.15: A percentage of vertices visited comparison of the selected successive augmentation procedures for the real world problem set, varying the number of vertices

Procedure name	Summary mean schedule cost	Standard deviation
Difference insertion	4001908.031	1009360.280
Farthest insertion	4174344.754	952353.422
Largest sum insertion	4259375.301	1015643.419
Cheapest insertion	4351373.214	1052569.612
Smallest sum insertion	4377738.803	983202.180
Nearest insertion	4393209.427	1061810.194
Nearest addition	4680802.728	1129738.263
Nearest neighbour	6793028.939	1935994.835

Table 3.12: A table showing the summary mean schedule cost and standard deviation of the successive augmentation procedures, varying the number of depots, ranked from the smallest summary mean schedule cost to the largest for the real world problem set

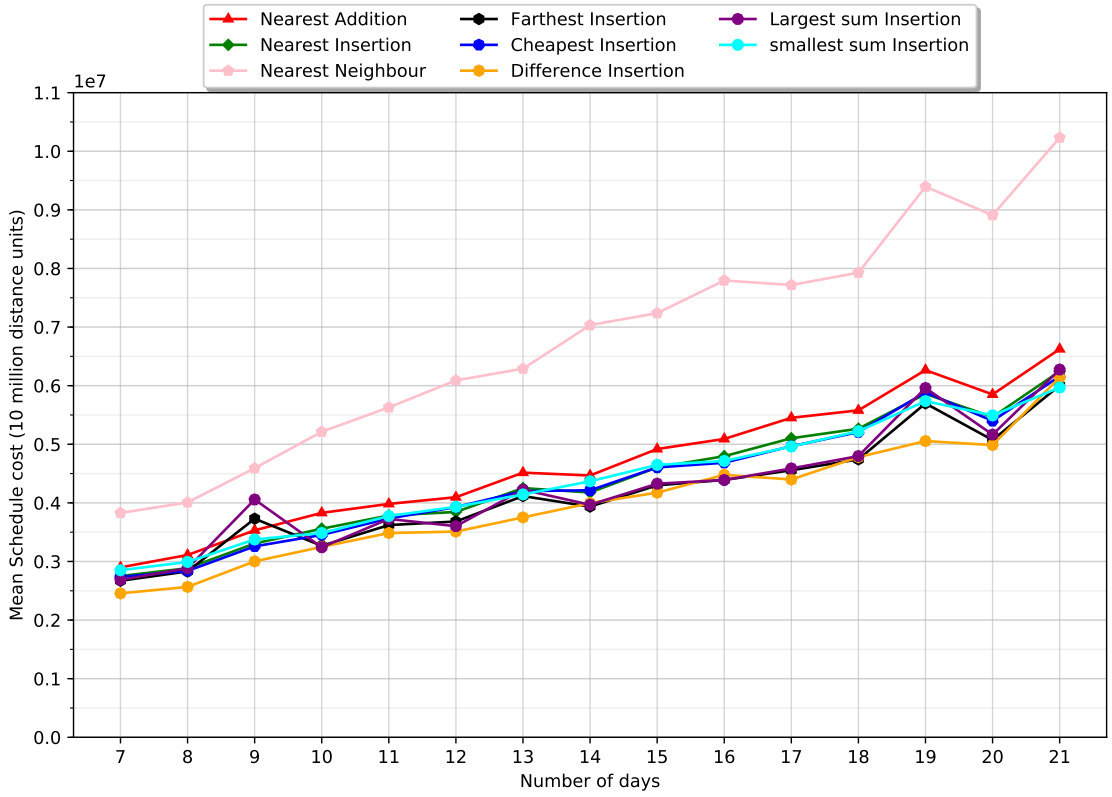


Figure 3.16: A percentage of vertices visited comparison of the selected successive augmentation procedures for the real world problem set, varying the number of days

tices that are geographically closest to the existing partial tour such as Cheapest Insertion, Smallest Sum Insertion, and Nearest Insertion. Procedures that do not use the insertion expansion criteria under perform in comparison with those that do, as Nearest Addition under performs in comparison to Nearest Insertion, with Nearest Neighbour performing worse than both.

3.6.4 Augmented results

Introducing the revisiting requirement ($\mathcal{R}_3$), and consequently generating multiple tours, adds additional factors that are not currently accounted for by the selection criteria designed for the TSP. We have identified two additional factors that we examine as augmentations to the existing selection criteria. The first is the cost of each tour which could be used in one of two ways. The first is to use a *Cheapest tour* preference, such that the cheapest tour that has available vertices in the potential vertex insertions is selected. The selection metric then performs as normal and selects the most appropriate of those vertices. The second is to use an *Expensive tour* preference, whereby the opposite occurs with the most expensive tour that has available vertices being selected.

The second additional factor is the size of the time window itself. This can also be used in two ways, the currently unplanned-for vertex that has the largest time window can be selected, and then the existing selection criteria can be used to determine which day and at which point in the tour that vertex should be inserted. We call this a *Largest time window* preference. The opposite is also possible, with the *Smallest time window* preference selecting the unplanned for vertex that has the smallest time window.

Each of the 8 selected procedures were augmented with the 4 preferences, resulting in 32 variants which were all executed on the same dataset as in Section 3.6.3. When comparing to the results in Section 3.6.3, we call the unmodified procedures *Default*. The mean performance of the each of the preferences in presented in Table 3.13a, this specifically refers to the mean performance of the preferences across the 8 procedures. A visualisation of the performance for the 4 preferences proposed in this Section can be seen in Figure 3.17. Each of these visualisations omits the Nearest Addition and Nearest Neighbour approaches as their performance was significantly worse than the others, and could not be visualised easily within the Figures whilst maintaining clarity. A representation of the mean PSCI performance for each procedure given the 4 preferences is provided in Table 3.14.

The best performing preference is Expensive tour, with a mean PSCI of 19.693%, as can

Procedure name	Mean PSCI
Expensive tour	19.693
Default	22.493
Small window	24.210
Large window	24.753
Cheapest tour	29.024

(a) Augmentation summary

Procedure name	Mean PSCI
Largest sum insertion	3.771
Cheapest insertion	3.919
Nearest insertion	3.999
Difference insertion	5.763
Smallest sum insertion	8.281
Farthest insertion	9.193
Nearest addition	74.324
Nearest neighbour	83.027

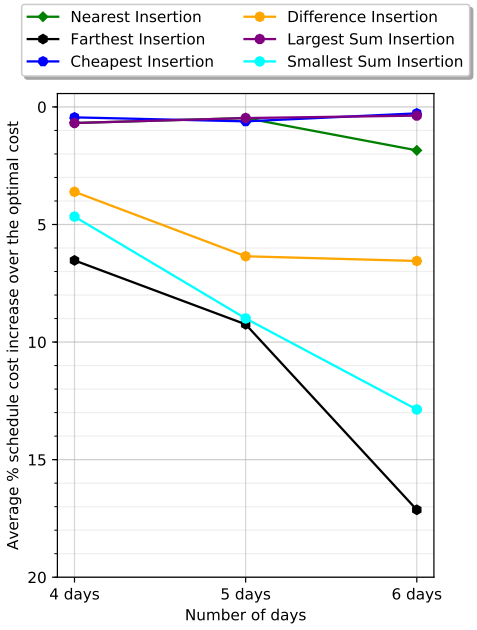
(b) Procedure summary

Table 3.13: The performance of the unmodified procedures (i.e. *Default*)

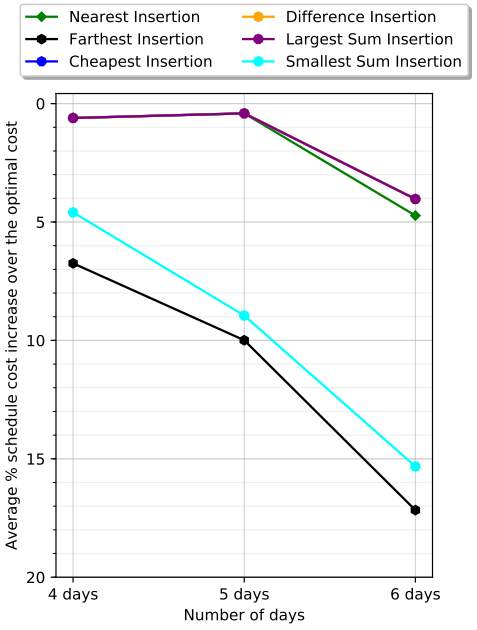
be seen in Figure 3.17d. This modification selects a tour and thus the problem becomes more similar to a TSP or OP. The best performing procedure using this preference is Farthest Insertion with a mean PSCI or 1.592%; this performance is broadly consistent when varying the number of days, with a standard deviation of 2.1. This is vastly different from how Farthest Insertion performs without the preference, as it had a default mean PSCI of 10.232%. Largest Sum Insertion has a similar strategy to Farthest Insertion as it generates an outline for a tour and then fills in the locations within that outline. It is the second best performing procedure with a mean PSCI of 1.593%, and a similar consistency with a standard deviation of 2.095.

The next best performing approach is Cheapest Insertion with a mean PSCI of 1.758%, which is better than it's default mean PSCI of 4.003%. As with the other procedures, it has a consistent performance when increasing the number of days, with a standard deviation of 1.928. Smallest Sum Insertion uses a similar strategy to Cheapest Insertion in that it attempts to generate small tours, by reducing the cost of additional insertions. The performance of these two procedures is also similar given the Expensive tour preference, as it has a mean PSCI of 1.767%. The fifth best performing procedure is Nearest Insertion, with a mean PSCI of 1.815, which is an improvement from it's default performance of 3.449%.

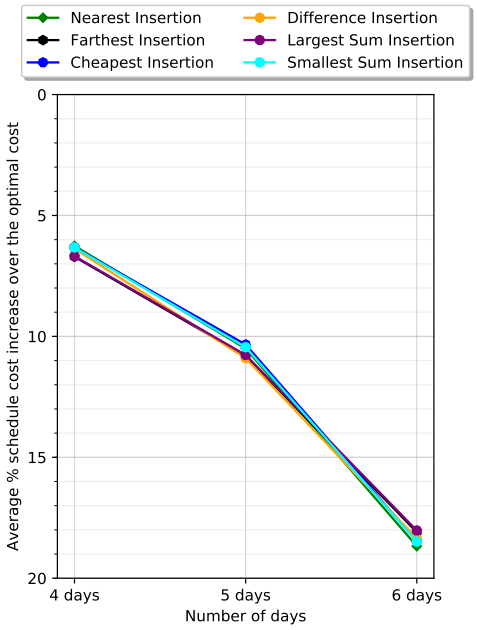
Difference Insertion did not perform as well as would be expected, given it's performance in Sections 3.4 and 3.5. It had a mean PSCI of 1.815%, and a standard deviation of 1.936. This is likely due to the small scale of the graph size, as Difference Insertion uses the difference between insertion costs as it's metric. Nearest addition had marginally



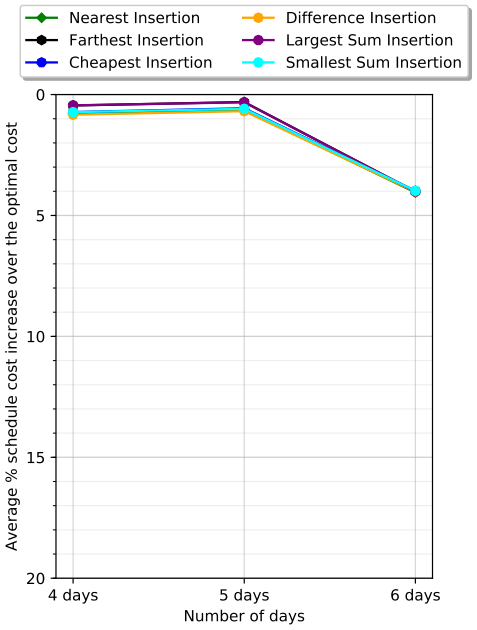
(a) A summary of performance with the small time window preference



(b) A summary of performance with the large time window preference



(c) A summary of performance with the cheapest tour preference



(d) A summary of performance with the expensive tour preference

Figure 3.17: A set of visualisations of the mean *Percentage Schedule Cost Increase* (PSCI) for each of the successive augmentation procedures across different planning horizons investigated, separated into a visualisation for each of the four preferences

Procedure name	Mean PSCI	Standard deviation
Cheapest insertion	0.444	0.170
Largest sum insertion	0.509	0.160
Nearest insertion	1.002	0.740
Difference insertion	5.504	1.644
Smallest sum insertion	8.843	4.104
Farthest insertion	10.969	5.509
Nearest addition	78.646	3.068
Nearest neighbour	87.766	6.280

(a) A summary of performance with the small time window preference

Procedure name	Mean PSCI	Standard deviation
Cheapest insertion	1.684	2.035
Difference insertion	1.684	2.035
Largest sum insertion	1.684	2.035
Nearest insertion	1.917	2.438
Smallest sum insertion	9.623	5.396
Farthest insertion	11.302	5.333
Nearest addition	80.189	3.236
Nearest neighbour	89.942	3.829

(b) A summary of performance with the large time window preference

Procedure name	Mean PSCI	Standard deviation
Cheapest insertion	11.706	6.189
Smallest sum insertion	11.753	6.186
Nearest insertion	11.811	6.311
Largest sum insertion	11.825	5.745
Farthest insertion	11.870	5.768
Difference insertion	11.878	6.052
Nearest addition	77.478	3.570
Nearest neighbour	83.873	3.586

(c) A summary of performance with the cheapest tour preference

Procedure name	Mean PSCI	Standard deviation
Farthest insertion	1.592	2.100
Largest sum insertion	1.593	2.095
Cheapest insertion	1.758	1.928
Smallest sum insertion	1.767	1.920
Nearest insertion	1.815	1.936
Difference insertion	1.848	1.884
Nearest addition	67.999	3.653
Nearest neighbour	79.174	4.252

(d) A summary of performance with the expensive tour preference

Table 3.14: A summary of the mean *Percentage Schedule Cost Increase* (PSCI) for each of the successive augmentation procedures across different planning horizons investigated, separated into tables for each of the four preferences using the PTSP-VDTW

worse performance with the Expensive Tour preference when compared so the default performance, with a mean PSCI of 67.999% against 67.307%. Nearest Neighbour also had a worse performance with the Expensive Tour preference, with a mean PSCI of 79.174%.

The remaining preferences had a mean performance across the procedures that were worse than the unmodified performance of 22.493%. The second best performing preference was Small Windows with a mean PSCI of 24.493%. The best three performing procedures are grouped closely together, with a similar mean PSCI and standard deviation. Cheapest Insertion and Nearest Insertion aims to create small tours using the cost and distance respectively, however Largest Sum Insertion uses the outline and fill in mechanism to create tours. These are Cheapest Insertion, Largest Sum Insertion, and Nearest Insertion

which were 3 procedures that performed well given the conditions of the OP in Section 3.5. Cheapest Insertion performed best with a mean PSCI of 0.444%, which was unaffected by an increase in the number of days with a standard deviation of 0.17. The second best performing in this group is Largest Sum Insertion with a mean PSCI of 0.509% and again a consistent performance when increasing the number of days with it's standard deviation of 0.16. The third best performing was Nearest Insertion with a mean PSCI that is marginally worse than the second best performing with 1.002%. The standard deviation for this procedure was 0.74.

The following three procedures are less closely grouped than the former three, with the fourth best performing procedure being Difference Insertion with a mean PSCI of 5.504%. The procedure is not significantly effected by increasing the number of days with a standard deviation of 1.644. This is followed by Smallest Sum Insertion with a mean PSCI of 8.843%, however the performance is much more significantly effected by an increase in the number of days. With 4 days the mean PSCI is 4.665% which increases significantly to 12.868% for 6 days. This is similar to this procedure's performance during the TSP and OP conditions. The next best performing is Farthest Insertion, which is the first of these procedures which has a worse performance when using the Small Window preference, which mirrors the performance pattern seen in Section 3.5 and indicates that when using this preference the selection criteria perform similarly to the The mean PSCI is 10.969% which is marginally worse than it's default performance of 10.232%. The performance is also significantly effected by an increase in the number of days, with a mean PSCI of 6.527% for 4 days, which increases to 17.133% for 6 days.

The two other expansion criteria are significantly impacted by using the Small Window preference. Nearest Addition has a mean PSCI of 78.646% which is significantly higher than the default performance of 67.307%. The standard deviation was particularly high with 3.068. The worst performing procedure was Nearest Neighbour, which has a mean PSCI of 87.766% and as with nearest neighbour is significantly worse than it's default performance of 74.382%. The standard deviation was also the worst with 6.28.

The third best preference is Large Window, with a mean PSCI for the 8 procedures of 24.753%. Three procedures all performed identically across the problem set, which are Cheapest Insertion, Difference Insertion, and Largest Sum Insertion. The mean PSCI is 1.684% with a standard deviation of 2.035, and the performance for each of the procedures is better than their default performance. These procedures had an almost identical performance across the dataset, with only a small number of differences between the approaches.

The fourth best performing procedure is Nearest Insertion, which performs similarly to the previous 3 procedures with a mean PSCI of 1.917% and a standard deviation of 2.438. These 4 procedures behave similarly when the preference is for Large Windows instead of Small Windows due to the order in which the vertices are selected. This is because the Small Window preference attempts to assign all of the vertices that will have the fewest range of days that they may be assigned to, then vertices that have a larger range are assigned around them based on the existing tours. The Large Window preference attempts to assign the vertices with the largest range first, however the partial tours will be less populated, and therefore it is less likely to be an optimal position. As the 4 procedures have the same expansion criteria, these procedures do not result in significantly different sets of tours.

The Smallest Sum and Farthest Insertion procedures have similar performance with both the Large and Small Windows preference. Smallest Sum has a mean PSCI of 9.623% which is marginally worse than it's default performance of 9.421%. With 4 days the mean PSCI is 4.594% and this increases to 15.323% for 6 days. The performance of Farthest Insertion procedure with the Large Windows preference is worse than it's default performance with a mean PSCI of 11.302% when compared to the 10.232% of default. The procedure is also significantly effected by an increase in the number of days, with 4 days having a mean PSCI of 6.745% which increases to 17.167% for 6 days.

As with the Small Window preference, the non-insertion expansion criteria perform worse than the insertion procedures. Nearest addition has a mean PSCI of 80.189% and a standard deviation of 3.236. Finally Nearest neighbour performs the worst out of all of the procedures with a mean PSCI of 89.942% and a standard deviation of 3.829.

As with the Expensive Tour preference, the performance of the procedures are all close with the first 6 procedures having a mean PSCI within 1% of the best and worst. The performance of all 8 procedures are however worse than their default counterparts. The number of days vastly effect of the mean PSCI, with 4 days having a PSCI of between 6.257% and 6.707%, and between 18.025% and 18.674% for 6 days. The best performing of these procedures is Cheapest Insertion with a mean PSCI of 11.706%. This is closely followed by Smallest Sum Insertion with a mean PSCI of 11.753%. The thirdly best performing procedure is Nearest Insertion with a mean PSCI of 11.811%. These three procedures all aim to produce small tours by selecting vertices that are close to vertices in the partial tour. The fourth best performing procedure is Largest Sum Insertion with a mean PSCI of 11.825%. This is followed by Farthest Insertion, with a mean PSCI of

11.87%. Both of these procedures aim to generate initially large tours and then fill in the remaining vertices at the best possible positions. The worst performing of the insertion procedures is Difference Insertion, with a mean PSCI of 11.878%. Finally both of the non-insertion procedures performed worse than the insertion procedures. Nearest Addition had a mean PSCI of 77.478% and Nearest Neighbour had a mean PSCI of 83.873%.

3.7 A comparison of successive augmentation procedures for the Periodic Maintenance Person Problem

The Periodic Maintenance Person Problem (PMPP), as formalised in Section 3.2, produces a set of tours, each associated with a day in the planning horizon, where the cost of each tour is within the tour cost limit and the revisiting constraint for each non-depot location is satisfied while minimising the total cost of all tours. In this section we discuss a modification to the successive augmentation mechanism that expands upon the modifications made to the mechanism in Sections 3.5 and 3.6. In Section 3.5 modifications were made to the selection metric, such that a selection that brings the total tour cost over the tour cost limit could not be made. In Section 3.6 changes were made to the mechanism to terminate only when a valid schedule is produced, and a data structure was proposed called the *potential visits* to ensure that each insertion increased the coverage of the vertices. For the PMPP however, these two modifications would not be sufficient to solve the problem as a situation may occur when an incomplete schedule does not have remaining space to entirely plan for a vertex. In this situation a backtracking mechanism can be employed to determine if different selections during previous iterations would allow for a valid schedule to be created. The method we have chosen is a modification of Branch and Bound [48], which we call *Search and Bound*.

Branch and bound uses a tree structure to explore a given search space in such a way as to quickly find a valid solution. Then areas of that search space that are guaranteed not to have better solutions are bounded off and the remaining space explored until the optimal answer is found. For Search and Bound a similar process is carried out, with the space explored and invalid portions bounded off, however the process terminates once a valid solution is found to the problem. We describe the process in detail in Section 3.7.1. We empirically evaluate the performances of the procedures and their variants as described in Section 3.6.1. This is done by testing the average percentage tour cost increase on the

same graphs as used in Section 3.6, however with a tour cost limit of 60% the optimal TSP tour cost for a given graph. We discuss our experiment settings in Section 3.7.2 and present our results in Section 3.7.3.

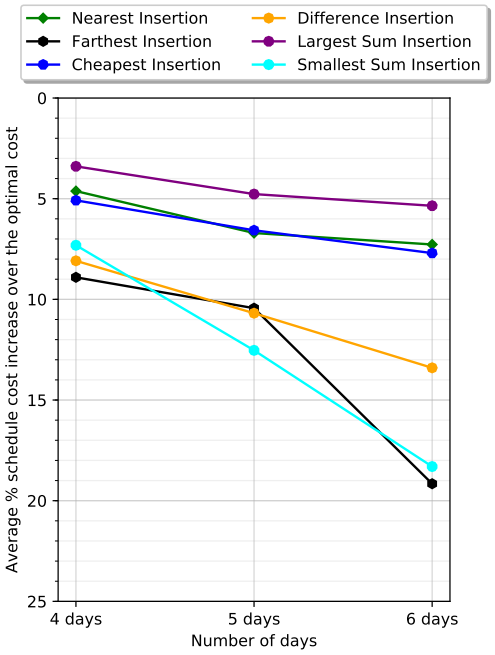
3.7.1 The Search and Bound method

The Search and Bound method is an adaption of the Branch and Bound method and can be viewed as a depth first search with a bounding mechanism [48]. The Branch and Bound method starts with an initial problem state and explores the problem state systematically. For example the branch and bound for the TSP starts at a given node, then explores all potential selections from that given node as separate branches on the data tree. The selection that is the best, given some selection criteria is then explored until a viable solution is found. Then every branch on the tree that is guaranteed to perform worse if explored is bounded off, this is often done using the lower bound of a selection.

Branch and search carries out a similar process, starting with the initial state of the problem, then selections from the possible vertex insertions data structure are ranked and explored. If at any point a vertex, that does not have full coverage, cannot be inserted in any of the tours without violating the tour cost limit, then that incomplete schedule will never be viable. As the triangle inequality holds for the graph, it is not possible that adding another vertex to those tours will allow the vertex to fit. Therefore a backtracking mechanism is used, whereby the current state is reverted to that of the parent node, and the next best selection is made.

3.7.2 Experimental Methodology

The same two problem sets, as used in Section 3.6, were reused in this section in order to provide a direct comparison between the PTSP and PMPP. The PMPP formalisation has two additional features that the PTSP does not have, which are a tour cost limit, and a service time for each location. In order to ensure that the results are more directly comparible, the service time for each non-depot vertex is 0 for this empirical analysis. The small scale problem set contains graph sizes of 6 vertices, with the number of days varying from 4 to 6. The real world problem set is also reused, with the same problem instances used from TSPLIB, with sizes ranging from 51 to 1000. The tour cost limit for each graph is set to 80% of the optimal TSP cost for that graph, in our testing we found that 100% rarely engaged the backtracking mechanism. The implementation of the modified



(a) A PSCI comparison for the selected successive augmentation procedures, not plotted are the results for Nearest Addition and Nearest Neighbour

Procedure name	Mean PSCI	Standard deviation
Largest sum insertion	4.506	1.006
Nearest insertion	6.200	1.396
Cheapest insertion	6.451	1.314
Difference insertion	10.725	2.658
Smallest sum insertion	12.716	5.498
Farthest insertion	12.834	5.530
Nearest addition	75.304	1.824
Nearest neighbour	82.684	1.595

(b) A table to show the average performance of each procedure tested, sorted by the percentage tour cost increase

Figure 3.18: The performance of different successive augmentation procedures when used with the MPP with different planning horizons

successive augmentation mechanism was done using python 3 on a windows 10 laptop with Intel Core i7-8750H CPU @ 2.20GHz.

3.7.3 Results

The results of our empirical testing is shown in Figure 3.18a and the average performance of each of the procedures is shown in Table 3.18b. Six of the eight procedures tested were insertion procedures, and these six procedures outperformed the addition and neighbour procedures, we therefore do not plot these procedures in Figure 3.18a. This is due to the difference between the worst performing insertion procedure and the best performing non-insertion procedure being 43%, which skews the scale of the diagram. For each of the procedures, an increased number of days in the planning horizon increases the difference

between the produced schedules and the optimal schedule costs.

The best performing procedure was largest sum insertion, with an average PSCI of 4.5%. The average PSCI for the 4 day problem set was 3.4%, increasing to 5.3% for the 6 day problem. Rosenkrantz et al determined that the strategy of selecting vertices farthest away from the vertices in the current partial tour performs well, in the context of the TSP, because it generates an outline of a tour then fills in the rest of the vertices given more information about the partial tour [83]. This strategy is also taken by the sixth best performing procedure which was farthest insertion. This procedure has an average PSCI of 12.8%, with the 4 day problem set having a PSCI of 8.9% increasing to 19.1% for the 6 day problem set. Farthest insertion uses a single edge to determine the selected vertex, however largest insertion uses the edge costs to each vertex in the partial tour. This indicates that for the farthest away strategy, a selection criteria that takes into account more information produces lower costing schedules and is less affected by an increased planning horizon.

The second best performing procedure was the Nearest Insertion procedure with an average PSCI of 6.2%, the PSCI from the 4 day problem set is 4.6% which increases to 7.2%. The strategy used by this approach is to identify which vertices are closest to the existing partial tour, with the intuition being that adding the vertex that is closest will add the lowest overall additional tour cost. This strategy is also used by smallest sum insertion, which is the fifth best performing procedure, and has average PSCI of 12.7% The PSCI of 7.3% for the 4 day problem set increases to 18.3% for the 6 day problem set. In contrast with the strategy of selecting vertices that are farthest away, finding the vertex that is closest on average to all vertices in the partial tour performs worse as the vertex selected is not closest to any one point in the tour. For this strategy the former approach is also less affected by an increasing planning horizon.

The third best performing procedure was the cheapest insertion with an average PSCI of 6.2%. The strategy employed by this procedure is to identify the vertex that would minimise the additional tour cost, in this way the locally optimal choice is made. The average PSCI for the 4 day problem set was 5%, increasing to a 7.7% PSCI for the 6 day problem set. Difference insertion is an extension of the cheapest insertion procedure that uses the difference between the cheapest insertion and the second cheapest insertion costs as a selection criteria. It performed worse then the cheapest insertion procedure with an average PSCI of 10.7%, which indicates that the actual cost of insertion is a better metric than the opportunity cost in the context of the PMPP.

Nearest Neighbour and Nearest Addition have a different expansion metric to the other

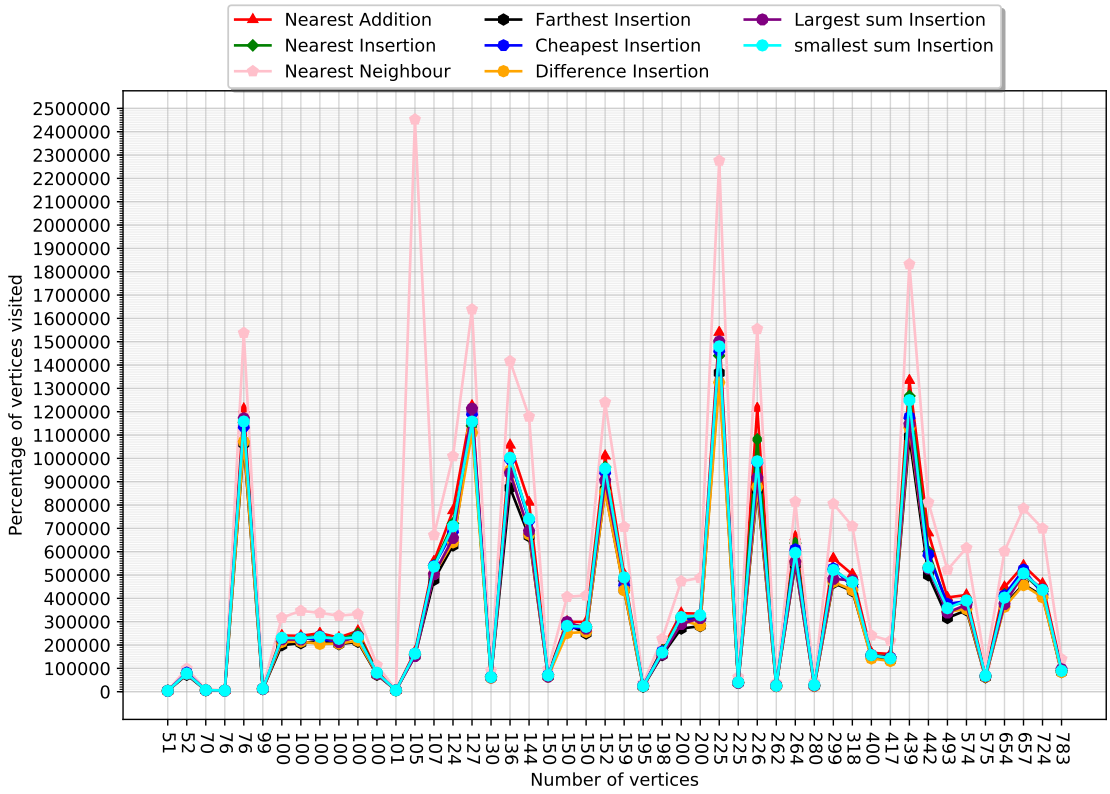


Figure 3.19: A percentage of vertices visited comparison of the selected successive augmentation procedures for the real world problem set, varying the number of days

procedures. As these procedures both perform significantly worse than the worst performing insertion procedure, this indicates that minimising the additional tour cost produces better quality tours and the insertion expansion criteria should be used. Nearest addition is the seventh best performing procedure, with an average PSCI of 55.3%, with nearest neighbour being the worst performing procedure, with an average PSCI of 82.6%.

The results for the real world problem are presented in Figure 3.19, and Table 3.15. These results follow a similar pattern demonstrated in Section 3.6, with the best performing procedures being those that first generate the general outline of a tour and then fill in the remaining vertices. For the PTSP, Difference Insertion outperformed Farthest Insertion however this does not apply here, which may indicate that the Difference Insertion selection criteria is less effective with the introduction of a tour cost limit. Difference Insertion does

Procedure name	Summary mean schedule cost	Standard deviation
Farthest insertion	346337.229	345578.163
Difference insertion	350209.396	346548.368
Largest sum insertion	369025.625	369967.288
Cheapest insertion	377789.062	374414.904
Smallest sum insertion	380724.771	378667.240
Nearest insertion	384752.979	380807.051
Nearest addition	411518.042	408076.245
Nearest neighbour	601754.646	614168.361

Table 3.15: A table showing the summary mean schedule cost and standard deviation of the successive augmentation procedures, varying the number of depots, ranked from the smallest summary mean schedule cost to the largest for the real world problem set

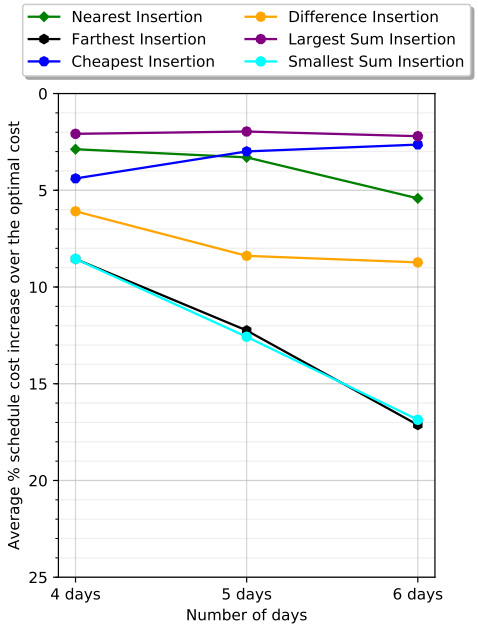
still outperform Cheapest Insertion, from which it is extended and therefore is still a beneficial change to the selection criteria. Nearest Neighbour performs worse under these conditions than in the PTSP, and therefore the expansion criteria may play a more crutial role in this problem than others.

3.7.4 Augmented results

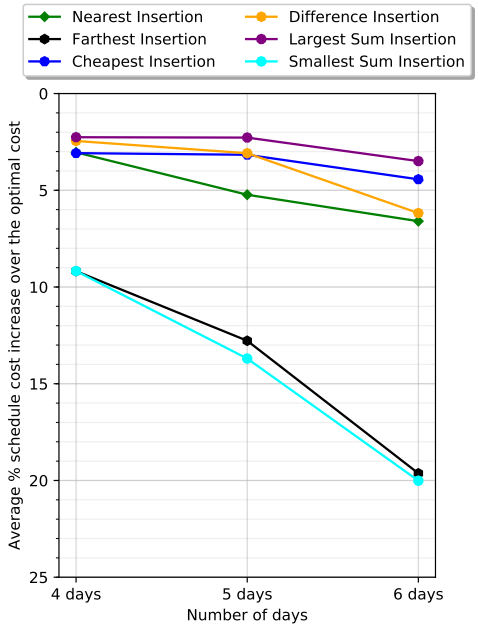
The augmented results, as presented as a visualisation in 3.20 and as a set of summary tables in 3.16, follow a similar pattern as those presented for PTSP-VDTW. The expensive tour preference out performed the default performance for a majority of the procedures.

3.8 Conclusion

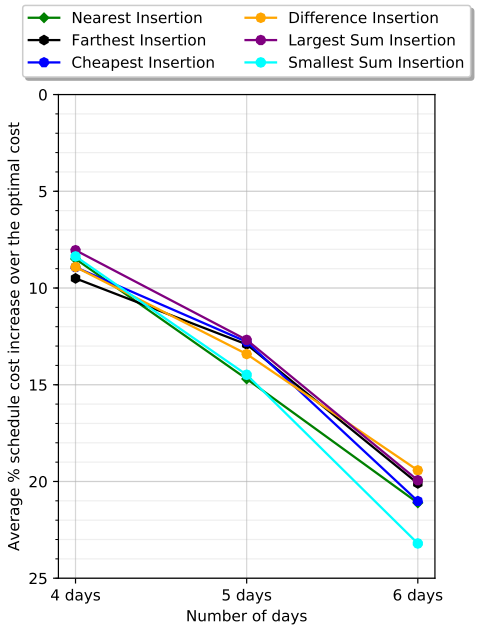
In this chapter we have described and formalised the Periodic Maintenance Person Problem (PMPP), and discuss our reasoning for developing a heuristic algorithm. A systematic approach was adopted to the generation of a tour construction mechanism for the PMPP, based on the Successive Augmentation Mechanism for the TSP. The PMPP has two additional features when compared to the TSP, which are the tour cost limit and the visit dependant time window revisiting constraint. We have carried out a number of different empirical studies for the TSP, and extensions of the TSP with each of those additional features, the OP and PTSP. Finally we have discussed the modifications required to accommodate both the additional features of the PMPP, and have empirically evaluated its



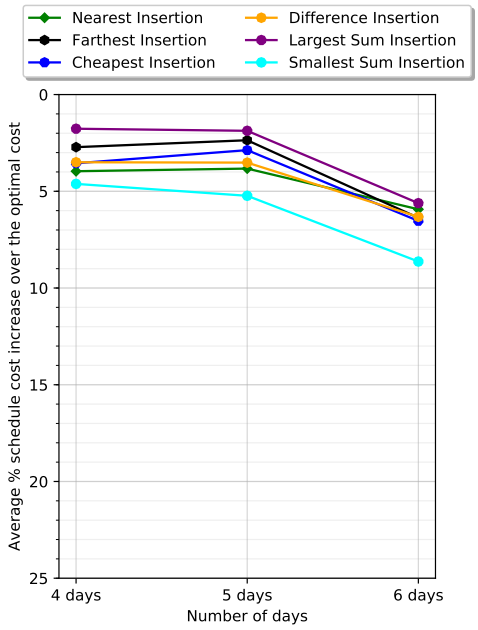
(a) A summary of performance with the small time window preference



(b) A summary of performance with the large time window preference



(c) A summary of performance with the cheapest tour preference



(d) A summary of performance with the expensive tour preference

Figure 3.20: A set of visualisations of the mean *Percentage Schedule Cost Increase* (PSCI) for each of the successive augmentation procedures across different planning horizons investigated, separated into a visualisation for each of the four preferences

Procedure name	Mean PCTI	Standard deviation
Largest sum insertion	2.083	0.121
Cheapest insertion	3.340	0.925
Nearest insertion	3.865	1.361
Difference insertion	7.734	1.437
Farthest insertion	12.642	4.301
Smallest sum insertion	12.661	4.158
Nearest addition	85.975	2.976
Nearest neighbour	96.483	5.739

(a) A summary of performance with the small time window preference

Procedure name	Mean PCTI	Standard deviation
Largest sum insertion	2.676	0.707
Cheapest insertion	3.559	0.758
Difference insertion	3.905	1.995
Nearest insertion	4.956	1.795
Farthest insertion	13.863	5.309
Smallest sum insertion	14.295	5.443
Nearest addition	86.937	4.349
Nearest neighbour	103.118	0.856

(b) A summary of performance with the large time window preference

Procedure name	Mean PCTI	Standard deviation
Largest sum insertion	13.556	5.993
Difference insertion	13.917	5.275
Farthest insertion	14.172	5.409
Cheapest insertion	14.239	6.176
Nearest insertion	14.767	6.305
Smallest sum insertion	15.351	7.457
Nearest addition	84.450	2.793
Nearest neighbour	93.812	4.035

(c) A summary of performance with the cheapest tour preference

Procedure name	Mean PCTI	Standard deviation
Largest sum insertion	3.085	2.191
Farthest insertion	3.818	2.219
Cheapest insertion	4.319	1.946
Difference insertion	4.446	1.624
Nearest insertion	4.574	1.177
Smallest sum insertion	6.161	2.161
Nearest addition	74.774	2.290
Nearest neighbour	87.198	4.585

(d) A summary of performance with the expensive tour preference

Table 3.16: A summary of the mean *Percentage Schedule Cost Increase* (PSCI) for each of the successive augmentation procedures across different planning horizons investigated, separated into tables for each of the four preferences using the PMPP

performance.

We have shown that for euclidean instances of the TSP and OP, Difference Insertion performs consistently better than other procedures, having an average tour cost increase from the optimal tour of 4.069% in the case of the TSP, and visiting 96.323% of all vertices for the OP. The Smallest Time Window variant of the Cheapest Insertion procedure performed best for the PTSP-VDTW, with an average percentage tour cost increase of 0.444%. In contrast to this, the Smallest Time Window variant of the Largest Sum Insertion performed best for the PMPP, with an average percentage tour cost increase of 2.083%.

Chapter 4

Multiple Travelling Salesperson Problem

4.1 Introduction

Each day a quality control company dispatches a group of geographically disparate workers (AKA *Samplers*) to perform tests at various *service locations* in the company's designated area. Each sampler starts from their home location, called a *depot* and visits a subset of service locations before returning to the depot, this is called a *tour*. A *scheduler* is responsible for creating a *schedule* for the workforce, which is an ordered list of locations for each Sampler to visit. The scheduler may aim to optimise for several different metrics, such as: total distance travelled by all Samplers, a balanced number of locations for each Sampler to visit, etc. The most common of these metrics is to minimise the maximum time worked by a sampler, as minimising this metric tends to balance the workload more equally across the workforce. A majority of sampling companies employee salaried workers, and their workload per day does not exceed a couple of hours, and therefore a maximum number of hours per tour is not imposed on the problem.

This problem closely maps onto the Multiple Travelling Salesperson Problem (MTSP), which is the broad term for the family of problems where multiple salesman each have a distinct tour that visits each of the locations exactly once [16]. The specific variant of the MTSP that we have chosen is the Multiple depot MinMax MTSP, which states that each salesman has a unique depot location and that the aim is to minimise the longest tour. As this problem is NP-hard, there exists a significant body of work in the literature that

is dedicated to providing heuristic algorithms for the problem [102]. A wide variety of different mechanisms have been explored, and recently a graph partitioning approach has been introduced by Vandermeulen et al [94]. The MTSP often models the problem as a graph, with the vertices corresponding to the locations and the edges weights corresponding to the travel distance or travel time between between the relevant locations. A graph partitioning approach to the MTSP first divides the graph into a set of distinct subgraphs, with each consisting of single depot and a set of locations to visit, then this subgraph is solved as a Travelling Salesperson Problem (TSP).

One contributing factor to why the graph partitioning approach has not received prior attention in the MTSP literature, is that determining the cost of a tour from a subgraph requires solving the NP-Hard TSP. Vandermeulen et al found that there is a correlation between average length of a graph and it's optimal tour cost, and as finding the average length of a graph can be found in $O(n^2)$ it is therefore a suitable proxy. In their paper, the authors present a greedy heuristic approach that performs a series transfers and swaps of vertices between subgraphs, until there does not exist a transfer or swap that would further improve the subgraph costs given the optimisation metric. The author's literature review discusses other MTSP algorithms, however does not discuss the existing graph partitioning literature, which we believe is a fruitful area for adaptation and further work.

In this chapter we take several selected graph partitioning algorithms and modify them to be suitable for use for the MTSP, in order to determine which characteristics of the algorithms produce high quality tours. There are two broad categories of graph partitioning mechanisms, which are: (1) mechanisms that take in a graph and output a set of subgraphs which we call *Construction mechanisms*; and (2) mechanisms that take in a set of subgraphs and exchange vertices between these subgraphs to optimise for a given metric which we call *Improvement mechanisms*.

In Section 4.2 we discuss the selected graph partitioning algorithms and deconstruct them into a set of Construction and Improvement mechanisms. We then provide a description of how these mechanisms have been modified to be suitable for the MTSP. In Section 4.3 we perform an average case empirical analysis on both the Construction and Improvement mechanisms proposed in Section 4.2, and provide a discussion of the characteristics of these methods that produce higher quality tours. Finally in Section 4.4 we discuss the overall performance of graph partitioning mechanisms for the MTSP, and the potential future work.

Algorithm name	Construction phase	Improvement phase
Vandermeulen et al [94]	Random partition	Transfer and swap, Transfer outliers
Kernighan-Lin [50]	Random partition	Swap and lock
Fiduccia-Mattheyses [28]	Random partition	Transfer and lock
Farhat [27]	Breadth first search	None
Gain [73]	Cut size change	None
Bubble (Tschöke) [73]	Simultaneous Breadth first search	Repeated seed changing partitioning

Table 4.1: A table showing the selected graph partitioning mechanisms, decomposed into Construction and Improvement mechanisms

4.2 Selected graph partitioning mechanisms

Vandermeulen et al presented an approach to the Multi-depot MinMax MTSP, in this section we generalise this approach into a framework for using graph partitioning mechanisms [94]. The general framework has two stages, the first is to divide the problem graph into a series of subgraphs, with each subgraph containing a single unique depot. The second stage is to treat each of these subgraphs as a TSP in order to provide a solution to the MTSP when combined. A graph partitioning algorithm can be decomposed into a collection of mechanisms, with those mechanisms classified as either *Construction* or *Improvement*. A construction mechanism takes in a graph and returns a set of distinct subgraphs, and an improvement mechanism takes in a set of subgraphs and returns a set of subgraphs optimised for a particular metric.

Using the framework, unmodified construction mechanisms can be used in a MTSP algorithm however as they optimise for minimising the number of edges between subgraphs they may not optimise in a way that lines up with the objectives of the MTSP. There are two modifications required to ensure that a graph partitioning mechanism is suitable for the Multiple Depot MinMax MTSP. The first is that a distinct depot must be assigned to each subgraph and cannot be transferred to another subgraph. The second is that the optimisation metric should more closely reflect the optimisation function of the MTSP. Vandermuelen et al found that there is a correlation between the average length of a graph and it's optimal tour cost, and as the average length is computationally less expensive to compute than the optimal tour cost this can be used as a proxy and therefore optimised for.

Several broad categories of graph partitioning exist, with one category being geographic based approaches [8]. 5 algorithms were selected from a survey by Preis and Diekmann [73] that specifically discusses this category of graph partitioning algorithms. This is therefore not an exhaustive list of all graph partitioning mechanisms that are suitable for modification for the MTSP, however it is instead intended to illustrate that this conversion and modification process is possible. Each approach is decomposed into a collection of Construction and Improvement mechanisms, which we outline in Table 4.1. These graph partitioning mechanisms do not produce subgraphs that optimise for either the MinSum or MinMax objective functions, but instead aim to minimise the number of edges between partitions and therefore modifications are required for each mechanism. As discussed in Chapter 2, the term *balanced* in a graph partitioning context refers to generating partitions that have as equal a number of vertices as possible in each subgraph. For example if there are 10 vertices being partitioned into 2 subgraphs then each should have 5 vertices, however if 10 vertices is partitioned into 3 subgraphs then the first subgraph has 4 vertices and the second and third have 3. Where possible we will provide a variant to a modified mechanism that both includes and does not include this constraint to determine it's effectiveness for the purposes of the MTSP. In Section 4.2.1 we discuss which construction mechanisms were selected, and the modifications required, and Section 4.2.2 similarly discusses the improvement mechanisms and their modifications.

4.2.1 Construction mechanisms

From the selected algorithms, 3 unique construction mechanisms were identified: Random partitioning, Breadth first search, and Cut size change. There can be several variants for each of these mechanisms, which we call *mechanism variants*, and these are presented in Table 4.2. The Random partitioning is described in two different ways in the selected algorithms. The first method randomly assigns each vertex to a subgraph, giving an equal probability to each of the subgraphs which results in an unbalanced partition. The second randomly assigns a fixed number of vertices to a subgraph before moving on and repeating the process for the next subgraph. This fixed number is as close as possible to an equal distribution of vertices as possible and therefore the subgraphs are balanced. Both of these approaches can be modified to accommodate a depot vertex by first populating each subgraph with a distinct depot, and then using the appropriate method to complete the subgraphs. Due to the random nature of this approach a metric is not used, and therefore

in this instance the second modification is not required. We call these methods *Random partitioning (balanced)* and *Random partitioning (unbalanced)*.

The second mechanism is Breadth first search, which generates a search tree with the vertex with the minimum degree being used as the root. A tree structure is then generated from this root node that selects the next vertex that is closest to any vertex currently in the tree, and connects it to the tree by that edge. This process repeats until a fixed number of vertices have been selected, which corresponds to as equal number of vertices per subgraph as possible. The next currently unselected vertex with a minimum degree to other unselected vertices is then selected as the next root node and the process is repeated. These trees can be turned into a set of subgraphs which will therefore be balanced. As the graphs we consider are complete and therefore using the degree to select the root vertex would not be possible. Instead we assign a distinct depot vertex to each of the subgraphs. As the graphs used by the MTSP is complete, using the minimium degree is not suitable, and therefore the minimum total cost for a vertex to each vertex in a tree can be used [86]. The tie breaking condition of the first vertex in the set is used. We call this modified mechanism *Breadth first search (local, sequential)*. It is local as it selects vertices based on the cost of the unvisited vertices to the currently examined tree. It is sequential in the manner in which each subgraph is examined, and it is balanced as it results in a balanced number of vertices in each subgraph.

Another variant of this mechanism in the selected algorithms is the Simultaneous breadth first search. A set of seed vertices are selected, and instead of each tree being built sequentially, each tree selects a vertex in rounds. This process still results in the same number of vertices being distributed amongst the trees as the standard Breadth first search, and is therefore also balanced. Each tree has a unique identifier, and if two trees happen to select the same vertex then the tree that has the earlier identifier selects it and the other moves on to it's next best choice. In a similar way to the main variant, this process can be modified with each of the depots being used as the seed vertices, and the metric does not require modification. We call this mechanism the *Breadth first search (global, simultaneous)*, as it considers costs across the entire graph and in a simultaneous manner and results in a balanced set of subgraphs.

The simultaneous variant selects subgraphs in rounds, however another modification is that instead of building each subgraph simultaneously, only the best selection is made each round. This process could have an equal number of vertices in each subgraph by only considering those subgraphs that are not at a balanced capacity, however the mechanism

allows for this constraint to be dropped and therefore we separate this into two variants. The first is called *Breadth first search (global, sequential, balanced)*, as it considers which selection would be best global and produces a balanced set of subgraphs. The other variant is called *Breadth first search (global, sequential, unbalanced)*, and is the same as the former except it does not ensure each subgraph has an equal number of vertices.

The final mechanism is Cut size change, which starts with a set of seed vertices and iteratively selects a vertex to add to each subgraph. Each of these subgraphs start with a cut size, which is the number of edges connecting this subgraph to all of the others. The vertex that would result in the smallest increase in cut size is selected and added to the relevant subgraph until all vertices have been assigned. This procedure can be modified with two changes, the first is to use each of the depots as the seed vertices, the second is to use the average length metric instead of cut size as selection. There are two different methods for this, the first is that the vertex and subgraph is selected with the minimial additional cost. Which we call *Average length change (local cost)*, as it is related to the average length and not the cut size and it takes into account the local additional cost. The second method is to select the vertex and subgraph that would result in the minimal largest subgraph cost, which we call *Average length change (subgraph cost)*.

4.2.2 Improvement mechanisms

The unique mechanisms identified from the selected algorithms were: Transfer and swap, Transfer outliers, Swap and lock, Transfer and lock, and Repeated seed changing partitioning. In contrast with Section 4.2.1, a balanced variant does not apply to an improvement mechanism and therefore each mechanism simply has a modified mechanism instead of a mechanism variant. A summary of the modified improvement mechanisms is provided in Table 4.3. Vandermuelen et al proposed two improvement mechanisms, the first being the *Transfer and swap* procedure. This mechanism works by iterating through each pair of subgraphs, transferring and swapping vertices between subgraphs that minimise the maximum average length of each pair of subgraphs. A transfer moves a vertex from subgraph i to j, whereas a swap simultaneously moves a vertex from i to j and a vertex from j to i. An eligible transfer or swap is one which reduces the maximum average length of the pair of subgraphs. The procedure begins by iterating through each unique pair of subgraphs. It was observed by Vandermuelen et al that for each pair there are fewer eligible transfers than swaps, therefore the process starts by evaluating the cost of each possible transfer.

Mechanism name	**Description**
Random partitioning (balanced)	As equal a number of vertices as possible are randomly assigned to each subgraph
Random partitioning (unbalanced)	Each vertex is randomly assigned to a subgraph, with no limits on the number of vertices in each subgraph
Breadth first search (local, sequential)	A tree structure is generated from the first depot with as equal a number of vertices as possible selecting the nearest unselected vertex. This process is repeated for each depot until all vertices have been assigned
Breadth first search (global, sequential, balanced)	Similar to Breadth first search (local, sequential, balanced), however each round a vertex and tree is selected instead of only building a single tree, as equal a number of vertices as possible is assigned to each tree
Breadth first search (global, sequential, unbalanced)	The same as Breadth first search (global, sequential, balanced) however there is no constraint on the number of vertices in each tree
Breadth first search (global, simultaneous)	The same as Breadth first search (global, sequential, balanced) however each round a vertex is assigned to each tree
Average length change (local cost)	Starts with a depot vertex in each subgraph, and the vertex and subgraph are selected where the additional cost to the average length of the subgraph is minimal
Average length change (subgraph cost)	The same as Average length change (local cost) however the vertex and subgraph are selected that results in the minimal cheapest subgraph average length

Table 4.2: A brief description of each of the construction mechanism variants

The best transfer is one which most reduces the maximum average length of the pair, and this transfer is carried out and the cost of each transfer is re-evaluated. This process repeats until there no longer exists a transfer that would reduce the maximum average length of the pair. Then a similar process occurs with the swaps, evaluating the cost and carrying out the best until there no longer exists an eligible swap. Once all of the swaps have been carried out the entire process of transfers and swaps is repeated until there does not exist a transfer or swap that would reduce the average length. This process iterates through the next pair of subgraphs until it has iterated through each unique pair and there does not exist an eligable swap or transfer, at which point the process terminates.

Evaluating the cost of each transfer and swap requires modelling the changes to both subgraphs. This can be done in $O(n^2)$ for each subgraph where n is the number of vertices in that subgraph. A different method is proposed by the authors however, each vertex is initially assigned a marginal cost which is the cost of all edges from that vertex to all other vertices in that subgraph. This marginal cost is used when modelling the cost of a transfer and swap, which takes $O(n)$ for each update. To avoid checking pairs of redundant subgraphs, once a pair has been checked it is marked as checked and this will not change until either of the subgraphs recieve or lose vertices. Vandermuelen et al graceously provided their code for our use, it is from this that we found that they order the subgraphs from the highest average length to the lowest average length. The most expensive was paired with the least expensive, then the second most expensive until all those pairs have been checked. Then the second most expensive is paired with the least expensive and iterates in the same manner until each unique pair has been checked. No changes need to be made to this procedure as it is designed for use with the average length metric.

The second mechanism proposed by Vandermuelen et al is *transfer outliers*. This procedure identifies and relocates vertices that have a disproportionate effect on the average length of a subgraph. The marginal vertex cost is defined as the total cost of all edges from a vertex to all other vertices in the subgraph. The average vertex cost of a subgraph is the total cost of edges in the graph divided by the number of vertices. If a vertex's marginal cost is above a pre-determined factor from that subgraph's average marginal vertex cost, then the vertex should be transferred to a subgraph with the lowest marginal cost for that vertex. This process is repeated until there are no more vertices with a disproportionate impact on the average length for their subgraph, or those that remain are in a position with the smallest marginal vertex cost. The intention of this procedure is to move vertices to

subgraphs that may not reduce the maximum average length between the pair in order to avoid only searching and optimising the local optima. The threshold for being considered disproporate by the authors was a factor of 1.5. As this procedure was designed for use with the multi depot MinMax MTSP, no modifications are required.

A popular graph partitioning approach is the Kernighan-Lin heuristic, which we will call *swap and lock*. This approach starts with a given initial partition, with the author's suggestion being to use random partitioning. The current cost of the cut is determined, which is the total cost of edges between subgraphs. Each pair of vertices are then evaluated to determine the effect that swapping them would have on the cost of the cut, with any reduction in cut cost being called the *gain*. The swap with the greatest gain is selected and carried out, and both vertices are locked into position in that neither are considered for any further swaps. This process repeats until all vertices have been locked. The state of the partitions after each swap is recorded, and once the procedure has completed the state that has the lowest cut cost is used as the result. It may be that the swap with the greatest gain increases the total cut cost, this is still carried out as exploring beyond the local optima may lead to better results. The first modification the algorithm requires is that each of the initial partitions must contain a depot, and these vertices cannot be considered for a swap throughout the procedure. The second modification is that the metric measured should should be the maximum average length, however only considering the subgraph with the maximum average length would mean that a large proportion of swaps would not effect that value and would therefore have a gain of 0. Instead of this the difference in the maximum average length of the subgraphs involved in the swap is used.

Another popular graph partitioning approach is the Fiduccia-Mattheyses heuristic, which we will call *Transfer and lock*. This approach works in a similar way to the Swap and lock heuristic, however the difference is that vertices are transferred between subgraphs as opposed to swapping a pair of vertices. A unique data structure is proposed with this mechanism that uses doubly linked lists, which allows for the gain values for each of the vertices to be updated only for the effected vertices. As this work aims to examine characteristics of graph partitioning approaches that produce good MTSP tours we will not be translating this structure. The modifications required for this approach are the same as for Swap and lock, with a depot in each of the subgraphs, and the metric used being average length in the same way.

The final mechanism is repeated seed changing, which starts by generating a set of initial tours using some construction mechanism from a set of initial seeds. The suggested

Mechanism name	Description
Transfer and swap	Between each pair of subgraphs, transfer and swap vertices until the maximum average length of the pair cannot be improved. Repeat this process until no more transfers and swaps can be made.
Transfer outliers	Determine if a vertex has a marginal vertex cost that is above a certain factor from the average for that graph. Move these vertices to a subgraph that minimises this marginal vertex cost.
Swap and lock	Determine the reduction in the maxmimum average length for each possible swap, make the best swap and do not move either of the vertices again. Repeat the procedure until all vertices have been locked, then revert to the partition with the lowest maximum average length.
Transfer and lock	The same process as swap and lock however with transfers.

Table 4.3: A brief description of each of the modified improvement mechanisms

method being to use simultaneous breadth first search. Then the vertex in each subgraph that has the lowest marginal vertex cost is selected as the seed for that subgraph, and then the process is repeated until two successive iterations generate the same set of seeds. As the nature of this improvement mechanism is that is modifies the seed, and that a modification required for the Multiple depot MinMax MTSP is that a depot node must be present in each subgraph, this mechanism does not present an evident method for modification.

4.3 Empirical analysis

In order to evaluate the performance of the modified graph partitioning mechanisms discussed in Section 4.2, each was empirically evaluated. This was done in two ways, comparing the performance of tours generated by the graph partitioning mechanisms against the optimal tours for small scale problem instances, and comparing the performance of tours generated by the graph partitioning mechanisms against each other. For both scales of problem, the number of vertices and the number of agents were varied independently in order to evaluate the effect of those variables on the performance of the mechanisms. For the non-optimal comparison, both artificial and real world graphs were compared in order to examine the effect of the topology. This creates a total of 5 problem sets. A

detailed description of the experiment settings is provided in Section 4.3.1. The results of the empirical analysis of the Construction mechanisms, and analysis of those results are presented in 4.3.2, and the analysis of Improvement mechanisms and it's discussion is presented in Section 4.3.3.

4.3.1 Experiment settings

In this section we provide the reasoning for the experimental design and implementation decisions for the average case empirical analysis. In Section 4.3.1 we describe the reasoning for selecting the particular graph structure used. In Section 4.3.1 we then discuss the problem sizes for the 5 problem sets examined. We discuss the two different metrics used for the problem sets in Section 4.3.1 and Finally in Section 4.3.1 we discuss the technical setup.

Graph structure

Six different problem sets were used, with 2 of them consisting of a small scale set of graphs, 2 of them consisting of 2 larger scale graphs, and 1 of them consisting of real world graph topologies. For each of the six data sets, complete undirected graphs were selected as to remove the level of connectivity as an additional variable to control. Each graph has a fixed number of vertices, with a subset of those vertices as depots corresponding to the number of agents for that graph. For example if there are 6 vertices and 2 depots, then vertices 1 and 2 are depots, and vertices 3, 4, 5, and 6 are locations to visit. Each combination of a number of vertices and depots is a *graph combination*. Each vertex has a random X and Y co-ordinate, which is a real number in a range between 0 and 100. This range was selected in order to provide consistency with Vandermeulen et al's empirical analysis that used the same range [94].

Each of the subgraphs are solved as a TSP, and in order to determine the optimal tour cost a program called Concorde was used [21]. The reasoning for this selection are described in more detail in Section 3.4.2. The form of edge weight most commonly used by this solver is referred to as 'EUC_2D', and returns an integer euclidean distance between two coordinates [79]. This variant of the euclidean equation uses the 'bankers rounding' method, whereby a value of 0.5 is rounded up and all others values are rounded to the nearest integer. For example, a value of 1.3 becomes 1, a value of 1.5 becomes 2, and a value of 1.7 becomes 2. If we represent the X coordinate of the first point with a and the

Y coordinate as b, then if we represent the X coordinate and Y coordinate of the second point as c and d respectively, then we can describe this variant of the euclidean equation as Equation 4.1.

$$round((c-a)^2 + (d-b)^2) \tag{4.1}$$

Problem instance sizes

We examine 5 problem sets, with 2 being small scale graph sizes, 2 being large scale graph sizes, and 1 being real world graphs. For the construction mechanisms, all of these problem sets were examined, however for the improvement mechanisms the 2 small scale problem sets were used. For both small and large scale graph sets the number of vertices and agents were varied independently, which results in two problem sets for each size. The first small scale problem set is that of varying the number of vertices while maintaining 2 agents. The number of vertices varies from 10 to 15, incrementing in steps of 1, which gives 6 graph combinations. The second set of varies the number of agents while maintaining the number of vertices as 15. The number of agents varies between 2 and 5, incrementing in steps of 1, which gives 4 graph combinations. Both problem sets share a graph combination of 15 vertices and 2 agents, therefore between the two sets there are 9 unique graph combinations. As the optimal tour cost is computationally complex to obtain, these small scale problems are used in order to give an indication of the overall performance of each of the mechanism variants.

For the larger scale graphs the same general structure applies. For the first set the number of vertices varies from 25 to 100 in increments of 5 vertices, with a constant of 5 agents, giving 16 graph combinations. The second set varies has a constant number of 100 vertices, with the number of agents varying from 5 to 20 in steps of 1, giving 16 graph combinations. In the same way as the small scale graphs, one graph combination is shared between the two sets and therefore the larger scale graphs has 15 unique graph combinations. For each unique graph combination, 1000 graphs were generated to minimise the effect of any one graph layout skewing the results. Therefore for the 9 unique graph combinations of the small scale problem sets there are a total of 9000 graphs generated, and for the 15 unique graph combinations of larger scale graphs there are a total of 15000 graphs.

For the real world problem set, the same problem instances from Section 3.4.2 are used

from the TSPLIB [79]. The instances with the number of vertices from 51 to 1000 vertices, with 49 unique graph sizes that use a real world topology. A list of these instances is provided in Table A.1, which can be found in Appendix A. During the empirical analyses of Section 3.4 and Section 3.5, it was determined that the larger problem instances did not vary significantly in performance from the smaller problem instances. These problem instances did incur a significant computational cost increase when compared to the smaller problem instances. The problem instance of size 1000 was chosen arbitrarily, and given a preference for numbers that are a multiple of 10.

Metric

Two different metrics are used to evaluate the performance of the different problem sizes. For small scale problem instances where the optimal maximum tour cost is known, the difference between the optimal maximum tour cost (OMTC) and the maximum tour cost from a heuristic algorithm (HMTC) can be used to evaluate the performance of that heuristic. For example, if the OMTC is 100 and the HMTC is 120 then the difference is 20. For a different graph combination, if the OMTC is 1000 and the HMTC is 1200 then the difference is 200. To normalise this and make the two problem instances comparable, the HMTC can be expressed as a ratio or percentage of the OMTC, with the prior two examples both being represented as a ratio or either 1.2 or 120%. We have chosen to represent the the difference as a percentage, as we believe it is more readable when compared to a ratio, with the prior examples resulting in a value of 20%. We call this metric the *Percentage Tour Cost Increase of the Maximum Tour* (PTCI). For large scale and real world problem instances the OMTC is not known and therefore the maximum tour cost is used.

Technical setup

Each graph was generated using the Python 3 programming language on a Windows 10 laptop with an Intel Core i7-8750H CPU @ 2.20GHz. The random co-ordinates require a random number generator and therefore a seed, which was the microsecond the graph was generated. A 10 microsecond delay was added between each graph generation to avoid the same seed being used due to read or write delays. Each of the small scale problem instances was solved using an Integer Programming formulation [41]. Laporte and Nobert's symmetric formulation is used, with the objective function modified to use the MinMax instead of MinSum function [53]. The subtour elimination constraint used

was Kulkarni and Bhave's [52] extension of the Miller-Tucker-Zemlin subtour elimination constraint [60]. This was programmed using the pyomo library for python 3, and was executed on the same Windows 10 laptop. Each modified graph partitioning mechanism was implemented in Python 3, and output each subgraph into the '.tsp' file format [80]. The Concorde TSP solver, for the reasons discussed in Chapter 3 Section 3.4.2, was used to find the optimal tour and tour cost for each subgraph [21]. Concorde by default assumes a rounded down integer euclidean distance, therefore this was used as the edge weight for the graphs. Concorde was developed for a UNIX system and was therefore ran on an Ubuntu 20.04 laptop with Intel Celeron Dual-core T3300 @ 2.00GHz.

4.3.2 Empirical evaluation of construction mechanisms

Each construction mechanism variant described in Section 4.2.1 was executed on each of the 5 problem sets. The deterministic variants, which are all mechanism variants excluding random partitioning (balanced) and random partitioning (unbalanced), were executed once on each problem instance as they will always return the same result. The non-deterministic variants were executed 100 times on each problem instance in order to minimise the effect of a particular random seed. The results for the small scale problem sets, varying the number of vertices and varying the number of agents are presented in Figures 4.1 and 4.2 respectively. The results for the larger scale problem sets are shown in Figures 4.3 and 4.4 in the same way. Finally the results for the real world problem sets are shown in Figures 4.6 and 4.5. The conclusions about each of the mechanism variants come from the small scale problem sets, as the PTCI is a more accurate reflection of the performance. The large scale and real world problem sets are used to illustrate how the general trend observed in the small scale problem sets continues.

Each of the figures consists of (a) which is a visual representation of the results, and (b) a Table that shows a performance summary of each mechanism variant. For the small scale visual representation in Figures 4.1 and 4.2, for the deterministic mechanisms each point represents the mean PTCI for each graph combination. That is to say the mean PTCI across the 1000 problem instances for that graph combination. For the non-deterministic mechanisms, the mean PTCI for each problem instance is found, which is the max PTCI of the 100 executions on that problem instance. Then the mean PTCI for that graph mechanism can be found from the 1000 problem instance, and each point represents this value. For the large scale visual representation in Figures 4.3 and 4.4, a similar process

is carried out however the maximum tour cost is used as the metric. The real world problem set visualisations are represented in Figure 4.6 and 4.5. This differs from the previous two methods as there is only a single problem instance for each problem size and number of depots, which we refer to as a graph combination. When varying the number of vertices, each data point represents the mean value of the maximum tour costs across all graph combinations relating to graph size. When varying the number of depots, each point represents the mean value of the maximum tour costs across all graph combinations relating to a number of depots.

Two standard deviation values are presented in the tables (b). The first is the mean standard deviation of the mean PTCI, which is the standard deviation of a mechanism variant's mean PTCI values with the values corresponding to each graph combination. The second is the mean standard deviation of the tour PTCI. For each set of tours, each tour is normalised against the optimal maximum tour cost to get the PTCI value. The standard deviation is then found between these tours, if at least two tours have a non-zero cost. The mean of those standard deviations found for each graph combination, and finally the mean is found for those graph combination values.

The best performing of the three unique construction mechanisms was Average length change with a mean PTCI across both mechanism variants of 21.8%. For both small scale problem sets, the subgraph cost variant outperforms the local cost variant, both in terms of producing tours that are closer to optimal and in the consistency of closeness to the optimal. The subgraph cost variant has a mean PTCI of 12.5% when increasing the number of vertices, which is almost half that of the local cost variant at 23.6%. The subgraph variant is also far more consistent, with it having a standard deviation of 0.9 compared to the local cost's 1.4. Increasing the number of depots shows similar results in favor of the subgraph variant, with a mean PTCI of 18.5% against local cost's 32.6%. The standard deviation of these PTCIs again has the same pattern with the subgraph cost variant having a standard deviation of 3.7 and local cost having 4.9. The reason for this improved performance is likely the balance of cost between the subgraphs and therefore tours, as when varying the number of vertices the subgraph cost variant has a standard deviation of 14.4 which is a vast improvement on local cost's 43.9. Again a similar performance increase is seen when varying the number of agents, with 24.2 against 47.0. The difference between the metrics used by both mechanism variants is that local cost uses just the additional cost, and subgraph cost aims to minimise the maximum average length. These results indicate that a graph partitioning mechanism with a construction strategy

that takes into account the entire subgraph cost will outperform those that do not.

The second best performing of the unique construction mechanisms was the Breadth first search, with a mean PTCI of 38.7% for the 4 mechanism variants. The 4 breadth first search mechanism variants can be separated into 2 pairs of approaches. The first pair is the balanced and unbalanced variant of the global sequential strategy, and the second is local sequential and global simultaneous. For the global sequential variants, the balancing refers to an equal number of vertices for each subgraph. The balanced variant marginally outperforms the unbalanced variant when increasing the number of vertices, with a mean PTCI of 28.5% to 29.7%. The performance of the balanced variant is consistent when increasing the number of vertices, with the mean PTCI values having a standard deviation of 0.5. The unbalanced variant however is significantly impacted, with the PTCI starting at 25.7% for 10 vertices and increasing to 34.1% for 15 vertices. This is initially better than the balanced variant's performance of 28.3% for 10 vertices, however from 12 vertices onwards the performance is worse. The variance between the cost of each tour also varies far more for the unbalanced variant with a standard deviation of 60.9 compared to the 39.2 of the balanced variant.

The opposite is true when increasing the number of agents, with the unbalanced variant having a mean PTCI of 42.6%, which outperforms the balanced variant of 58.3%. The unbalanced variant has a consistent performance when increasing the number of agents, as shown by the standard deviation of 5.8. By contrast the balanced variant starts with a PTCI of 28.6% for 2 agents, which outperforms the unbalanced variant's 34.1%, then drops significantly to 83.6% with 5 agents. The balanced variant does however still does produce more consistent tours, with a standard deviation of 48 to the unbalanced variant's 61.1. These results indicate that when the ratio between the number of vertices and agents is in favor of the vertices, then selecting the balanced variant would be the better choice. However when the ratio is in favor of the agents, with fewer vertices, the unbalanced variant is a superior choice.

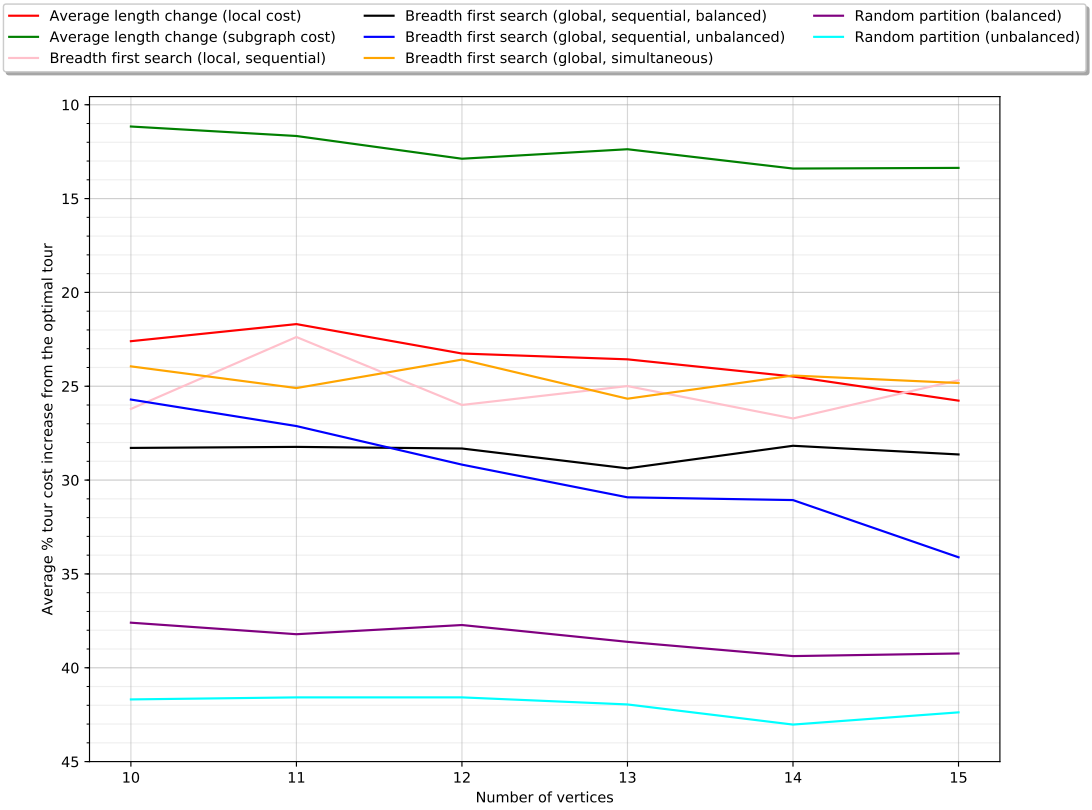
The local sequential and global simultaneous variants are similar in that both distribute an equal number of vertices to each of the subgraphs, however the local sequential variant distributes these vertices in a sequential fashion and the simultaneous variant distributes a vertex to each subgraph in rounds. The global simultaneous variant outperforms the local sequential variant when increasing the number of vertices, with a mean PTCI of 24.6% in contrast to 25.2%. Both approaches have a consistent performance, with local sequential having a standard deviation of 1.6 and global simultaneous having 0.8. When increasing the

number of agents the local sequential is the better performing variant, with a mean PTCI of 49.4% against global simultaneous with 51.0%. Both of these approaches are significantly affected by the number of agents. Local sequential starts with a PTCI of 24.7& with 2 agents, and increasing to 70% with 5 agents. Global simultaneous starts with a PTCI of 24.8% for 2 agents, and 74.6% for 5 agents. The performance of both approaches are similar, with similar performance trends when varying the number of vertices and agents. This indicates that for the breadth first search, the difference between selection in rounds and sequential selection is minimal.

The Random partitioning mechanism variants were the worst performing of the three unique mechanisms, with a mean PTCI from both variants of 57.8%. For both problem sets, increasing the number of vertices and increasing the number of agents, the balanced variant outperforms the unbalanced variant by almost all of the metrics being observed. When increasing the number of vertices the balanced variant has a mean PTCI of 38.5% and the unbalanced variant has a mean PTCI of 42%. The performance of both variants is consistent across the graph sizes examined, with the balanced variant having a standard deviation of 0.8 and the unbalanced variant having 0.6. When increasing the number of agents, the balanced variant has a mean PTCI of 72.7% with unbalanced having 78.1%. The performance is greatly dependant on the number of agents however, with the balanced variant starting with a mean PTCI of 39.2% with 2 agents and increasing to 94.5% with 5 agents. The unbalanced variant starts with a mean PTCI of 42.4% and with 5 agents this becomes 103.7%. For both problem sets the balanced variant produces tours with a smaller standard deviation, with 17 outperforming 24.8 when increasing the vertices and 28.7 outperforming 40.3 when increasing the number of agents. As the balancing constraint is the only difference in the approaches, this demonstrates that for the MinMax objective this is a beneficial constraint to apply.

4.3.3 Empirical evaluation of improvement mechanisms

Each of the modified improvement mechanisms described in Section 4.2.2, were executed on the same 4 data sets used in Section 4.2.1. In contrast to a construction mechanism, an improvement mechanism requires a partition as input instead of a graph. Therefore the resulting partitions from the Random partitioning, both balanced and unbalanced, for the 4 data sets was used as the input. For one of the mechanisms, transfer outliers, a threshold value is used to determine the factor above which is selected for transfer between

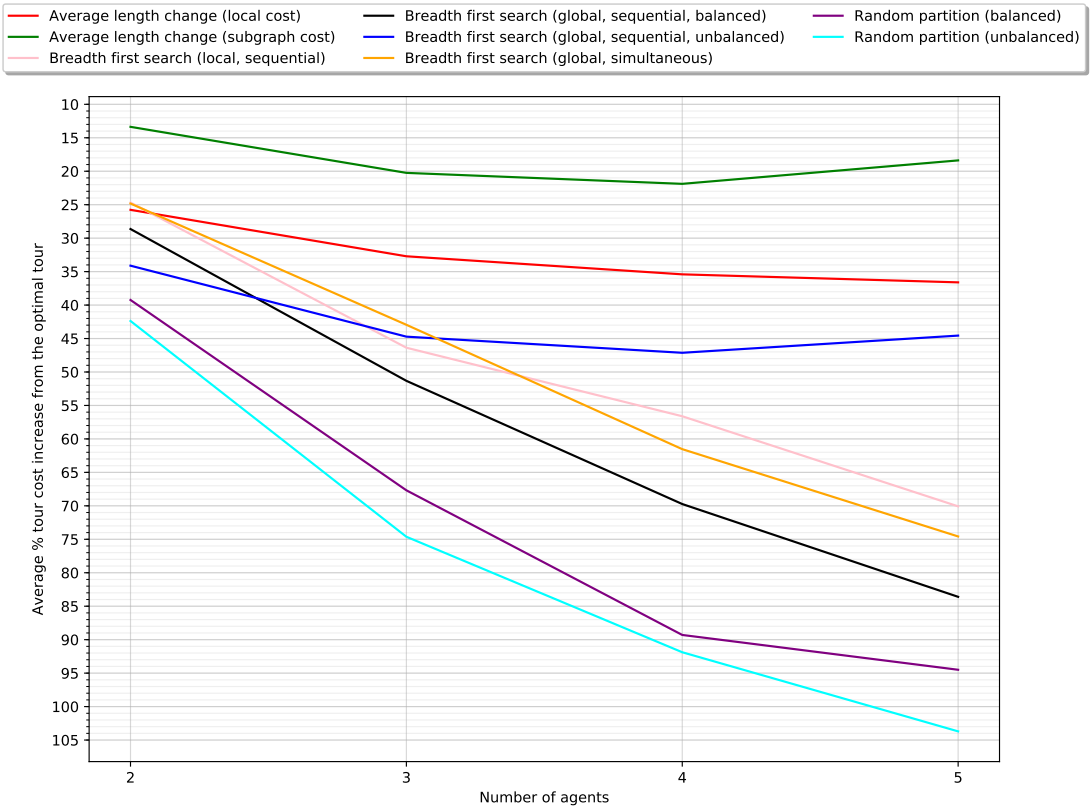


(a) A mean PTCI comparison of the selected graph partitioning mechanism variants, varying the number of vertices.

Procedure name		Mean PTCI (%)	Mean standard deviation of mean PTCI (%)	Mean standard deviation of tour PTCI (%)
Average length change	(subgraph cost)	12.5	0.9	14.4
	(local cost)	23.6	1.4	43.9
Breadth first search	(global, simultaneous)	24.6	0.8	24.7
	(local, sequential)	25.2	1.6	27.8
	(global, sequential, balanced)	28.5	0.5	34.9
	(global, sequential, unbalanced)	29.7	3.0	54.1
Random partition	(balanced)	38.5	0.8	17.0
	(unbalanced)	42.0	0.6	24.8

(b) Table to show a comparison between: the mean PTCI, the mean standard deviation of the mean PTCI, and the mean standard deviation of tour PTCI for the selected graph partitioning mechanism variants.

Figure 4.1: A summary of the results of the small scale empirical evaluation of construction mechanism variants, when increasing the number of vertices. Figure 4.1a shows a visual representation of the mean PTCI for the mechanism variants, with Table 4.1b presenting a summary of the variants performance.

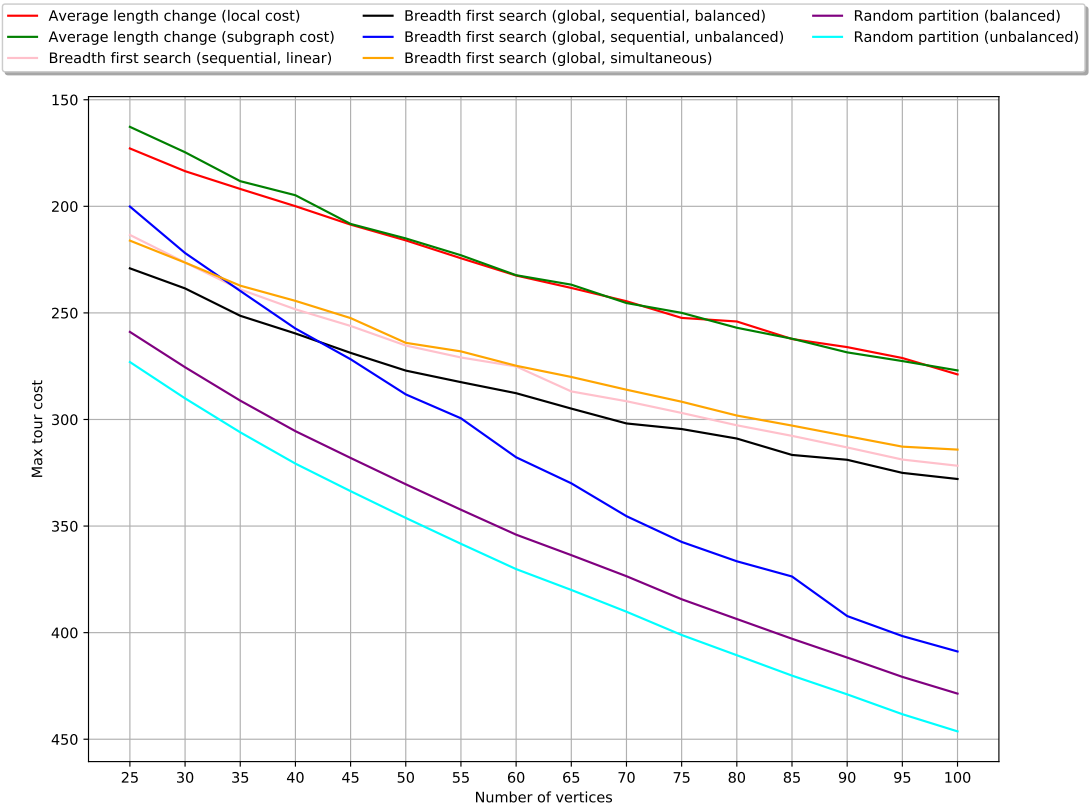


(a) A mean PTCI comparison of the selected graph partitioning mechanism variants, varying the number of agents

Procedure name		Mean PTCI (%)	Mean standard deviation of mean PTCI (%)	Mean standard deviation of tour PTCI (%)
Average length change	(subgraph cost)	18.5	3.7	24.2
	(local cost)	32.6	4.9	47.0
Breadth first search	(global, sequential, unbalanced)	42.6	5.8	57.5
	(local, sequential)	49.4	19.1	39.1
	(global, simultaneous)	51.0	21.7	38.6
	(global, sequential, balanced)	58.3	23.8	48.0
Random partition	(balanced)	72.7	25.1	28.7
	(unbalanced)	78.1	26.7	40.3

(b) Table to show a comparison between: the mean PTCI, the mean standard deviation of the mean PTCI, and the mean standard deviation of tour PTCI for the selected graph partitioning mechanism variants.

Figure 4.2: A summary of the results of the small scale empirical evaluation of construction mechanism variants, when increasing the number of agents. Figure 4.2a shows a visual representation of the mean PTCI for the mechanism variants, with Table 4.2b presenting a summary of the variants performance.

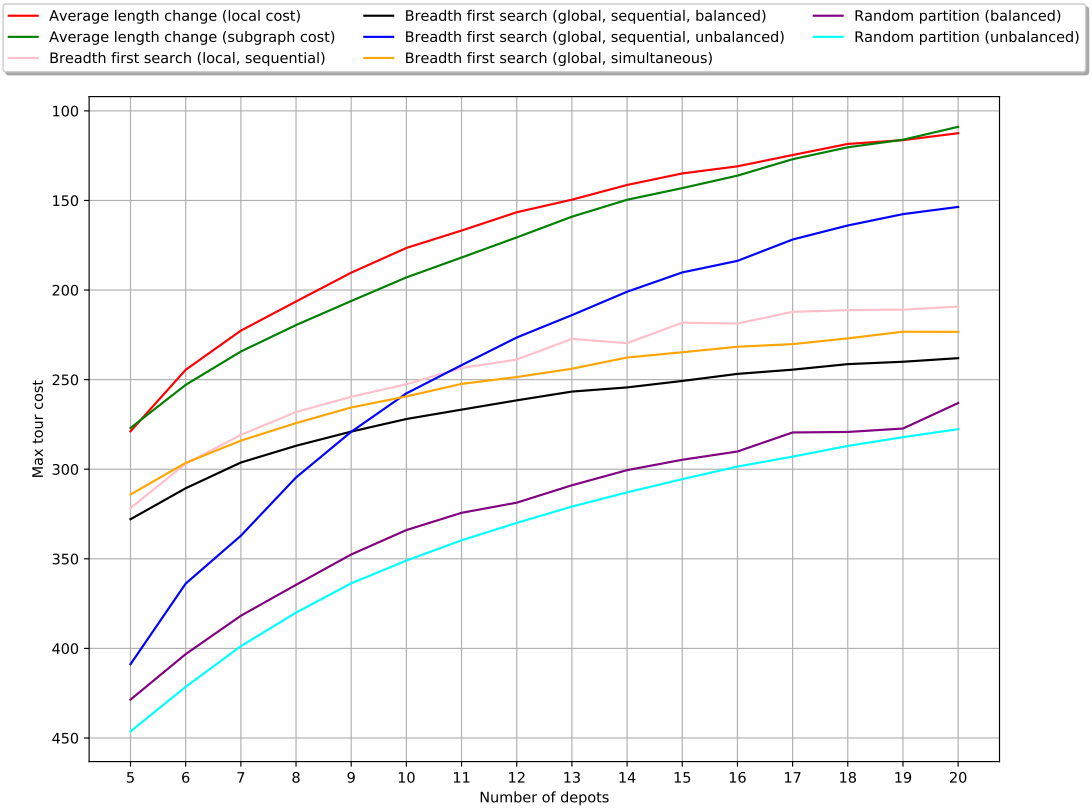


(a) A maximum tour cost comparison of the selected graph partitioning mechanism variants, varying the number of vertices.

Procedure name		Mean maximum tour cost	Standard deviation
Average length change	(subgraph cost)	229.3	35.8
	(local cost)	231.1	32.9
Breadth first search	(global, simultaneous)	273.6	31.2
	(local, sequential)	277.1	33.5
	(global, sequential, balanced)	287.1	30.8
	(global, sequential, unbalanced)	317.0	65.8
Random partition	(balanced)	353.4	53.0
	(unbalanced)	369.6	54.1

(b) Table to show a comparison between the mean maximum tour cost and standard deviation of the mechanism variants, sorted by the mean maximum tour cost.

Figure 4.3: A summary of the results of the large scale empirical evaluation of construction mechanism variants, when increasing the number of vertices. Figure 4.3a shows a visual representation of the mean PTCI for the mechanism variants, with Table 4.3b presenting a summary of the variants performance.

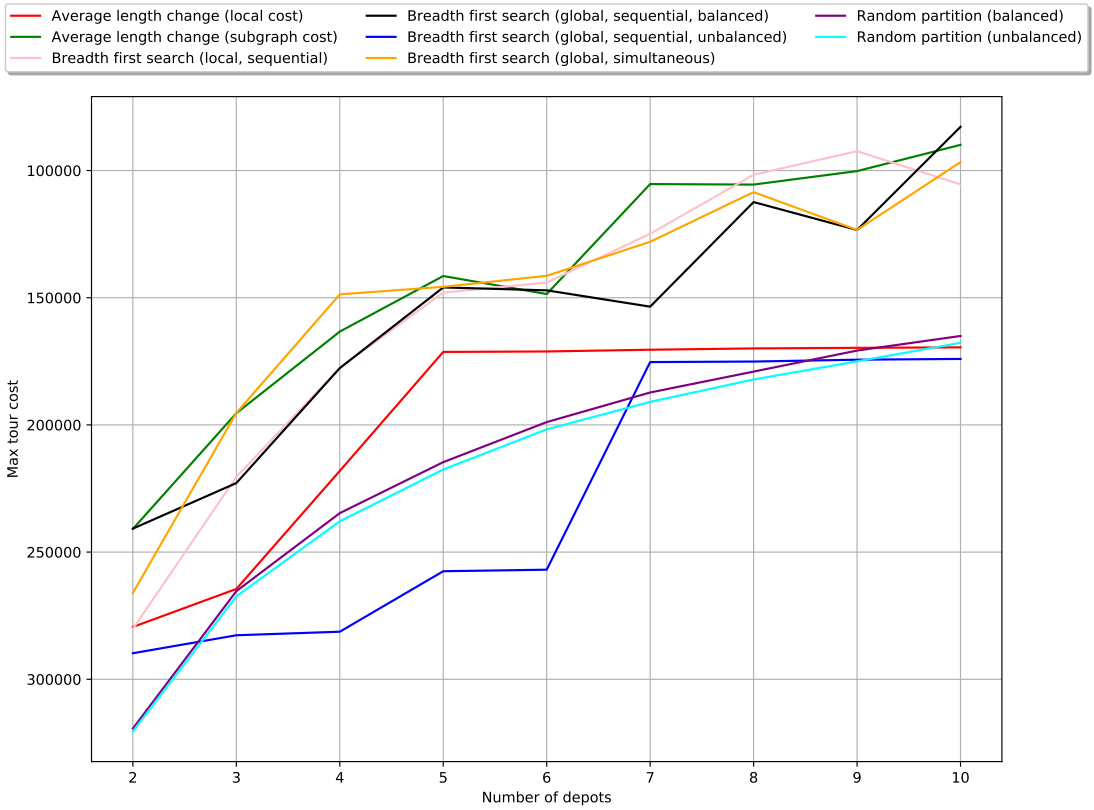


(a) A maximum tour cost comparison of the selected graph partitioning mechanism variants, varying the number of agents.

Procedure name		Mean maximum tour cost	Standard deviation
Average length change	(local cost)	167.0	49.6
	(subgraph cost)	174.7	51.6
Breadth first search	(global, sequential, unbalanced)	241.0	78.3
	(local, sequential)	243.7	34.0
	(global, simultaneous)	253.0	27.5
	(global, sequential, balanced)	267.1	26.8
Random partition	(balanced)	324.8	48.7
	(unbalanced)	338.0	51.6

(b) Table to show a comparison between the mean maximum tour cost and standard deviation of the mechanism variants, sorted by the mean maximum tour cost.

Figure 4.4: A summary of the results of the large scale empirical evaluation of construction mechanism variants, when increasing the number of agents. Figure 4.4a shows a visual representation of the mean PTCI for the mechanism variants, with Table 4.4b presenting a summary of the variants performance.

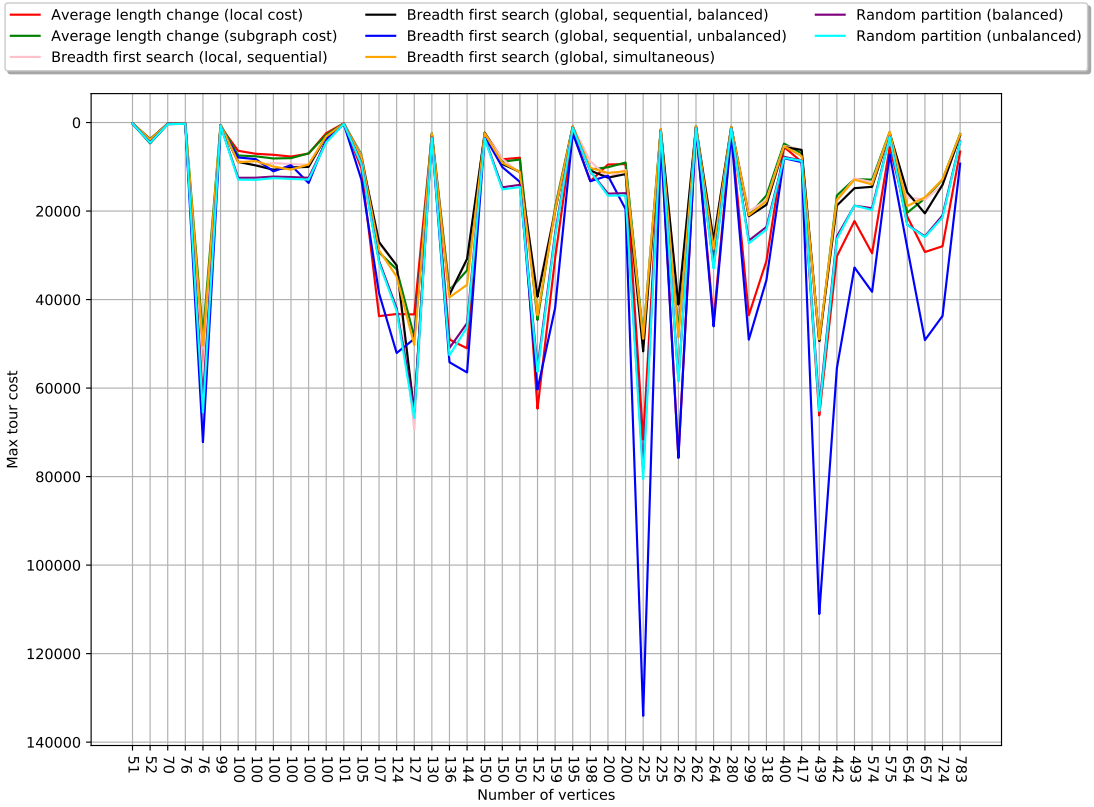


(a) A maximum tour cost comparison of the selected graph partitioning mechanism variants, varying the number of depots.

Procedure name		Mean maximum tour cost	Standard deviation
Average length change	(local cost)	198225.1	44789.2
	(subgraph cost)	143401.7	50308.7
Breadth first search	(global, sequential, unbalanced)	154952.7	62007.8
	(local, sequential)	156274.0	50786.0
	(global, simultaneous)	229669.5	53260.6
	(global, sequential, balanced)	150410.5	51678.0
Random partition	(balanced)	215006.1	50727.3
	(unbalanced)	217874.5	49991.0

(b) Table to show a comparison between the mean maximum tour cost and standard deviation of the mechanism variants, sorted by the mean maximum tour cost.

Figure 4.5: A summary of the results of the empirical evaluation of construction mechanism variants for the real world problem set, when increasing the number of depots. Figure 4.5a shows a visual representation of the mean PTCI for the mechanism variants, with Table 4.5b presenting a summary of the variants performance.



(a) A maximum tour cost comparison of the selected graph partitioning mechanism variants, varying the number of vertices. With the last data point of 1000 vertices excluded as the cost varies so greatly from the rest of the data set as to render it unreadable.

Procedure name		Mean maximum tour cost	Standard deviation
Average length change	(local cost)	198224.7	1239321.9
	(subgraph cost)	143401.2	898250.7
Breadth first search	(global, sequential, unbalanced)	154952.3	975087.2
	(local, sequential)	156273.5	981107.3
	(global, simultaneous)	229669.1	1416023.8
	(global, sequential, balanced)	150410.0	942947.5
Random partition	(balanced)	215005.6	1361562.6
	(unbalanced)	217874.1	1378457.1

(b) Table to show a comparison between the mean maximum tour cost and standard deviation of the mechanism variants, sorted by the mean maximum tour cost.

Figure 4.6: A summary of the results of the empirical evaluation of construction mechanism variants for the real world problem set, when increasing the number of depots. Figure 4.6a shows a visual representation of the mean PTCI for the mechanism variants, with Table 4.6b presenting a summary of the variants performance.

subgraphs, and we have chosen to place this threshold at 1.1. The results for the small scale problem sets, varying the number of vertices and varying the number of agents are presented in Figures 4.7 and 4.8 respectively. Each of the figures consists of (a) which is a visual representation of the results, and (b) a Table that shows a performance summary of each mechanism variant. The results for a mechanism using the random partitioning (balanced) has the (balanced) suffix, and the same is true for the results of the random partitioning (unbalanced) with the (unbalanced) suffix. For each graph combination there are 1000 problem instances, and for each problem instance there are 100 partitions for both the balanced and unbalanced variant. The mean PTCI for a mechanism across the 100 partitions is found, then the mean PTCI for the 1000 problem instances in that graph combination is found and this is represented as a point in visualisation (a).

Two standard deviation values are presented in the tables. The first is the mean standard deviation of the mean PTCI, which is the standard deviation of a mechanism variant's mean PTCI values with the values corresponding to each graph combination. The second is the mean standard deviation of the tour PTCI. For each set of tours, each tour is normalised against the optimal maximum tour cost to get the PTCI value. The standard deviation is then found between these tours, if at least two tours have a cost. The mean of those standard deviations found for each graph combination, and finally the mean is found for those graph combination values.

The best performing mechanism was transfer and swap, with a mean PTCI of 14.5% across the 4 data sets and balanced and unbalanced partitions. When increasing the number of vertices, using a balanced partition produces better results with a mean PTCI of 12.6% which is marginaly higher than unbalanced's 12.8%. The variation in performance is the same for both balanced and unbalanced with a standard deviation of 1.3, and the standard deviation between tours of 10.8 and 10.9 respectively. When increasing the number of agents, unbalanced marginally outperforms balanced with a mean PTCI of 16.2% against 16.5%. As with increasing the number of vertices, the standard deviations of both balanced and unbalanced are 2.2, and a standard deviation between tours of 15.5 for both. The closeness of both standard deviations as well as the closeness of mean PTCIs indicates that regardless of the initial partition, for a given problem instance, the transfer and swap mechanism tends to converge to the same the final partition. The approach is also relatively unaffected by increasing either the number of vertices or agents, which indicates that it's performance is not dependant on the ratio between vertices and agents favoring either side, and may indicate that the approach scales well across graph combinations.

The second best performing mechanism was swap and lock, with an overall mean PTCI or 21.5% for the 4 data sets, both balanced and unbalanced. When increasing the number of vertices the balanced partitions benefited more, with a mean PTCI of 12.8% compared to unbalanced's 18.7%. The resulting tours from balanced partitions also outperform unbalanced partitions in terms of standard deviation between tours, with 10.3 and 15.5 respectively. The performance was not significantly effected by increasing the number of vertices, with a standard deviation of 1 for balanced and 0.5 for unbalanced. A similar pattern occurs when increasing the number of agents, with balanced outperforming unbalanced with 20% to 34.4%. The performance from balanced partitions is relatively consistent across the number of agents, with s standard deviation of 4.6. Unbalanced however, starts with a mean PTCI of 19.1% for 2 agents, and increasing to 44.2% for 4 agents. A similar difference in standard deviation in tour costs occurs with balanced having 15.9 and unbalanced having 24.1. Swap and lock differs from transfer and swap in two ways, it does not transfer individual vertices between subgraphs, and it does not re-evaluate the position of a vertex once moved. A combination of these factors mean that this mechanism does not converge in the same way as transfer and swap, however it shows good performance for balanced partitions.

The third best performing mechanism was transfer outliers, with a mean PTCI of 30.6% across the balanced and unbalanced results from the 4 data sets. When increasing the number of vertices, using balanced and unbalanced partitions result in a similar final performance, with a mean PTCI of 19% and 19.2% respectively. Unbalanced partitions had a lower variance when increasing the number of vertices, with a standard deviation of 0.8 compared to balanced's 1.2. Both also have a similar standard deviation between tour costs with balanced having 17 and unbalanced having 18. When increasing the number of agents, starting from an unbalanced partition outperforms the balanced partition with a mean PTCI of 18.6% against 20.3%. The performance is greatly impacted by increasing the number of agents, with the balanced partitions returning tours with a mean PTCI of 17.9% for 2 agents, and 65.1%. Using an unbalanced initial partition produces marginally better results when increasing the number of agents, as it starts with 18.2% for 2 agents and increases to 61.3%. The intention for the mechanism by Vandermuelen et al is to displace vertices that are disproportionately effecting the subgraph average length, and therefore avoid the local optima search problem that the transfer and swap method may encounter. This method has similar performance, given the correct threshold value, to swap and lock when increasing the number of vertices. It does however have significantly

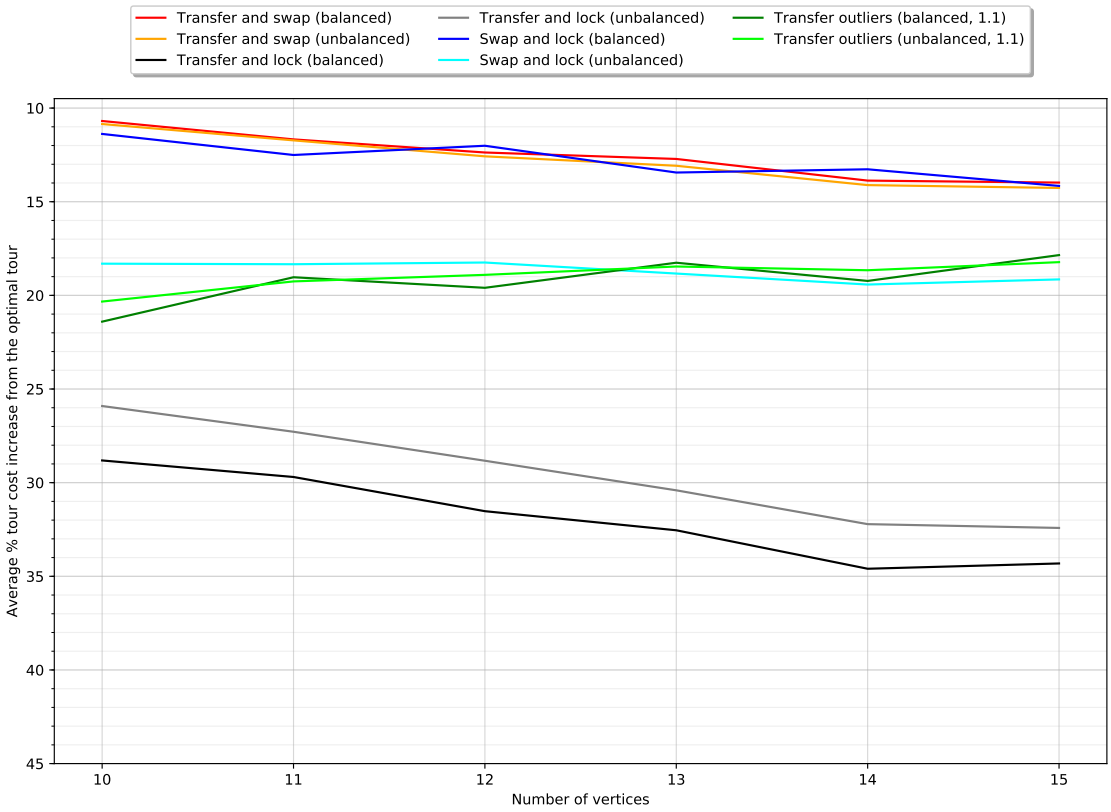
worse performance when increasing the number of agents.

The worst performing mechanism was transfer and lock, with a mean PTCI of 34.3%. When increasing the number of vertices, the tours generated from an unbalanced partition had a mean PTCI of 29.5% which outperformed the balanced partition's 31.9%. Using both balanced and unbalanced initial partitions, the performance decrease with an increase in vertices. Balanced starts with a mean PTCI for 28.8% for 10 vertices and strays further from optimal to 34.3% with 15 vertices. Unbalanced follows a similar pattern, starting with a mean PTCI for 25.9% for 10 vertices and increasing to 32.4% for 15 vertices. The standard deviation between tours for both are comparatively close with 12.4 for balanced and 13 for unbalanced. When increasing the number of agents, the tours generated from the unbalanced partition also outperformed that from the balanced partition, with a mean PTCI of 35% against 40.6%. The standard deviation of the unbalanced is 2.4 which is almost half of the balanced partitions 4.4. The standard deviation between tours is also incredibly similar with 17.7 and 17.8 respectively.

4.4 Conclusion

In this chapter we have discussed the graph partitioning approach to the Multi Depot MinMax MTSP proposed by Vandermeulen can be generalised into a framework, and that other graph partitioning mechanisms can be modified for use with this framework. For both construction and improvement mechanisms, we have identified through empirical testing which characteristics may provide beneficial results. For construction mechanisms, we have identified that when using random selection such as the random partitioning approach, that applying the balancing constraint produces better quality tours. This however has limited effect when using a edge weight metric such as Breadth first search, where the balancing only provided better solutions when the ratio between the number of vertices and agents was heavily in favor of the vertices. The best performing metric was that of using the average length, with using the average length of the subgraph performing better than using the local metric of the additional cost to the average length of adding a vertex. This indicates that modifying graph partitioning mechanisms to use the average length is likely to produce mechanisms that return high quality tours.

For improvement mechanisms, we have identified that the Transfer and Swap procedure is the best performing mechanism of those examined. Finding that swapping vertices tended to produce better quality results than the transferring of vertices between

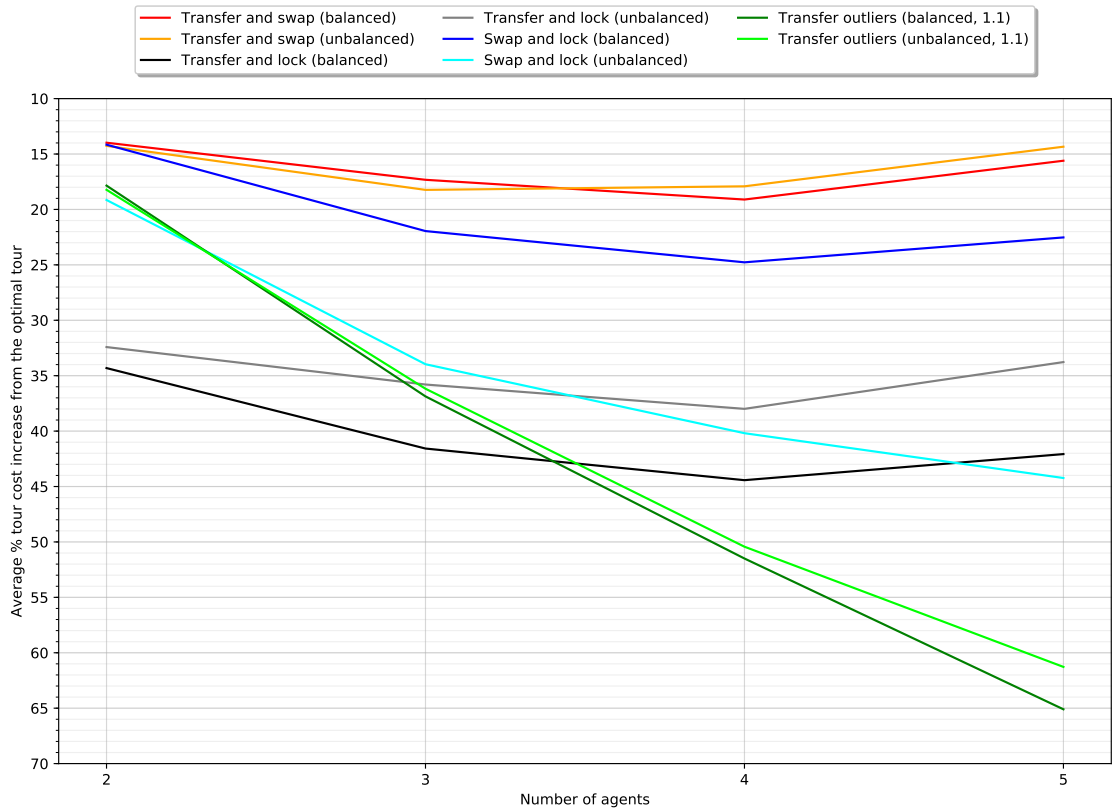


(a) A mean PTCI comparison of the selected graph partitioning mechanism variants, varying the number of vertices.

Procedure name	Mean PTCI	Mean standard deviation of mean PTCI	Mean standard deviation of tour PTCI
Transfer and swap (balanced)	12.6	1.3	10.8
Swap and lock (balanced)	12.8	1.0	10.3
Transfer and swap (unbalanced)	12.8	1.3	10.9
Swap and lock (unbalanced)	18.7	0.5	15.5
Transfer outliers (unbalanced, 1.1)	19.0	0.8	18.0
Transfer outliers (balanced, 1.1)	19.2	1.2	17.0
Transfer and lock (unbalanced)	29.5	2.6	13.0
Transfer and lock (balanced)	31.9	2.4	12.4

(b) Table to show a comparison between: the mean PTCI, the mean standard deviation of the mean PTCI, and the mean standard deviation of tour PTCI for the selected graph partitioning mechanism variants.

Figure 4.7: A summary of the results of the small scale empirical evaluation of improvement mechanism variants, when increasing the number of vertices. Figure 4.7a shows a visual representation of the mean PTCI for the mechanism variants, with Table 4.7b presenting a summary of the variants performance.



(a) A mean PTCI comparison of the selected graph partitioning mechanism variants, varying the number of agents.

Procedure name	Mean PTCI	Mean standard deviation of mean PTCI	Mean standard deviation of tour PTCI
Transfer and swap (balanced)	16.5	2.2	15.5
Transfer and swap (unbalanced)	16.2	2.2	15.5
Swap and lock (balanced)	20.9	4.6	15.9
Swap and lock (unbalanced)	34.4	11.0	24.1
Transfer and lock (unbalanced)	35.0	2.4	17.8
Transfer and lock (balanced)	40.6	4.4	17.7
Transfer outliers (unbalanced, 1.1)	41.5	18.6	33.1
Transfer outliers (balanced, 1.1)	42.8	20.3	32.3

(b) Table to show a comparison between: the mean PTCI, the mean standard deviation of the mean PTCI, and the mean standard deviation of tour PTCI for the selected graph partitioning mechanism variants.

Figure 4.8: A summary of the results of the small scale empirical evaluation of improvement mechanism variants, when increasing the number of vertices. Figure 4.8a shows a visual representation of the mean PTCI for the mechanism variants, with Table 4.8b presenting a summary of the variants performance.

subgraphs. As the transfer and swap mechanism can have a high time complexity from the repeated searches between subgraphs, even with the complexity limited features of not-checking unmodified subgraphs, future work should examine graph partitioning mechanisms that use vertex swapping.

Chapter 5

Multiple Periodic Maintenance Person Problem

5.1 Introduction

Maintenance and inspection companies schedule a team of technicians to carry out the regular servicing of equipment at service locations, over a given period of time called the planning horizon. We call this problem the *Multiple Periodic Maintenance Person Problem (MPMPP)*. This scheduling can be approached in several different ways. One method is to divide the area covered by the company into several smaller areas, each based around a technician. These areas can then be treated as an Maintenance Person Problem as discussed in Chapter 3. This approach has the advantage that it reduces the complexity of the problem and the processing of a solution can be carried out simultaneously. The insular nature of each of the sub problems means that a workload cannot be balanced on a given day between technicians. For example the technician assigned to one area may have a large number of service locations to visit, and the technician of the adjacent area may have comparatively few.

Another method is to determine which service locations should be visited on each day, and then each day can be treated as a *Multiple Travelling Salesperson Problem (MTSP)*, as discussed in Chapter 4. This has the advantage that the workload can be balanced between each technicians, and there exists a wide variety of literature related to MTSP to draw from. This does however require that the revisiting constraint must be satisfied manually first, which may not be possible as determining if the workload is too much for the team

scheduled to work that day requires simultaneously solving the MTSP while developing the revisiting schedule. The method we will explore in this chapter is a combination of these two approaches, where the scheduling of multiple technicians is carried out simultaneously for the planning horizon. We present two heuristic mechanisms using this approach. The first is an extension of the modified successive augmentation mechanism proposed in Chapter 3. An initial partial tour, including the relevant depot for each of the technicians is generated for each of the days in the planning horizon. The selection criteria is then extended to accommodate the multiple agents, with each selection being ranked in the usual way.

The second is to combine the approaches from Chapters 3 and 4. The modified successive augmentation mechanism evaluates, for each day, the cost of every vertex from the potential vertex insertion data structure at every position specified by the expansion criteria. Given the specifics of that expansion criteria this may be every position in each of the partial tours. The best performing of the construction mechanisms examined in Section 4.3.2 was Average Length Change, which builds a set of subgraphs iteratively. The Average Length Change graph partitioning mechanism can be used as a selection criteria. This has the advantage of evaluating far fewer options, as only each of the subgraphs need to be evaluated instead of each position in the partial tours. Each of the subgraphs can then be solved as Travelling Salesperson Problems (TSP) after the addition of each new vertex in order to determine if the addition of a vertex has exceeded the tour cost limit. In Section 5.2 we discuss both of the modifications to the successive augmentation procedure. In Section 5.4 we perform an empirical analysis on these two approaches to determine which performs better, and present the results of this. Finally in Section 5.5 we discuss the findings from this work.

5.2 Proposed mechanism

We propose two methods for adapting the successive augmentation mechanism to accommodate: multiple agents, tour cost limit, and revisiting constraint ($\mathcal{R}_3$). The first is an extension of the modified successive augmentation mechanism proposed in Chapter 3. This modification already accommodates a tour cost limit and revisiting constraint ($\mathcal{R}_3$), however does not take into account multiple agents, and therefore further modifications are required. We call this method the *Extended modified successive augmentation mechanism (EMSAM)*, and this is described in detail in Section 5.2.1. The second method is an extension of EMSAM that uses a graph partitioning mechanism to distribute vertices between

the agents. The graph partitioning construction mechanism Average Length Change proposed in 4 is iterative in nature, assigning a vertex to an agent's subgraph each iteration until all vertices have been assigned. This mechanism can be combined with the EMSAM, with each iteration assigning a vertex using the graph partitioning mechanism instead of a selection and expansion criteria to generate subgraphs instead of tours. These subgraphs can then be turned into a tour using an exact or heuristic method to determine if the tour cost limit was violated. We provide a full description of this modification, which we call the *Graph Partitioning and modified successive augmentation mechanism (GPMSAM)* in Section 5.2.2

5.2.1 Extended modified successive augmentation mechanism

As described in Chapter 3, the modified successive augmentation mechanism for the MPP extends the successive augmentation mechanism in five ways. The first is that there is an initial tour associated with each day in the planning horizon. The second is that the selection and expansion criteria do not simply evaluate all of the candidate vertex insertions across each of these multiple tours. The third is that the current state of the candidate vertices are kept track of using the potential vertex insertions, which we model as a list with m tuples. Each of these tuples is a list containing each of the candidate vertex insertions associated with that day in the planning horizon. The fourth is that the termination criteria is changed from when there are no more unvisited vertices to when each vertex has a vertex coverage that extends beyond the planning horizon. The fifth is that a search and bound mechanism is added, which explores the problem space and determines if a sequence of selections can produce a valid schedule. If at any point an invalid schedule is detected, the mechanism backtracks and explores another potential solution.

Extending the modified successive augmentation mechanism entails two changes. The first change is that an initial tour is not only assigned to each of the days in the planning horizon, but also assigned to each of the agents on each of days. For k agents there are k partial tours, each containing the corresponding depot for the agent. This is repeated for each of the days m in the planning horizon, resulting in km initial tours. The second change is that for each day in the planning horizon, not only the candidate insertions for a single tour is considered but instead each of the candidate insertions across the multiple tours are considered. These additional modifications also require an updated set of tie breaking conditions.

If two candidate insertions have the same cost, given the selection criteria, a unique attribute of these candidate insertions is used as a tie breaking condition. There are a hierarchy of these tie breaking conditions, and this structure is followed to determine the ordering of these possible candidate insertions. Currently the hierarchy is: vertex, day, and finally position with a tour. This hierarchy is extended to: vertex, day, agent, and finally position within a tour. If two candidate insertions have the same cost, however the selected vertex for each of these is different then either an earlier or later vertex preference can be used. Each vertex has a unique position in the set of vertices, and this is used to determine if it is earlier or later in the set. For example if it would be either vertex 5 or 9, and an earlier preference is applied then vertex 5 would be chosen, with the inverse being true for a later preference.

If two candidate insertions both have the same cost and use the same vertex, however are associated with different days in the planning horizon then a later day preference is applied. Each day is unique in the set of days, for example days 3 and 11. Inserting a vertex into a day's tour increases a vertex's coverage by a fixed amount, that day plus the time window. So if a vertex with a time window of 2 is inserted into the tour for day 5 then the coverage of that vertex becomes 7. In this way, inserting a vertex into a later tour will always result in a higher vertex coverage and therefore is the only preference in this tie breaking condition. If two candidate insertions have the same: cost, vertex, and day then the agent's tour can be used as a tie breaking condition. As with the vertices and days, the agents each have a unique position within the set of agents and each of the tours are associated with one of these agents. Therefore this can be used in a similar way to the vertex as there is no significant effect of having an earlier or later preference for the agents.

Finally the position within the tour can be used as the final tie breaking condition within the hierarchy. Two candidate insertions may have the same: cost, vertex, day, and agent however they cannot have the same position within that agent's tour. In the same way as the vertex and agent there is no reason why a preference may be for an earlier or later position in the tour. The other modifications made to the successive augmentation mechanism for the MPP do not require further modification for the MPMPP. These additional changes are the potential vertex insertions, termination criteria, and search and bound method. The potential vertex insertions maintain the current state of available vertices for insertion into each day of the planning horizon, and therefore the agent's tour that the vertex is assigned to or even the position in that tour is irrelevant. As the po-

tential vertex insertions has not been modified, and one of it's purposes is to keep track of the vertex coverage for each vertex then the termination criteria is also unmodified. Finally the search and bound method navigates the problem space for the MPP, with each iteration selecting a vertex to be inserted on a given day at a position within it's tour. For the MPMPP the search and bound method navigates the problem space in the same way except also selecting which tour within a given day.

5.2.2 Graph partitioning and modified successive augmentation mechanism

The Extended Modified Successive Augmentation Mechanism can be further extended to combine with a Graph Partitioning Mechanism, and specifically Average Length Change. The EMSAG evaluates the cost of inserting each of the valid vertices at each of the position in the tours that are specified by the expansion criteria. Depending on the insertion criteria this may be each position within each of the partial tours for a given day. For each iteration in navigating the problem space by the search and bound method, each of these candidate insertions are considered. The combination of EMSAG with Graph Partitioning is primarily motivated by a reduction in complexity, which therefore allows for quicker navigation by the search and bound method. A modified construction graph partitioning mechanism shares similarities to a construction mechanism for a TSP related problem, in that both assign vertices to a tour or subgraph permanently instead of transferring those between the tours or subgraphs. Therefore the modified graph partitioning mechanism of Average Length Change was selected, as it was the best performing of these mechanisms.

The selection and expansion criteria of the EMSAG is replaced in the GPMSAG by Average Length Change's selection and expansion method. For each iteration, the effect on the subgraph costs of each vertex is evaluated and a selection is made based on the Subgraph cost and Local cost metrics of the two variants. The Subgraph cost variant selects the vertex which results in the minimal largest subgraph cost. The Local cost variant selects the vertex which results in the smallest increase to any subgraph's cost. This replaces the selection criteria for the EMSAG, and the expansion criteria becomes adding the selected vertex to the relevant subgraph. This generates subgraphs instead of tours however, and does not indicate if the tour cost limit has been violated. Therefore a tour is generated for each subgraph after a new vertex has been added, and this can be done using an exact or heuristic TSP algorithm.

The current bounding mechanism for the EMSAG checks if a vertex can be inserted into any of the position in the current tours, and if not will then backtrack. This is not possible for the GPMSAG, and therefore this part of the bounding mechanism is removed. The tie breaking condition for the GPMSAM differs from that of the EMSAM, as there is no partial tour to consider. Therefore the order of consideration is: vertex, day, and then finally subgraph.

5.3 Experimental Methodology

In order to evaluate the performance the EMSAG and GPMSAG, 2 problem sets were generated and each procedure relating to the two approaches were executed on these problem sets. The EMSAG has 8 selected procedures which were outlined in detail in Chapter 3. The GPMSAM has two procedures, that of Average Length Change (Subgraph cost) and Average Length Change (Local cost), with a detailed description provided in Chapter 4. The graphs used were complete euclidean graphs, for the reasons described in both of the previous 2 chapters. The edge weight for these graphs was the euclidean distance was rounded down to the nearest integer in order to allow for a consistent comparison with the previous work. Each vertex has a random x and y co-ordinate, with both values being a real number in the range of 0 to 100. The tour cost limit is 150% of the maximum tour cost generated by the modified Average Length Change (Subgraph cost) method. This is an an arbitrary value that we found through some experimentation engaged the backtracking mechanism but did not result in a situation where all tours were maxed out at this value. The service time for each service location is a random integer value between 1 and 10, and the time window varies depending on if the graph is generated for the small scale or large scale problem sets.

The small scale problem set consists of a single graph size at a single number of days. This is due to the high computationally complexity associated with with finding the optimal tour for a problem size. For this reason we selected a graph with 8 vertices, 2 of which are assigned as depots and a planning horizon of 3 days. The time window for each of the service locations is a random integer value from 1 until the number of days in the planning horizon. During previous chapters we have enforced two visits for each location to guarantee the revisiting mechanism is used, however with a planning horizon of 3 days this is not possible. Using the method outlined in Section 5.3.1, the optimal tour cost was found for 250 instances of the small scale problem size.

The large scale problem set increases the number of days in the planning horizon to determine it's effect on each of the selection and expansion criteria. We do not use an additional problem set to determine the effect of the number of vertices, as this is addressed in Chapter 3, were we examined the effect of increasing the number of vertices on each of the procedures for the EMSAG, and in Chapter 4 were we examined the effect on the graph size and number of agents for the construction mechanisms, including Average Length Change. The planning horizon increases from 7 days to 14 days in steps of 1 day, this leads to 8 different problem sizes. The graphs were generated on a Windows 10 laptop with an Intel Core i7-8750H CPU @ 2.20GHz using the Python 3 programming language. The co-ordinates for each vertex was generated using a seeded random number generator, with the seed being the current time in milliseconds. A 1 millisecond wait gap was given in between generation to ensure that a read or write error did not cause two graphs to use the same seed.

The GPMSAG generates subgraphs that are then solved using either a heuristic or exact solver. We have chosen to use the Concorde solver to find the optimal tour cost [21], in order to only evaluate the performance of the mechanism and selection metric on the result instead of also examining the effect of a procedure on the TSP. For the small scale problem set the Percentage Schedule Cost Increase (PSCI) was used as a metric to evaluate the performance, however for the large scale problem set PSCI could not be used as the optimal schedule cost is not known. The EMSAG procedures optimise for the total tour cost, as is the objective function of the MPP, and the Average Length Change optimises for the maximum tour cost. As there methods are optimising for different objective functions we have instead chosen to use the mean schedule tour cost as an evaluation metric. The mean value was found for all tours on a given day, the mean of these values was then found to represent the mean tour cost from across the entire schedule.

5.3.1 MPMPP exhaustive algorithm

In order to find the optimal set of tours for MPMPP problem instances, an exhaustive algorithm was used. For each day within the planning horizon, each vertex may be assigned one of $m + 1$ states where m is the number of agents. One state is unassigned, and the remaining m states correspond with being assigned to each of the agents. $n - m$ represents the number of non-depot vertices to visit, and therefore there are $(m + 1)^{n-m}$ possible permutations. For example if there are 6 vertices to visit by 2 agents, then a permutation

may be $1, 0, 2, 2, 0, 1$. In this example vertices 2 and 5 are not visited, with vertices 1 and 6 being visited by agent 1, and vertices 3 and 4 being visited by agent 2.

For each of these permutations, given the vertices visited by an agent the optimal tour can be found using a TSP exact algorithm. Each of the permutations that results in at least one tour with a cost that is greater than the tour cost limit is removed as the permutation is not valid for the MPMPP. Each day in the planning horizon may have any of the valid permutations, therefore at most there are $((m+1)^{n-m})^k$ possible schedules where k is the number of days in the planning horizon. Each possible schedule is then checked to determine if it satisfies the revisiting constraint for each non-depot location. Then the total cost of the valid schedules is found by totalling the cost of the optimal tours for each of the days in that schedule, then the schedule with the minimal cost is returned by the algorithm.

5.4 Results

The results of the empirical analysis on the small scale problem set, against the optimal schedule cost, is presented in Table 5.1. The results of the empirical analysis of the large scale problem set is presented in Figure 5.1. This consists of two elements, the first is Figure 5.1a and is a visualisation of the large scale problem set. Each point represents the mean schedule tour cost for a given number of days for each of the procedures. The second element is Table 5.1b and shows a summary of the performance for each of the procedures.

The Graph partitioning and modified successive augmentation mechanism (GPMSAM) variants, subgraph cost and local cost, outperformed all of the Extended modified successive augmentation mechanism (EMSAM) variants without a preference. That is to say that both variants of GPMSAM outperformed the default preference procedures using the EMSAM. The best performing of the GPMSAM procedures was Average Length Change (Subgraph cost), with a mean PSCI of 8.143% with the small scale problem set and a mean schedule tour cost of 156.622. This outperformed the other GPMSAM procedure, Average Length Change (Local cost), which has a mean PSCI of 15.375% for the small scale problem and with a mean schedule tour cost of 202.756. These results reflect a similar pattern of performance as the examination of the graph partitioning construction mechanisms in Section 4.3.2.

The best performing of the EMSAM variants is Smallest Sum Insertion, with a mean PSCI of 17.436% for the small scale problem set and a mean schedule tour cost of 247.538

for the large scale problem set. This is followed closely by Cheapest Insertion, which has a mean PSCI of 17.546% for the small scale problem set, and a mean schedule tour cost of 250.226 for the large scale problem set. The fifth best performing procedure is Nearest Insertion, which has a mean PSCI of 19.287% for the small scale problem set, and a mean schedule tour cost of 260.413 for the large scale problem set. These three procedures are the best performing of the EMSAM procedures, and all adopt the same strategy which is to generate tours that are as small as possible. Cheapest Insertion aims to use the additional cost to make a small tour, with Smallest Sum Insertion trying to find the vertex that is closest to all of the vertices in the partial tour. Nearest Insertion also finds the vertex that is closest but only uses a single edge to identify this. The order of performance for these three procedures show that using the multiple edges is a better approach than using the cost, with using a single edge under performing against both of those strategies.

The sixth best performing procedure is Farthest Insertion with a mean PSCI of 20.594% for the small scale problem set and a mean schedule tour cost for the large scale problem set of 293.809. The seventh best performing procedure is Difference Insertion with a mean PSCI of 22.517% for the small scale problem set and a mean schedule tour cost of 312.397 for the larger problem set. Largest sum insertion is the eighth best performing procedure, with a mean SCI of 22.845% for the small scale problem set, and for the large problem set a mean schedule tour cost of 313.655. Farthest Insertion and Difference Insertion procedure both aim to generate an initial outline of the tour by selecting the vertices that are furthest away and then filling this outline in with the remaining vertices given more knowledge of the general shape of the tour. This is one of the best performing strategies in for the TSP, as demonstrated in Section 3.4.3, however when applying a tour cost limit, in Section 3.5.3, the performance was impacted as the outline of vertices was not filled in. As there are multiple tours using the same set of vertices, a similar issue occurs here with the procedures generating outlines of of the tour but not filling in the remaining vertices as they must be distributed amongst each of the tours. Difference insertion has a similar issue, in that it relies on the assumption that a vertex will be inserted into one of the possible positions in a tour and therefore the difference in these two costs can be used as a metric. As a vertex is not necessarily assigned to a tour, and may instead be assigned to another tour, this metric is less effective.

The final two procedures were the worst performing procedures, Nearest Addition and Nearest Neighbour. The second worst performing procedure is Nearest Addition with a mean PSCI of 86.651% for the small scale problem set, and a mean schedule tour cost

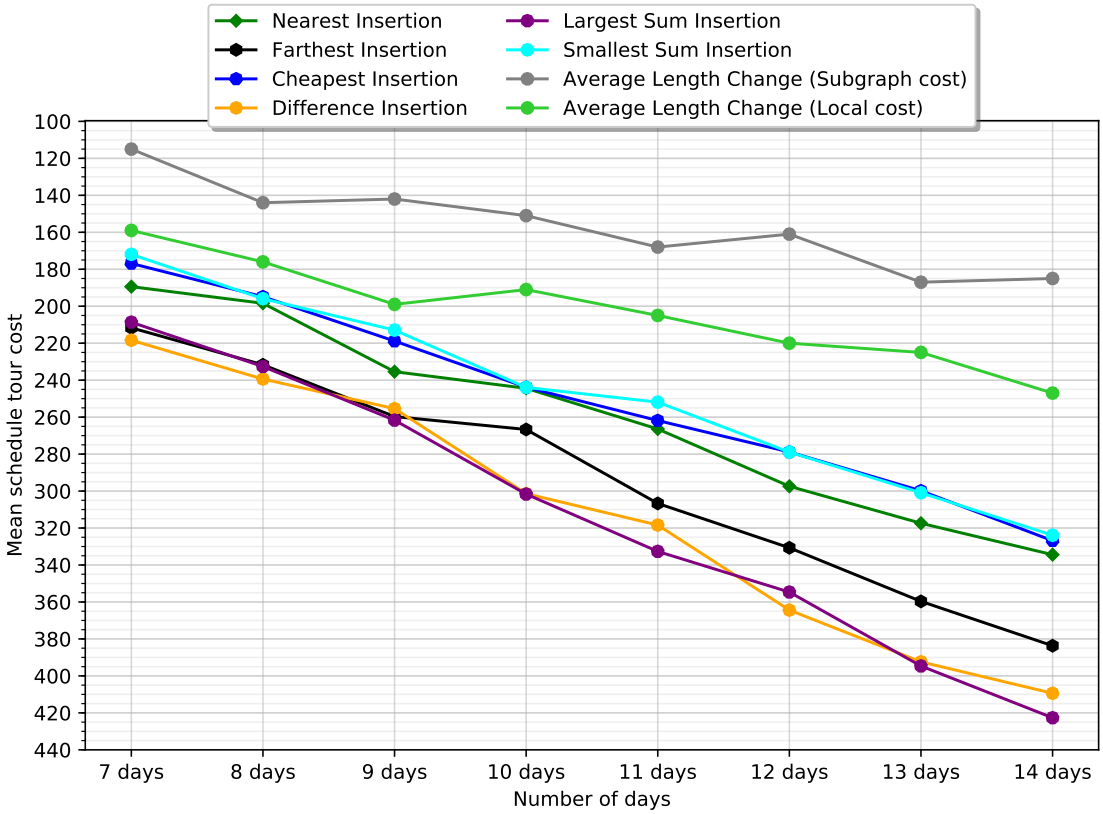
Procedure name	Mean PSCI (%)	Standard deviation
Average length change (subgraph cost)	8.143	0.524
Average length change (local cost)	15.375	0.873
Smallest sum insertion	17.436	2.586
Cheapest insertion	17.546	2.849
Nearest insertion	19.287	2.945
Farthest insertion	20.594	4.273
Difference insertion	22.517	7.066
Largest sum insertion	22.845	6.286
Nearest addition	86.651	4.417
Nearest neighbour	93.393	4.991

Table 5.1: A summary of the mean percentage schedule cost increase and standard deviation of the selected procedures using the EMSAG or GPMSAG, ranked from the smallest *Percentage Schedule Cost Increase (PSCI)* to the largest

of 764.211 for the large scale problem set. The worst performing procedure is Nearest Neighbour, with a mean PSCI of 93.393% for the small scale problem set and for the large scale problem set a mean schedule tour cost of 1129.785. As with the empirical analysis of the EMSAG for the MPP and PTSP, these expansion criteria are less effective than the Insertion expansion criteria.

5.5 Conclusions

Our empirical analysis indicates that the Graph Partitioning and Modified Successive Augmentation Mechanism outperforms the Extended Modified Successive Augmentation. This is due to the better distribution of vertices across the tours and therefore a reduction the maximum tour cost provided by the graph partitioning mechanism. This work could be extended, using the additional variables introduced by having multiple tours in a similar way as demonstrated in Chapter 3. Other graph partitioning mechanisms that have been adapted for use with the MTSP can also be used in order to determine if better performance could be achieved.



(a) A visualisation of the mean schedule tour cost for selected procedures using the EMSAG and GPMSAG on the large scale problem set.

Procedure name	Mean schedule tour cost	Standard deviation
Average length change (subgraph cost)	156.622	23.945
Average length change (local cost)	202.756	28.126
Smallest sum insertion	247.538	52.484
Cheapest insertion	250.226	51.774
Nearest insertion	260.413	53.313
Farthest insertion	293.809	61.466
Difference insertion	312.397	71.797
Largest sum insertion	313.655	76.455
Nearest addition	764.211	120.349
Nearest neighbour	1129.785	146.230

(b) A summary of the mean schedule tour cost and standard deviation of the selected procedures using the EMSAG or GPMSAG, ranked from the smallest mean schedule tour cost to the largest

Figure 5.1: A summary of the performance of each of the EMSAG and GPMSAG procedures on the large scale problem set. Figure 5.1a shows a visualisation of the performance when increasing the number of days in the planning horizon, and Table 5.1b summaries the performance of each procedure.

Chapter 6

Discussion and Future work

In this thesis we have examined several questions posed around various aspects of the the MPMPP.

1. *When examining a variant of the MPMPP where only a single agent is examined, what mechanisms can be used in order to solve this problem optimally and heuristically, what are the limitations of these mechanisms?* In Chapter 3 we explored that the single agent variant of the MPMPP which we call the PMPP. For the optimal solutions to the problem, the NP-hard nature of the problem means that only a small scale set of solutions can be examined. For generating a heuristic solution to the problem, a modified successive augmentation mechanism was presented. The selection and expansion criteria can have a significant impact on the result, and we identified several strategies that resulted in better quality tours and identified the characteristics of these approaches.

2. *How do different heuristics for the Periodic Maintenance Person Problem (PMPP) affect the quality of the tours, when considering different revisiting requirements?* In Chapter 3 we found that by using a successive augmentation mechanism, the selection and expansion criteria can have a significant impact on the result. We identified several strategies that resulted in better quality tours and identified the characteristics of these approaches. Adding additional features such as a tour cost limit can alter which strategies perform the best, and this was discussed in each of the relevant sections.

3. *Given the complexity of the problem, is an exact or heuristic algorithm more suitable*

for the MPMPP? In Chapter 3 we sketched a proof that the MPMPP is an NP-hard problem and therefore a heuristic approach is more suitable to solve non-trivial sizes of this problem. The problem sizes explored in this problem reflect problem sizes for similarly non-trivial extensions of the TSP and future developments in areas relating to these problems may have applications for the MPMPP.

4. *What sort of mechanisms are required to accommodate the visit dependant revisiting requirement?* In Section 3.3 we discussed how an additional step of vertex selection can be applied before using a TSP construction mechanism in order to accommodate the revisiting requirement. We also discussed a method for updating and maintaining a list potential insertion vertices that can be used to keep track of which vertices are available for each day, and using vertex coverage, which vertices have been placed in a sufficient number of tours to not require further insertions.

5. *What are the challenges involved when using graph partitioning when scheduling multiple tours in a single day?* As discussed in Chapter 4 the two problems have different objective functions and therefore require some modification. It was found that the average length of a graph is a suitable proxy for it's optimal tour cost, and therefore the objective function of graph partitioning was changed for minimising the total edge weight between subgraphs to that of minimising the largest average length for the set of subgraphs.

6. *Can graph partitioning mechanisms be adapted to be suitable as an approach to the Multiple Traveling Salesperson Problem (MTSP)?* In Chapter 4 we first discuss the framework introduced by Vandermeulen et al. for converting a set of subgraphs into a set of MTSP tours, with average length being introduced as a metric to determine the suitability of a set of subgraphs [94]. In Section 4.2 we discussed the two categories of graph partitioning algorithm, that of construction and improvement. In Sections 4.2.1 and 4.2.2 we discuss what modifications a graph partitioning mechanism may require in order to be suitable.

7. *Can the use of graph partitioning be combined with heuristic approaches to the Periodic Maintenance Person Problem to find solutions to the Multiple Periodic Maintenance Person Problem (MPMPP)?.* In Chapter 5 we examine two approaches, the first is an extension of the work presented in Chapter 3 which we call the Extended Modified Successive Augmentation Mechanism (EMSAM). The second approach we

examined is a further modification of the EMSAM, using the graph partitioning construction mechanism Average Length Change, which we call the Graph Partitioning and Modified Successive Augmentation Mechanism (GPMSAM). Using a combination of the search and bound mechanism and potential vertex insertions, the possible vertices for insertions were kept track of. The respective mechanisms were then used to distribute the vertices across the agents.

8. *How can we accommodate multiple agents with a revisiting constraint?* We have provided two methods for accomplishing this. The first is to extend an approach intended for the TSP, as presented by the EMSAM. The second is to extend the EMSAM with a graph partitioning approach, as it seen with the the GPMSAM. We found that this second approach produced better quality tours as agents were better distributed between tours.

6.1 Future work

6.1.1 A survey of successive augmentation procedures

The successive augmentation mechanism is ubiquitously used in heuristic algorithms for various problems closely related to the TSP. Due to it's importance there have been a variety of survey papers that have discussed the performance of the various procedures, such as those by: Huang and Yu [45], Ursani and Corne, [93], and Gutin and Punnen [39]. These surveys however all share two common issues, the first is that they often address only a subgroup of successive augmentation mechanisms such as Insertion procedures or Nearest Neighbour variants. The second is that they only examine procedures such as Nearest Insertion or Nearest Neighbour, instead of examining the effect of each of the selection and expansion criteria separately. Each selection criteria can be paired with each expansion criteria in order to more accurately determine the effect that they have have on the resulting tours. Selection criteria can also be categorised based on their characteristics, such as using single edge, multiple edge, geometric based, etc. We believe that a survey into the effects of these characteristics could lead to a better understanding of under which conditions each selection criteria performs well. Finally the existing literature only examine the performance of these procedures on symmetric euclidean graphs, and an evaluation, empirical or otherwise, of the performance of these procedures on asymmetric graphs may prove useful.

6.1.2 Successive augmentation and the orienteering problem

A natural extension of applying the successive augmentation mechanism to the TSP is to apply it to the OP. As discussed in Section 3.5 the successive augmentation mechanism can be modified to better suit the orienteering problem. There are modifications to the successive augmentation mechanism that were intended for the TSP but may prove useful in providing high quality tours for the OP. An example of one such modification is successive augmentation with regret as proposed by Hassin and Keinan [42]. For each iteration, when a new vertex is selected the additional cost is checked against the contribution to the tour cost of each of the vertices in the partial tour. If an existing vertex contributes more to the partial tour cost than the newly added vertex, then it is removed and therefore the new vertex was swapped for this one. In this way a vertex added that is a poor decision and causes 'regret' can be removed and swapped with a better selection. This mechanism removes the guaranteed time complexity of $O(n^2)$, however has been shown to improve the tour cost for the Cheapest Insertion procedure, and may prove beneficial to other selection and expansion criteria.

6.1.3 An extension of the Periodic Maintenance Person Problem

The PMPP addresses the core problem of a single technician over a time period of multiple days, however there are several additional constraints that could be explored. One would be enforcing a minimum number of days between revisits to a service location. This may be relevant in situations where a company must allow the element they are sampling for a period of time before re-examination. Another aspect that has the potential for fruitful work is the development of an improvement mechanism for the MPP. An approach that could be taken is to determine which vertices in a tour have an additional number of days in their time window before their previous visit. The cost of moving these vertices to tours later in their time window could be evaluated and explored. This could be done in a way that only makes a move if it reduces the total schedule cost, or could continue to ensure that options outside of the local optima are explored. The GPMSAM approach discussed in Chapter 5 introduces a mechanism that after each addition of a vertex to a subgraph, uses an exact TSP algorithm to find the optimal tour given that set of vertices. An addition of this method to the modified successive augmentation mechanism may result in better quality tours, though would carry a much larger computational complexity.

6.1.4 Extension of graph partitioning work for the Multiple Travelling Salesperson Problem

The graph partitioning mechanisms presented in Chapter 4 are common geometric based approaches to the problem. The graph partitioning literature is expansive, with a variety of different approaches that have been proposed. An examination on how these approaches could also be adapted to use Vandermeulen et al.'s framework may reveal mechanisms that perform better than those explored in this thesis [94]. The time complexity of the existing approaches we have presented may also provide a useful insight in determining which mechanisms have a good ratio between it's time complexity and the effectiveness of the resulting tours. The approaches we have examined could also be shown in comparison with other MTSP algorithms, to contextualise their performance.

Appendix A

TSPLIB table

Problem ID	Problem name	Number of vertices	Optimal tour cost
1	eil51	51	426
2	berlin52	52	7542
3	st70	70	675
4	eil76	76	538
5	pr76	76	108159
6	rat99	99	1211
7	kroA100	100	21282
8	kroB100	100	22141
9	kroC100	100	20749
10	kroD100	100	21294
11	kroE100	100	22068
12	rd100	100	7910
13	eil101	101	629
14	lin105	105	14379
15	pr107	107	44303
16	pr124	124	59030
17	bier127	127	118282
18	ch130	130	6110
19	pr136	136	96772
20	pr144	144	58537
21	ch150	150	6528
22	kroA150	150	26524
23	kroB150	150	26130
24	pr152	152	73682
25	u159	159	42080
26	rat195	195	2323
27	d198	198	15780
28	kroA200	200	29368
29	kroB200	200	29437
30	ts225	225	126643
31	tsp225	225	3916
32	pr226	226	80369
33	gil262	262	2378
34	pr264	264	49135
35	a280	280	2579
36	pr299	299	48191
37	lin318	318	42029
38	rd400	400	15281
39	fl417	417	11861
40	pr439	439	107217
41	pcb442	442	50778
42	d493	493	35002
43	u574	574	36905
44	rat575	575	6773
45	p654	654	34643
46	d657	657	48912
47	u724	724	41910
48	rat783	783	8806
49	dsj1000	1000	18659688
50	pr1002	1002	259045
51	u1060	1060	224094
52	vm1084	1084	239297
53	pcb1173	1173	56892
54	d1291	1291	50801
55	rl1304	1304	252948
56	rl1323	1323	270199
57	nrw1379	1379	56638
58	fl1400	1400	20127
59	u1432	1432	152970
60	fl1577	1577	22249
61	d1655	1655	62128
62	vm1748	1748	336556
63	u1817	1817	57201
64	rl1889	1889	316536
65	d2103	2103	80450
66	u2152	2152	64253
67	u2319	2319	234256
68	pr2392	2392	378032
69	pcb3038	3038	137694
70	fl3795	3795	28772
71	fnl4461	4461	182566
72	rl5915	5915	565530
73	rl5934	5934	556045
74	rl11849	11849	923288
75	usa13509	13509	19982859
76	brd14051	14051	469385
77	d15112	15112	1573084
78	d18512	18512	645238

Table A.1: A table showing each of the selected problem instances from TSPLIB

Bibliography

[1] A Iqbal Ali and Jeff L Kennington. The asymmetric m-travelling salesmen problem: A duality based branch-and-bound algorithm. *Discrete Applied Mathematics*, 13(2):259 – 276, 1986.

[2] Alexander Barvinok, Sandor Fekete, David Johnson, Arie Tamir, Gerhard Woeginger, and Russ Woodroofe. The geometric maximum traveling salesman problem. *Journal of the ACM*, 50, 05 2002.

[3] Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006.

[4] M. Bellmore and G. L. Nemhauser. The traveling salesman problem: A survey. *Operations Research*, 16(3):538–558, 1968.

[5] Jon Jouis Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–411, 1992.

[6] Dimitri P. Bertsekas. A new algorithm for the assignment problem. *Mathematical Programming*, 21(1):152–171, Dec 1981.

[7] Janez Brest and Janez Zerovnik. A heuristic for the asymmetric traveling salesman problem. In *6th Metaheuristics International Conference*, pages 145–150, 2005.

[8] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. *Recent Advances in Graph Partitioning*, pages 117–158. Springer International Publishing, Cham, 2016.

[9] Edmund Burke, Peter Cowling, Patrick De Causmaecker, and Greet Vanden Berghe. A memetic approach to the nurse rostering problem. *Applied Intelligence*, 15(3):199–214, Nov 2001.

[10] Alberto Caprara, Matteo Fischetti, and Paolo Toth. Modeling and solving the train timetabling problem. *Operations Research*, 50(5):851–861, 2002.

[11] M.Y. Chan and F.Y.L. Chin. General schedulers for the pinwheel problem based on double-integer reduction. *IEEE Transactions on Computers*, 41(6):755–768, 1992.

[12] Yupo Chan and S.F. Baker. The multiple depot, multiple traveling salesmen facility-location problem: Vehicle range, service frequency, and heuristic implementations. *Mathematical and Computer Modelling*, 41:1035–1053, 04 2005.

[13] I.-Ming Chao, Bruce L. Golden, and Edward A. Wasil. A new heuristic for the period traveling salesman problem. *Comput. Oper. Res.*, 22(5):553–565, May 1995.

[14] I-Ming Chao, Bruce L. Golden, and Edward A. Wasil. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88(3):475–489, 1996.

[15] Vassilis Charitopoulos, Vivek Dua, and Lazaros G Papageorgiou. Travelling salesman problem (tsp) based integration of planning, scheduling and optimal control for continuous processes. *Industrial and Engineering Chemistry Research*, 56, 08 2017.

[16] Omar Cheikhrouhou and Ines Khoufi. A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy. *Computer Science Review*, 40:100369, 2021.

[17] N. Christofides and J. E. Beasley. The period routing problem. *Networks*, 14(2):237–256, 1984.

[18] N. Christofides, A. Mingozzi, and P. Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20(1):255–282, Dec 1981.

[19] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report ADA025602, Carnegie Mellon University, Management Sciences Research Group, 0 1976.

[20] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[21] William Cook. Concorde home. `https://www.math.uwaterloo.ca/tsp/concorde.html`, 2020. Accessed: 2021-10-11.

[22] Jean-François Cordeau, Michel Gendreau, and Gilbert Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119, 1997.

[23] Éric D. Taillard, Philippe Badeau, Michel Gendreau, François Guertin, and Jean-Yves Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31:170–186, 05 1997.

[24] M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.

[25] K A Dowsland and J M Thompson. Solving a nurse scheduling problem with knapsacks, networks and tabu search. *Journal of the Operational Research Society*, 51(7):825–833, Jul 2000.

[26] Vu Duc Minh, Mike Hewitt, Natashia Boland, and Martin Savelsbergh. Solving time dependent traveling salesman problem with time windows. *Optimization Online*, 06 2018.

[27] Charbel Farhat. A simple and efficient automatic fem domain decomposer. *Computers & Structures*, 28(5):579–602, 1988.

[28] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181, 1982.

[29] Leszek Gasieniec, Ralf Klasing, Christos Levcopoulos, Andrzej Lingas, Jie Min, and Tomasz Radzik. Bamboo garden trimming problem (perpetual maintenance of machines with different attendance urgency factors). In Bernhard Steffen, Christel Baier, Mark van den Brand, Johann Eder, Mike Hinchey, and Tiziana Margaria, editors, *SOFSEM 2017: Theory and Practice of Computer Science*, pages 229–240, Cham, 2017. Springer International Publishing.

[30] Bezalel Gavish and Kizhanathan Srikanth. An optimal solution method for large-scale multiple traveling salesmen problems. *Operations Research*, 34(5):698–717, 1986.

[31] Michel Gendreau, Alain Hertz, Gilbert Laporte, and Mihnea Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3):330–335, 1998.

[32] Michel Gendreau, Gilbert Laporte, and Frédéric Semet. A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, 106(2):539–545, 1998.

[33] Fred Glover, Gregory Gutin, Anders Yeo, and Alexey Zverovich. Construction heuristics for the asymmetric tsp. *European Journal of Operational Research*, 129(3):555–568, 2001.

[34] Asvin Goel. Vehicle scheduling and routing with drivers' working hours. *Transportation Science*, 43:17–26, 02 2009.

[35] B. Golden, L. Bodin, T. Doyle, and W. Stewart. Approximate traveling salesman algorithms. *Operations Research*, 28(3-part-ii):694–711, 1980.

[36] Bruce L. Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

[37] Ralph E. Gomory. Early integer programming. *Operations Research*, 50(1):78–81, 2002.

[38] J. Gromicho, J. Paixão, and I. Bronco. Exact solution of multiple traveling salesman problems. In Mustafa Akgül, Horst W. Hamacher, and Süleyman Tüfekçi, editors, *Combinatorial Optimization*, pages 291–292, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[39] G. Gutin and A.P. Punnen. *Experimental Analysis of Heuristics for the STSP*, pages 369–443. Springer-Verlag US 2007, 2007.

[40] William W. Hardgrave and George L. Nemhauser. On the relation between the traveling-salesman and the longest-path problems. *Operations Research*, 10(5):647–657, 1962.

[41] William E Hart, Jean-Paul Watson, and David L Woodruff. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011.

[42] Refael Hassin and Ariel Keinan. Greedy heuristics with regret, with application to the cheapest insertion algorithm for the tsp. *Operations Research Letters*, 36(2):243 – 246, 2008.

[43] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[44] Vera Hemmelmayr, Karl Doerner, and Richard Hartl. A variable neighborhood search heuristic for periodic routing problems. *European Journal of Operational Research*, 195:791–802, 06 2009.

[45] Weihuang Huang and Jeffrey Yu. Investigating tsp heuristics for location-based services. *Data Science and Engineering*, 2, 03 2017.

[46] Barbar Katja Höck. *An examination of heuristic algorithms for the travelling salesman problem.* PhD thesis, University of Cape Town, 1988.

[47] K. Ilavarasi and K. S. Joseph. Variants of travelling salesman problem: A survey. In *International Conference on Information Communication and Embedded Systems (ICICES2014)*, pages 1–7, 2014.

[48] David S. Johnson and Lyle A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. Wiley, 1997.

[49] Robert L. Karg and Gerald L. Thompson. A heuristic approach to solving travelling salesman problems. *Management Science*, 10(2):225–248, 1964.

[50] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

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

[52] R.V. Kulkarni and P.R. Bhave. Integer programming formulations of vehicle routing problems. *European Journal of Operational Research*, 20(1):58–67, 1985.

[53] Gilbert Laporte and Yves Nobert. A cutting planes algorithm for the m-salesmen problem. *Journal of the Operational Research Society*, 31(11):1017–1023, 1980.

[54] Joseph Leung, Laurie Kelly, and James H. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Inc., Boca Raton, FL, USA, 2004.

[55] John D. C. Little, Katta G. Murty, Dura W. Sweeney, and Caroline Karel. An algorithm for the traveling salesman problem. *Operations Research*, 11(6):972–989, 1963.

[56] Y. Liu, I. H. Khalifa, and A. E. Kamel. The multi-period and multi-depot dynamic vehicle routing problem with time windows. In *2016 3rd International Conference on Logistics Operations Management (GOL)*, pages 1–6, May 2016.

[57] Zhenyuan Liu, Zaisheng Liu, Zhipeng Zhu, Yindong Shen, and Junwu Dong. Simulated annealing for a multi-level nurse rostering problem in hemodialysis service. *Applied Soft Computing*, 64:148 – 160, 2018.

[58] Jens Lysgaard, Adam Letchford, and Richard W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100:423–445, 06 2004.

[59] Thiago A.S. Masutti and Leandro N. de Castro. A self-organizing neural network using ideas from the immune system to solve the traveling salesman problem. *Information Sciences*, 179(10):1454 – 1468, 2009. Including Special Issue on Artificial Imune Systems.

[60] C. E. Miller, Albert W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7:326–329, 1960.

[61] A. Modares, S. Somhom, and T. Enkawa. A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems. *International Transactions in Operational Research*, 6(6):591–606, 1999.

[62] John P. Norback and Robert F. Love. Geometric approaches to solving the traveling salesman problem. *Management Science*, 23(11):1208–1223, 1977.

[63] Michael G. Norman and Pablo Moscato. The euclidean traveling salesman problem and a space-filling curve. *Chaos, Solitons and Fractals*, 6:389–397, 1995. Complex Systems in Computational Physics.

[64] H.L. Ong and H.C. Huang. Asymptotic expected performance of some tsp heuristics: An empirical evaluation. *European Journal of Operational Research*, 43(2):231–238, 1989.

[65] G. Paletta. A multiperiod traveling salesman problem: Heuristic algorithms. *Computers and Operations Research*, 19(8):789–795, 1992.

[66] Giuseppe Paletta. The period traveling salesman problem: a new heuristic algorithm. *Computers and Operations Research*, 29(10):1343–1352, 2002.

[67] Ulrich Pferschy and Rostislav Stanek. Generating subtour elimination constraints for the tsp from pure integer solutions. *Central European Journal of Operations Research*, 25, 03 2017.

[68] Michael Pinedo. *Scheduling*, pages 375–398. Springer International Publishing, 2015.

[69] Loren K. Platzman and John J. Bartholdi. Spacefilling curves and the planar travelling salesman problem. *J. ACM*, 36(4):719–737, oct 1989.

[70] Ira Pohl. Practical and theoretical considerations in heuristic search algorithms. *Machine Intelligence 8*, pages 55–72, 01 1977.

[71] Michael Polacek, Karl F. Doerner, Richard F. Hartl, Guenter Kiechle, and Marc Reimann. Scheduling periodic customer visits for a traveling salesperson. *European Journal of Operational Research*, 179(3):823–837, 2007.

[72] Jean-Yves Potvin, Guy Lapalme, and Jean-Marc Rousseau. A generalized k-opt exchange procedure for the mtsp. *INFOR: Information Systems and Operational Research*, 27(4):474–481, 1989.

[73] R. Preis and R. Diekmann. Party - a software library for graph partitioning. *Advances in Computational Mechanics for Parallel and Distributed Processing*, pages 63–71, 1997.

[74] Harilaos N. Psaraftis, Min Wen, and Christos A. Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Netw.*, 67(1):3–31, January 2016.

[75] R. Ramesh and Kathleen M. Brown. An efficient four-phase heuristic for the generalized orienteering problem. *Computers and Operations Research*, 18(2):151–165, 1991.

[76] T. C. Raymond. Heuristic algorithm for the traveling-salesman problem. *IBM J. Res. Dev.*, 13(4):400–407, July 1969.

[77] César Rego and Fred Glover. *Local Search and Metaheuristics*, pages 309–368. Springer, Boston, MA, 2007.

[78] G. Reinelt. *Construction Heuristics*, pages 73–99. Springer-Verlag Berlin Heidelberg, 1994.

[79] Gerhard Reinelt. Tsplib 95. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf`, 1995. Accessed: 2022-07-27.

[80] Gerhard Reinelt. Tsplib. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`, 2013. Accessed: 2021-10-11.

[81] Roberto Roberti and Mario Ruthmair. Exact methods for the traveling salesman problem with drone. *Transportation Science*, 55(2):315–335, 2021.

[82] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis. Approximate algorithms for the traveling salesperson problem. In *15th Annual Symposium on Switching and Automata Theory (swat 1974)*, pages 33–42, 1974.

[83] Daniel Rosenkrantz, Richard Stearns, and Philip Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6:563–581, 09 1977.

[84] Robert A. Russell. An effective heuristic for the m-tour traveling salesman problem with some side conditions. *Operations Research*, 25(3):517–524, 1977.

[85] J.L. Ryan, T.G. Bailey, J.T. Moore, and W.B. Carlton. Reactive tabu search in unmanned aerial reconnaissance simulations. In *1998 Winter Simulation Conference. Proceedings (Cat. No.98CH36274)*, volume 1, pages 873–879 vol.1, 1998.

[86] Shakila Saad, Wan Nurhadani Wan Jaafar, and Siti Jasmida Jamil. Solving standard traveling salesman problem and multiple traveling salesman problem by using branch-and-bound. *AIP Conference Proceedings*, 1522(1):1406–1411, 2013.

[87] Danilo Sanches, Darrell Whitley, and Renato Tinós. Improving an exact solver for the traveling salesman problem using partition crossover. In *Proceedings of the Genetic*

and Evolutionary Computation Conference, GECCO '17, page 337–344, New York, NY, USA, 2017. ACM.

[88] Chi-Hwa Song, Kyunghee Lee, and Won Lee. Extended simulated annealing for augmented tsp and multisalsemen tsp. *Proceedings of the International Joint Conference on Neural Networks*, 3:2340 – 2343 vol.3, 08 2003.

[89] Kenneth J. Supowit, David A. Plaisted, and Edward M. Reingold. Heuristics for weighted perfect matching. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, page 398–419, New York, NY, USA, 1980. Association for Computing Machinery.

[90] Hao Tang, Elise Miller-Hooks, and Robert Tomastik. Scheduling technicians for planned maintenance of geographically distributed equipment. *Transportation Research Part E: Logistics and Transportation Review*, 43:591–609, 09 2007.

[91] Lixin Tang, Jiyin Liu, Aiying Rong, and Zihou Yang. A multiple traveling salesman problem model for hot rolling scheduling in shanghai baoshan iron & steel complex. *European Journal of Operational Research*, 124(2):267–282, 2000.

[92] Theodore Tsiligiridis. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35:797–809, 09 1984.

[93] Ziauddin Ursani and David Corne. Introducing complexity curtailing techniques for the tour construction heuristics for the travelling salesperson problem. *Journal of Optimization*, 2016, 08 2016.

[94] Isaac Vandermeulen, Roderich Groß, and Andreas Kolling. Balanced task allocation by partitioning the multiple traveling salesperson problem. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, page 1479–1487, 2019.

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

[96] F. P. Vasilyev and A. Yu. Ivanitskiy. *Dual Simplex Method*, pages 119–166. Springer Netherlands, Dordrecht, 2001.

[97] E Wacholder, J Han, and R˙ C Mann. A neural network algorithm for the multiple traveling salesmen problem. *Biological Cybernetics*, 61(1):11–19, 1989.

[98] M. H. J. Webb. Some methods of producing approximate solutions to travelling salesman problems with hundreds or thousands of cities. *Journal of the Operational Research Society*, 22(1):49–66, 1971.

[99] Gerhard J. Woeginger. *Exact Algorithms for NP-Hard Problems: A Survey*, pages 185–207. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[100] Marlin Wolf Ulmer. Approximate dynamic programming for dynamic vehicle routing. *Operations Research/ Computer Science Interfaces Series*, 61, 01 2017.

[101] Zuoyong Xiang, Zhenyu Chen, Xingyu Gao, Xinjun Wang, Fangchun Di, Lixin Li, Guangyi Liu, and Yi Zhang. Solving large-scale tsp using a fast wedging insertion partitioning approach. *Mathematical Problems in Engineering*, 2015:854218, Nov 2015.

[102] Majid Yousefikhoshbakht, Farzad Didehvar, and Farhad Rahmati. Modification of the ant colony optimization for solving the multiple traveling salesman problem. *Romanian Journal of Information Science and Technology*, 16:65–80, 12 2013.

[103] Zhong Yu, Liang Jinhai, Gu Guochang, Zhang Rubo, and Yang Haiyan. An implementation of evolutionary computation for path planning of cooperative mobile robots. In *Proceedings of the 4th World Congress on Intelligent Control and Automation (Cat. No.02EX527)*, volume 3, pages 1798–1802 vol.3, 2002.

[104] Tiehua Zhang, W.A. Gruver, and M.H. Smith. Team scheduling by genetic search. In *Proceedings of the Second International Conference on Intelligent Processing and Manufacturing of Materials. IPMM'99 (Cat. No.99EX296)*, volume 2, pages 839–844 vol.2, 1999.