Optimisation for Large-scale Maintenance, Scheduling and Vehicle Routing Problems

Yujie Chen

Doctor of Engineering

University of York
Computer Science

September, 2016

Abstract

Solving real-world combinatorial problems is involved in many industry fields to minimise operational cost or to maximise profit, or both. Along with continuous growth in computing power, many asset management decision-making processes that were originally solved by hand now tend to be based on big data analysis. Larger scale problem can be solved and more detailed operation instructions can be delivered.

In this thesis, we investigate models and algorithms to solve large scale Geographically Distributed asset Maintenance Problems (GDMP). Our study of the problem was motivated by our business partner, Gaist solutions Ltd., to optimise scheduling of maintenance actions for a drainage system in an urban area. The models and solution methods proposed in the thesis can be applied to many similar issues arising in other industry fields.

The thesis contains three parts. We firstly built a risk driven model considering vehicle routing problems and the asset degradation information. A hyperheuristic method embedded with customised low-level heuristics is employed to solve our real-world drainage maintenance problem in Blackpool. Computational results show that our hyperheuristic approach can, within reasonable CPU time, produce much higher quality solutions than the scheduling strategy currently implemented by Blackpool council.

We then attempt to develop more efficient solution approaches to tackle our GDMP. We study various hyperheuristics and propose efficient local search strategies in part II. We present computational results on standard periodic vehicle routing problem instances and our GDMP instances. Based on manifold experimental evidences, we summarise the principles of designing heuristic based solution approaches to solve combinatorial problems.

Last bu not least, we investigate a related decision making problem from highway maintenance, that is again of interest to Gaist solutions Ltd. We aim to make a strategical decision to choose a cost effective method of delivering the road inspection at a national scale. We build the analysis based on the Chinese Postman Problem and theoretically proof the modelling feasibility in real-world road inspection situations. We also propose a novel graph reduction process to allow effective computation over very large data sets.

Contents

List of Tables

List of Figures

Acknowledgements

I firstly would like to thank my main supervisor, Dr Fiona Polack for her guidance, support on the research and help with writing the thesis. Very importantly, her encouragement helps me to go through the journey of EngD tasks. I would also like to thank my co-supervisor Professor Peter Cowling for his supervision and help during my entire EngD.

I am also very grateful to my industrial partners, Gaist Solutions Ltd., for providing real-life data and information for this research. Especially, I would like to thank Dr Stephen Remde and Steve Birdsall from Gaist, for their supervision, domain knowledge support and the invitations to many research meetings with local councils in the UK. Without their help, many research tasks could not go forward.

I would also like to thank the Engineering and physical sciences research council (EP-SRC) for the funding of this research project.

I would also like to thank my colleagues and friends in Artificial Intelligence group, for their discussions and inspirations from various research domains. These discussions give me a general view about many related techniques that can be applied in solving different interesting problems. Especially, I would like to thank my friend, Philip Mourdjis, for his countless discussions and supports during my research.

Finally, I would like to thank my parents for their supports and love throughout my life.

Author's declaration

I declare that this thesis is a presentation of original work and I am the sole author. All information in this thesis has been presented in accordance with academic rules. The research in this thesis includes work that has been previously published or submitted for publication.

The following papers are about the modelling and solving of our drainage system maintenance scheduling problem using existing meta-heuristic and hyperheuristic approaches.

1. Chen, Y., Cowling, P. and Remde, S. (2014). Dynamic Period Routing for a Complex Real-World System : A Case Study in Storm Drain Maintenance. In *Evolutionary Computation in Combinatorial Optimisation*. 8600: 109–120.

   This paper introduces a very early version of the model for our drainage system maintenance problem. In this work, some artificial resource data is used to deliver analysis, due to the lack of specific domain knowledge at the time. To start the research, I solved the maintenance scheduling problem for our drainage system with half the number of asset points, using variable neighbourhood search (Hansen et al. (2010)). In Part I of this thesis, a slightly modified model with deeper analysis of resource data is presented. Furthermore, techniques such as problem size reduction and an enhanced problem solver allow us to deal with the full sized problem. Therefore, this thesis only covers the new model and the analysis based on the new model.

2. Chen, Y., Polack, F., Cowling, P., Mourdjis, P. and Remde, S. (2016f). Risk Driven Analysis of Maintenance for a Large-scale Drainage System. In *Proceedings of 5th the International Conference on Operations Research and Enterprise Systems*: 296–303 (Won the best application paper award and included in Chapter 5.)

3. Chen, Y., Polack, F., Cowling, P., Mourdjis, P. and Remde, S. (2016e). Exploring Techniques to Improve Large-Scale Drainage System Maintenance Scheduling Using a Risk Driven Model. In *Communications in Computer and Information Science*

(submitted) (Included in Chapter 5.)

4. Chen, Y., Cowling, P., Polack, F., Remde, S. and Mourdjis, P. (2016b). Dynamic optimization of preventative and corrective maintenance schedules for a large scale urban drainage system. *European journal of operational research* 257(2): 494–510 (Chapter 6.)

In the following papers, we focus on the study of local search, meta-heuristic and hyperheuristic. To understand the behaviour of these algorithms, we test them on benchmark problem instances that are closely related to our drainage system maintenance problem, and on the problem itself.

1. Chen, Y., Mourdjis, P., Polack, F., Cowling, P. and Remde, S. (2016d). Evaluating Hyperheuristics and Local Search Operators for Periodic Routing Problems. In *Evolutionary Computation in Combinatorial Optimisation*: 104–120 (Chapter 7.)

2. Chen, Y., Cowling, P., Polack, F. and Mourdjis, P. (2016a). A multi-arm bandit neighbourhood search for routing and scheduling problems. *Journal of Heuristics (submitted)* (Chapter 8.)

The content in Chapter 9 has also been described in following publications.

1. Chen, Y., Cowling, P., Remde, S. and Polack, F. (2016c). Efficient Large-scale Road Inspection Routing. In *Proceedings of 5th the International Conference on Operations Research and Enterprise Systems*: 304–312

2. Chen, Y., Polack, F., Cowling, P. and Remde, S. (2016g). A comparison of one-pass and bi-directional approaches applied to large-scale road inspection. In *Communications in Computer and Information Science (submitted)*

Chapter 1

Introduction

1.1 Motivation and research target

Public asset management has asserted its importance in many sectors, including water supply, energy distribution, waste disposal, transportation system management and many others. In an advanced society, it is important to keep the city's fundamental infrastructures performing their designed functions.

In transportation system management, the performance of the highway network has a serious impact on the social, local economic development and environmental well being of the community (Department for Transport (2013)). Due to the nature of ageing behaviour of highway infrastructure and severe weather effects such as flooding, maintenance planning is vital to keep the highway system in good condition. Poor road conditions may cause significant losses to business (Asphalt Industry Alliance (2010)) and increase the chances of accidents and disruption.

The highway system includes assets such as carriageways, foot-ways, trees, lighting system, street furniture and drainage system. In recent years, local authorities have increased their awareness of the importance of asset maintenance, which aims to deliver more efficient and effective approaches to manage the highway infrastructures through continuous planning. Many projects have been deployed in the UK to produce more "intelligent" services that achieve the same outcome with lower cost or better outcomes. Example projects can be found in highway asset management plans carried out in London (Transport For London (2007)) and Hampshire Hamphshire (2016).

In cooperation with our business partner, Gaist Solutions Ltd., this research investigates a maintenance scheduling task for a drainage system, and uses it as a case study to come up with a general model for many Geographically Distributed asset Maintenance Problems

(GDMP). The aim is to provide a data supported decision making system that delivers robust asset maintenance operation strategies in various scenarios.

In many cases in the UK, the drainage system maintenance is also tightly coupled to environmental concerns such as water safety issues (e.g. biochemistry) and flooding risk control. Our research with Gaist seeks a solution that includes a high level estimation and quantification of the condition of a local authority's drainage system, and a detailed despatching system that prioritises maintenance actions with consideration of the suitability of the drainage system to deal with present and future flood risk (Department for Transport (2012)).

In a typical city in the UK, a drainage system contains about 10,000 to over 100,000 gully-pots located across the city road network, as shown in Figure 1.1. A maintenance team is responsible for ensuring that the gully-pot system performs its designed function – to help the surface water to drain away from the roads quickly. To deliver an efficient and effective service to such a system, there is an attempt to reduce the operation cost of visiting these assets, focusing limited resources towards the greatest need. Many approaches could be taken from operational research (OR) techniques; modelling of the vehicle routing problem (VRP) is a good place to start. The VRP models are considered whenever multiple locations need to be visited within one period or over multiple periods, given limited vehicle/human resource. During the past decades, a considerable amount of research on vehicle routing and scheduling problems has been carried out. In order to cope with real-world complexity, additional features and constraints could be added to the standard VRP models to fit our real-world needs.

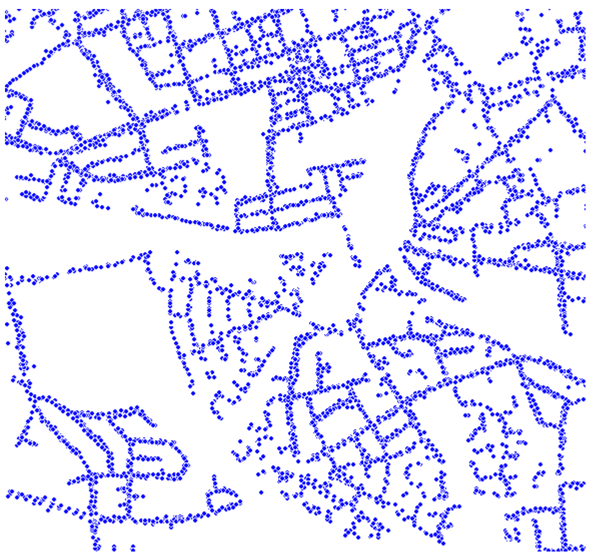


Figure 1.1: Geographically distributed gully-pots (a small area in Stockport, UK)

Solving a real-wrold OR problem consists both of problem modelling and solving the modelled problem. The outcome of a problem solver is a series of actions that could be taken in the real-world scenarios. Problem modelling and solving are equally important to deliver a complete successful solution. Developing well-performing algorithms for optimisation in OR has been much studied. However, research still faces challenges in tackling many of the issues that arise from real-world applications, such as large scale, uncertain information, limited computation power and so on. More effort should be devoted to developing more efficient, robust, and easily applicable algorithms. The second aim of this thesis therefore is to investigate the development of good solution methods for large scale optimisation problems.

Meanwhile, we also address a related highway management issue, the road inspection problem. It provides the latest key information on road and surrounding asset condition, which helps decision making on the following maintenance planning. This is another specific interest of our business partner, Gaist Solutions Ltd.. The interest of this study is not only in the problem itself, but also in its large scale. We aim to make a strategic decision to choose a cost-effective approach to inspect the road networks in the UK at a national scale.

1.2 Achievements and thesis overview

This thesis starts with an introduction to problem modelling and solving techniques for general vehicle routing and scheduling problems in Chapter 2. Then Chapter 3 discusses a number of advanced techniques that are studied in heuristic approaches, which inspires the design of our algorithms later in the thesis. The rest of the thesis can be divided into three parts, summarised as follows.

Part I We start with a comprehensive analysis of the drainage system maintenance problem in Chapter 4, and propose a risk-driven model that captures the critical features of the problem. We believe this model could also be directly used or slightly modified to solve similar asset maintenance problems mentioned in Section 1.1. We deliver a series of simulation based analysis to discover the weakness of the current maintenance scheduling strategy in Chapter 5. In further research, Chapter 6 introduces a predictive scheduling strategy that automatically adjust the maintenance actions according to environment and system status information changes. To deal with large scale problems, we propose a systematic way of grouping asset points, resulting in substantial reduction of problem size and little effects to the solution quality. A detailed comparison of the long term impact between using different maintenance policies is also presented.

Part II Having proposed a successful heuristic based solution method to produce the optimised schedule in Chapter 6, there remains ample room for improvement in method development. In the later stages of my EngD journey, I moved towards designing heuristic based search algorithms with machine learning. Hyperheuristics and statistical learning based local search techniques are presented in Chapter 7 and Chapter 8, respectively. In Chapter 7, we aim to discover the essential components of developing a successful hyperheuristic algorithm and to analyse the behaviours of hyperheuristics solving problems with different characteristics. Detailed performance analysis of a range of hyperheuristics is presented. In Chapter 8, we propose a novel dynamic multi-arm bandit neighbourhood search algorithm, which utilises some interesting properties of our drainage system maintenance problem to deliver an efficient search process. This algorithm introduces various search techniques and successfully out-performs many existing algorithms studied in the earlier chapters.

Part III In the last part of this thesis, we focus on the road inspection problem. The target is not to build a detailed vehicle dispatching schedule for daily inspection. Rather, we consider a high level decision making in OR, to determine a cost effective way of delivering the entire road inspection project (Chapter 9). More specifically, we compare the total distance vehicles should travel using two potential road inspection approaches, one-pass and bi-directional. In order to deal with very large-scale road networks, we develop a graph reduction method that allows the accurate analysis of a problem that was previously computationally infeasible. Our graph reduction approach does not lose any necessary information to derive the exact solution. With assists from Gaist of domain knowledge, we estimate an annual saving across all UK local authorities of £4.32 million, if the more efficient one-pass road inspection strategy is applied. Gaist is now using our approach in planning its national scale road inspection programme.

Chapter 2

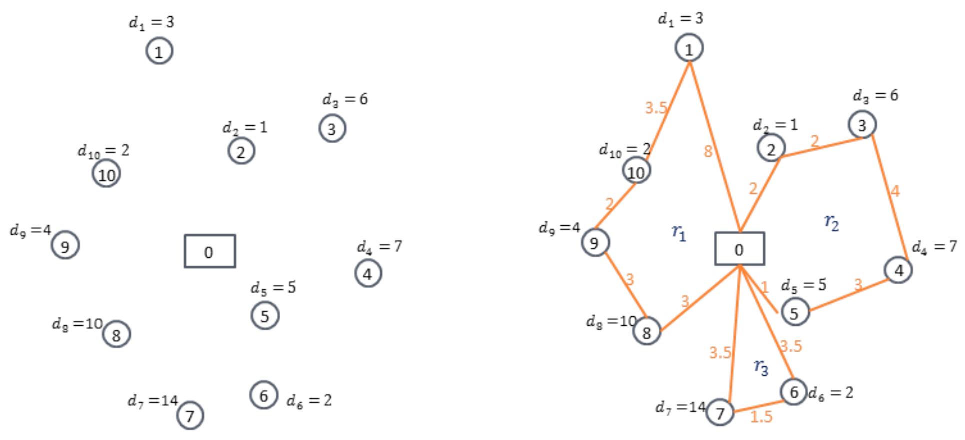
Real-world complex routing problem

Many real-world on-site service and maintenance problems are modelled as vehicle routing problems with various objectives and complex constraints. The objective of this chapter is to have an overview of how to model practical routing problems and discusses the solvers that are commonly used. In order to do so, we firstly introduce the basic version of vehicle routing problem (VRP). Then, four well-studied variations of VRP are introduced, as they share some common features with our geographically distributed asset maintenance problem (GDMP, Chapter 4). Finally, we summarise the aspects that need to be considered when modelling real-world problems.

2.1 Vehicle routing problems

The VRP is often described as a problem of designing a set of optimal delivery routes from a central depot or several depots to a number of geographically distributed customers subject to constraints (Toth and Vigo (2002)). The most basic version of VRP, capacitated VRP (CVRP), only considers the vehicle capacity constraints and one central depot. Figure 2.1 presents an example of CVRP and one of its feasible solutions.

Formally, let $G = (V, A)$ be a complete graph, where $V = \{0, ...n\}$ is the node set and A is the arc set. There are $i \in V/\{0\}$ customers to be served by a central depot 0, using K independent vehicles of an identical capacity Q. Each arc (i, j) is associated with a cost $c_{i,j}$. If $\forall\ (i, j) \in A, c_{i,j} = c_{j,i}$, this problem is a symmetric CVRP, and it is asymmetric otherwise. Each customer i has a non-negative demand q_i. To ensure feasibility, we assume that $\forall\ i \in V, q_i \leq Q$. The objective is to construct a set of least-cost routes that satisfies



(a) Geographically distributed customers and their demand information d_i

(b) Example of a feasible solution of the CVRP in (a)

Figure 2.1: An example of CVRP and its solution. In this problem, we have three vehicles to serve ten customers. Each vehicle has a capacity of 20. The demand of each customer and their locations are illustrated in sub-figure (a). A feasible solution is presented in sub-figure (b) and its total travel distance is 40. Distances are labelled next to each edge.

the following constraints:

1. Each route starts and ends at the depot, represented as a sequence $r = (0, ., i, .., j, ., 0)$;
2. Each customer i is visited exactly once by exactly one vehicle;
3. The sum of demands of all customers visited by each vehicle does not exceed the vehicle capacity. $\sum_{i \in r} q_i \leq Q$;

CVRP is one of the simplest, but still NP-hard (Toth and Vigo (2002)), variations of the VRP. It was firstly defined by Dantzing and Ramser (1959). In the last half century, a large quantity of solution methods have been proposed, including exact and heuristic methods. Exact methods can obtain optimal solutions. However, this type of approaches is normally used to solve small-sized instances. To solve large-scale problems, heuristic approaches are often used. In the following subsections, we firstly introduce a few commonly used mathematical models and exact methods to solve VRPs. Then we focus on the heuristic approaches.

2.1.1 Common models and exact methods

In literature, a large quantity of mathematical models are proposed to solve the VRPs since 1960s. Generally speaking, most of the models can be divided into three groups (Laporte and Nobert (1987)): 1) vehicle flow formulations; 2) commodity flow formulations and 3) set partitioning formulations. The first group of models, such as the two-index and three-index vehicle-flow models for CVRP, are the most widely used by far. The second group of formulations were firstly proposed by Garvin et al. (1957) in an oil delivery problem. Compared to the vehicle flow formulations, the commodity flow formulations add an associate flow variable with each arc (i, j), to emphasis how much of the demand destined for a customer i travelled on arc (i, j). The commodity flow model can easily be transformed to deal with the situation of heterogeneous fleet (Magnanti (1981)) and separable goods delivery. The set partitioning formulations are often used to tackle larger VRPs. In the following, we select a two-index vehicle flow model by Laporte et al. (1985) and a set partitioning model by Balinski and Quandt (1964) to present in detail, due to their large number of successful applications. For more examples of VRP models, we suggest a survey paper by Laporte (1992).

2.1.1.1 Two-index vehicle-flow model of CVRP

Laporte et al. (1985) proposed the following two-index formulation for the symmetric CVRP. Let the integer variable $x_{i,j}(i < j)$ represent the number of times that arc $(i, j) \in A$ is traversed in the solution.

$$minimise \sum_{i<j;i,j\in V} c_{i,j}x_{i,j} \tag{2.1}$$

$$\sum_{j\in V/\{0\}} x_{0,j} = 2K \tag{2.2}$$

$$\sum_{i<l} x_{i,l} + \sum_{j>l} x_{l,j} = 2, \quad \forall l \in V/\{0\} \tag{2.3}$$

$$\sum_{i,j\in S} x_{i,j} \leq |S| - v(S), \quad S \subseteq V/\{0\}, 2 \leq |S| \leq |V| - 2 \tag{2.4}$$

$$x_{i,j} \in \{0, 1\}, \quad \forall i, j \in V/\{0\} \tag{2.5}$$

$$x_{0,j} \in \{0, 1, 2\}, \quad \forall j \in V/\{0\} \tag{2.6}$$

The objective function (2.1) is straightforward that aims to minimise the total cost of arcs included in the solution. The constraint (2.2) ensures K number of vehicles are used and the degree constraint (2.3) ensures each customer is visited exactly once in the solution. The constraint (2.4) is the route elimination formulation, where $v(S)$ denotes the minimum number of vehicles required to visit the node set S. The value of $v(S)$ can be obtained by solving a bin packing problem with the item set S and bins of capacity Q. In the same paper, Laporte et al. (1985) described a constraint relaxation algorithm that solved the problem with up to 60 nodes.

By adopting similar models, a large number of exact methods based on branch and bound (B&B) have been developed. The key idea of B&B is to divide the search space into sub-areas (branching). Then the evaluations of lower/upper bounds of these sub-areas are calculated (bounding). If the evaluation value indicates that a sub-area does not contain the optimal solution, the sub-area is pruned. A general approach to estimate a bound is to solve a relaxed problem (such as removing or relaxing a constraint of an integer linear programming model). Depending on the mechanism used to calculate the bounds, algorithms in this family, such as branch-and-cut (e.g. Blasum and Hochstättler (2002); Lysgaard et al. (2004)) and branch-and-price (e.g. Dell'Amico et al. (2006)), have been successfully applied to vehicle routing problems.

2.1.1.2 Set partitioning model of CVRP

Another well known formulation of the CVRP is the set partitioning formulation, in which all the feasible routes are enumerated (Balinski and Quandt (1964)). This model uses a very different concept and its objective is to find an optimal set of routes that form a feasible solution with minimum travelling cost. Let R denote the set of all feasible routes, where each route $r \in R$ does not exceed the vehicle capacity constraint. The cost of each route is measured in distance c_r. Note that c_r is the cost of the shortest route that visits all the customers included in r. A variable a_i^r describes whether a customer i is visited by route r. A decision variable x_r equals 1 if route r is included in the solution and equals 0 otherwise. The model is presented as follows:

$$minimise \sum_{r \in R} c_r x_r \tag{2.7}$$

$$\sum_{r \in R} a_i^r x_r = 1, \forall i \in V/\{0\} \tag{2.8}$$

$$\sum_{r \in R} x_r \leq K \tag{2.9}$$

$$x_r \in \{0,1\}, \forall r \in R \tag{2.10}$$

The objective function (2.7) selects a set of routes that minimise the total travelling cost. Constraint (2.8) ensures each customer is served by one selected route. Constraint (2.9) presents the maximum number of routes can be selected into the solution.

Using the set partitioning formulations to solve VRPs usually involves a very large set of feasible routes, which makes the enumeration impossible. A natural way around the difficulties is to use a column generation algorithm. An example solution procedure can be found in Desrosiers et al. (1984). As one can expect that this model will work better on more tightly constrained problems, as the number of feasible routes (columns) is smaller (Laporte (1992)).

2.1.1.3 Further discussion

The progress made in the last ten years on the exact CVRP algorithms is considerable. Various techniques are combined together to enhance exact solvers' efficiency. Fukasawa et al. (2006) combine branch-and-cut and column generation to generate more effective bounds than each of those methods taken alone. This method is named as branch-cut-and-price (Fukasawa et al. (2006)), and solves a number of previously unsolvable CVRP benchmark instances. Subsequently, a number of improved implementations of branch-cut-and-price algorithm have been developed in recent years (e.g. Ropke (2012); Pecin et al. (2016)).

More details of these methods and recent surveys on exact methods for the CVRP and its variations can be found in (Baldacci et al. (2007); Cordeau et al. (2007); Baldacci et al. (2012); Poggi and Uchoa (2014)). To our knowledge, the state-of-the-art exact methods can solve CVRP instances with 200 customers in a consistent way (Uchoa et al. (2016)). Some larger instances (Golden et al. (1998)), ranging from 240 to 360 customers, are also solved using branch-cut-and-price by Pecin et al. (2016). However, when we face larger scale real-world routing problems (with over thousands of customers and more complex constraints), even the best exact algorithm will face huge challenges. In addition, the computational time needed for exact methods is significantly longer than heuristic based algorithms (next section). In the experiment set by Uchoa et al. (2016)[1], the average CPU time used to solve CVRP instances with size from 150 to 300 customers is about 1,000 seconds, whereas

[1]These tests have been conducted on a Xeon CPU with 3.07 GHz and 16 GB of RAM, running under Oracle Linux Server 6.4.

heuristic based approaches (e.g. iterative local search) used on average about 5 seconds to produce solutions that are less than 1% above optimal for the same instances. In this thesis, we study a very large-scale GDMP, with about 30,000 nodes, and therefore we focus on heuristic based approaches.

2.1.2 Heuristic methods

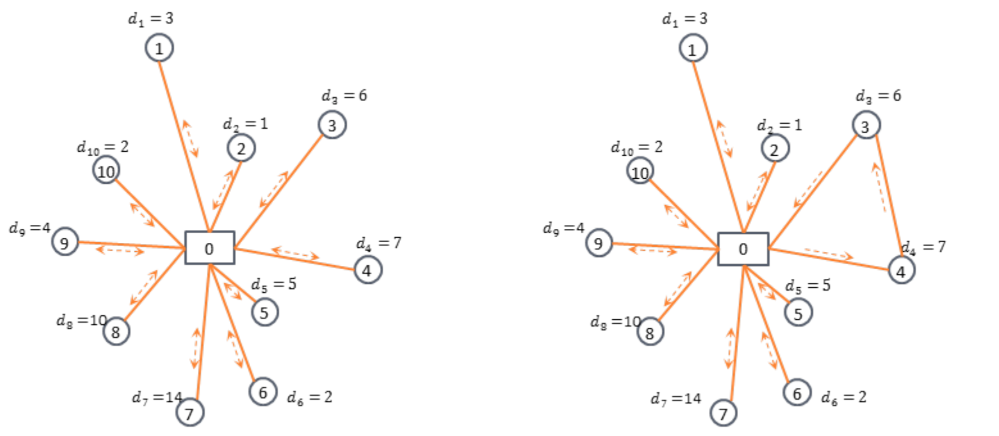
Heuristics play an important role in solving large-scale combinational optimisation problems. These methods can be considered as a search process that iteratively constructs or modifies a single or several candidate solutions. Different heuristics apply different rules to walk within the solution space. Comparing to exact approaches using systematic search structure, heuristics usually use local information from recently checked solutions. To intelligently and efficiently move towards good solutions is the key to the success of a heuristic.

Along with the evolution of heuristic solvers for VRPs, we have witnessed the developing trend from simple heuristics to hybridisation of several techniques initially developed independently. This section aims to give a general view of using heuristic to solve VRPs. In the following, we will introduce algorithms from constructive heuristics, improvement heuristics and meta-heuristics. We will also briefly discuss the developing trend of heuristic approaches. In the next chapter, we give a deeper discussion of more advanced techniques designed to improve heuristic search efficiency.

2.1.2.1 Constructive heuristics

Constructive heuristics iteratively build a complete (usually feasible) solution from scratch. The earliest research can be traced back to 1964, a distance saving heuristic proposed by Clarke and Wright (1964). Later on, more constructive heuristics for VRP, such as insertion heuristic (e.g. Christofides (1976a)) and two-phase heuristic (e.g. Gillett and Miller (1974)) gradually appeared from 1970s to 1990s. In the recent years, constructive heuristics are usually employed to generate an initial solution for improvement heuristics and meta-heuristics. However, to improve search diversity, many modern meta-heuristics multi-start the search from randomly generated solutions. Here, we present two representative algorithms that are still widely applied in modern solvers.

The Clarke and Wright algorithm The Clarke-Wright algorithm (Clarke and Wright (1964)) is designed to solve CVRP that starts from constructing n routes $r = (0, i, 0)$ for a problem with n customers ($i = 1, ..., n$). There is no limitation on the number of vehicles used in the solved problem. The heuristic process repeatedly merges two selected routes $r_1 = (0, ..., i, 0)$ and $r_2 = (0, j, ..., 0)$ into a single route $r' = (0, ..., i, j, ..., 0)$, if the new generated route is still feasible (i.e. load constraint satisfied). When two routes are merged, a distance saving $s_{i,j} = c_{i,0} + c_{0,j} - c_{i,j}$ is generated, as shown in Figure 2.2. At each iteration, the feasible merger with the largest savings is performed. The algorithm stops when there are no more routes that can be feasibly merged.



(a) Construct back and forth route for each customer from deport

(b) Identify "saving" for each pair of customers $s_{i,j} = c_{i,0} + c_{0,j} - c_{i,j}$; take shortcuts and merge them into a single route, as long as the resulting route satisfy constraints

Figure 2.2: Clarke-Wright algorithm

The Clarke-Wright heuristic is a greedy algorithm that builds good routes at the beginning and consequently gives less consideration to the routes generated later. To overcome this problem, Gaskell (1967) and Yellow (1970) modify the saving function to $s_{i,j} = c_{i,0} + c_{0,j} - \lambda c_{i,j}$. The larger the λ, the less the impact from the relative location of the depot. Further enhancement of this saving function has also been made by Paessens (1988) and by Altnel and Öncan (2005), adding terms such as the difference in distances from customers to the depot (i.e. $|c_{0,i} - c_{j,0}|$) and customer demand information to the function. Although the algorithm of Altnel and Öncan (2005) performs the best on benchmark problems among these algorithms according their experiments, when employed as an initial solution generator for meta-heuristics, there is no obvious advantage. In Chapter 6, a slightly modified Clarke-Wright algorithm is applied to generate parts of the initial solution for our GDMP.

Insertion heuristics Insertion heuristics (IHs) are also widely used today, especially for solving VRP with tight constraints such as time window constraints for each customer (e.g.

Campbell and Savelsbergh (2004); Brysy and Gendreau (2005); Joubert and Claasen (2006); Meidan et al. (2010); Hosny (2011)). Except when employed as an initial routes generator, IHs are also often used as solution repair or reconstruct operators in meta-heuristics (e.g. Adaptive Large Neighbourhood Search Ropke and Pisinger (2005)).

IHs build a feasible solution by repeatedly and greedily inserting an un-routed customer into a partially constructed solution. Different IHs use different criteria to make the two key decisions at each insertion-iteration: 1) which un-routed customer to insert? 2) where to insert the customer in the partial solution?

One of the most used IHs is the sequential insertion heuristic (Mole and Jameson (1976)). The sequential insertion heuristic expands one route at a time. At each insertion-iteration, the algorithm selects the customer with the least cost that is measured by the increased distance from the insertion of the customer to the route, and the distance between the customer and the depot.

It is rather flexible to modify the customer selection and insertion criteria according to problems. Therefore, IHs can easily be tailored to handle VRP with complex constraints. Some example applications can be found in the fleet size and mix vehicle routing problem (Liu and Shen (1999)) and VRP with back hauling (Salhi and Nagy (1999)).

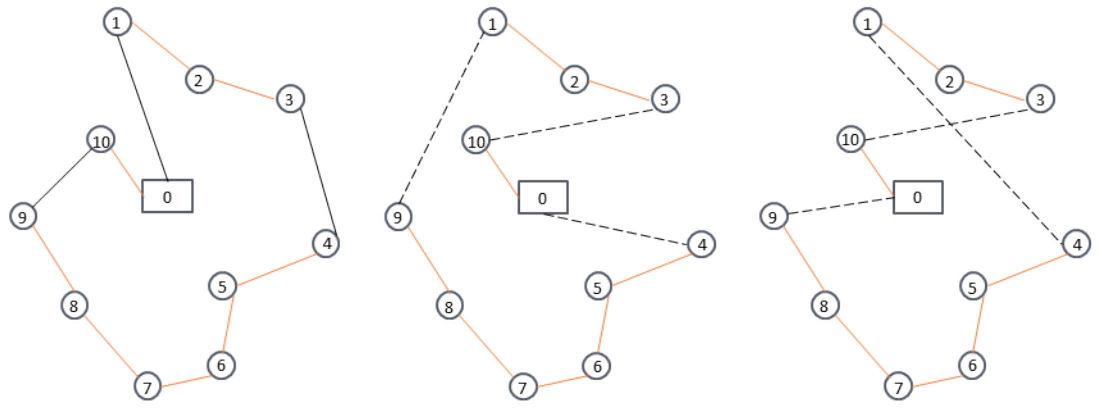
2.1.2.2 Improvement heuristics

Classical improvement heuristics adopt a local search (neighbourhood search) concept that perturbs a complete solution to improve its quality slightly at each iteration. A large number of moves to modify the structure of the solution have been proposed since 1960s. Until now, almost all successful VRP related problem solvers still use some of these basic moves, as they are efficient ways to manipulate routes structures. In this section, we review a few basic but commonly used moves for VRPs.

Classical improvement heuristics for VRPs perform moves on either a single route or multiple routes at a time, classified as intra-route and inter-route moves, respectively. In the first case, the λ-*opt* is one of the famous moves, proposed by Lin (1965). In each step, λ edges are replaced by other λ edges to achieve a shorter route. Figure 2.3 illustrates two possible changes after applying a λ-*opt* ($\lambda = 3$) move. The λ-*opt* is based on the concept of λ-optimality, larger values of λ are more likely to lead to an optimal final route (Helsgaun (2000)). Due to the fact that a larger value of λ results in a higher computational cost, the values $\lambda = 2$ and $\lambda = 3$ are mostly used in literature.

Motivated by finding a good λ with consideration of both computational time and result quality, Lin and Kernighan (1973) modify the value of λ dynamically throughout the search.

Or (1976) introduces an *Or-opt* move that attempts to improve a route by replacing a chain of consecutive nodes in a different position in the route.

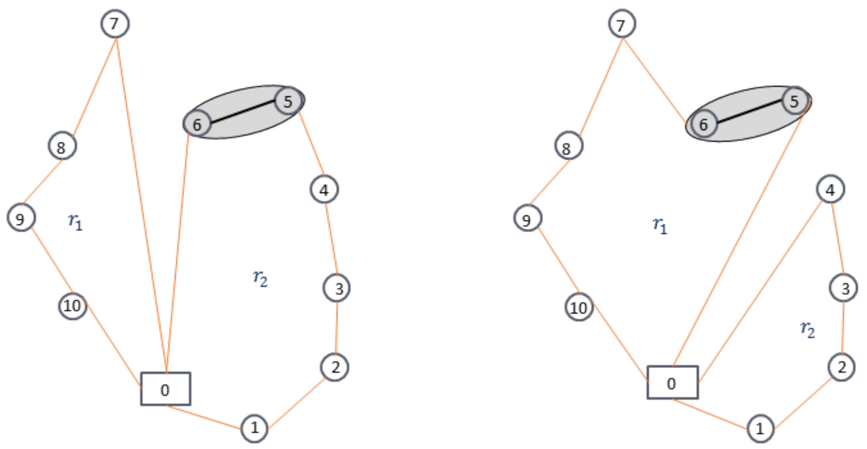


(a) Before (b) After(a) (c) After(b)

Figure 2.3: Two possible 3-opt moves

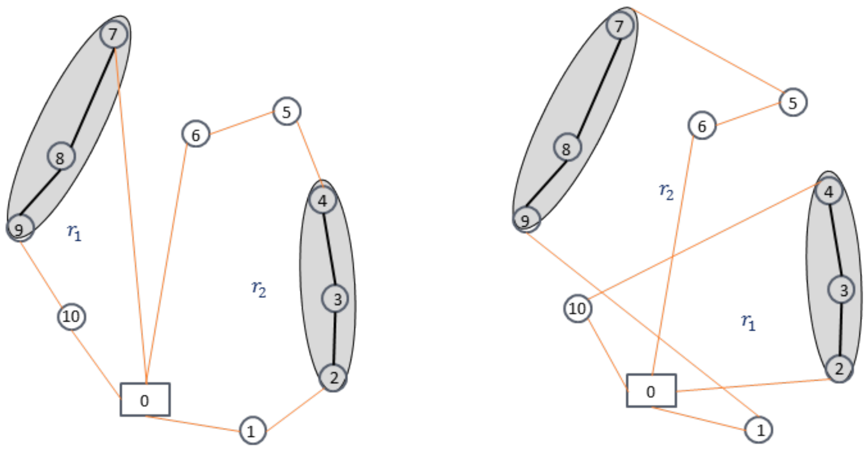
For VRPs with multiple constructed routes, the inclusion of inter-route moves is necessary to build more solutions structures. A wide range of moves has been developed (e.g. Dror and Levy (1986); Fahrion and Wrede (1990); Savelsbergh (1991)). Examples include moving a chain of customers from their current route to another one (*Relocate*, shown in Figure 2.4), exchanging positions of two chains of customers from two routes (*Cross-exchange*, shown in Figure 2.5), and removing two edges from different routes and reconnecting the parts differently into two new routes (*2-opt**, shown in Figure 2.6). To specify the length of chains modified in *Relocate* and *Cross-exchange* moves, we parametrise them by defining the longest chain changed in the move, denoted as *i-relocate* and *i-cross-exchange*. A comprehensive survey about moves for VRPs is provided by Groër et al. (2010).

A complete search of neighbour solutions that are generated by operating one specific move on the current solution is computational expensive, especially for large-scale routing problems. In recent years, applications of VRPs tend to capture more complex features of real-world problems, so customised moves are usually designed to satisfy some specific constraints. To tackle these problems, a set of moves is designed as a toolbox. Comparing to classical improvement heuristics, recent literature places more effort on designing sophisticated management methods to apply the set of moves in an intelligent way. In Chapter 3, a further discussion is given for several advanced neighbourhood search techniques and strategies of adopting appropriate moves at different search stages.



(a) Before (b) After

Figure 2.4: An example of *2-relocate* move. A chain contains customer (5,6) is relocated from route r_2 to r_1



(a) Before (b) After

Figure 2.5: An example of *3-cross-exchange* move.

2.1.2.3 Meta-heuristics

Meta-heuristic is a core research domain in solvers for combinatorial optimisation problem. A large number of meta-heuristic algorithms have been proposed for VRPs in the last twenty years. With respect to the classical improvement heuristics, meta-heuristics aim to search the solution space more widely and thoroughly. Several representative meta-heuristics will be introduced here. Each of them proposes a distinctive way of escaping from valleys of the search landscape and many of these methods have been successfully applied to a large variety of optimisation problems. Surveys of meta-heuristics for VRPs can be found in Cordeau et al. (2005) and in Vidal et al. (2013). Existing meta-heuristics can be generally classified as local search based methods and population based methods.

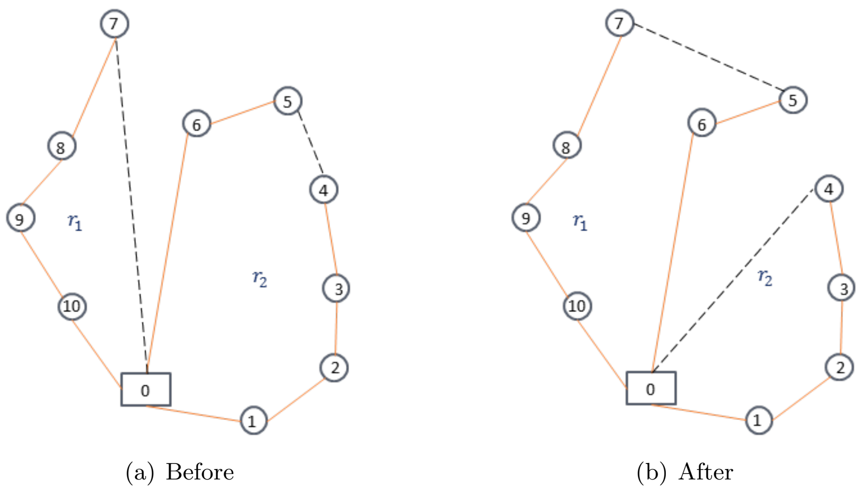


Figure 2.6: An example of *2-opt** move.

Local search based methods Local search based meta-heuristics have similarity to some classical improvement heuristics, as a way of iteratively exploring the neighbourhoods of a single incumbent solution to improve its quality. However, meta-heuristics introduce various mechanism to avoid the search becoming stuck in local optima.

A simple, but widely used, strategy is to apply a large random modification to the incumbent solution. Iterated local search (ILS) (Lourenço et al. (2010)) is a good example of using this technique. ILS applies successively an improvement phase, which ends up in a local optimum, and a perturbation phase to escape from the local optimum. Even though using the word "perturbation", the strength of the perturbation move should not be so weak that it always leads the search back to the same local optimum in the improvement phase. A similar strategy is to restart the search from a new randomly constructed solution.

Another idea is to temporarily accept worse solutions. Typical examples include simulated annealing (SA)(Kirkpatrick (1984)) and tabu search (TS) (Glover (1990a)).

At any time during SA, a worse solution is accepted with a probability governed by a statistical process. Intuitively, SA favours a more random exploration of the solution space by frequently accepting solution degradation at the beginning of a search process. Then, SA gradually decreases the probability of accepting worse solutions to focus on moving towards good solutions. Some early applications using SA to solve CVRP can be found in Robuste et al. (1990); Alfa et al. (1991); Osman (1993). In recent literature, the SA strategy and its variations is usually employed by more sophisticated meta-heuristics and hyperheuristics as a solution acceptance criteria (e.g. Hemmelmayr et al. (2009)).

Tabu search (TS) (e.g. Glover (1990b); Toth and Vigo (2003); Cordeau and Maischberger (2012)) introduces memory techniques to maintain information about the search trajectory. At each search step, TS replaces the incumbent solution with the best neighbour solution which has not been tabued, even if the neighbour solution is worse than the incumbent solution. To avoid cycling search, TS uses a short-term memory to reject solutions that contain recently examined tabu elements (tabued solutions). TS also allows exceptions to accept a tabued solution if it satisfies some aspiration criteria, such as "the best found solution". To enhance the robustness of TS, a number of long-term memory strategies are proposed, either to diversify the search by moving to a less explored area of the search space, or to intensify the search in a promising region. In the applications of TS for VRPs (e.g. Taillard (1993); Xu and Kelly (1996); Cordeau et al. (1997); Brandão and Mercer (1997); Alonso et al. (2007)), designing of tabu elements is critical for the success of TS. Therefore, problem domain knowledge and expert experience is very important here.

Evolving with the increasing complexity of combinatorial optimisation problems, a novel, simple but powerful search schema is proposed by Hansen and Mladenović (2001), called Variable neighbourhood search (VNS). VNS works with a set of neighbourhoods $(N_1, N_2, ..., N_k)$, which is often (but not necessarily) defined based on the use of move types (e.g. N_{2opt}, $N_{Relocate}$). Starting from an incumbent solution x, standard VNS applies a perturbation (or shaking) $(x' \leftarrow RandomPick(N_i(x)))$ and then an improvement local search $(x'' \leftarrow LS(N_j(x')))$, at each search iteration. Shaking is essential for VNS schema as this mechanism leads to the successive local searches starting at a slightly different point in the solution space. VNS iteratively examines solutions from these neighbourhoods in a systematic fashion. VNS is a parameter free algorithm, which makes it relatively easy to implement for various problems. Many successful VNS applications for VRPs can be found in (Fleszar et al. (2009); Hemmelmayr et al. (2009); Pirkwieser and Raidl (2010)).

Population based methods Whereas local search based methods generate a single search trace, population based methods manage a set of solutions all the time by generating several new solutions out of combinations of existing ones.

One of the most famous members of this class is the genetic algorithm (GA), which can be traced back to work from 1950s and was popularised by Holland (1962). GAs mimic the process of natural selection and improve solution quality by applying elitist selection, recombination (or crossover) and mutation techniques. Though GA has been successfully applied to many other problem domains, traditional GA shows slow convergence when applied to VRPs (Vidal et al. (2013)). To make GA work for VRPs, researches have added various enhancement mechanisms, such as a local search phase (so called "education") (e.g.

Moscato (1989)). By hybridising with other search techniques, enhanced GA can achieved impressive success in solving VRPs (e.g. Bell and McMullen (2004); Nagata et al. (2010); Vidal et al. (2012))

Ant Colony Optimization (ACO) is another nature inspired population-based method that has been successfully applied to many optimisation problems. When ants search for food, each one marks the travelled paths with an amount of "pheromone" depending on the quality of the food source. Applied to VRPs, each ant applies constructive heuristics with information collected from the search history (i.e. "pheromone"). Comparing to GAs, ACO labels good elements (e.g. edges of a good route) for constructing solutions, whereas GAs look for elite complete solutions. Some successful VRP applications can be found in Matos and Oliveira (2004); Yu and Yang (2011). As with many applications using GAs, ACO is often combined with local search methods to enhance the solution quality.

2.1.2.4 Further discussion

In recent years, there is a growing interest in combining various search techniques to deliver high quality solutions. In the section above, we have noted many examples of combining population based methods with local search. The recombination of large parts of good solutions (used in population based method) quickly finds high-quality starting points for a later local search procedure. And local search intensively guides the search into local optima. This complementary combination may explain the success of recent hypird-GAs (e.g. Nagata et al. (2010); Vidal et al. (2012)).

Another trend in hybridisation combines meta-heuristics with exact solvers (e.g. Mixed integer programming, constraint programming). Usually, exact solvers are used to solve a sub-problem during the search, such as reconstructing promising elements of solutions into complete solutions (e.g. Alvarenga et al. (2007)), or searching within a very large neighbourhood (e.g. Salari et al. (2010)). For solvers designed for more complex VRPs, exact methods are often used to solve a sub-problem of the original problem, such as the customer assignment problem for periodic VRP (e.g. Gulczynski et al. (2011); Crainic et al. (2012)).

Meta-heuristics have experienced a big growth and achieved many successes in a variety of VRP related problems. A large number of algorithms have been proposed. Each of them presents its own specifications, resulting in different behaviours of walking within the solution search space. Comprehensive discussion and detailed description of these algorithms can be found in Talbi (2009).

2.2 VRP with interesting features

In this section, we look at four well-studied variation of VRPs, each with additional interesting features. Among these variations, the periodic vehicle routing problem has strong similarities to our GDMP (Chapter 4). The other three VRPs also capture some important properties of our problem.

2.2.1 Periodic vehicle routing problems

The Periodic Vehicle Routing Problem (PVRP) (Christofides and Beasley (1984); Chao et al. (1995); Cordeau et al. (1997); Gulczynski et al. (2011)) is widely used as a mathematical model for real-world problems, such as inventory servicing, periodic maintenance, and on-site service planning. Case studies can be found in milk collection (Claassen and Hendriks (2007)), periodical grocery supply (Gaur and Fisher (2004)), waste collection (e.g. Teixeira et al. (2004); Shih and Chang (2001)), elevator maintenance (Blakeley et al. (2003)), remote healthcare services (An et al. (2012)) and many others. For more information, see the survey by Campbell and Wilson (2014).

A PVRP comprises K vehicles which can be used to service the demands of a set of customers over a period of days. Each customer i has a set of available visit patterns, denoted as Λ_i. For example, a customer might require two service visits per week, on either Monday and Thursday or Tuesday and Friday, giving two available patterns. Each PVRP has constraints that must be met: all vehicles start and end their journey at a single depot; no more than K routes are built on each day; capacity restrictions of vehicles and travelling duration are respected; a customer's request should be serviced in one time slot by one vehicle; only one service pattern $\lambda \in \Lambda_i$ is chosen for each customer i. The PVRP objective is to design a set of daily routes, comprising feasible patterns for each customer, that minimises the total travelling cost and satisfies the PVRP constraints.

To solve PVRP-related problems, two main types of approach are commonly considered. The first type of approaches (e.g. Alegre et al. (2007)) assigns customers to days according to their service pattern and then solves a vehicle routing problem (VRP) for each day. Use of this solving process transforms a PVRP to a multiple depot VRP (MDVRP) (Pisinger and Ropke (2007)). The second type of approaches (e.g. Tang et al. (2007)) is to simplify a PVRP to periodic travelling salesman problem (PTSP) by assigning customers to each vehicle/salesman. Routes are then built up and scheduled to days. This second approach is usually used when the service fleet is heterogeneous, or when strong ties exist between specific service personnel and customers.

Baldacci et al. (2011) propose a successful exact algorithm for solving the PVRP. The fundamental idea uses bounding methods to generate a reduced problem from the original problem, which ensures the the optimal solution of the reduced problem is also optimal for the original problem. Then the reduced problem is solved by means of an integer linear programming solver. To our knowledge, this paper presents the largest PVRP solved by an exact algorithm, of 153 customers, and using about 14,500 CPU seconds [2].

Meta-heuristics, which are capable of solving large-scale real-world problems, are the most common PVRP solvers in literature. Different meta-heuristics apply different search strategies, resulting in variance in solution quality. Tested on benchmark instances with customer numbers from 50 to about 400, meta-heuristic solvers are generally able to obtain solutions within 2% of the best known solutions, using hundreds of seconds. Chao et al. (1995) present a two-stage record-to-record algorithm that constructs solutions using several local moves applied one after another. Cordeau et al. (1997) were the first to use a tabu search heuristic for PVRP. During the search, infeasible solutions are allowed and controlled using an adaptive penalty function. Alegre et al. (2007) apply a scatter search framework (Laguna and Marti (2012)) to solve PVRP. The algorithm solves a problem of assigning calendars to customers in a periodic vehicle loading problem (Delgado et al. (2005)). Hemmelmayr et al. (2009) apply a VNS embedded with customer reassignment and inter-route moves as “shaking” operators and intra-route moves for local search process. Pirkwieser and Raidl (2010) add a coarsening and refinement process to VNS, called multilevel VNS for PVRP.

More recently, hybrid meta-heuristics present very competitive results in terms of both solution quality and computational time. Gulczynski et al. (2011) describe an integer programming-based heuristic (IPH): in this approach, the reassignment and daily routing processes are repeatedly applied until little or no improvement is found in the current iteration, when a restart initial solution is generated. Gulczynski et al. (2011) report that IPH out-performs the algorithms proposed by Chao et al. (1995); Cordeau et al. (1997); Alegre et al. (2007); Hemmelmayr et al. (2009). Vidal et al. (2012) propose a hybrid genetic algorithm that combines local search and sophisticated population management strategies to guide the search, an approach shown to perform better than all the above algorithms. Cordeau and Maischberger (2012) combine tabu search and iterated local search to give a competitive, broad exploration of the search space. Crainic et al. (2012) propose a

[2]The experiments were performed on a Fujitsu TX200S3 server equipped with an Intel Xeon E5310 processor at 1.6 GHz and 8 Gb of RAM

modular heuristic algorithm (MHA) that introduces a reference set to guide exploration and exploitation during the search for solutions minimising the number of vehicles used. In addition, the authors also present a self-learning mechanism that leads the search to assign better customer visit patterns as the solution evolves.

2.2.2 Vehicle routing problems with profits

Another interesting additional feature to VRP is profit, which is the most significant concern in many real-world applications. Some application examples include design of tourist trips (Vansteenwegen and Oudheusden (2007)), collection of oil (Goncalves et al. (2005)), and operations of a steel rolling mill (Balas (2007)). Recent reviews of VRP with profit can be found in Vansteenwegen et al. (2011); Archetti et al. (2014).

Based on the number of vehicles used, the standard model of VRP with profits can be classified as an orienteering problem (OP) (Golden and Wasil (1987)) and a team OP (TOP) (Chao et al. (1996b)). In the OP, there is a set of N locations and each with an associated profit p_i. The travelling time between any two locations $i, j \in N$ is denoted as $t_{i,j}$. There is one vehicle available, with a maximum route duration T_{max}. The vehicle should start and end its journey at pre-defined locations. The goal is to find a route that visits some of the locations, in order to maximise the total collected profit while satisfying the maximum duration constraint. The OP has several other names in literature such as the selective travelling salesman problem (Laporte and Martello (1990)), and the maximum collection problem (Kataoka and Morito (1988)). TOP extends OP by allowing at most K vehicles to collect profit.

Compared to other VRP variations, there are two decisions made simultaneously in VRP with profits: which locations to visit and in what order to visit them. The locations each have an associated profit value, making it more or less attractive.

In early 1990s, several researchers proposed exact methods such as branch-and-bound and branch-and-cut to solve the OP (e.g. Laporte and Martello (1990); Gendreau et al. (1998a)). Along with the developing trend of heuristic and meta-heuristics, methods including constructive heuristics, multiple stage improvement heuristics (e.g. Chao et al. (1996a)), tabu search (e.g. Gendreau et al. (1998b)) and ant colony optimisation approach (ACO) (e.g. Liang et al. (2002)) are gradually being applied to the OP.

In recent years, the literature has shown more interest in solving TOP and its variants such as additional time window constraints of visiting each location (e.g. Vansteenwegen et al. (2009b)). Meta-heuristics are the dominant solvers. A few example can be found in tabu search (e.g. Archetti et al. (2007)), VNS (e.g. Archetti et al. (2007)), and guided local search (e.g. Vansteenwegen et al. (2009a)).

2.2.3 Heterogeneous vehicle routing problems

The standard orienteering problem above only considers vehicle duration constraints. In many applications, the vehicle load constraint is equally important. Depending on the various types of vehicles available in a fleet (not limited by the duration and load information), a problem can be classified as a heterogeneous or a homogeneous VRP. Whereas many standard VRPs study homogeneous problems, real-world problems often consider a heterogeneous VRP.

In the literature of heterogeneous VRP, two major classes of models are defined: heterogeneous fleet VRP (HFVRP) and fleet size and mix VRP (FSMVRP). HFVRP considers a problem when the fleet composition is given (e.g. Tarantilis et al. (2004)), whilst FSMVRP assumes infinite vehicles of each type (e.g. Choi and Tcha (2007)). HFVRP aims to optimise the operational cost given existing vehicle resources. In comparison, FSMVRP emphasises a strategic investment decision on an optimal fleet composition to deliver efficient service.

Generally, heterogeneous VRP manages a heterogeneous fleet composed by M types of vehicles. For each type $m \in M$, there is k_m number of vehicles available (in FSMVRP, $k_m = +\infty, \forall m \in M$), each having a capacity Q_m. A set of N customers are given and each has a demand of q_i from the depot. In addition, the travelling cost may depend on the type of vehicle, denoted as $c^m_{i,j}$. In some variations of the heterogeneous VRP, an additional fixed cost F_m is associated with each type of vehicles, which models rental or maintenance costs. The goal is to find a set of routes that services all customers with the minimum operational cost, subject to the capacity constraint and vehicle number constraint.

A more closely related variation to our GDMP (Chapter 4) is the site dependent VRP (SDVRP) (Chao and Liou (2005)). In SDVRP, a limited heterogeneous fleet is given and there is no fixed cost considered $F_m = 0, \forall m \in M$. SDVRP uses a vehicle-independent travelling cost, where $c^{m_1}_{i,j} = c^{m_2}_{i,j}, \forall m_1, m_2 \in M$. However, each customer is restricted to the vehicle types. For example, a customer can be visited by m_1 and m_2 types of vehicle (represented as $A_i = \{m_1, m_2\}$) and another customer can be visited by m_1 and m_3, etc.. Here, we can see a strong similarity between the SDVRP and the PVRP, in which an assignment problem and a routing problem are involved. Cordeau and Laporte (2001) show that the SDVRP can be converted into a special case of PVRP and they use the same solver designed for PVRP in a previous paper (Cordeau et al. (1997)) to tackle SDVRP.

Baldacci et al. (2008) provide a comprehensive survey of solutions to heterogeneous VRP. Many of the proposed successful algorithms are adapted from solvers developed for other VRP variants, such as record-to-record heuristic (Li et al. (2007)), tabu search (e.g. Brandão (2011)) and hybrid ILS with VND (e.g. Penna et al. (2013)). Li et al. (2007) adapt a record-to-record heuristic (RTR) for the VRP to handle the HFVRP. RTR can be considered as a deterministic variant of simulated annealing. Two clearly defined search stages, uphill and downhill (improvement) stages, are introduced by RTR. In the uphill stage, a neighbour solution is accepted as long as it is not worse than the current (or best) solution by $x\%$. Brandão (2011) introduce a tabu search framework embedded with a number of customised design for HFVRP, including design of moves, strategy of managing the moves, attributes in tabu list and aspiration criteria, etc.. Penna et al. (2013) propose an iterated local search (ILS) meta-heuristic which uses a variable neighbourhood descent (VND) procedure, with a random neighbourhood ordering, in the local search stage.

2.2.4 Dynamic and stochastic vehicle routing problems

Along with the fast development of information technology, there is an increasing interest in dynamic scheduling and routing problems. The most common scenarios of dynamism emerge from taxi business (e.g. Caramia et al. (2002)), city logistics (e.g. Barceló et al. (2007)) and vehicle routing in supply chain management (e.g. Giaglis et al. (2004)). In recent years, applications in maintenance scheduling (e.g. Tagmouti et al. (2011)), have also started to be aware of the dynamic features of real-world situations. Depending on the problem scenario, the concept of dynamism has been introduced from various perspectives, such as unknown time of demand arrivals, fluctuating travel time due to traffic, uncertain service time at each site and changing service priority. A recent survey can be found in Pillac et al. (2013).

By means of dynamic planning, we can change or update the plan during the execution of the plan. Figure 2.7 illustrates a simple example of a dynamic vehicle routing problem with only one vehicle to schedule.

To solve dynamic VRPs, an intuitive approach is to periodically solve a static problem according to the current state information. The replanning time can be whenever the available information changes (e.g. a task complete, new customer appeared), or at fixed time intervals.

Another concept often involved in real-world dynamic VRPs is stochastic input infor-

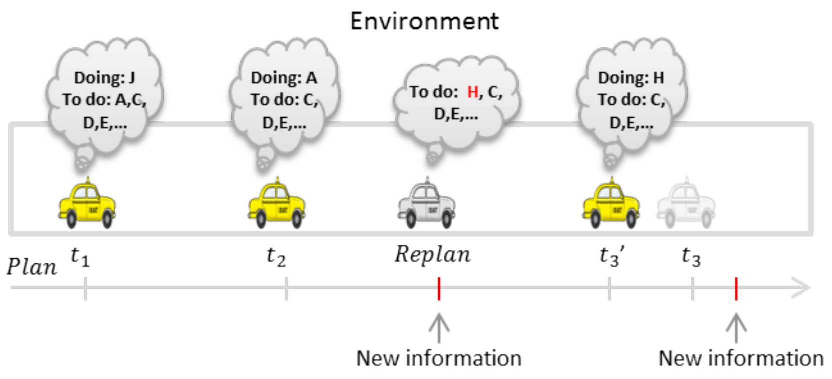


Figure 2.7: Dynamic planning for future tasks of a single vehicle.

mation, categorised as stochastic VRP (Gendreau et al. (1996)). In addition to the current known information, stochastic VRP introduces additional uncertain information to the problem. In real-world applications, the uncertain information could be obtained from estimation using historical domain knowledge. The motivation is to enhance the robustness of a solution in an uncertain changing environment. A large body of literature has reviewed dynamic and stochastic VRP. Here, we suggest a review paper (Larsen et al. (2008)) and a book (Zeimpekis et al. (2007)) for detailed discussion of problems in this area.

Looking at the dynamic idea applied in situations closer to our GDMP (Chapter 4), Angelelli et al. (2007) introduce a dynamic multi-period routing problem (DMPRP) where a two time-slots scenario is analysed. At the beginning of the first time-slot, a set of orders arrives that have to be serviced either immediately or in the next time-slot. Thus, the decision should be made as to whether the requests should be postponed. The objective of the problem is still to minimize the total distance travelled during the entire planning horizon. The authors propose a $SMART(p)$ algorithm that decides whether to postpone the customers depending on a ratio of incremental distance after adding the customers. A theoretical proof is given that $p = 2$ is an optimal algorithm for Euclidean distance instances. As an extension of the previous work, Angelelli et al. (2009) further classifies orders into on-line and off-line requests and replanning during the journey (diversion) is allowed. Here, instead of simple rule based decision making, a heuristic based method, VNS, is used to produce good solutions. Wen et al. (2010) solve a real-world variation of DMPRP; whether to serve an order immediately is a vital decision in this case study. Their tested instances have 10-day to 15-day planning horizons and 80 emerging orders on average every day. A PVRP variation is solved every day in the horizon by applying a three-phase rolling horizon heuristic, which includes a customer selection stage, a route optimisation stage with respect to the overall objective function and a post-optimization stage for daily route distance minimisation.

2.3 Real-world vehicle routing problems

VRPs have attracted a lot of attention in the field of transport related operational research. These problems are usually concerned with real-world applications where multiple locations need to be visited. For example, a supermarket needs to plan the routes for delivering goods to each customer; garbage collocation service predicts the quantity of garbage from each site and plans the routes for collecting the garbage. Comparing to the CVRP, additional features are captured in real-world applications using various constraints and sophisticated objective functions, commonly known as rich VRP (Caceres-cruz et al. (2014)). In Table 2.1, we summarise six aspects that are usually considered when modelling a real-world vehicle routing problem.

Table 2.1: Feature considerations when model a real-world routing problem

Concern aspect	**Description**
Customer	Many features can be extracted from real-life, such as service type (e.g. pick up, deliver), service time, vehicle type requirement and cost/profit of visiting.
Vehicle/Technician	Homogeneous or heterogeneous fleet defined by features like capacity, service duration, type. In this context, a type of vehicle/technician can only provide a certain type of service, such as fixing an air conditioner.
Depot	Depot defines the starting and ending location of planned routes. Multiple depots, and related features such as depot capacity can be introduced. In open VRP, vehicles are not required to return to the depot.
Planning horizon	The objective is to optimise the operation over multiple periods.
Dynamic	During the planning process, not all information related to the problem is revealed at the beginning. For example, the customers' requirements appear over time when the vehicles have already been sent to carry out tasks.
Objective	In a real-world scenario, the objective function can be very tricky to design. There may be multiple objectives to optimise. In different situations, these objectives may or may not have direct impact to each other. Common solutions include combining multiple objectives into a single function, or solving each objective in a different stage of an optimisation process.

In addition, multiple stakeholders are usually involved in real-world problem scenarios. To understand the relation between all of their concerned problems is important. An example scenario can be as follows. Department $D1$ has assigned budget $B1$ to survey road surface condition every year and department $D2$ has an assigned budget $B2$ to maintain the drainage system in the urban area every year. Is it possible and beneficial to combine these two problems? This topic is out of the scope of this thesis and more close to project management. However, this is the type of problem that researchers need to be aware of when considering solvers for real-world situations.

A number of survey papers attempt to classify the variants of VRPs (e.g Vidal et al. (2013); Lahyani et al. (2015)). Here, we are more interested in the developing trend of rich VRP and its applications in real-world scenario. We refer to the diagram proposed by Caceres-cruz et al. (2014) (Figure 2.8). At the bottom of the diagram, many classical VRP models and benchmark instances have been proposed. These models capture most of the essential features of VRPs, but the benchmark instances have small size. These classical models have clear mathematical definitions that allow theoretical analysis and comparison between various algorithms proposed by different authors.

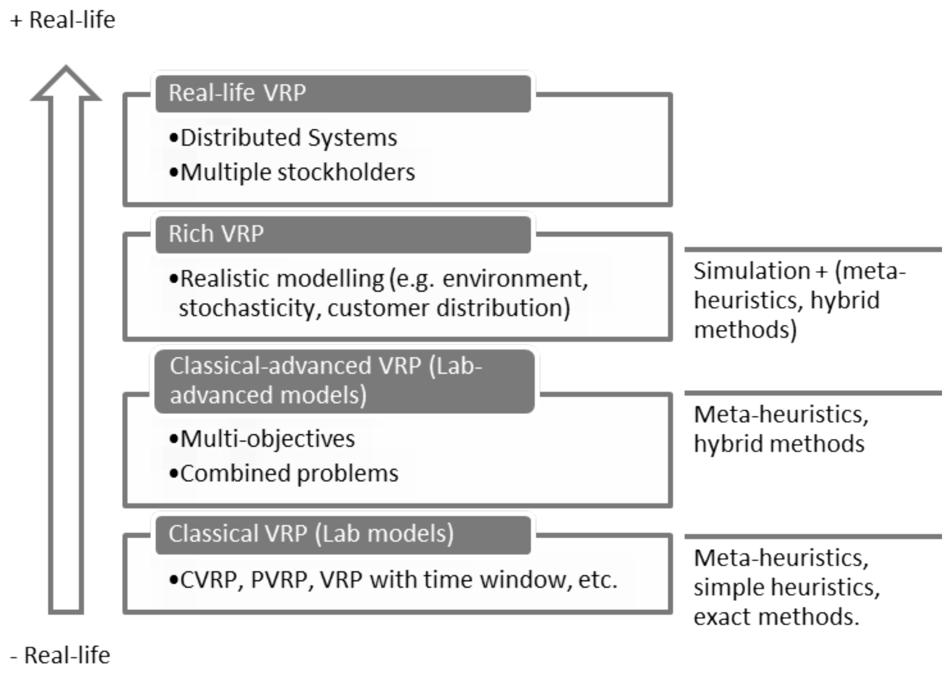

+ Real-life
Real-life VRP
•Distributed Systems
•Multiple stockholders
Rich VRP
•Realistic modelling (e.g. environment, stochasticity, customer distribution)
Simulation + (meta-heuristics, hybrid methods)
Classical-advanced VRP (Lab-advanced models)
•Multi-objectives
•Combined problems
Meta-heuristics, hybrid methods
Classical VRP (Lab models)
•CVRP, PVRP, VRP with time window, etc.
Meta-heuristics, simple heuristics, exact methods.
- Real-life

Figure 2.8: From laboratory VRPs to real-world applications (modified from Caceres-cruz et al. (2014)).

To cope with more sophisticated scenarios emerging from real-world requirements (e.g. Li et al. (2010)), we sometimes need to consider multiple related problems for the model, such as container packing, inventory management, and the like. Moving to the third level of VRPs (Figure 2.8), in order to understand the solution quality (or more correctly the solution impact to real-world situations), problem solvers and simulation based analysis are usually provided in literature (e.g. Faccio et al. (2011)). The solvers that are needed to deploy a VRP support system in real-world scenarios (top level, Figure 2.8), requires sophisticated software engineering as well as advanced search techniques. These solvers consider distributed computing, robustness of the system (e.g. more complex environment than simulation, large-scale demands), safety concerns (e.g. route-planning in self-driving vehicles), and so on.

In this thesis, we focus our research on the third level, using modelling and simulation based approach to evaluate the impact of our heuristic solvers and to compare with the manual scheduling strategies that are currently widely used in the corresponding field.

2.3.1 Common techniques for solving real-world problems

After analysing and modelling the given problems, we need to propose some realistic solutions that can deal with challenges such as large-scale, uncertainty, real-time, etc.. According to our experience of solving drainage system maintenance scheduling and a road inspection problem in this thesis, we summarise a few useful techniques for handling real-world problems:

1. Problem size reduction. Existing methods (including both exact approach and heuristics) are usually able to handle VRPs with less than 1,000 nodes in CPU minutes. However, most of the real-world problems contain thousands of geographically distributed nodes. Depending on the CPU budget, efficient solvers are normally preferred. Pre-processing to decrease the problem size is usually more effective than optimising algorithm. This is mainly because we are solving NP-hard problems (Toth and Vigo (2002)). Common problem size reduction techniques include node clustering (e.g. two-phase heuristics), node aggregating (e.g. Wen et al. (2011)), small rolling horizon (e.g. Chen et al. (2014)), and so on.

2. Incrementally solving a problem with updated information. In a dynamic environment, at any instant time t, we can solve a small problem that starts the search from historically built good solutions or good elements of solutions.

3. Good design of solution representation and neighbourhood structure. Heuristic based approaches for VRPs normally require the design of local moves. It is important that a sequence of moves can reach any feasible solution in the solution space. Also, techniques like neighbourhood pruning (e.g. Toth and Vigo (2003)), statistically examining promising neighbours (Chapter 8) usually significantly helps to enhance the search efficiency for large problems.

4. Estimation. In many real-world scenario, not all information (e.g. customer demands) is known when solving the problem. To better solve dynamic and uncertain real-world problems, solvers often integrate estimation techniques with heuristics for scheduling and routing (e.g. Nuortio et al. (2006)).

2.4 Summary

This chapter has reviewed problems that can be commonly modelled as a vehicle routing problem and its variations. We start with the very basic capacitated VRP to get an overview of the elements (e.g. nodes, arc, load information) included in these problems. Then we reviewed various interesting elements added to CVRP (e.g. multi-period, heterogeneous fleet), that capture different scenarios from real-world experiences. Exact approaches like branch-and-cut algorithm, heuristic based approaches like local search and genetic algorithm have been proposed to solve VRP-like problems.

With the evolving of solution approaches, researches attempt to model more and more sophisticated problem that arise from a diversified real-world situation. To cope with new challenges, hybrid algorithms that integrate assorted techniques originally developed independently, now become to be the trend of solution approach design.

Our research in this thesis faces a significantly larger problem (Chapter 4) compared to similar problems studied in literature. Traditional ways of solving such a problem may be very computational expensive, and/or do not converge within reasonable CPU time. However, reviewing individual techniques employed in various solvers, such as elite solution (or element) management, rolling planning horizon and many others, helps us to build efficient solvers for our needs.

Chapter 3

Techniques applied in heuristic search

In Chapter 2, we have presented methods to model and solve VRP-like problems. Various heuristic approaches have been mentioned. In this chapter, we focus on the algorithms and further discuss the advanced search techniques that are applied in modern heuristic solvers in more detail. We review the fundamental concept of local search methods and introduces the concept of hyperheuristics. Meanwhile, we clarify a few terminologies, which are widely used but many of them are ambiguous in the literature. This chapter builds the basis of the algorithms developed in this thesis.

3.1 Local Search (neighbourhood search)

Local search (LS) and the concept of neighbourhoods are the basis of most heuristic search techniques. LS walks in a candidate solution space, starting from its current position and moving to a neighbour that is acceptable according to a cost function and the pre-defined acceptance rules. An LS move takes the current solution as an input and modifies it by slightly changing part structure of the current solution. At all times, the LS heuristic has a current solution, and it iteratively applies moves until it reaches a predefined stopping criteria.

LS-based methods use local information to guide the move directions in the solution space. The guiding rules are often based on experience or intuition and can be either predefined or adapted in course of the search. Because of limited information, any LS-based method is not able to predict the exact landscape of the distant search scape.

In this chapter, we introduce local search formally. The presentation is a modification of Funke et al. (2005).

In a given combinatorial optimisation problem instance, we use X to represent the solution space that includes all feasible solutions of the problem instance. A mapping $f : X \longrightarrow Q$ is a cost function that measures the quality of a candidate solution $x \in X$. For example, X is the set of tours for a travelling salesman problem (TSP) and $f(x)$ is the travelling distance of a tour x. X is finite, but could also be a very large set.

We assume that the given problem instance is a minimisation problem, and then the aim is to find a solution x^* satisfying $f(x^*) \leq f(x)$, $\forall x \in X$ (Funke et al. (2005)).

We define a *move* as an operation on a solution x, which can transform it into a different solution x'. A *move* is the elementary concept of all local search-based methods. Normally, an LS uses a set of *moves* M, which modify different part(s) of a solution or the same part of a solution differently. A solution x' is defined as a neighbour of x, if x' can be transformed from x by operating one *move* $m \in M$ on x. The set of all solutions that can be reached from the current solution x using one *move* from the given set of *moves* define the *neighbourhood* of the current solution, which is denoted as $N(x)$. If, for all $x' \in N(x)$, $f(x') \geq f(x)$, then we call the solution x a *local optimum* (Funke et al. (2005)).

A general LS method is described in Algorithm 3.1.1. This method looks for at least one improving neighbour solution $x' \in N(x)$ at each iteration.

Algorithm 3.1.1 Generic Local Search based on Funke et al. (2005)

```
1: Initialisation: construct a feasible solution $x \in X$
2: **Repeat**
3: Search for an improving neighbour $x' \in N(x)$, where $f(x') < f(x)$
4: **if** the search found an improving neighbour $x' \in N(x)$ **then**
5:     $x = x'$
6: **end if**
7: **Until** no more improvements are found
```

We note that line 3 of Algorithm 3.1.1 does not determine the search strategy within a neighbourhood $N(x)$. A detailed discussion of this is presented in the next section. To accept an improvement (lines 4-5), the two most common strategies are either to perform first improvement (FI) or best improvement (BI) where the entire neighbourhood is explored to identify the best solution. Of course, you can also apply the $n-$best search that chooses the best *move* once you have found n improving neighbours. Intuitively, we can think that when not using BI, the order of examining potential *moves* becomes very important, especially when a solution has large neighbourhoods.

Algorithm 3.1.1 could also be called an *improvement heuristic* (Laporte et al. (2000)), a local search process that only applies *moves* that lead to an improved solution. This process introduces a simple but efficient search strategy.

3.1.1 Search within a neighbourhood

A given set of *moves* defines the neighbourhood structure of the current solution. Finding an improving neighbour quickly becomes the next step. We summarise the neighbourhood search strategies in the four following classes: Solution Structure Oriented Search (SSOS), Feature Sequential Search (FSS), Sequential Search (SS), and Neighbourhood Partition Search (NPS).

SSOS The natural approach for developing a search algorithm is to, in turn, examine parts of the solution. For example, vehicle routing problems (VRPs) normally code candidate solutions as sequences of nodes (referring to the customers). SSOS identifies k-changing elements (e.g. 2 edges of a route) of a *move* (e.g. *2-opt*) using k-nested loops following the current solution structure. When more than one *move* is designed, a *heuristic* is employed to search through the neighbourhood by examining every move for each node/edge (or identified k elements), either in a pre-defined order or a random order. Applications can be found in Gulczynski et al. (2011) and Vidal et al. (2012).

FSS Instead of guiding the search by examining the *moves* following the current solution structure, FSS identifies k-changing elements of a *move* according to the cost of the elements. FSS usually considers a candidate list sorted by features (e.g. length of the edges in a *2-opt move*). One example is an edge-modification *move*, which examines the longest edges. FSS aims to give priority to the areas of a neighbourhood, which are perceived as promising. However, the trade-off to potentially decrease the number of examined neighbours is the extra effort involved in feature sorting. Early applications of FSS include the travelling salesman problem (TSP) solving (Bentley (1992)). In recent literature, FSS is rarely seen, especially when many different *moves* need to be designed for a rich VRP. In Chapter 8, this search strategy is embedded in a meta-heuristic framework to solve PVRP and GDMP.

SS Sequential Search is based on a pre-condition that a *move* of a neighbourhood can be decomposed into so-called *partial moves*, which are cost-independent. A decomposition is cost-independent if the change in the objective function for the complete move is the sum of the gains of all the partial moves. Then, a complete move can be sequentially formed by determining good *partial moves* first. By doing so, it becomes possible to avoid checking many potential *moves*. This method was first used by Lin and Kernighan (1973) for solving the TSP. Later Funke et al. (2005) introduce SS for common CVRP neighbourhoods where experiments show a significant speeding in the search compared to SSOS.

NPS The idea of dividing a neighbourhood gradually emerges when more and more rich VRPs are built as models of real-world problems. Variable neighbourhood descent (VND) (Hansen et al. (2010)), specifically utilises this concept. In many implementations of VND, a set of neighbourhoods $N_1, N_2, ..., N_k$ is defined based on the *moves* operating on the current solution. For example, we can note N_{2opt} as a neighbourhood that is composed of all solutions modified from the current solution x using a *2-opt move*. To solve a VRP, we then define a set $\{N_{2opt}, N_{Relocate}, ...\}$, and in turn, VND searches through each neighbourhood at each iteration. It must be noted that the terminology *neighbourhoods* in VND and adaptive large neighbourhood search (ALNS) (Ropke and Pisinger (2005)) can be considered as sub-neighbourhoods as defined in LS (Section 3.1), as it divides the neighbourhood of solutions based on *moves*. By dividing a neighbourhood, VND indeed introduces a search strategy to determine which neighbour(s) to examine first within LS.

Thus, a big challenge while designing a good VND algorithm is to decide in what order the neighbourhoods should be searched. One way, which is guided by intuition, is to order them based on the level of complexity of searching a neighbourhood, such that one starts with the least complex neighbourhoods, and then gradually includes the more expensive *moves*. A typical example is as follows: given a set of parametrised k-relocated neighbourhoods, $k = \{1, 2, .., n\}$, a reasonable search order would follow the order of $k = 1, 2, ..., n$. When it is difficult to determine the complexity of searching in different neighbourhoods or when an algorithm introduces more diversity, a random order has never been seen to be a bad idea. In fact, ALNS specifically applies a roulette wheel selection of neighbourhoods to introduce randomness to the search.

A more intensive way of dividing neighbourhoods into sub-neighbourhoods is proposed by the guided fast local search (Voudouris et al. (2010)). The authors present examples from various problem domains using different measures to create and evaluate sub-neighbourhoods and effective solutions depend on the derivation of suitably small sub-neighbourhoods. For example, an N_{2opt} neighbourhood can be further divided based on the nodes that a *move* involves. Then a tabu mechanism is used to decide whether to search a sub-neighbourhood or not.

Reduce search space In addition to the various strategies for defining search order through a neighbourhood, other techniques such as neighbourhood restriction could also affect the efficiency of LS. *Neighbourhood restriction approaches* reduce the CPU time spent on each iteration of LS. Toth and Vigo (2003) derive granular neighbourhoods from a neighbourhood by discarding moves that have none of the promising elements, which would be likely to belong to high-quality solutions. The elements, in this case, are the arcs of a routing problem. An element is labelled as promising based on characteristics such as arc length, incidence of the arc to the depot, and whether the arc has been used in one of the best solutions encountered so far. The granular neighbourhoods approach is embedded in a tabu search (Section 2.1.2.3), and the algorithm is tested on VRP instances with up to around 500 customers. The experimental results show that the method is efficient in computational time.

In fact, the SS (above) also decreases the total number of candidate moves to be examined at each LS iteration, constructing a complete move by sequentially determining good partial moves. *Fast guided local search* (above) uses a neighbourhood restriction in which moves are only evaluated if they belong to an activated sub-neighbourhood.

3.2 From heuristics to hyperheuristics

So far, we have reviewed a large number of (*meta-*)*heuristics* proposed in literature and applied to vehicle routing problems. In recent years, a large amount of research has shown that some *meta-heuristics* perform better for some type of problems. In addition, for the same problem, different *heuristics* or *meta-heuristics* perform better for different instances. Furthermore, different (*meta-*)*heuristics* perform better at different stages of a search for the same instance. Consequently, many *hybrid meta-heuristics* are appearing, aiming to take advantage of the abilities of different (*meta-*)*heuristics*.

A similar concept has been introduced as *hyperheuristics*, and is now a big branch of heuristic research. In the last 15 years, hyperheuristics have experienced rapid growth. Successful applications can be found across a large variety of problem domains, such as timetabling (Bilgin et al. (2007); Burke et al. (2007b); Bai et al. (2012)), vehicle routing (Pisinger and Ropke (2007); Garrido and Riff (2010); Misir et al. (2011); Walker et al. (2012)), and space allocation (e.g. Bai and Kendall (2005)). Summaries of state of the art hyperheuristic techniques can be found in survey papers by Özcan et al. (2008); Chakhlevitch and Cowling (2008); Burke et al. (2010, 2013). In Chapter 6, we successfully applied a tabu-based *hyperheuristic* to solve our large-scale drainage system maintenance problem.

A *hyperheuristic* is often described as a heuristic that chooses heuristics (Cowling et al. (2001); Ross et al. (2003)). In other words, *hyperheuristics* define a series of rules to choose between several sub-ordinate heuristics or the so-called *Low Level Heuristic (LLH).* It schedules the order of *LLH* to solve a problem instance at hand. Also, it can be used to analyse what *LLH* fits best with which instance.

Some *hybrid meta-heuristics* can be considered as human designed *hyperheuristics.* There is certainly an overlap between these two terminologies. However, most of the implementations of *hybrid meta-heuristics* describe rather complicated integrations of different search strategies, which often specifically benefit a certain problem type. The difference with *hyperheuristics* is that the decision making for an *LLH* selection is totally based on problem-independent measures such as the change in the quality of a solution. Once implemented, *hyperheuristics* can be directly used in another problem domain with different problem specific *LLHs.*

Another noticeable feature of some *hyperheuristics* is the introduction of machine learning to an *LLH* selection,which truly frees design from the requirement for domain experts. Since the terminology *hyperheuristic* was first described in the context of solving combinatorial problems (Cowling et al. (2001)), many interpretations, implementations, and classifications have emerged in the available literature on this subject matter. As a conclusion, we would define *hyperheuristic* as any approach, which attempts to automate the design of heuristic algorithms for solving difficult computational problems, given some necessary tools (*LLHs*).

3.2.1 Low-level heuristics design

Typically, for complex combinatorial optimisation problems, solvers with multiple neighbourhood structures are designed. With the large quantity of hyperheuristic applications in various problem domains, the design of an LLH set is varied as well. LLHs directly affect the search strategy through neighbourhoods. We summarise three common designs of LLH, which brings the first insight to development of our hyperheuristics and local search algorithms in the following chapters.

The first group of algorithms use FI, BI, or even LS heuristics as LLHs. The LLHs need to further define the search sequence within a selected neighbourhood (e.g., random, lexicographic, etc.).

The second group of algorithms typically use a random move from a selected neighbourhood structure. The best-studied example typically uses adaptive operator selection

(AOS) (Fialho et al. (2010b)) as part of an evolutionary algorithm. In recent years, AOS has also been applied in hyperheuristic approaches (Soria-Alcaraz et al. (2014); Sabar et al. (2014)). Comparing these algorithms with the first group, the uncontrolled use of LLHs (random moves rather than improvement heuristics) may result in unproductive revisiting of the same move and eventually lead to an inefficient search. In these situations, the intelligent LLH-management mechanism in hyperheuristics is rather critical to the success of the entire algorithm.

The third approach is employed in the well-known adaptive large neighbourhood search (Ropke and Pisinger (2005)). Here, the LLHs are not specified in advance and are instead created programmatically from a known set of destroy and construct methods. The system learns which combinations work effectively and focuses the search on these.

3.3 Summary

In this chapter, we have reviewed the fundamental concepts of *local search* and the common terminologies in literature. Interesting questions arise from both search strategy within a neighbourhood (low-level) and management strategy (high-level) over a set of LLH. Part II of this thesis will specifically investigates these two aspects of heuristic-based approaches. Chapter 7 analyses the search performance and behaviours of a number of representative *hyperheuristics*. Chapter 8 introduces a learning mechanism into the local search process.

Part I

Large Scale Geographically Distributed Asset Maintenance Scheduling Problem

This research looks at large scale Geographically Distributed asset Maintenance Problems (GDMP). In collaboration with our industrial partner Gaist Solutions Ltd., we focus on a gully-pot system maintenance problem and use it as a case study to build a general GDMP model. This model is able to capture two main sub-issues that exist in this type of problem, namely the routing problem and the despatching problem with consideration of asset condition deterioration. Power grid maintenance is another good example of real-world applications in this area.

Chapter 4

A gully-pot system maintenance scheduling problem

In this chapter, we focus on the modelling of a real-world gully-pot system maintenance problem. While the content is included in our papers (Chen et al. (2016b,f,e)), a more complete presentation of the general geographically distributed asset maintenance model is given here, and further discussion is delivered.

4.1 Background information

Gully-pots compose an important part of the storm drain system that prevents solids and sediment from flushing into sewers, where they cause blockages in the underground surface water collection infrastructure (Butler et al. (1995)). Regular cleaning is required for gully-pots to function effectively (Karlsson and Viklander (2008); Scott (2012)): typical strategies are to clean all gully-pots once or twice a year. If gully-pots are not cleaned regularly, partial or complete blockages and accelerated deterioration of the gully-pots increases the likelihood of surface water flooding. In extreme situations such as intensive rainfall, a clogged drainage system may cause serious property loss (i.e. BBC (2011, 2012); Shieldsgazette (2012); Leylandguardian (2015); Yorkpress (2016)).

In the UK, gully-pot maintenance is undertaken by local councils, each using its own strategy. Our research focuses on gully-pot system maintenance problem from Blackpool, UK, as a case study, with data from the local council and from consultants, Gaist Solutions Ltd. Blackpool's gully-pot maintenance system records 28,149 gullies in an area of about 36.1 km^2. Analysis of real-world gully-pot maintenance records shows that human and environmental factors play a critical role: leaf-fall causes many gully-pot blockages; strong winds can blow sand or dirt into gully-pots causing partial blockages; reporting of gully-pot issues by local residents varies across the seasons; and parked vehicles affect the cleaning plan.

Blackpool local council has two gully cleaning vehicles but only one cleaning team. On any day, the team either takes out the normal cleaning machine, which uses hydrodynamic pressure and a vacuum to loosen and remove solids and liquids from a gully-pot (Karlsson and Viklander (2008)), or uses a specialist machine, equipped for fixing broken gully-pots. Currently, each day there is a schedule of gully-pots to visit, starting and ending at the depot. Either maintenance vehicle departs the depot at 09:00 and returns no later than 17:00. During servicing, some gully-pots are inaccessible, usually due to parked vehicles. If the team encounters a broken gully-pot during normal cleaning, it is recorded and subsequently added to the schedule of the specialised vehicle. Scheduling also needs to take account of residents' reports of problematic gully-pots: depending on the local risk, these emerging events should be scheduled 5 to 20 days from when they are recorded.

Currently, the maintenance schedule is produced manually by experienced managers. The control level is down to wards of the city and each day's maintenance route is planned by experienced drivers. Our research aims to improve this situation and produce self-adaptive scheduling and routing supported by data analysis.

4.1.1 Preventative and corrective maintenance approaches

Maintenance is a series of actions that aims to retain an object in, or restore it to, the state where it performs its required function (Besnard et al. (2010)). As shown in Figure 4.1, maintenance is generally categorised into corrective and preventative maintenance (Duffuaa et al. (2001); Besnard et al. (2010); Ahmad and Kamaruddin (2012)).

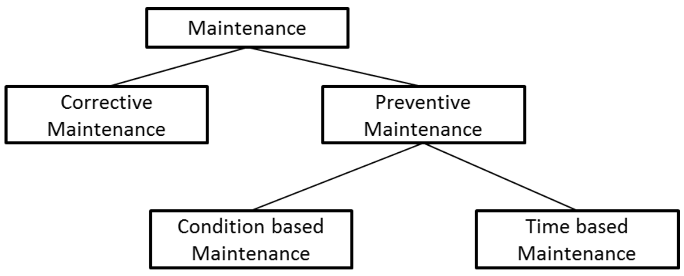


Figure 4.1: Classification of maintenance strategies (Besnard et al. (2010))

Corrective maintenance (CM) usually happens after failures occur. It includes actions such as repair and replacement. Tsang (1995) notes that the consequence of doing only corrective maintenance is a high risk of machine downtime and property loss. In our gully-pot system maintenance case study, we define failure as broken or blocked gullies and residents calling events. These failures cause a dynamic scheduling and routing situation, where unscheduled maintenance actions should be carried out. This also leads to high maintenance costs.

Preventative maintenance (PM) is an alternative strategy that aims to reduce these risks. In industry, preventative maintenance typically takes place at regular time intervals, based on experience. Operational research on PM introduces decision making, based on data analysis, with techniques such as time-based (TBM) (e.g. Scarf and Cavalcante (2010); Wu et al. (2010)) and condition-based maintenance (CBM) (e.g. Carnero Moya (2004); Campos (2009)). TBM can be applied when the failure rate is predictable, whilst CBM is employed where conditions are continuously monitored by sensors or any appropriate indicators. There is little research combining PM and CM strategies: Kenne and Nkeungoue (2008) introduce a PM/CM rate control strategy, obtaining a near-optimal maintenance policy for a manufacturing system.

In comparison to other maintenance literature, the gully-pot system maintenance problem involves geographically distributed points and a strictly-limited service resource. Instead of finding an optimal maintenance policy for each individual object, the focus of this research is to produce an optimal maintenance schedule covering all objects within time and resource constraints.

4.1.2 Case studies of geographical distributed maintenance problem

In this section, we review various models used in real-world maintenance and on-site service problem scenarios. The focus here is the applications of the models; the definition of these models and their common solution methods are discussed in Section 2.2.

4.1.2.1 Periodic vehicle routing problem (PVRP)

Many geographically distributed maintenance and on-site service problems are modelled as rich vehicle routing problems. The most widely used model is the PVRP in which a planning period of several days is considered and each customer (or asset) in the problem must be visited at specified days.

Blakeley et al. (2003) use a multiple-objective PVRP to model a real-world elevator and escalator maintenance problem, which includes periodically checking customers' equipment and reacting to call-outs. Travelling time, workload balancing, visiting time window violation and overworking time are considered in a weighted linear objective function. Jang et al. (2006) solve a problem of routing lottery sales representatives to visit lottery retail locations using a similar model. Again, a weighted objective function composed of travelling distance and routes balance is applied. The route duration and visiting time window limitations are considered as soft constraints. During the optimisation process, while minimising the objective function, the algorithm also simultaneously decreases the constraints violations. Alegre et al. (2007) analyse a real-world periodic pick-up of raw materials problem and model it as PVRP. The notable characteristic of this research is the very long planning horizon (90 days) compared to other literature. Related work has also been analysed in remote healthcare services. An et al. (2012) consider the home healthcare problem, which needs to provide periodical services to various patients. Maya et al. (2012) help an education institution to provide periodical services for disabled children. This problem is considered as a multiple depot PVRP as each teaching assistant starts and ends their journey from home. García et al. (2013) consider a perishable products supply problem for a bakery company. Weekly delivery routes from the depot to distributors are generated. This problem introduces a certain flexibility in the delivery date. The authors introduce a bi-objective model that minimises the total travelled distance and the total stock over the planning horizon simultaneously.

4.1.2.2 Profit based objectives

Another element introduced to many maintenance or on-site service scheduling models is profit. In these models, the visiting frequency of each customer/site is not pre-specified. Instead, each customer/site is associated with a profit value and the number of visits within the planning period becomes a decision variable. These models have similarity to PVRP, in that a schedule is produced for a given period. A well studied standard model with the profit maximising objective is the team orienteering problem (TOP) (Section 2.2.2), which requires the determination of a set of routes maximising the total reward of nodes visited with a time duration limit. In many real-world problem modelling, the elements of TOP and PVRP are combined. Baptista et al. (2002) model a paper recycling problem as a PVRP with profit, which maximise the total profit from selling the collected paper given limited operational resource (e.g. vehicle, working hours). Their objective function is composed of a linear cost function including income and outcome aspects of the operation for a given period. Goncalves et al. (2005) consider a similar model for an oil collection problem and they try to maximise the amount of oil collected by all vehicles while some specific constraints for the problem are satisfied (i.e. a well site can only be revisited after its recovery day interval). Tang et al. (2007) model a geographically distributed equipment maintenance scheduling problem as a Multiple Tour Maximum Collection Problem with Time Dependent rewards (MTMCPTD). The rewards are decided based on manufacturer maintenance interval suggestions. The objective is not to minimise the travelling cost but to maximise the reward from completing tasks (i.e fixing or checking a machine). In this case, not all equipments are visited within the planning horizon. A similar model has also been applied for a inspector scheduling issue for one of the largest retailers in the world (Zhang et al. (2013)).

4.1.2.3 Heterogeneous fleet

Some on-site service scheduling problems also involve human resource and vehicle resource management. Cappanera et al. (2011) propose a skill VRP that originates from the despatching of technicians to customers in after-sales service management. The service team has a set of technicians and each with a skill level represented by an integer value. The requirement from a customer i should be serviced by any technician having a skill level at least s_i. The skill VRP can be considered as a special case of site-dependent VRP (Section 2.2.3). The difference exists in the ordering relation among the technicians, whereas there is no hierarchy among the vehicles in site-dependent VRP. Amorim et al. (2014) study a rich VRP arising from a Portuguese food distribution company. A fleet is composed of refrigerated and normal trucks of various sizes. For each customer, there is a set of vehicles that is able to serve this customer. To capture all the features that exist in this real-world problem, a heterogeneous fleet VRP with site-dependent and time window constraints is introduced.

4.1.2.4 Dynamic problems

In most of the case studies considered above, all of the customers or assets that need to be serviced are known in advance. In contrast, the service requests in dynamic scenarios emerge continuously over time. Wen et al. (2010) consider a large distributor operating in Sweden and model it as a dynamic multi-period routing problem. Customer orders and their feasible service periods are dynamically revealed over time. The objective function is a linear combination of total travel costs, customer waiting, and the balance of daily workload over the planning horizon. The dynamic aspect is also studied in a winter road gritting problem by Tagmouti et al. (2011). The problem is dynamic as the best service time for each segment is affected by a moving storm and a decision should be remade as the environment changes. Every time that the update of weather forecasting information is received, an updated solution is generated by solving a static problem with the newest state information.

4.2 A risk driven model

Having reviewed many related case studies of real-world maintenance and on-site service problems, this section proposes a risk driven model that is based on the problem characteristic and requirement analysis.

4.2.1 Problem analysis

For our gully-pot system maintenance problem and many other asset maintenance problems, the high level aim is usually to ensure our system performs its required function. This form of service differs from many on-site service delivery problems (Section 4.1.2.1) in terms of its unclear or very variable visiting pattern. How to decide when to visit which asset becomes a key question, unlike PVRP where this is known in advance. Why is it necessary to visit some of the assets first? What is the measurement of maintenance schedule quality? It seems that a high level dispatching decision should be made. The traditional vehicle routing models with a fixed value of each vertex (e.g. TOP) or without any vertex evaluation (e.g. VRP) are not able to give a satisfactory answer. Meanwhile, routing is a crucial aspect when we face a large set of geographical distributed asset points. Routing aims to optimise resource utilisation.

In terms of heterogeneous fleet management, the gully maintenance problem considered in Blackpool has a fixed bound between a gully-pot state and the required service, so there is no decision to be made on which vehicle to use.

Another interesting property of our problem is that it is dynamic in the sense that unexpected failures are revealed incrementally over time. In this scenario, rescheduling may be required. At any decision point, planning must determine which assets should be visited soon and in which sequence the vehicles should visit them.

In this section, we propose a risk driven model that combines the risk driven asset management concept with dynamic periodic routing model.

4.2.2 General model of the risk driven asset maintenance scheduling

Our risk driven model captures the common features of many GDMPs. A geographically distributed system is considered as a directed complete graph $G = (V, A)$, where V is the set containing an assets set $N = \{1, 2, ..., n\}$ and one depot (denoted as $\{0\}$). A is the set of arcs (i, j), where $i, j \in V$. These assets need maintenance in a continuous time or over a very long period of days D (e.g. $D = \{1, 2, ..., h\}$, where $h = 1825$ days). Each asset i is associated with a risk impact r_i, which measures the value of this asset to its stakeholders. There is a failure probability of each asset that changes over time, and can be obtained by a function $P_i(d)$, which measures the probability that asset i is in a failure state on day $d \in D$. We have a maintenance team with a heterogeneous fleet. We use K to denote the set of vehicles. Each vehicle $k \in K$ is able to deliver a subset (denoted as L_k) of all types of maintenance actions in the set L. Accordingly, we classify vehicles into groups and vehicles in each group can deliver the same types of services. We use K_m to represent the set of vehicles from the same group. $\bigcup_{m \in M} K_m = K$, where M is the set of all vehicle types. Other input parameters include the following:

- T_{max}: the maximum travelling time allowed for each route;
- c_{ij}: distance, in terms of travelling time, from asset i to j;
- K^m_{max}: the total number of type m vehicles.
- H_{max}: the total number routes allowed each day (usually due to human resource limitation).
- t_l: service time of completing type l maintenance actions.
- a_{il}: a variable equals 1 if the asset i requires type l maintenance action, and the value equals 0 otherwise.
- b_{kl}: a variable equals 1 if $l \in L_k$, and the value equals 0 otherwise.

Here, we present the formulation for the planning problem of the upcoming period of W from the current day ($W = \{1, ..., g\}, |W| \ll |D|$). A judicious subset of assets is scheduled in the next short maintenance period W (e.g. a week or 2 weeks). A binary variable that describes the scheduling decision is shown below:

$$x^d_{ik} = \begin{cases} 1, & \text{if asset } i \text{ is visited by vehicle } k \text{ on day } d \\ 0, & otherwise \end{cases} .$$

The objective is to minimise the total risk caused by asset failure in the period of W:

$$\sum_{d\in W}\sum_{i\in N} r_i P_i(d) \tag{4.1}$$

To calculate the objective function (4.1), we define our $P_i(d)$ as follows. $F_i(d, d_i)$ represents an approach to estimate the lifetime of asset i given its last service date information $d_i \in D$. For example, Section 4.3.2 describes a Weibull distribution used for gully-pot lifetime estimation in this study. Alternatives are also discussed.

$$P_i(d) = \begin{cases} 0, & \text{if } x_{ik}^d = 1,\, a_{il} = 1,\, b_{kl} = 1 \\ F_i(d, d_i), & \text{otherwise} \end{cases} .$$

Our secondary objective is to minimise the total distance travelled during the planning period:

$$\sum_{d\in W}\sum_{(i,j)\in A}\sum_{k\in K} c_{ij} y_{ijk}^d \tag{4.2}$$

$$y_{ijk}^d = \begin{cases} 1, & \text{if the scheduled route visits arc } (i,j) \text{ by vehicle } k \text{ on day } d \\ 0, & \text{otherwise} \end{cases} .$$

Subject to:

$$\sum_{i\in N}\sum_{l\in L} x_{ik}^d a_{il} b_{kl} t_l + \sum_{(i,j)\in A} c_{ij} y_{ijk}^d \le T_{max}; \qquad \forall d \in W, \forall k \in K \tag{4.3}$$

$$x_{jk}^d = \sum_{i\in V} y_{ijk}^d; \qquad \forall d \in W, \forall k \in K, \forall j \in N \tag{4.4}$$

$$\sum_{i\in V} y_{ijk}^d \le 1; \qquad \forall d \in W, \forall k \in K, \forall j \in N \tag{4.5}$$

$$\sum_{i\in V} y_{ijk}^d = \sum_{u\in V} y_{juk}^d; \qquad \forall d \in W, \forall k \in K, \forall j \in V \tag{4.6}$$

$$\sum_{k\in K}\sum_{j\in N} y_{0jk}^d \le H_{max}; \qquad \forall d \in W \tag{4.7}$$

$$\sum_{k\in K_m}\sum_{j\in N} y_{0jk}^d \le K_{max}^m; \qquad \forall d \in W, \forall l \tag{4.8}$$

Constraint (4.3) guarantees that the time spent on each daily route is less than the limit T_{max}. Constraints (4.4) and (4.5) explain the relation between two decision variables and ensure that an asset point is only visited once in one route. Constraint (4.6) guarantees flow conservation for each route. Constraint (4.7) makes sure that the number of vehicles used on each day does not exceed H_{max}. Constraint (4.8) limits the usage of vehicles for each type. Please note that we do not include vehicle capacity constraint in this model. For most of the maintenance situations, the vehicle capacity is sufficient for a normal daily work. In our gully system maintenance case, the waste disposal process after daily work is beyond our scheduling programme and a fully prepared vehicle is despatched every day.

4.3 An application to gully-pot system maintenance

To apply the risk model to our gully-pot system maintenance problem, this section introduces the detailed analysis and calculation for the two components of the objective Function 4.1, risk impact of each asset r_i and the estimated failure probability on future days $P_i(d)$.

4.3.1 Estimating the risk impact per gully-pot

A potential hazard (i.e. surface water flooding) could be exacerbated by both geographic factors (i.e. elevation, soil type) and social-related factors, which are usually influenced by economic, demographic and building types (Cutter et al. (2003)). A higher risk impact here implies that, if a particular gully-pot is blocked and flood happen, it results in relatively larger economic and social losses. In other words, we prefer to clean the gully-pots with larger impact more frequently to keep them working properly. Co-operating with Gaist Solutions Ltd. and Blackpool local council, we firstly decided a list of social concerns with awareness of their economic and population influence, as shown in Table 4.1. Then, each gully-pot is evaluated by its location and the related social concerns.

Based on the existing data from Blackpool council, social concerns are classified in to three groups: 1) residential property; 2) commercial and industrial areas including local and district centres, business zones, and employment sites; 3) public services including schools, hospitals, doctors and public transport routes. In Table 4.1, the estimated value of each item in group 1 is the average residential house price in Blackpool UK GOV (2015). Group 2 takes account of footfall and critical building prices for each item. The estimated value of items in group 3 is based on average daily operation costs.

Flooding impact analysis involves large uncertainties. Research has shown historic flooding from different perspectives (Changnon (1999); Thieken et al. (2008); Brouwer and Ek (2004); Merz et al. (2004)). We do not expect a precise assessment of impact. Instead,

Table 4.1: Average daily risk impact estimation of each gully-pot

Group	**Social Concerns**	**Estimated value**	**Value loss from flooding**	**Risk impact**
1	Residential	£113,000	3%	£34
2	Local center	£1,130,000	5%	£580
	District center	£1,695,000	5%	£870
	Business area	£565,000	5%	£290
	Employment sites	£226,000	5%	£116
3	School	£5,168	4%	£71
	Large hospital	£917,808	4%	£377
	Doctors	£9,178	4%	£73
	Bus route	£220	100%	£37

we aim to find values that are able to guide gully-pot maintenance actions in decision making. Here, we mainly focus on direct economic losses using a damage function which relates to property type and water level. Thieken et al. (2008) propose the impact from a range of flood water levels on different building types. After consulting Blackpool Council and Gaist Solutions Ltd., we decide to focus on the impact of floodwater levels of less than 21 cm. This gives value-loss figures (Table 4.1, third column) of 5%, 3% and 4% for commercial, residential and public service areas, respectively. For public transport we focus on bus routes, estimating the cost of closing a road section due to surface water flooding.

By analysing Blackpool historic flooding frequency (Blackpool (2009)), the probability of flooding events is used to map the flooding value loss to the daily risk impact per gully-pot according to its location (last column of Table 4.1). We assume that gullies in the same section of a street evenly share the responsibility for the risk impact evaluated in that area. Figure 4.2 illustrates the geographic distribution of gully-pot risk impact in Blackpool.

4.3.2 Estimating the process of a gully-pot blocking

Ahmad and Kamaruddin (2012) suggest that time-based maintenance is the normal strategy in situations where equipment has a fixed lifespan or predictable failure behaviour. After analysis of historic gully-pot records, we model the gully-pot blocking process using the Weibull distribution model (Weibull (1950); O'Connor (1997)), from reliability theory. We define:

$$F_i(d, d_i) = 1 - e^{-((d-d_i)/\lambda)^{\alpha}}$$

The parameters of this form of Weibull distribution are the shape parameter α, and the scale parameter λ. In our study, we define $\alpha = 6$, based on historical data analysis;

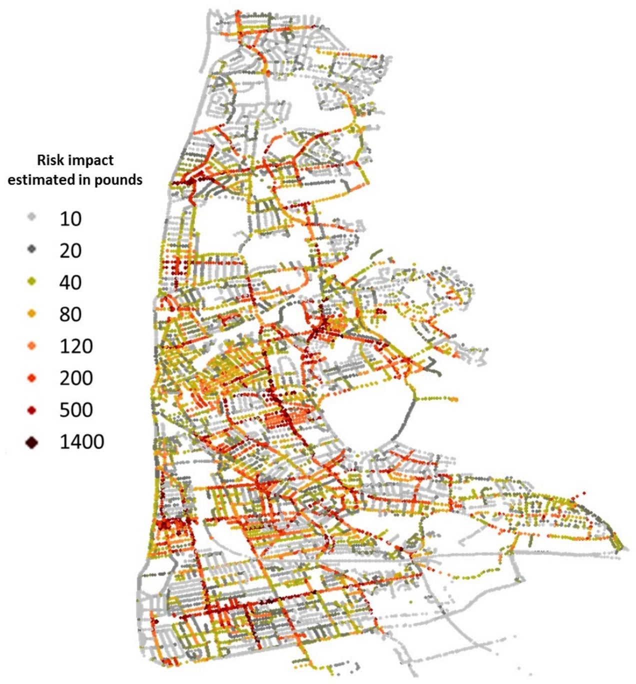
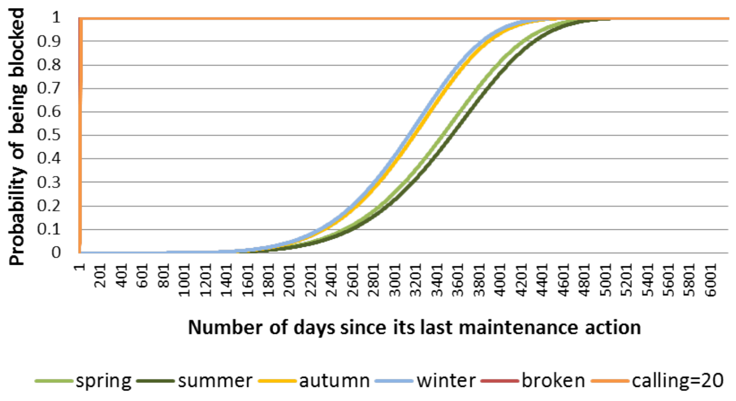


Figure 4.2: Gully pots' risk impact distribution in Blackpool

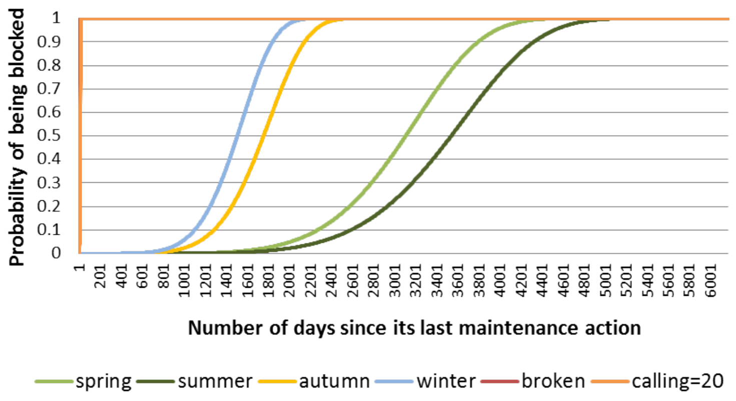
this captures a realistically increasing blocking rate over time. The scale parameter λ, capturing lifetime behaviour, is affected by location and seasonal factors, according to a simple linear function:

$$\lambda = \begin{cases} 10 & \text{... if gully-pot recorded as broken} \\ E_{calling} & \text{... a calling event} \\ max(90, E - \sum_{f \in F} n_f * s_f) & \text{... normal state} \end{cases}$$

$E_{calling}$ represents the expected number of days from a report on a gully-pot to its servicing. E is the expected number of days that it would take a normal gully-pot to become blocked since its last service. Here, $E = 10.3$ years, again based on historical data analysis. F is a set of factors that may affect gully-pot lifetime, such as street type, number of trees nearby, and blown sand effect: n_f represents the effect level from a specific factor $f \in F$ to a gully-pot; s_f adjusts the effect from factor f according to seasonal information. For example, if a gully-pot is on a street with five deciduous trees nearby, then historical analysis gives $n_f = 5$ with $s_f = 93, 1, 389, 433$ in spring, summer, autumn and winter respectively. If a gully-pot location is not affected by factor f, we simply assign $n_f = 0$. All values are based on our statistical analysis of the Blackpool data. Figure 4.3 illustrates two examples of gully-pot lifetime estimation taking account of the surrounding environment.



(a) Example of a gully-pot lifetime with 1 tree nearby at different seasons



(b) Example of a gully-pot lifetime with 5 tree nearby at different seasons

Figure 4.3: Probability of a gully-pot being blocked since last maintenance action

Alternatives to Weibull distribution For our gully system, we use a Weibull distribution with two parameters to estimate each gully-pot's life time, based on historical gully-pot failure behaviour analysis and consulting from our business partner, Gaist Solutions Ltd.. Alternatively, there is a large amount of research focusing on the asset life cycle management, that predicts the asset breakage and degradation process of different types of assets. Literature shows some related research in bridge, pavement and water pipe maintenance systems (Madanat and Ibrahim (1995); Morcous et al. (2002); Baik et al. (2006)). We could plug in any realistic life time estimation method in to the risk driven model, in order to apply our optimisation algorithm (Chapter 6) to create a maintenance schedule targeting the appropriate assets.

Here, we summarise two groups of approaches that can be applied for other asset maintenance problems in the future. First, functional based models such as exponential (Shamir and Howard (1979)) and time-powered models (Kleiner and Rajani (2001)) have been used to determine the optimal timing of water pipe inspection and replacement. Time-dependent Poisson (Constantine et al. (1996)) and the accelerated Weibull hazard models (Le Gat and Eisenbeis (2000)) are also commonly used functions in industry. The second group of methods use Markov chain-based deterioration models. A number of real-world applications can be found in Madanat and Ibrahim (1995); Morcous et al. (2002); Baik et al. (2006). Different to the functional based models, Markov chain-based models focus on the transition probabilities between different grades, which also implies the asset life time are evaluated discretely.

4.3.3 Other parameter settings for gully-pot system maintenance

Having introduced the objective function's two components (i.e. risk impact and probability of failure state) for the gully system, we have the working duration constraint $T_{max} = 8$ hours each day; at any time, a gully-pot can be in one of the state $L = \{normal, calling, broken\}$; $H_{max} = 1$ allows one vehicle work each day; $K^m_{max} = 1$ for $m \in \{broken, normal\}$, where the type $m = broken$ vehicle can repair any gully-pot in its broken state, and the type $m = normal$ vehicle delivers servicing for gully-pots in a normal state and those registered as calling reports. Note that a gully-pot registered as a calling report can be either in a *calling* state or a *broken* state. However, the accurate information could only be revealed after a first visit. Normally, a $m = normal$ vehicle is able to fix the problem of a calling report; in the special case that the $m = normal$ vehicle finds the reported gull is in a *broken* state, a visit from $m = broken$ vehicle and a rescheduling is required.

4.4 Conclusion and discussion

In this chapter, we propose a risk driven model to capture the major characteristics of a general GDMP. Many other models have been discussed in Section 4.1.2. We would like to summarise the differences between our risk driven model and two closely related standard models, PVRP and TOP. The aim is to clarify the suitable real world situations for each approach and to summarise the required inputs for each model. As shown in Table 4.2, the main difference between our risk driven model and PVRP is whether visiting a customer/asset is a decision variable or a constraint. The TOP model also decides whether to visit a certain customer. Comparing the risk driven model with the TOP, our risk driven model introduces a dynamic value (risk, measured by $r_i P_i(d)$) of each asset, based on time

Table 4.2: Summary of difference between models

	Risk driven model	**PVRP**	**TOP**
Expected output	A solution that minimises the risk caused by estimated failures in the system for future periods.	A solution that minimises the operation cost within planning periods (e.g. travelling distance) while satisfying all visiting requirements.	A solution that maximises total collected profits from visited sites.
Required inputs	1. Available vehicles; 2. Risk value estimation as a function of time for each asset ($r_iP_i(d)$).	1. Available vehicles; 2. Clear visit pattern/frequency for each customer within the planning period.	1. Available vehicles; 2. Value/profit of each customer.
Maintenance Application Scenarios	1. When visiting or not is a decision variable rather than a constraint; 2. When the failure rate of each site is dynamic.	1. When each customer has clear visiting requirements.	1. When visiting or not is a decision variable rather than a constraint; 2. When the value/profit of each customer is fixed.

information d, instead of a fixed value or profit from each site.

In comparison to the multi-period profit maximisation models for real-world problems (Section 4.1.2.2), apart from being a risk minimisation problem, rather than a profit maximisation problem, our risk driven model runs in a continuously dynamic scenario, where failure events may happen at any time. Over a short period ($|W|$ days), our risk driven model has strong similarities to multi-period profit maximisation models. Both of the two models introduce a dynamic value prediction or measurement of each asset over time. For example, our risk driven model uses an estimated risk of each asset on any day and Baptista et al. (2002) use an estimated number of papers at each collection site on any day. These values are critical to make despatching decisions and designing these value measurements requires domain specific knowledge. On the other hand, the risk driven model is typically applied over a much longer period. This is broken down into shorter sections of $|W|$ days for computability. To deliver an optimal scheduling strategy in the long term, the parameter $|W|$ should be chosen wisely.

In the following chapters, we firstly use a simulation based approach to investigate the weakness of a manual maintenance scheduling strategy that is widely used across local councils in the UK (Chapter 5). Chapter 6 focuses on automating the maintenance scheduling supported by intelligent data analysis and estimation.

Chapter 5

Risk driven analysis of a manual strategy for gully-pot system maintenance scheduling

In Chapter 4, we introduce the gully-pot system maintenance problem and propose a risk driven model that captures the risk impact of gully-pot failure and failure behaviour. In this chapter, we use a simulation based approach to understand the weaknesses in the manual strategy. We propose to evaluate the quality of maintenance scheduling strategy using the average daily risk caused by gully-pot failures in the system and conduct what-if analysis on the manual strategy. The data required for simulation is provided by Blackpool council, a client of Gaist Solutions Ltd.. The content in this chapter has been approved by the city's consultant from Gaist, and has also been published in Chen et al. (2016f,e)

5.1 Introduction

This chapter focuses on two factors that may affect the scheduling of maintenance actions: the issue of parked cars and up-to-date gully-pot status information. In the current gully-pot system maintenance situation, during the preventative maintenance, some gully-pots are inaccessible due to parked vehicles. Historical maintenance records show that this is a striking issue: about 8.3% of gully-pots are not serviced each year because of parked cars. Apart from the parking issue, we also notice another weakness of current maintenance scheduling strategy, namely untimely system status information. Currently, all the broken or blocked gully-pots are either reported by local residents or found through preventative maintenance. This passive situation potentially leads to uncontrolled surface water flooding.

In order to discover techniques or policies that could improve current gully-pot maintenance, this chapter considers the gully-pot maintenance as a risk-driven problem, as described in Chapter 4. The current widely used manual maintenance strategy, which includes both preventative and corrective actions, is evaluated by the risk caused by failures in the system, across various scenarios.

The remainder of this chapter is organized as follows. Section 5.2 states the daily risk evaluation function for the gully-pot system. Section 5.3 introduces our simulation process of the current manual scheduling strategy. Section 5.4 presents additional simulation assumptions and environment settings. Section 5.5 analyse the impact of parked cars and Section 5.6 illustrates the benefits of conditional based maintenance. A summary of investment suggestions based on our simulations are provided in Section 5.7.

5.2 Evaluation of maintenance schedule strategy quality

We use the risk measurement function to evaluate the quality of maintenance schedules over the simulated period. Each day, we calculate the risk of surface water flooding due to blocked/broken gully-pots. This is evaluated by Function 5.1:

$$\sum_{i \in N} r_i P_i(d) \tag{5.1}$$

Note that the Function 5.1 is the daily gully-pot system risk element of Function 4.1 (Chapter 4). The methods of deriving each part of the risk measurement for each gully-pot are introduced in Section 4.3

5.3 Simulation of current manual scheduling strategy

In order to discover techniques or policies that could improve the current gully-pot maintenance, we would like to simulate the actual scheduling strategy that is widely applied across local authorities. In the real world, maintenance schedules are generated at varying levels of granularity, from long term (yearly) to short term (weekly). To mimic the manual schedule impact, we directly consider the short term planning that continuously creates the daily based despatching schedule. We first generate a set of fixed preventative routes for all gully-pots in the system and adjust our schedule plan weekly with consideration of emerging failure events. The simulation process captures the key points of the current manual strategy in Blackpool and is approved by Gaist Solutions Ltd.. We summarise the procedure as follows.

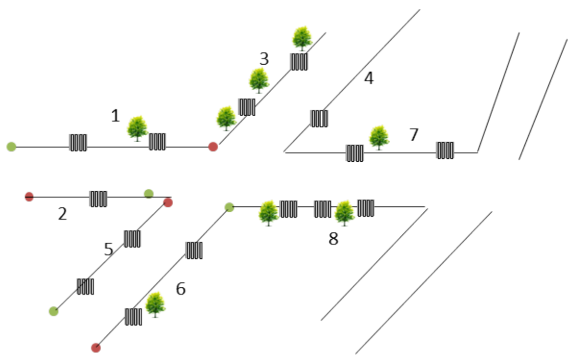
- **Step1**: Construct efficient preventative maintenance routes S_{fixed} for all gully-pots in the system (Section 5.3.2).
- **Step2**: Collect recent information on emerging broken/blocked gully-pots from either local residents' reports or daily preventative maintenance. Generate reactive routes for these problematic gully-pots (Section 5.3.3).
- **Step3**: Generate a maintenance schedule for the near future (Section 5.3.4).

This overview of the simulation process of the manual scheduling strategy, presents a number of technical issues that need to be solved. In the following subsections, we present the detailed techniques used to generate a large set of distance optimised routes.

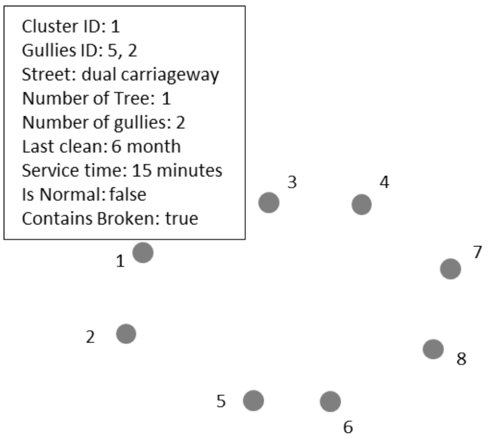
5.3.1 Reduce the problem size

Gully-pot system maintenance is a large-scale problem. Simulating the scheduling process for such a system is a computational challenge. To be able to deliver the analysis within reasonable CPU time whilst retaining enough information to build feasible cleaning routes and track gully-pot condition, we group gully-pots located on the same section of street. As shown in Figure 5.1(a), we assume that these gully-pots share the same environmental factors (Section 4.3.2). Gully pots in the same group are always scheduled together for preventative maintenance. The service time of a group includes both cleaning time for the gully-pots and travelling time inside this section of a road. This representation also maintains traffic distance: for instance, the distance between group point 1 to group point 6, in Figure 5.1(b), is the road distance measured from the red node of road 1 to the green node of road 6, in Figure 5.1(a). Furthermore, individual gully-pot states (i.e. normal, calling, broken) are still recorded, because unexpected damage or blockage events may happen to any of them: this allows corrective actions to be accurately tracked. A gully-pot-cluster is labelled as in normal state only if all the gully-pots included are in normal state. The risk of a gully-pot-cluster is the sum of all included gully-pots' risk at any given time.

By applying this grouping strategy, we reduce the preventative maintenance problem size from 28,149 to 9,277 points. For corrective actions, routes are built on problematic gully-pots, which only comprise a small vehicle routing problem.



(a) Example of gully-pot on street



(b) Example of grouped information

Figure 5.1: Reduce the size of a gully-pot system maintenance problem

5.3.2 Generate preventative route set

To build up the fixed preventative route set, we consider solving a vehicle routing problem (VRP) (Section 2.1). The objective is to minimise the total travelling distance, with constraints including: 1) all gully-pots in the system should be visited at least once; 2) all routes should start and end at the depot; 3) no route travelling time should exceed the working hours constraint.

VRP solver The VRP solver starts from an initial vehicle routing solution, constructed using the Clarke-Wright (CW) Savings heuristic (Section 2.1.2.1). After an initial solution is constructed, the improvement phase uses variable neighbourhood search (Section 2.1.2.3) embedded with *i-relocate* and *i-cross-exchange* shaking operators (Section 2.1.2.2) and a local search phase. A similar process is used by Hemmelmayr et al. (2009) in their daily VRP solving stage. In total, 12 neighbourhoods are implemented. The order of neighbourhoods is *i-relocate* (i = 1; 2; 3; 4; 5; 6) and then *i-cross-exchange* (i = 1; 2; 3; 4; 5; 6).

In order to enhance the solution quality, a local search strategy is used after a solution is obtained through "shaking". The single route operator, *3-opt* (Section 2.1.2.2) is adopted in an iterative first improvement procedure. Only the two modified routes have to be re-optimized.

Maximise usage of working time After finding the optimised VRP solution, we still cannot guarantee that every route maximises the use of the daily time limitation. Therefore, for each route in the preventative route set S_{fixed}, we try to insert the closest points which are not already included using least cost insertion (Section 2.1.2.1), until no more points can be inserted without breaking the working time limitation.

Discussion In our simulation, all the preventative gully maintenance routes minimise the travelling distance and maximise the usage of daily working time. This assumption produces routes that are better than the routes used in reality. Delivering on-site service normally has considerable variance in terms of service time and total working time. We use the average service and travel time, assuming that the impact of variance will cancel out over the long period of analysis.

5.3.3 Generate reactive routes set

Before producing the schedule for the following week, we create the reactive routes set, based on emerging events information. During the last week, a normal callings and b broken reports are received. Calling reports that have not been addressed and the a new calling reports make up the set V_{calls}. In the same way, we also get a set V_{broken}. When a call is received, we register the cluster ID (Figure 5.1) so that the schedule can inspect gully-pots around the reportedly problematic ones; however, when broken pots are discovered through preventative maintenance or inspection, we register them individually.

The VRP solver described above (Section 5.3.2) is used for both V_{calls} and V_{broken} to create candidate route sets S_{calls} and S_{broken}, respectively.

At this point, we have a candidate routes set, S_{all}, (including preventative routes, reactive routes that contain reported gullies and reactive routes that contain broken gullies) optimised in distance:

$$S_{all} = S_{fixed} \cup S_{calls} \cup S_{broken}$$

5.3.4 Produce schedule manually

At the end of every simulated week, we select seven routes from the candidate route set and assign them into the following working days in the next week. Only one route is executed for each day in the planning horizon. Priority is given to the routes that contain broken and blocked gullies (corrective maintenance). When all the reported problematic gully-pots have been serviced, the crew comes back to the preventative maintenance in the following day if there are still some days left in this week. The preventative routes $s \in S_{fixed}$ are serviced in turn and are arranged in descending order of risk $\sum_{i \in s} r_i P_i(d)$ measured at the beginning of a year.

5.4 Simulation assumptions and environment settings

After introducing the process to mimic the manual scheduling strategy, we make a few additional assumptions in our simulation, as follows.

1. In the real world, the number of working days every week varies depending on local council requirements: in the following experiments, we assume seven working days per week; holidays are not considered.
2. Parking issues: inaccessibility during maintenance due to parking predominantly affects preventative maintenance. For reactive actions, including servicing for both resident reports and broken gully-pots, our simulation assumes that the team always has access.

Environment simulation settings In the simulated environment, we consider each gully-pot's blocking behaviour according to seasonal and location information. Also, we add some random broken events to add some unpredictable elements to the environment. Detailed descriptions are as follows.

1. Total number of gully-pots in the system: 28,149.
2. Broken events: Blackpool council estimates about 1.1% to 1.8% of gully-pots are broken every year. This is represented by each gully-pot becoming broken randomly with probability from $p_b = 0.00003$ to $p_b = 0.00005$ per day in our simulation.
3. Blocking probability: a gully-pot lifetime is estimated by a Weibull distribution (Section 4.3.2). Every day, each gully-pot has a probability of becoming blocked according to its failure rate function $h_i(d) = \frac{R_i(d-1)-R_i(d)}{R_i(d-1)}$, where $R_i(d) = 1 - F_i(d)$ is the reliability function.
4. Seasonal factors F: the Blackpool data only allows us to include trees and leaf-fall in our simulation. Seasonal factors related to the number of trees nearby highly affect the lifetime of gully-pots, and on average, each gully-pot is affected by 0.4 trees in Blackpool.
5. Resident calling behaviour: about 1700 calls are received every year by the Blackpool gully maintenance team, and most of the calls concern blocked or damaged gully-pots. Over 50% of all calls occur during the autumn, as shown in Figure 5.2. Our statistical

analysis determined that, to match the resident calling behaviour in our simulation, on any given day, the probability of receiving a call if a gully-pot is already broken or blocked is $p_{calls}(i) = \{0.0033, 0.005, 0.0056, 0.002\}$ for spring through winter, respectively. If a gully-pot is not broken, there is still a small chance that a call is received, related to its current condition. The simulation probability is $p_{calls}(i) = P_i(d) * \gamma$, where $\gamma = 10.62$ is the value to adjust the calling probability to match the real data and has been measured experimentally.

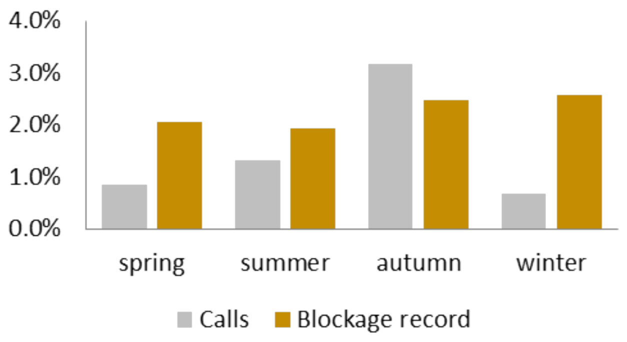


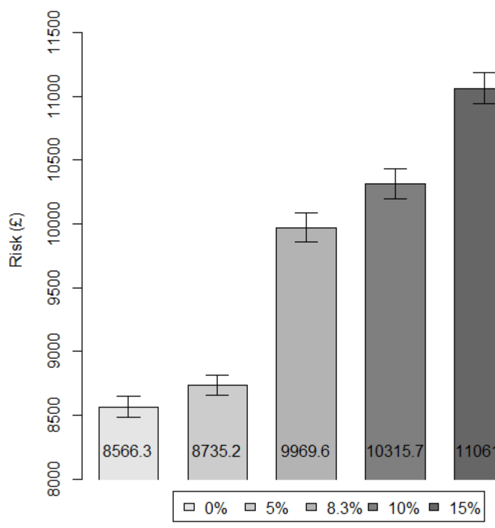
Figure 5.2: Seasonal calls and blockages as a percentage of the total number of gully-pots in Blackpool.

These parameters and assumption have been discussed with Gaist Solutions Ltd. and agreed to be a realistic representation of gully-pot behaviour in Blackpool. All simulations were implemented in C# and executed on a cluster composed of 8 Windows computers, each with 8 core Intel Xeon E3-1230 CPU and 16GB RAM. We evaluate the maintenance quality using average daily surface flooding risk caused by the gully-pot system over simulated four years.

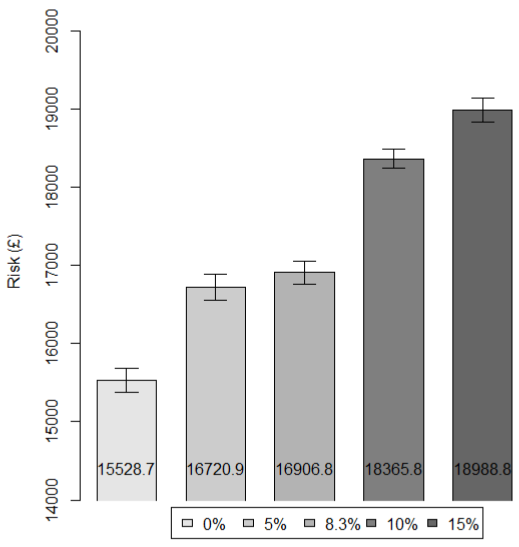
5.5 The impact of parked vehicles

According to the maintenance records, parked vehicles have been identified as a major problem that decreases the maintenance working efficiency, especially in the old town, where no extra space was designed for parked cars. Our simulation helps us to understand the impact of parking on gully-pot maintenance performance. Therefore, potential strategies can be proposed such as "banning parking" when a maintenance visit for a certain street is scheduled.

In simulation, we can test the effect of inaccessible gully-pots using a parameter, x, to represent the percentage of gully-pots that cannot be accessed during preventative maintenance each year. The values of x that we test are 0, 5, 8.3 (the actual value for Blackpool), 10 and 15 percent. Each parameter setting is run over 4 simulated years, with corresponding seasonal factors and residential report behaviours. We test two environments with the random broken events probability setting equal to either $p_b = 0.00003$ or $p_b = 0.00005$. We refer to these environments as stable and dynamic, respectively. The aim is to test the effect of parked vehicles in different degrees of uncertainty.



(a) Stable environment, $p_b = 0.00003$



(b) Dynamic environment, $p_b = 0.00005$

Figure 5.3: The average daily risk using the manual maintenance schedule, with different in accessibility settings during preventative maintenance. The bar with the setting of 8.3% is the current real-world situation. Error bars show 95% confidence intervals.

The results of simulation are shown in Figure 5.3. First, we can see that there is an increase in flooding risk as the percentage of inaccessible gully-pots increases in both scenarios. The simulation result suggests that a policy of banning parking on streets to be serviced might improve maintenance efficiency by about 14% and 8% in the stable and dynamic environments, respectively. In reality, a "banning parking" policy may only partially decrease the number of parked cars, if the percentage of inaccessible gullies is 5%, very different behaviours are shown in different environments. A nearly 12% risk reduction can be obtained in the stable environment, but little difference in risk can be observed in the dynamic environment. As the rate of car ownership continuous to increase, the impact of parked cars could become more of a problem, when the percentage of inaccessible gullies increases up to 15%, the surface flooding risk increases significantly by about 10% and 12% respectively.

This result tells us that parked cars pose a significant risk in both environments. Comparing the stable and dynamic environments, we can see the different stages of risk increase with different percentages of inaccessible gullies. The results suggest that different degrees of "banning parking" for different environments should be considered. For the situation in Blackpool, further investigation and more accurate estimation of the environment is needed to deliver the optimal policy while using the least effort.

5.6 What if we could do condition-based maintenance (CBM)?

Aside from parking issues, seasonal changes and untimely system status information are identified as other factors that affect the efficiency of drainage system maintenance. Seasonal change is an uncontrollable factor. On the other hand, improving low-cost sensor techniques makes it potentially feasible to continuously monitor gully-pot condition. This would allow our scheduling strategies to be combined with CBM, discussed in Section 4.1.1. Currently, we only find out that a gully-pot is blocked or broken either during preventative maintenance or if it is reported; because of this incomplete system information (i.e. the actual gully-pot status information at any time), it is difficult to produce any optimal schedules.

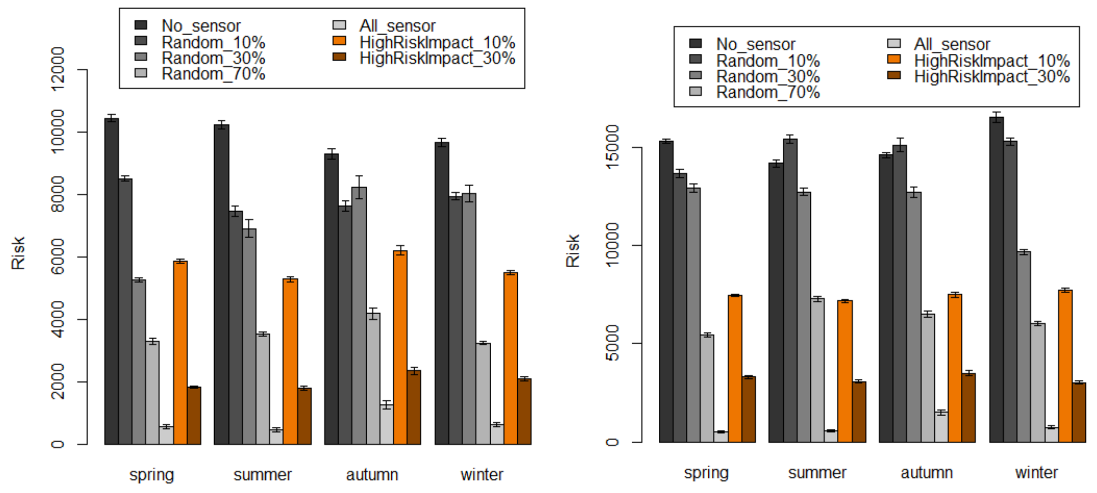
In simulation, we can test the importance of real time failure monitoring by varying the proportion of gully-pot failures that are known immediately, as if the gully-pot had a real-time sensor. In these experiments we consider more scenarios than the stable and dynamic environments introduced in Section 5.5. For each of the environments, we consider three starting conditions: a normal, well maintained system; a well maintained system after a natural disaster (recover-1); and a badly maintained system after a natural disaster (recover-2). Parameter settings for each of scenario are presented in Table 5.1.

As the simulation scenarios shown in Table 5.1, the normal state assumes that the entire system is well maintained and the number of days since the last maintenance action for each gully is uniformly distributed across 1.1 years and 1.5 years in the environment with two different uncertainty degrees respectively. In addition, we also test scenarios that assume the system is recovering from a natural disaster such that a large number of gullies are broken or blocked initially regardless of prior maintenance. Both a well maintained gully-pot system and a system that has had bad maintenance are tested (see Figure 5.5).

Figure 5.4 presents the average daily risk of systems with various coverage of sensors over a set of four-year simulations. As shown in Figure 5.4(a), in comparison to the simulation of current passive corrective maintenance (*No_sensor*), the instant information simulation (*All_sensor*) shows a reduction in risk of about 92%. A similar impact of instant information can also be observed in the dynamic environment, which brings a risk reduction of 94% (Figure 5.4(b)). In the current situation simulation (*No_sensor* in Figure 5.4), residents'

Table 5.1: "since last maintenance" and "initial broken gullies" set the system's initial state: for all gully-pots, the days since their last service are evenly distributed in θ years. We randomly assign a percentage of gully-pots to be in the broken state.

	Stable ($p_b = 0.00003$)		Dynamic ($p_b = 0.00005$)	
	Since last maintenance θ	**Initial broken gullies**	**Since last maintenance** θ	**Initial broken gullies**
normal	1.1	0.4%	1.5	0.7%
recovery-1	1.1	2%	1.5	2%
recovery-2	3	2%	3	2%

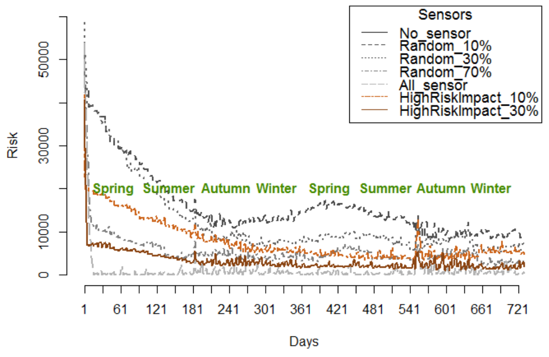


(a) Stable environment, $p_b = 0.00003$ (b) Dynamic environment, $p_b = 0.00005$

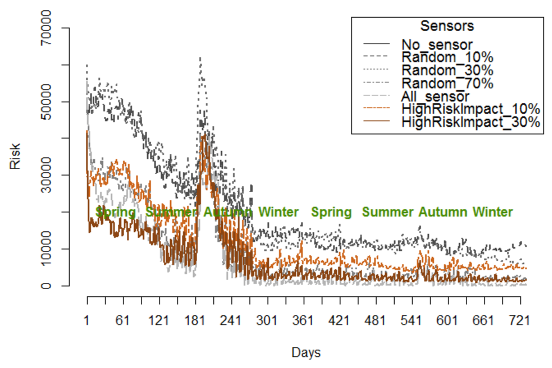
Figure 5.4: Performance of the manual scheudling strategy in normal scenario in Table 5.1, with sensors of different sensor installation capacity. Error bars show 95% confidence intervals.

reporting behaviour in different seasons strongly affects the response time for attending failed gullies. If a gully-pot breaks or blocks in winter, this may be only found through preventative maintenance in the next year. Such late responses to problematic gully-pots gradually accumulates risk over time. This may explain the skewed risk distribution across different seasons. The result reveals that depending entirely on reporting by local residents effectively hides the dangers to the system. For the case where all gully-pots have instant (sensor) information (*All_sensor*), the results clearly show the impact of seasonal factors: falling leaves and wet weather in autumn increase risk by about two times compared to other seasons.

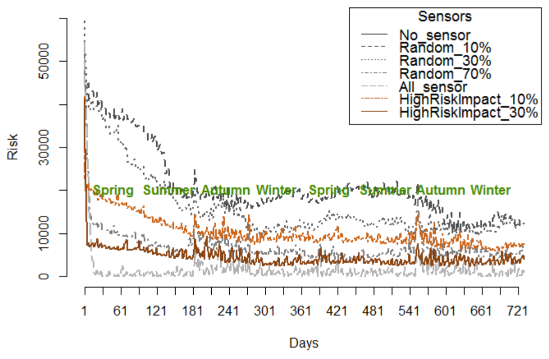
To provide further insight into how the availability of information on gully-pots affects flooding risk, we adapt the simulation to provide instant information from only some locations, simulating the localised installation of sensors. Setting 10% of gullies to have sensors allows us to compare an even distribution of sensors (*Random_10%*) to the results when sensors are focused on critical areas of the city (*HighRiskImpact_10%*). We find that focusing on high risk areas reduces the daily risk, on average, by about 28% in the stable environment (Figure 5.4(a)) and about 50% in the dynamic environment (Figure 5.4(b)). When monitoring is increased to cover 30% of the gullies, the comparable risk reductions are 75% and 72%, in the stable and dynamic environments respectively. This result shows that it is much more useful to track the status information from gullies located in critical areas.



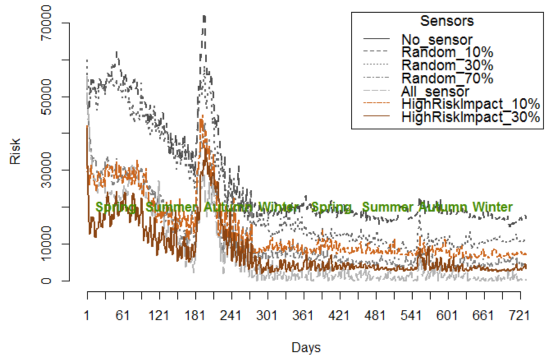
(a) Daily risk tracking of scenario recovery-1, $p_b = 0.00003$



(b) Daily risk tracking of scenario recovery-2, $p_b = 0.00003$



(c) Daily risk tracking of scenario recovery-1, $p_b = 0.00005$



(d) Daily risk tracking of scenario recovery-2, $p_b = 0.00005$

Figure 5.5: Performance of the manual scheudling strategy in recovery state with sensors of different install capacity.

Figure 5.5 illustrates the daily risk change over two years in the recovery states. In scenario recovery-1 (Figure 5.5(a) 5.5(c)), the system with full sensoring performs the best in terms of recovery speed, followed by the *HighRiskImpact_30%*. The faster recovery also implies lower total surface water flooding risk through the recovery period. In scenario recovery-2 (Figure 5.5(b) 5.5(d)), due to the previously poor system maintenance, the recovery period is significantly longer in all cases compared to recovery-1. Also, the peak point uncovers the vulnerability of a badly maintained system during the high-risk season (autumn). However, the sensoring still helps the maintenance team to produce a more informed schedule, which results in less total risk during the recovery period.

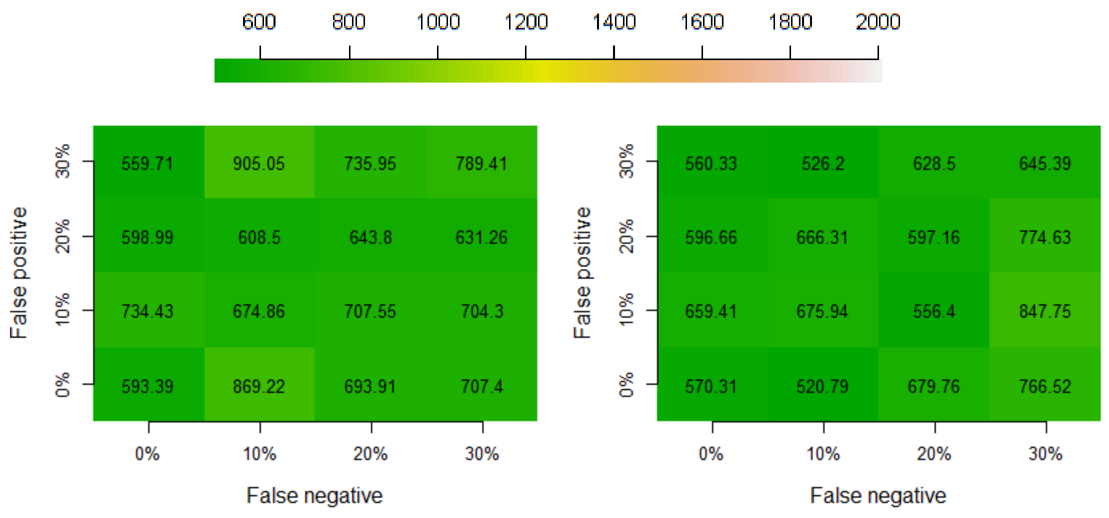
Comparing the recovery behaviour under stable and dynamic environments (Table 5.1), the system recovers slightly slower in the dynamic environment. Even though the absolute value of normal state risk and risk peak points are affected by the environment uncertainty, we can see very similar overall patterns in both environments settings. In the following experiments, we focus our analysis on the dynamic environment ($p_b = 0.00005$), which should give us a robust understanding of the impact of maintenance on the flooding risk of the gully-pot system.

5.6.1 Reliability

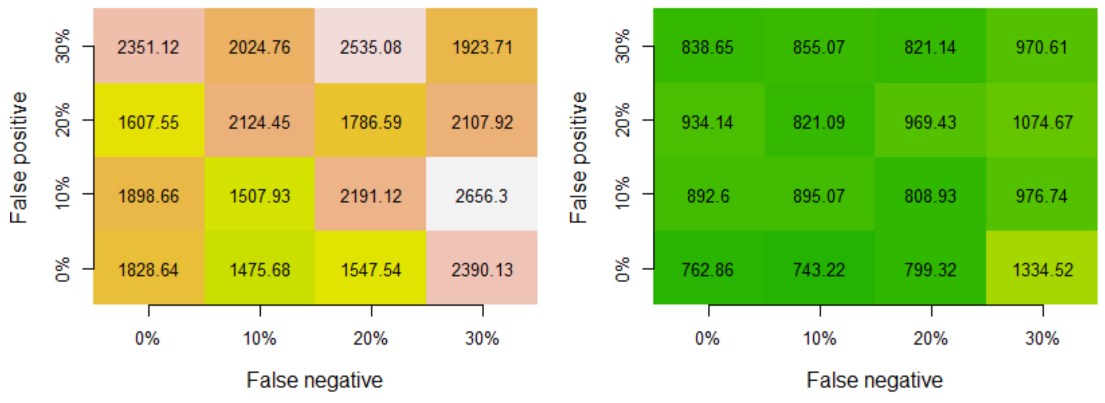
The above simulations show the contribution of timely information to improving the gully-pot system maintenance quality. However, installing and maintaining a sensor system also increases the management complexity, where extra cost and manpower are needed to ensure that the system is always working correctly. Furthermore, we assume in our simulation that instant gully-pot condition information can be received with no errors, which is hypothetical. In practice, current sensor techniques can achieve up to 85% reliability (See et al. (2012)). To justify the benefit from potential sensor technique in more realistic scenarios, we test various situations in which false negative and false positive error information is sent by sensors.

In the previous simulations, a blocked gully-pot is reported immediately if a sensor is installed. In the following experiments, we assume that a sensor may fail to report the gully-pot failures with a probability from 0% to 30%, labelled as false negative error in Figure 5.6. If a sensor fails, the gully-pot failure information relies on traditional reporting from local residents. Meanwhile, for any gully-pot in its normal state, an installed sensor may send a false alarm (also called false positive error) with probability from 0% to 30%. We run the maintenance simulation as if there is a full sensoring system, over four years, in the normal scenario (see Table 5.1).

The results of average daily surface water flooding risk are illustrated in Figure 5.6. By comparing Figure 5.4 and Figure 5.6, we can see that the overall maintenance quality in a full sensoring system, in terms of surface water flooding risk management, is still well controlled across all seasons, even with the probability of both false positive and false negative errors up to 30%. In the relatively stable seasons (spring and summer), the false reports show no strong effects on maintenance quality. In the autumn period, many reactive actions emerge due to the increasing failure rate of each gully-pot and the number of local residents' complaints. A large number of false alarms disrupts necessary



(b) Spring (c) Summer



(d) Autumn (e) Winter

Figure 5.6: Average daily surface water flooding risk over a four years simulation in the normal scenario. A full sensoring system with false alarm and false negative errors is considered.

maintenance in this period, resulting in a risk increase of about 28%. Interestingly, when the false negative error slightly increases in this very dynamic season, the maintenance quality slightly improves. A further investigation suggests that delayed reporting of problematic gullies in the dynamic season helps to construct more efficient maintenance routes to some extent. This result actually reveals that always giving priority to fixing problematic gullies is not an optimal strategy. More experiments related to optimising the scheduling strategy are discussed in Chapter 6. In the winter period, when the failure rate of each gully-pot is high but residents' reports are rare, large risk increases can be observed when the sensors' false negative rate increases up to 30%. This result reveals the importance of a reliable sensoring system, particularly if it is the dominant information resource.

Further discussion Whilst the use of sensors might be of benefit in maintenance scheduling and risk reduction, the realism of this approach needs further consideration. Accurate sensor information depends not just on the sensor detecting problems, but also on communication performance, which decreases in weather condition such as rain or snow (See et al. (2012)). The gully-pot system maintenance should combine a risk estimation approach (i.e. Section 5.2) with sensors to deliver optimised scheduling.

Our simulation shows large advantages when sensors are installed in high-risk areas. However, since sensors must be close enough to communicate wirelessly with each other, optimisation of the sensor network topology must be considered (See et al. (2012); Yick et al. (2008)).

5.6.2 Can we reduce maintenance frequency when providing CBM?

Historic gully-pot maintenance records from several local councils of the UK, show that the working frequency and pattern vary according to local policies. Our simulation experiments so far have assumed that the maintenance schedule is updated every week and the crew works 7 days a week. In order to further explore whether the installation of sensors is worthwhile, we compare the impact of reducing maintenance frequency on a no-sensoring system and full-sensoring system. To set up the simulations, the same scheduling policy is used (see Section 5.3.4), except that the maintenance crew only works for the first x days every week.

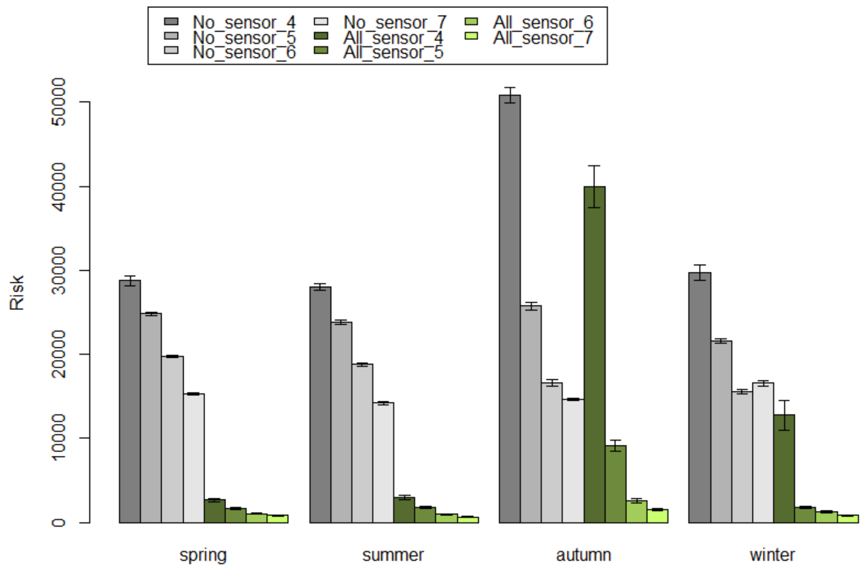


Figure 5.7: Performance of the manual scheduling strategy with different working frequency in the normal scenario: from four days per week to seven days per week. Error bars show 95% confidence intervals. Legend: sensorInfo_workingDays

Figure 5.7 shows the average daily risk over a four-year simulation with different working frequency settings. Firstly, we can see that the risk increases as the working frequency decreases in both no-sensoring and sensoring system. In the relatively low-risk seasons of spring and summer, the advantage from using a sensoring system is still apparent when we reduce the working frequency down to four days per week. However, the lack of maintenance leads to a series of problems in high-risk seasons (autumn and winter). Our results suggest that local authorities might make savings by using different working frequencies according to seasonal information. Comparing the result of five working days with sensoring to seven working days without sensoring, we can potentially improve the maintenance quality by about 76%, whilst reducing working time by 30%.

Figure 5.8 illustrates detailed daily risk changes over five-year simulations under various working frequencies. All simulations start from the same initial state as the normal scenario shown in Table 5.1, which assumes that the previous working frequency is seven days per week. Firstly, we can see that, when applying sensoring, the gully-pot system can be restored within about a month. In the first year, the daily surface water flooding risk caused by failures in the gully-pot system fluctuates at about £1,200 and £16,000, for the system with and without sensoring respectively. A dramatic risk increase can be observed in the second year, in the autumn period when we decrease the working frequency down to four days per week in both no-sensoring and sensoring system. This risk fluctuation pattern repeats in subsequent autumns. Once again, the result shows that insufficient maintenance will lead the system to a vulnerable situation, especially during high-risk seasons. Furthermore, this effect persists, even when perfect system status information is given by the sensors.

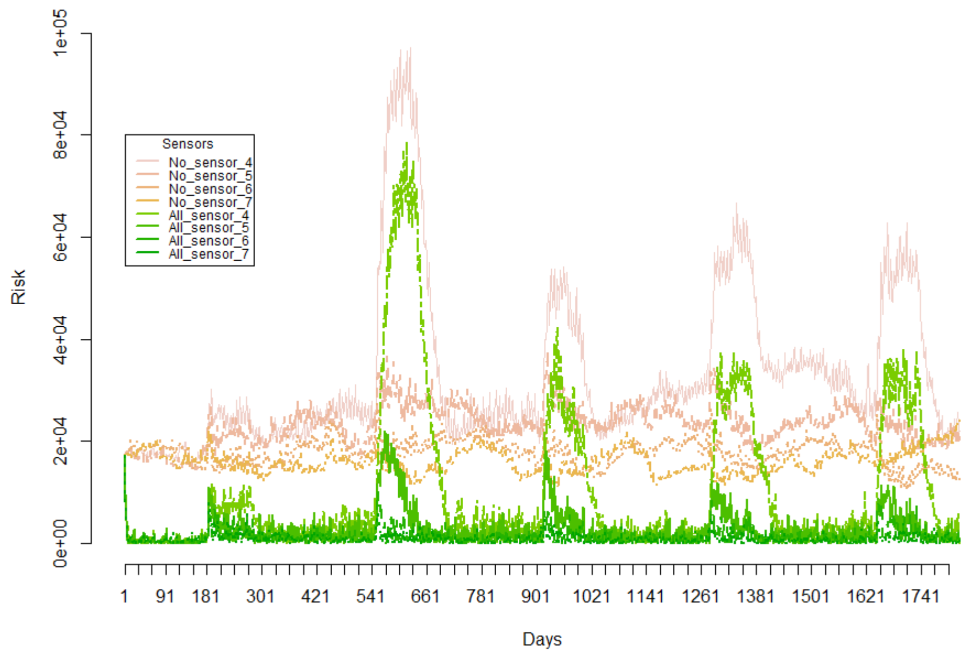


Figure 5.8: Daily risk tracked over a simulation of five years using the manual scheduling strategy with different working frequency in the normal scenario. Legend: sensorInfo_workingDays. (Lower risk is better)

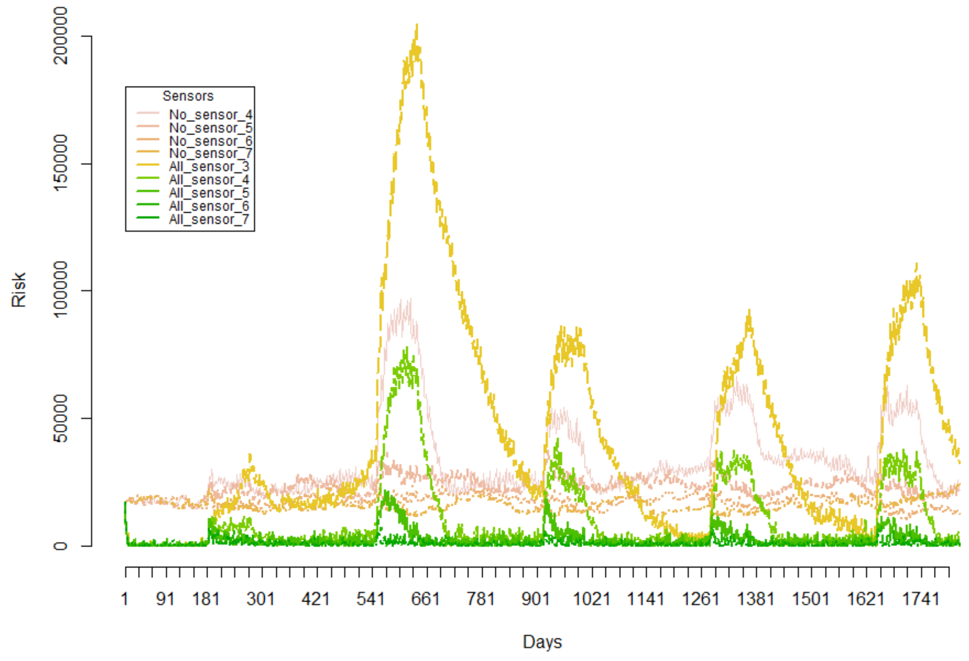


Figure 5.9: Daily risk tracked over a simulation of five years using the manual scheduling strategy with different working frequency in the normal scenario.

The results above illustrate the minimum number of days (i.e. five days a week) required to maintain the advantage due to timely system status information. To further explore the damage of insufficient maintenance frequency, we decrease the number of working days down to three days per week. The results, from Figure 5.9, show that there is no significant effect from this reduction of working frequency in the first year. However, the reduced maintenance results in gradually accumulated hazards, and these are suddenly exposed later, in what appears to be a critical threshold effect.

Focusing on the peak values, Figure 5.10 plots the worst risk value found in the simulation runs for the different working frequencies. The results show a clear, exponential risk increase in the worst situation when we decrease the working frequency every week. This result again highlights the negative consequences of insufficient system maintenance.

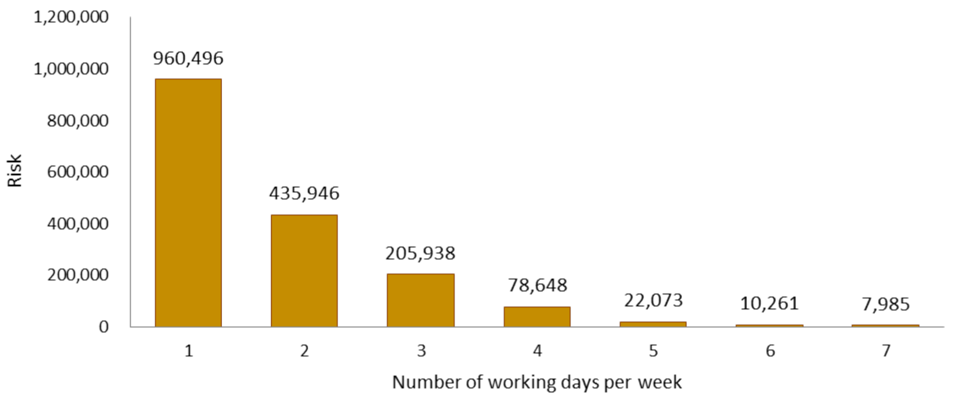


Figure 5.10: Peak risk value using different working frequency for a gully-pot system with full sensoring.

5.7 Conclusion

This chapter considers two factors that decrease the maintenance performance of the current manual schedule policy. We use simulation to explore the effect of “parking issues” and “untimely system status information” on maintenance scheduling, both of which are known weaknesses of the current manual maintenance approach.

Our simulation results suggest that a “banning parking” policy might reduce the surface water fooling risk to some extent. However, this policy may increase management complexity and residents’ complaints. In different scenarios, “banning parking” policy with different strength may achieve optimal effects while using least effort.

When we analyse the scenarios in which timely gully-pot status information can guide our maintenance schedule, the results show that the “untimely information” is a significant factor in lowering the efficiency of maintenance. Exploring the hypothetical use of sensors to provide timely information, our simulation results show that significant risk reduction can be obtained by sensor informed maintenance. Low-cost wireless sensor techniques could be a good investment to help produce an informed maintenance schedule and lower risk.

Even in practice, where the sensor technique cannot achieve up to 100% reliability, our preliminary simulations show that a full-sensoring system can cope with up to 30% false positive and false negative error information. Furthermore, the benefit of sensoring can still hold when we reduce the working week by two days.

Further work is needed to form a cost/benefit analysis to discover the optimal quantity of sensors to deploy, their locations and network topology. New scheduling approaches may be required to make best use of the potentially large amount of data generated by the sensors.

In the next chapter, we will look at how to improve the current manual maintenance scheduling using the available gully-pot information (estimated and known). We propose a predictive scheduling strategy that automatically adjusts despatching of preventative and corrective actions according to environment changes.

Chapter 6

Optimisation of a gully-pot system maintenance scheduling

In this chapter, we continue the research on the gully-pot system maintenance problem, moving from investigating policies that can improve efficiency with existing manual scheduling approaches (Chapter 5) to improving the scheduling process itself. We aim to automate the scheduling process, using a risk driven model to guide despatching of maintenance actions and optimise the service route simultaneously. This work has been published in Chen et al. (2016b).

6.1 Introduction

To solve our gully-pot system maintenance scheduling problem, we consider the risk driven model presented in Chapter 4. For completeness, we recall the high level objective (Equation 4.1) that is to select a judicious subset of gullies from N and assign them to days of the following short period, in order to minimise the risk in this period:

$$\sum_{d \in W} \sum_{i \in N} r_i P_i(d)$$

In the above equation, each gully-pot i in the system is associated with a risk impact value r_i. In a general sense, a higher risk impact here implies that if a particular gully-pot is blocked and floods happen, it results in relatively larger economic and social losses. Details of the estimation of r_i for each gully-pot in Blackpool is introduced in Section 4.3.1. $P_i(d)$ describes the probability that a gully-pot i is in its failure state on day d. We introduce a Weibull distribution that considers seasonal and local factors to adjust each gully's life time estimation (Section 4.3.2).

An interesting feature of our problem is that the objective function is designed for a short-term scheduling problem, but the overall aim is to analyse the scheduling impact for long-term risk management. Due to the changing environment and unexpected emerging situations, we cannot assume any repeated schedules between periods. To solve the long-term scheduling problem, a rolling horizon approach is devised, in which the short-term problem is solved repeatedly, given updates to environment and gully-pot status.

We propose an approach that maintains a set of distance optimised routes evolving with the environmental changes over time. We apply a tabu-based hyperheuristic – binary exponential back off (BEBO) (Remde et al. (2009)) which manages a set of route-adapting and scheduling low level heuristics to improve the solution iteratively.

There is good evidence that hyperheuristics can be successfully applied to various combinatorial problems, such as timetabling (Burke et al. (2007b); Bai et al. (2012)) and vehicle routing (Garrido and Riff (2010); Misir et al. (2011); Walker et al. (2012)).

The remainder of the chapter is organised as follows. Section 6.2 describes a predictive scheduling strategy to solve the gully-pot system maintenance problem. A comprehensive discussion and analysis of a set of gully-pot system maintenance policies is given in Section 6.3. Finally, we present the conclusion and directions for future research in Section 6.4.

6.2 Solution approach – predictive scheduling strategy (PSS)

In the simulation of the current manual scheduling strategy (Section 5.3), we described how to generate the candidate route set for preventative and corrective actions. Our PSS maintains a similar candidate route set. However, in order to schedule the routes in a smarter way, we add a new data structure to efficiently re-evaluate the risk of each route at any decision point. Furthermore, adaptations are described to dynamically modify the route structure according to changes in the environment.

6.2.1 Solution representation

Figure 6.1 shows the data structure used to store the solution. A $|W|$-days schedule contains selected $|W|$ number of candidate routes' IDs. Each route in the candidate set is optimised on distance. A route is composed of an ID, route information and the actual tour. Route information includes up to date gully-pot condition which helps to produce schedules, shown as follows:

1. route length;
2. number of gully-pots;
3. current route risk, which is the sum of the risk impact for each gully-pot multiplied by that pot's current failure rate;
4. tabu tenure l in days, which is used to stop the revisiting of the same preventative route in the near future.

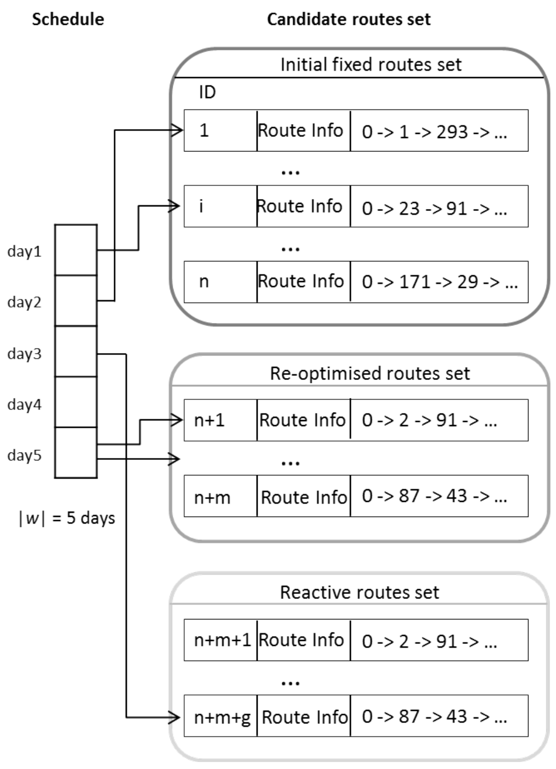

Schedule
Candidate routes set
Initial fixed routes set
ID
1 Route Info 0 -> 1 -> 293 -> ...
...
i Route Info 0 -> 23 -> 91 -> ...
...
n Route Info 0 -> 171 -> 29 -> ...
day1
day2
day3
day4
day5
|w| = 5 days
Re-optimised routes set
n+1 Route Info 0 -> 2 -> 91 -> ...
...
n+m Route Info 0 -> 87 -> 43 -> ...
Reactive routes set
n+m+1 Route Info 0 -> 2 -> 91 -> ...
...
n+m+g Route Info 0 -> 87 -> 43 -> ...

Figure 6.1: Solution representation and data structure for storing candidate routes, where the notations n, m, g represent the number of routes stored in the "initial fixed", "re-optimised" and "reactive" routes set respectively[1]. In the example above, we have a 5-days schedule and a route from the candidate routes set is selected to each day of the 5-days planning period.

6.2.2 Candidate route set management

The candidate route set (Figure 6.1) consists of an initial fixed route set (Section 6.2.2.1), a re-optimised route set (Section 6.2.2.2) and a reactive route set (Section 6.2.2.3). A solution to our problem selects $|W|$-routes from this set and specifies the order in which they are serviced.

[1]These notations have no relations with those used to describe our GDMP model in Chapter 4.

6.2.2.1 The initial fixed route set

Routes optimisation is very CPU intensive, especially for such large problems. Constructing routes repeatedly in a rolling planning schema is not efficient. Once the initial fixed routes set is built, these tours are not changed during execution; the fixed routes set stores the initial solutions for preventative maintenance.

Here, we start by finding a group of optimised candidate routes that can be scheduled directly or adjusted based on updated information before the use of the route in future days. At this stage, we treat the problem as a static VRP without considering any risk impact or lifetime information. To build the fixed preventative route set, we use the same process as in the manual approach (Section 5.3.2).

At the end of the initial route set construction, we have a route set S_{fixed} that visits every gully-pot at least once and each route is minimised in distance and maximised in the usage of time limitation T_{max}.

6.2.2.2 The re-optimised route set

Routes in the re-optimised routes set, denoted as S_{reopt}, are repeatedly updated during optimisation, as a result of environmental changes that cause gully-pot status changes. Section 6.2.3 describes the process of generating and maintaining routes in this set in detail. The size of the re-optimised route set is fixed. We can consider the S_{reopt} as a short memory buffer to store recently constructed high risk routes and these m number of routes are optimised in distance. When the set is full and new routes are generated, the oldest route is replaced. m is a user defined parameter. An impact analysis of m is given in Section 6.3.1.2.

6.2.2.3 The reactive routes set

Reactive routes are built using the same process introduced in manual scheduling (Section 5.3.3); before scheduling routes into days, we create candidate routes in the reactive routes set based on emerging events information. All these routes are discarded when a schedule solution is executed.

Again, the same method is applied to create candidate route sets S_{calls} and S_{broken} (Section 5.3.3). In addition to the S_{calls} result produced in manual scheduling simulation, each route in S_{calls} is treated as an opportunity to clean more normal-state gully-pots on the journey, as the same vehicle is used for the task. So, for each route in S_{calls}, we try to insert the closest gully-pot-cluster (see Section 5.3.1) that are in a normal state, and whose time since last service is longer than 30 days. We use least cost insertion, until no more points can be inserted without breaking the schedule duration constraint based on T_{max}. We denote the further optimised route set as S^*_{calls}

At this point, we have a candidate routes set (including initial fixed routes, routes that mostly contain reported gullies and routes that only contain broken gullies) optimised in distance:

$$S_{all} = S_{fixed} \cup S^*_{calls} \cup S_{broken}$$

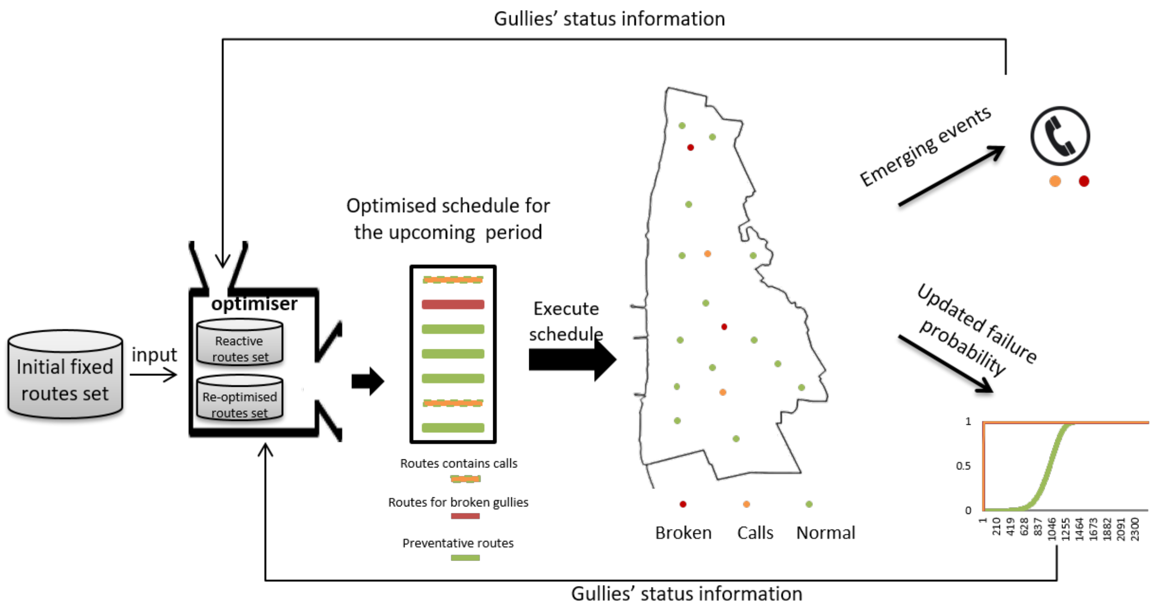


Figure 6.2: Overview of system operation

6.2.3 Producing a schedule

The PSS runs in continuous time. Figure 6.2 illustrates an overview of the system information flow, and Algorithm 6.2.1 describes our rolling horizon optimiser that automatically selects appropriate maintenance actions (either preventative or corrective) for the upcoming period.

6.2.3.1 Initialization

The initial schedule simply chooses the $|W|$ number of routes with tabu tenure l_s equal to zero, from all candidate routes, S_{all}, with the highest risk, $\sum_{i \in s} r_i P_i(today)$.

Algorithm 6.2.1 Rolling horizon optimiser – algorithm sketch

```
Define:
    $S_{fixed}$ is the initial fixed route set containing distance optimised routes.
    $S_{reopt}$ is a set of distance optimised routes that are updated during the search
  according to the recent gully-pots risk information; initially $S_{reopt} = \emptyset$
    $l_s$ is the tabu tenure of route $s$ in days, to stop revisiting of this route in near future
  (Section 6.2.1).
    $u_s$ is a flag parameter to prevent cyclic testing of route $s$ when using the scheduling-
  related $LLHs$ (i.e. $LLH_3$,$LLH_4$ in Section 6.2.3.2)
Rolling horizon repeat every $|W|$ days:
    1. Generate $S_{calls}$ and $S_{broken}$ based on emerging events. $\forall s \in S^*_{calls} \cup S_{broken}, l_s = 0$
    2. Get the candidate routes set $S_{all} = S_{fixed} \cup S_{reopt} \cup S^*_{calls} \cup S_{broken}$. $\forall s \in S_{all}, u_s =$
  $false$
    3. Generate $|W|$ days schedule solution $x$ that minimises the objective function 4.1
  (Section 6.2.3.1 – 6.2.3.3).
    4. Update risk information for routes $s \in S_{fixed} \cup S_{reopt}$, based on the changing
  condition of gullies;
    5. If any route $s \in S_{fixed} \cup S_{reopt}$ is scheduled in $x$, $l_s = 30(days)$, otherwise
  $l_s = l_s - |W|$
```

6.2.3.2 Improve the schedule using BEBO heuristic

The improvement stage is developed from a tabu search based hyperheuristic method – binary exponential back off (BEBO), proposed by Remde et al. (2009). BEBO has the fundamental structure of a hyperheuristic search strategy – a trial set of low level heuristics (*LLHs*) and systematic rules that control the usage of each *LLH*. BEBO uses dynamically adapted tabu tenures (Glover and Laguna (2013)) during the search process, which is especially useful when a large number of neighbourhoods are involved. If a *LLH* performs poorly recently, it is disabled for a number of iterations. If the *LLH* performs poorly continuously, the number of forbidden iterations increases, governed by a "*backoff*" value. The detailed searching framework is shown in Algorithm 6.2.2 (Remde et al. (2012)).

Apart from the hyperheuristic framework, a well-designed set of *LLHs* is crucial to successfully applying a hyperheuristic. In our implementation, the *LLHs* are designed from two aspects: 1) route-related moves that modify routes by changing segments or points in or between routes; 2) schedule-related moves that assign an optimised route to a day. The value of a solution is measured by the objective function in Equation 4.1.

▼ Route related moves:

The following route related moves are only applied to preventative routes ($s \in S_{fixed}$) and routes that contain mostly reported calls ($s \in S^*_{calls}$). Fixing broken gully-pots is carried out by a different vehicle. These reactive routes $s \in S_{broken}$ are constructed as described in Section 6.2.2.3 and no more route structure optimisation is processed.

LLH_1 *i-cross-exchange.* For any two scheduled routes r_1 and r_2, apply *i-cross-exchange* (Section 2.1.2.2). If any resulting route visits one point more than once, the points adjacent to longer edges are removed. Moves are examined for each pair of routes in a nested loop, the first yielding an improvement being implemented. $(1 \leq i \leq 5)$.

LLH_2 i-worst point insertion $(5 \leq i \leq 20)$. This *LLH* improves the next $|W|$ days' scheduled routes by finding the i highest risk points not appearing in the current schedule solution x. These i points are then inserted into the $|W|$ days schedule using a cheapest insertion heuristic (Section 2.1.2.1) with a relaxed time limit. If any target route scheduled in W now exceeds the T_{max} limitation, we repeatedly remove the best-condition point from that route until it becomes feasible.

The two *LLHs* above keep a copy of the original routes and generate new routes through operations. New routes are stored in the re-optimised routes set S_{reopt}. Though these modified routes may not generate improvements for the current iteration or the current short planning horizon, they normally contain relatively high risk gully-pot-clusters in recent time. Hence, they are still likely to be picked up using schedule related moves later or contribute to the near-future plan.

▼ Schedule related moves:

LLH_3 n-replace schedule $(1 \leq n \leq |W|)$ (see Algorithm 6.2.3). Replace the last n days' schedule with n other routes from the candidate set S_{all}, that are not included in the current solution, and whose tabu tenures l_s equals zero and has not been tested during the search $(u_s = false)$. We sort the candidate set S_{all} to check the higher risk routes first as these moves are more likely to produce improvements.

LLH_4 n-replace schedule random $(1 \leq n \leq |W| - 1)$. Same as LLH_3, except that we choose the n day's schedule to replace randomly, instead of the last n day's schedule.

LLH_5 switch two days' schedule (see Algorithm 6.2.4). First improvement scheme is applied.

Algorithm 6.2.2 BEBO hyperheuristic

```
Define:
    $x$ is the current solution;
    $LLH_i$ is a low level heuristic;
    $\triangle(x, LLH_i)$ returns the fitness value of a neighbour solution from applying $LLH_i$ to
current solution $x$, calculated based on the objective function 4.1;
    $tabu_i$ is the tabu tenure of $LLH_i$
    $backoff_{min} = 5$ is the minimum backoff value
    $backoff_i$ is the backoff value of $LLH_i$, where $backoff_i \geq backoff_{min}$
for all $i$ do
  set $backoff_i = backoff_{min}$
  $tabu_i = 0$  //we allow all LLHs to try at least once at the beginning
end for
while $\exists i$ that $tabu_i = 0$ do
  $bestvalue = x.value$
  for all $LLH_i$ do
    if $tabu_i = 0$ then
      if $\triangle(x, LLH_i) < x.value$ then
        $backoff_i = backoff_{min}$
        if $\triangle(x, LLH_i) < bestvalue$ then
          $bestvalue = \triangle(x, LLH_i)$
          $besti = i$
        end if
      else
        $backoff_i = backoff_i * 2$
        choose $tabu_i$ randomly from $\{0, 1, ..., backoff_i\}$
      end if
    else
      $tabu_i = tabu_i - 1$
    end if
  end for
  if $bestvalue < x.value$ then
    $x \Leftarrow apply(x, LLH_{besti})$
  end if
end while
```

Algorithm 6.2.3 n-replace heuristic

```
$S_{all}$ is a list of candidates routes sorted by risk in descending order
for each day $i$ in the last $n$ days' schedule do
  for each route $s$ in $S_{all}$ where $u_s = false$ && $l_s = 0$ && $s \notin x$ do
    $x' =$ replace the route scheduled in day $i$ with $s$;
    if $x'.value < x.value$ then
      $x = x'$
      $u_s = true$
      $break$
    end if
  end for
end for
return $x$
```

Algorithm 6.2.4 switch heuristic

```
for $i = 0; i < x.length; i = i + 1$ do
  for $j = i + 1; j < x.length; j = j + 1$ do
    $x'$ = switch the $i$th and $j$th days' schedule;
    if $x'.value < x.value$ then
      return $x'$
    end if
  end for
end for
return $x$
```

Algorithm 6.2.5 pop up (entry, stop)

```
Define: $entry$: the route to pop up
        $stop$: the target day;
for $i = entry; i > stop; i = i - 1$ do
  for $j = i - 1; j \geq stop; j = j - 1$ do
    $x'$ = pop up $i$th day's schedule to $j$th day;
    if $x'.value < x.value$ then
      return $x'$
    end if
  end for
end for
return $x$
```

LLH_6 pop up (Algorithm 6.2.5). Pop up ith day's schedule to a target position j. For example, one neighbour of solution 1,2,3,4 can be 1,4,2,3 by popping up 4 to the second position. In Algorithm 6.2.5, entry=$|W|$ and stop=1 are used in following experiments.

In summary, if we need to produce a $|W| = 7$ days schedule, in total 36 *LLH* will be called. The *LLHs* set contains both route structure adaptation and schedule modification according to risk estimation.

Our preliminary experiments show that all the *LLHs* contribute to the final solution quality. Among them, LLH_2 makes the most improvements. Also, LLH_2 helps the solver continuously add new elements to the candidate routes set. Our *LLHs* do not allow any individual route to visit one point more than once. However there is no rule to eliminate a solution that contains a gully-pot-cluster more than once during the period W: our experiments suggest that such a sub-optimal solution is easy for the algorithm to improve using LLH_2, LLH_3, LLH_4, which is thus rarely seen in practice. If a resulting solution suggests to visit a point more than once within W, the heuristic is opportunistically visiting a recently cleaned gully that lies close to the current route.

6.2.3.3 Improve current solution by partial rebuilding

At the end of the BEBO improvement stage, a local optimum solution x is returned and a reinitialisation process is applied to escape the local optimum, by partially destroying and rebuilding x. Then the BEBO improvement and reinitialization repeats for a given CPU time. The global best solution is remembered.

Destroy: For a $|W|$-days schedule solution, we randomly remove y days of the schedule, where $y \leq |W|/2$;

Rebuild: Here, we build y new routes that can replace the y removed schedules. First, from the optimised routes information stored in the fixed memory, we know the average number of points $\bar{n}$ included in a route. We then select n_{worst} number of points with the highest risk under the current environment and n_{random} random points that have not been visited in the $|W| - y$ unchanged scheduled routes, where $n_{random} = n_{worst} = \bar{n} * y/2$. Next, the entire process in Section 6.2.2.1 is applied to the selected points, resulting in z distance-optimised routes, which are stored in the the re-optimised routes set S_{reopt}. Finally, y out of the z routes are randomly assigned to replace the removed schedules.

6.3 Risk driven analysis of PSS

In this section, we first introduce our simulation settings. Then, we determine the rolling planning horizon by experimenting with its effect on risk management under different environmental conditions. Finally, we test the PSS and compare it with the current manual approach and a few other common scheduling policies. We aim to understand how different maintenance policies affect the surface water flooding risk due to blocked gully-pots in the long term. All simulations were implemented in C# and executed on a cluster composed of 8 Windows computers with 8 core, Intel Xeon E3-1230 CPU, and 16GB RAM.

6.3.1 Data & parameters

6.3.1.1 Simulation settings

Gully pot information comprises location, surrounding properties, nearby trees and historical maintenance actions from Blackpool local council, a client of Gaist Solutions Ltd. Details of simulation assumptions and environment settings are the same as introduced in Section 5.3. We use the inaccessibility figure of 8.3% to simulate the current parking issues, which implies not all gullies can be cleaned during preventative maintenance. We set the random broken event rate at $p_b = 0.00005$ that simulates the dynamic environment from Chapter 5.

6.3.1.2 Search parameter settings

The BEBO heuristic described in Section 6.2.3.2 is parameter-free, since all *LLHs* are given and it always chooses the best *LLH* at each decision point.

The termination criterion of the entire search process, composed of BEBO and reinitialisation, is controlled by a pre-set CPU time. Many heuristic search strategies find good solutions in the very early stages, but to find more improvements becomes harder and harder. To avoid either too early termination or unnecessary CPU consumption, we test the effects of limited computation time for various sizes of planning horizon, $|W|$. According to our experiments, about 0.002, 15, 68, 319 and 1189 minutes are required respectively for planning horizons $|W| = 1, 5, 7, 10, 14$ to achieve results that are within 2% of the best found solutions in preliminary experiments run over 48 hours of CPU times (see Figure 6.3). These CPU time limitations are used in the subsequent experiments.

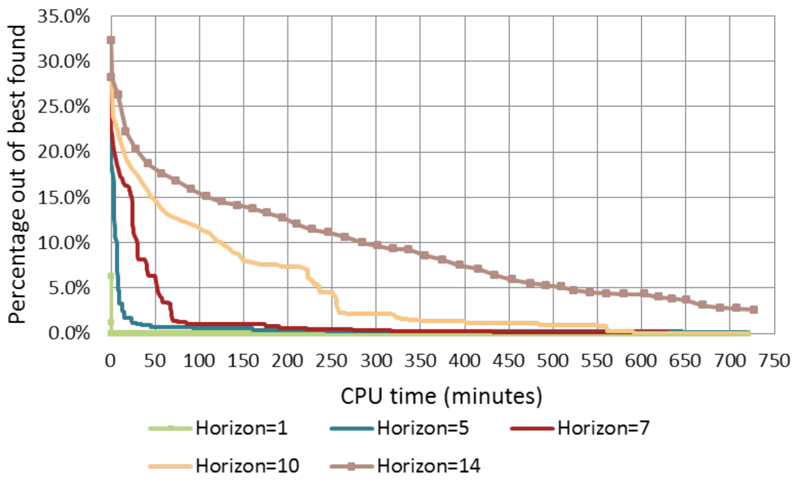


Figure 6.3: Effects of limiting computation time for different planning horizons

Considering the size m of the re-optimised routes set S_{reopt}, large values of m result in a more diverse set of routes, which may lead to a better solution. However, if m is too large, the increased CPU time (for schedule-related *LLHs* (Section 6.2.3.2) to find their local optima) does not yield better solutions in the time available. The route diversity due to larger m contains too many old updates during the search, which increases the searching complexity. If m is too small, route-related moves repeatedly generate the same or very similar routes. For our case, $m = 25\% * |S_{fixed}|$ is found to give the best balance between these two effects in preliminary experiments.

6.3.2 Impact of planning horizon $|W|$ on risk management in different environments

As we have seen in Chapter 5, gully-pot lifetimes are affected by seasonal factors, and peoples' reporting behaviour is different at different times of year. A short planning horizon may result in many reactive actions, and require more frequent information updates, whereas a longer planning horizon is better at balancing preventative and corrective maintenance. However, when $|W|$ is too large, it leads to a plan based on insufficiently up-to-date information.

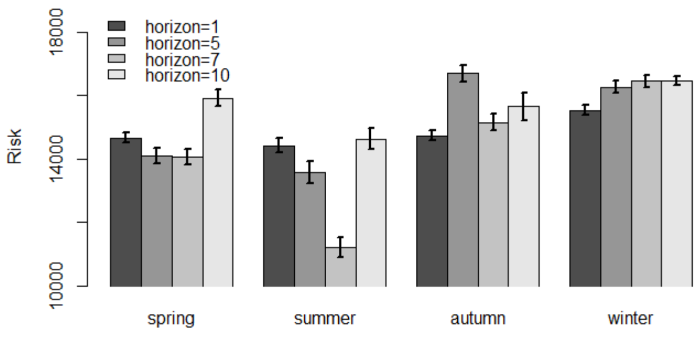
This section explores the impact of the planning horizon size $|W|$ on the maintenance performance in four seasons with the gully-pot system in either a normal or recovery scenario (Section 5.6). The parameter settings of the two scenarios are shown in Table 6.1. ($|W| = 1, 5, 7, 10$ are tested).

Table 6.1: "since last maintenance" and "initial broken gullies" set the system's initial state: for all gully-pots, the days since their last service are evenly distributed in θ years. We randomly assign a percentage of gully-pots to be in the broken state. In the simulation, $p_b = 0.00005$

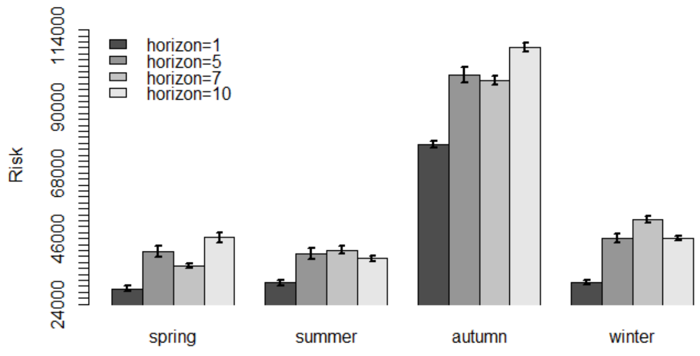
State	**Definition**	**Since last maintenance θ**	**Initial broken gullies**
normal	Based on the real-world situation, running simulation of maintenance actions for a long period until the overall system risk becomes stable; used as the initial state for all normal scenarios	1.5	0.7%
recovery	Start with very poor gully-pot conditions, and entire system in a high risk state	3	2%

Figure 6.4 shows the average daily surface water flooding risk caused by clogged gullies in Blackpool by using different planning horizons under different scenarios. We can be relatively sure that there is a genuine difference in risk, when both the mean values and the 95% confidence intervals differ. In the normal scenarios (Figure 6.4(a)), $|W| = 5$ and $|W| = 7$ perform better than other settings during spring and summer. Over autumn and winter, when the number of blocked gully-pots and calls significantly increases, $|W| = 1$ produces the best schedules, as it updates system and environment information most frequently. $|W| = 10$ performs badly in all seasons, due to lack of up-to-date information.

In the recovery scenarios (Figure 6.4(b)), the overall risk is about 2 to 3 times greater than when the system is in the corresponding normal state. In particular, if there is a lack of maintenance in autumn, this may lead to serious consequences. Again, $|W| = 1$ always produces the best schedules in the recovery state. This is because there is a significant number of emerging situations every day. Updating the system and environment information every day brings considerable advantages. In the recovery scenario, it is difficult to identify a single best value among $|W| = 5, 7, 10$; it is hard to balance the preventative and reactive actions by adjusting planning horizons, when recovering from a disaster.



(a) Normal



(b) Recovery

Figure 6.4: Impact of planing horizon on maintenance performance in different scenarios. Error bars show 95% confidence interval on each mean.

Table 6.2 presents scheduling performance in terms of corrective actions. As we described in Section 6.2, our solution does not impose hard constraints on the time taken to respond to residents' calls. Instead, the hyperheuristics automatically choose maintenance actions that minimise the entire system's risk. On average, all tested planning horizons react to emerging events in less than 7 days in a normal scenario. In the recovery scenario, $|W| = 1$ gives the fastest reaction to these emerging problem gully-pots. However, even with $|W| = 1$ there are big challenges in the autumn period, when the average delay between identification and correction of a problematic gully-pot rises to 34 days.

Table 6.2: The effect of planning horizon on corrective maintenance performance. Emergings per day: the average number of identified problematic gully-pots.

	Spring		**Summer**		**Autumn**		**Winter**	
	Average response	Emergings per day	Average response	Emergings per day	Average response	Emergings per day	Average response	Emergings per day
	Normal							
1	1.56	0.53	1.68	0.64	2.55	5.18	3.00	2.54
5	3.82	0.67	3.97	0.33	3.88	4.85	4.24	2.37
7	4.58	0.57	4.28	0.51	4.36	4.87	4.97	2.51
10	5.98	0.63	6.23	0.43	4.58	4.80	3.71	2.32
	Recover							
1	14.74	25.02	14.22	24.27	34.41	65.22	16.59	28.29
5	18.26	24.73	18.82	23.22	36.80	64.52	25.04	29.33
7	20.07	23.41	22.46	23.01	36.40	65.06	22.15	28.38
10	17.63	23.26	19.86	23.07	36.12	63.44	19.32	27.69

6.3.3 Effect of maintenance policies on risk in continuous time

Our essential aim is to reduce the surface water flooding risk for the entire city in continuous time. In the previous section, we seek the best-performing rolling planning horizon length. $|W| = 1$ requires the shortest computation time and produces the best schedule when the system is under pressure, but collecting the system and environment information every day is not feasible in real-world team management. When the gully-pot system is in its normal scenario, $|W| = 7$ shows the best ability to cope with seasonal changes. After consultation with Gaist Solutions Ltd., $|W| = 7$ is applied in the long period maintenance policy testing, since this balances team management requirements and scheduling performance.

In order to test the impact of how we manage preventative and corrective maintenance, we designed six policies that combine preventative and corrective actions with different rules. In these experiments, all scheduled routes are optimised on distance.

- Policy0: Pure reactive policy. Every week, we produce a $|W| = 7$ days schedule for reported problematic gully-pots only, according to up-to-date information. Priority is given to the emerging events with the highest risk. After finishing these planned tasks, we take a rest until the plan for the next week is produced.

- Policy01: Alternative pure reactive policy. Every day, we produce a $|W| = 7$ days schedule for reported problematic gully-pots only, according to up-to-date information. Only the first day schedule is executed, then we re-plan for the following week.

- Policy1: Pure reactive policy0 in autumn, PSS (see Section 6.2) with planning horizon $|W| = 7$ in other seasons.

- Policy2: PSS, introduced in Section 6.2, for all seasons.

- Policy3: Fixed manual schedule. All preventative routes are generated at the beginning of a year, giving the routes stored in fixed memory, see Figure 6.1. These routes are arranged in descending order of risk measured at the initial time. Every week, we use the first two days to deal with emerging events and use the remaining 5 days to deploy preventative actions, in order. During corrective time, we give priority to routes with the highest risk, $\sum_{i \in s} r_i P_i(today)$.
- Policy4: Dynamic manual schedule. Similar to policy 3, but priority is given to corrective maintenance. This is the manual approach we tested in Chapter 5.

We evaluate the performance of each policy from three aspects: overall risk management, agility to emerging events, and running cost. These six policies are firstly tested in a normal scenario and then we test their recovery speeds in a variety of bad initial situations. The daily risk is evaluated from the actual blocked and broken gully-pots with their associated risk impact.

6.3.3.1 Performance in the normal scenario

We simulate each policy on the Blackpool gully-pot system over four years, with corresponding seasonal settings and residents' reporting behaviour. Five random runs are carried out for each policy. We evaluate the average daily risk based on these experiments.

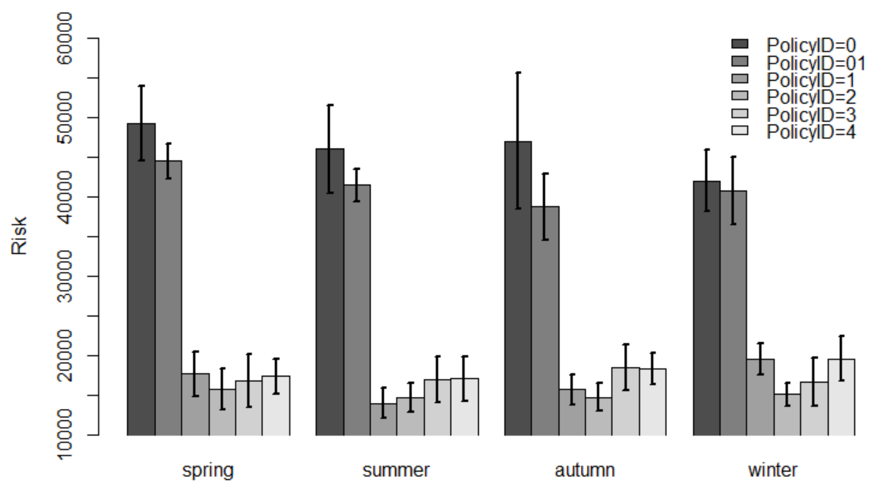
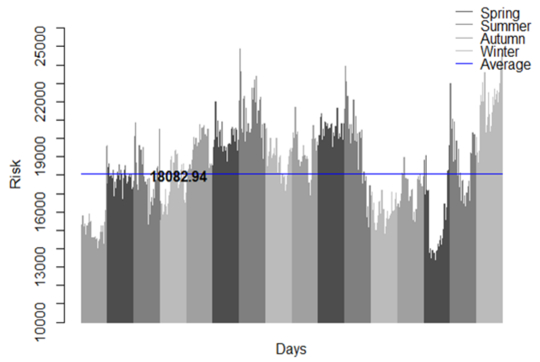
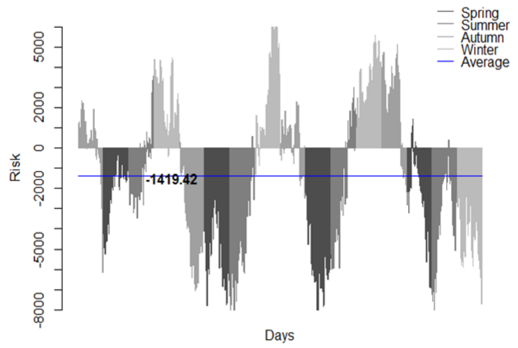


Figure 6.5: Policy performance in the normal scenario. Error bars show one standard deviation of daily risks for each season.

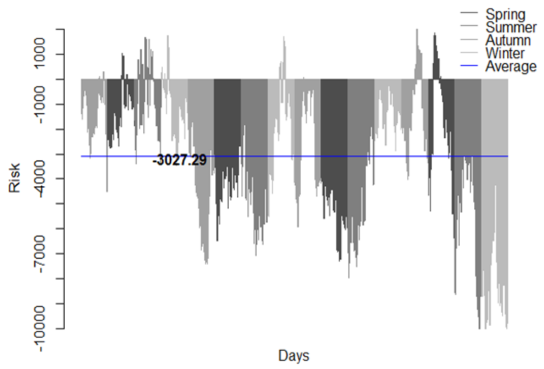
Risk management Figure 6.5 shows the average daily risk when applying the different maintenance policies in a normal scenario. Pure reactive policy 0 produces the highest risk all the time, and is about three times worse than any preventative and corrective combined



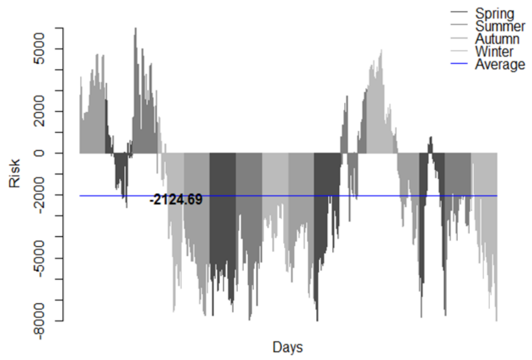
(a) PolicyID=4



(b) Policy 1 VS. Policy 4



(c) Policy 2 VS. Policy 4



(d) Policy 3 VS. Policy 4

Figure 6.6: Daily risk change over 4 years in the normal scenario using 4 types of preventative plus corrective maintenance policy.

policy. Even if we reschedule every day (policy 01), pure reactive maintenance still preforms significantly worse than other policies. The performance of pure reactive policies in autumn is not significantly worse than their performance in other seasons. However, their data shows very big deviations in autumn, which suggests large fluctuations happen. In the daily performance tracking, we find serious risk increases at the beginning of autumn, due to the lack of maintenance in other seasons and environmental factors. Also, in autumn, residents' reporting behaviour helps to prompt a large number of reactive actions.

Among all preventative policies 1 to 4, the PSS (policy 2) achieves the best overall performance. It is significantly better than manual scheduling in summer, autumn and winter. In spring, there is not much difference in applying any of the preventative plus corrective policies. To track the daily risk change over time, we apply these four policies in exactly the same environment simulation for four years. As illustrated in Figure 6.6, policy 4 is used as a base line and the other three policies are compared against it. When applying policy 4 in a normal scenario, the estimated surface flooding risk is £18,082 on average per day. By just rearranging the preventative and corrective tasks, policy 3 achieves an average risk decrease of about 12% per day, which suggests that always giving priority to emerging events may lead to poor working efficiency. The best result is for policy 2, which produces schedules that out perform the base line (policy 4) in 91% of days over 4 years; on average, policy 2 decreases risk by about 17% per day.

Agility Table 6.3 presents the average number of days to respond to calls. All policies except policy 3 are able to react to emerging calls in less than 5 days on average. Policy 3 uses a very straightforward scheduling rule, which may be good for team management, but shows serious latency for emerging requests. When only applying reactive actions (policy 0 and 01), on average about 3 times more residents' calls are received per day. This also exposes one reason for the poor performance of these policies in risk management (Figure 6.5): lack of preventative maintenance leads to more corrective maintenance.

Table 6.3: Agility analysis of different maintenance policies

	Average response	**Emergings per day**
Policy 0	4.88	11.14
Policy 01	2.24	10.88
Policy 1	3.93	4.31
Policy 2	4.34	3.30
Policy 3	**20.82**	2.83
Policy 4	4.67	3.19

Working efficiency analysis To discover how the PSS (policy 2) out performs other policies, we focus on time usage and work efficiency. Figure 6.7 illustrates the percentage of time spent in different types of activity. First, we can see that the reactive policy 0 shows high dependence on resident reports, resulting in working time of only about 45% percent during spring, summer and winter. Policy 3 follows a very straightforward rule, to do maintenance throughout the year. The fixed rule lacks the ability to adapt to seasonal changes. As Figure 6.5 shows, policy 3 has the largest fluctuations in all seasons compared to other preventative and corrective combined policies.

The time usage distributions of policy 2 and the manual schedule policy 4 show very similar patterns in Figure 6.7. Table 6.4 compares the daily working efficiency of policies 2 and 4. On average, policy 2 manages to service 10 more gully-pots every day within the same working time constraints. One reason is because policy 2 treats the resident calls and normal preventative maintenance together, so more efficient routes can be found and emerging blockages can be solved at the same time. Compared to the fixed preventative

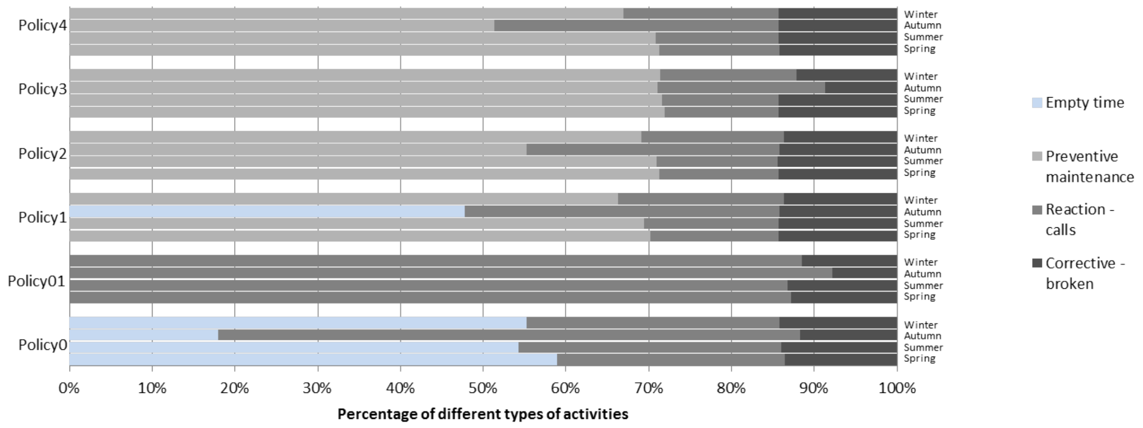


Figure 6.7: How do different policies use their time to do maintenance?

routes managed by policy 4, policy 2 always attempts to insert more high risk gully-pots into the current scheduled routes (Section 6.2.3.2, LLH_2), which results in automatically rescheduling of any missed gullies from previous preventative maintenance. Figure 6.8 illustrates further evidence that policy 2 produces better schedules. Comparing Figure 4.2 (the gully-pot risk impact map of Blackpool) and Figure 6.8 (the service frequency map under policies 2 & 4), we find policy 2 successfully targets the geographical areas which have been evaluated as highest risk. In contrast, policy 4 schedules the service times more evenly, which results in too many visits to low criticality areas.

Table 6.4: Average number of gully-pots serviced per day by policy 2 and 4

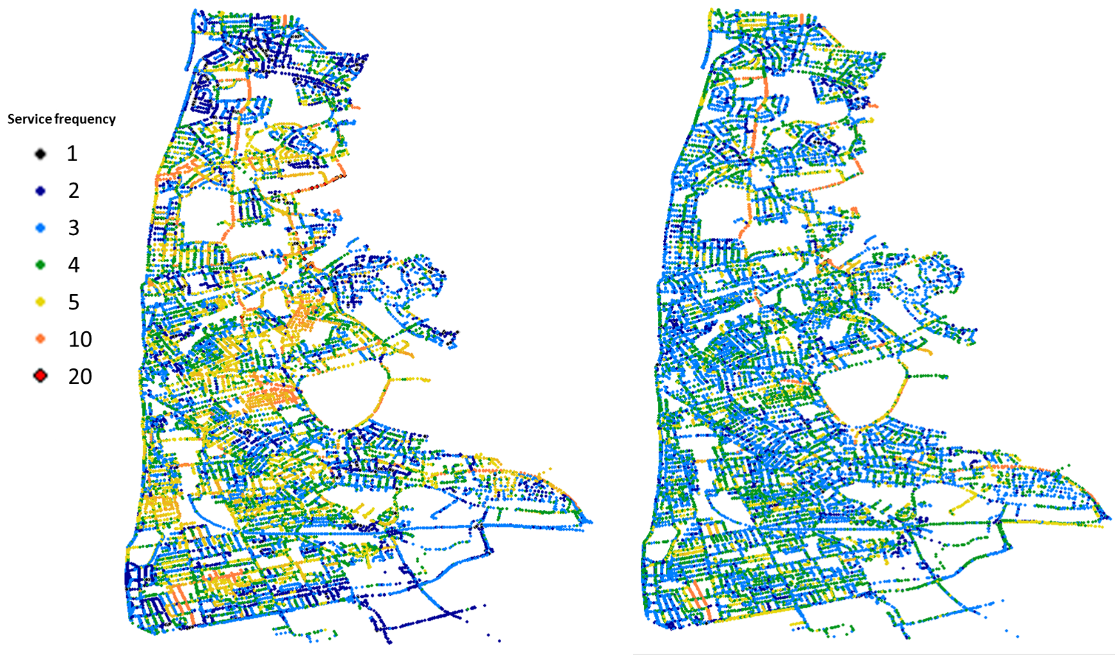
	Spring	**Summer**	**Autumn**	**Winter**
Policy2	81.90	82.09	83.41	81.42
Policy4	71.03	71.45	74.60	72.28

Cost using Blackpool's current operational costs (Table 6.5), we can estimate the annual cost of each maintenance policy. This allows us to explore the cost of extra effort required for preventative maintenance.

Table 6.5: Operation costs of gully-pot maintenance

	Cost	**Unit**
Travelling	£0.28	per km
Vehicle maintenance	£20,000	per year
Human resource	£56,000	per year
Preventative	£3.25	per gully
Calls response	£19.00	per gully
Broken	£225.00	per gully

The cost estimates for the different policies are shown in Figure 6.9. All of the preventa-



(a) Policy 2 (b) Policy 4

Figure 6.8: Geographic distribution of service frequency over a 4-year simulation

tive and corrective combined policies show expenditure of £280,000 to £300,000 annually. This means that, compared to the pure reactive policy 0, an extra 10% of expenditure could reduce potential risk by as much as a factor of 3, over time (Figure 6.5).

Comparing the predictive policy 2 to the current manual policy 4, about £8,000 more would need to be spent each year, due to the additional preventative work. However, these extra preventative actions would result in about £3,000 of risk reduction every day, or over £1 million per year.

Due to data limitations, our current simulation of gully-pot breakage behaviour uses a fixed probability, giving roughly 500 broken gully-pots a year, generated at random times. This simplistic breakage regime, which is the same under each policy, results in all policies presenting similar effort to tackle broken gully-pots. In practice, policies with regular preventative maintenance would slow deterioration, and might decrease the chance of breakage. We would expect a more realistic model of breakage probability to reduce the apparent cost of policies 1 to 4 to less than the cost of policies 0 and 01.

In conclusion, preventative maintenance could significantly ameliorate the surface water flooding risk caused by blocked gully-pots at a reasonable additional cost; these costs are more than justified by service quality improvement.

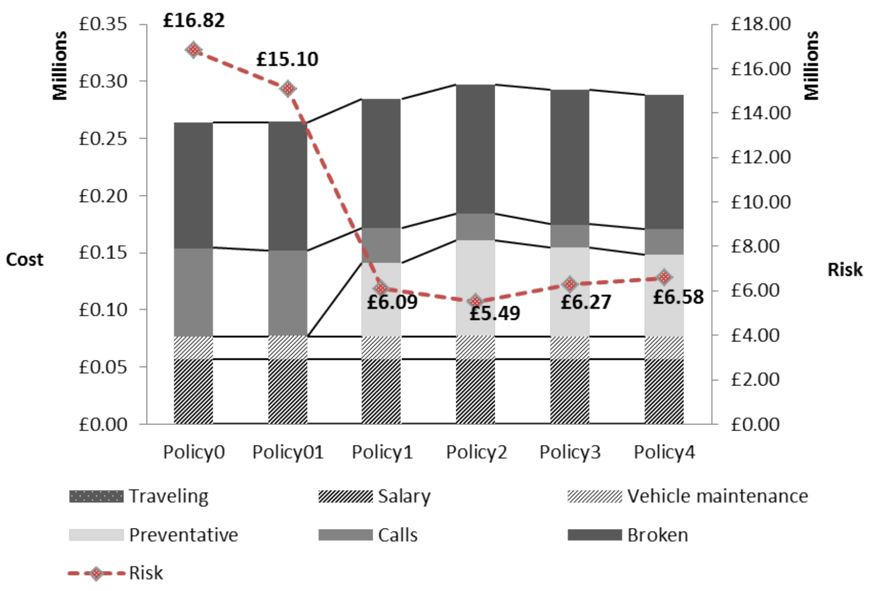


Figure 6.9: Annual operation cost and surface water flooding risk caused by clogged gully-pots for the different policies

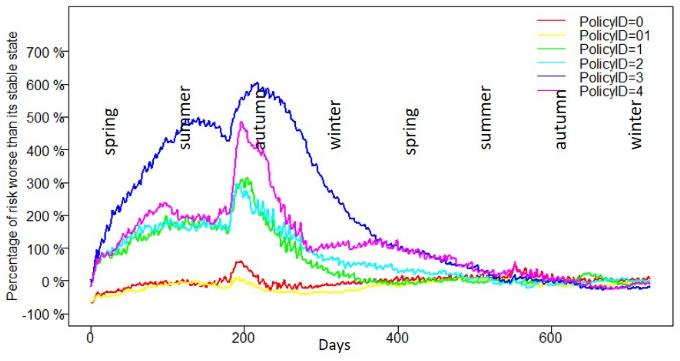
6.3.3.2 Performance in recovery scenario

Here, we test the robustness of each policy by starting from a very bad initial condition. We explore how long it takes for each maintenance policy to take the system from a poor initial state to a normal scenario. The average risk of applying each policy in a normal scenario (Section 6.3.3.1) is used as the policy's base line. As presented in Table 6.6, four recovery scenarios are tested. For each scenario, we report the average of 10 runs of a two-year simulation.

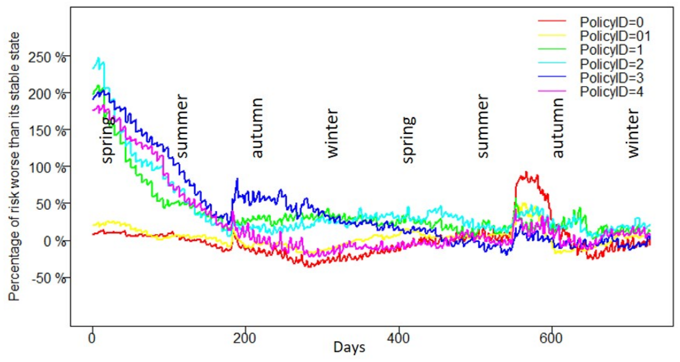
Table 6.6: "since last maintenance" and "initial broken gullies" set the system's initial state: for all gully-pots, the days since their last service are evenly distributed in θ years. We randomly assign a percentage of gully-pots to be in the broken state. In the simulation, $p_b = 0.00005$. (Scenarios 2 and 3 have the same experiment settings as recover-1 and 2 shown in Table 5.1)

	Scenario 1	**Scenario 2**	**Scenario 3**	**Scenario 4**
Since last maintenance θ	3 years	1.5 years	3 years	4 years
Initial broken gully-pots	0.7%	2%	2%	3%

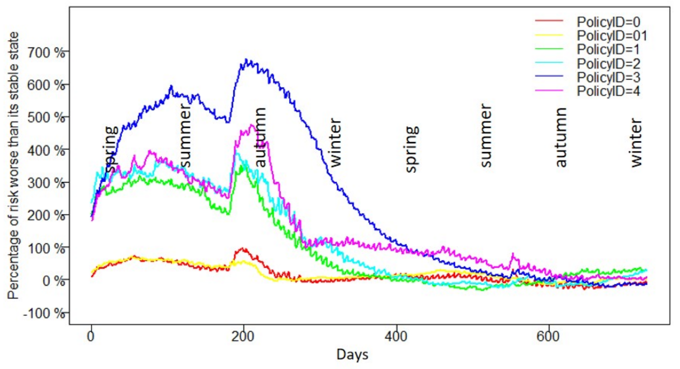
Recovery speed From Figure 6.10, we can see that the initial situation in scenario 2 is very close to the normal state of reactive policy 0. Comparing scenario 1 to scenario 2, the overall shortage of preventative maintenance (scenario 1) is more difficult to recover from than a small amount of very broken gully-pots in the system (scenario 2). On average, policies 1 to 4 need about 7 months to restore the system to its normal state in scenario 2 (see Figure 6.10(b)), whilst they need about 19 months to recover from initial situation in



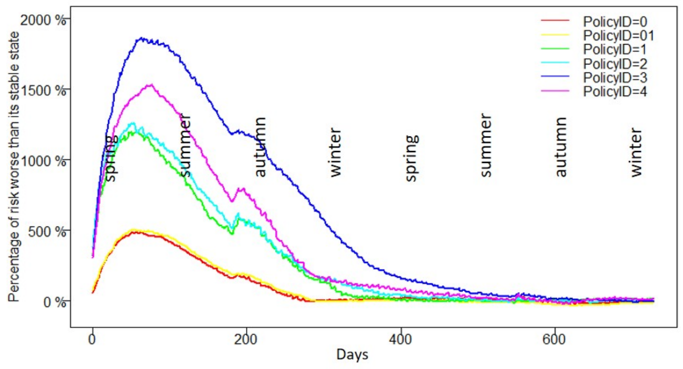
(a) Scenario 1



(b) Scenario 2

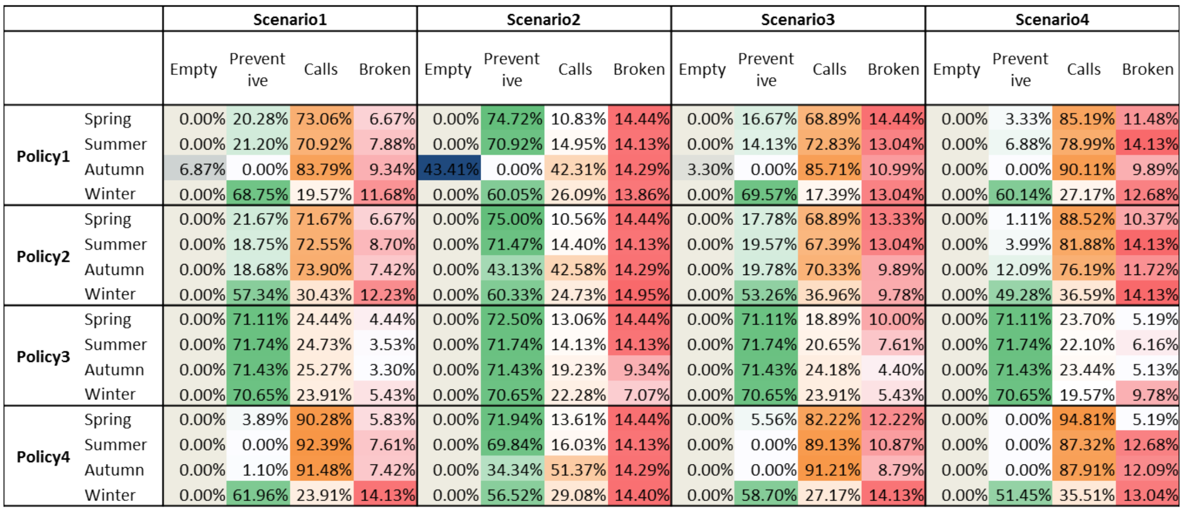


(c) Scenario 3



(d) Scenario 4

Figure 6.10: Recovery speed using different policies. The percentage of risk is calculated as $(r - \widetilde{r})/\widetilde{r} * 100\%$, where r represents the daily surface flooding risk and $\widetilde{r}$ is the average daily risk of applying the corresponding policy in its normal scenario (see Section 6.3.3.1).



		Scenario1				Scenario2				Scenario3				Scenario4			
		Empty	Preventive	Calls	Broken	Empty	Preventive	Calls	Broken	Empty	Preventive	Calls	Broken	Empty	Preventive	Calls	Broken
Policy1	Spring	0.00%	20.28%	73.06%	6.67%	0.00%	74.72%	10.83%	14.44%	0.00%	16.67%	68.89%	14.44%	0.00%	3.33%	85.19%	11.48%
	Summer	0.00%	21.20%	70.92%	7.88%	0.00%	70.92%	14.95%	14.13%	0.00%	14.13%	72.83%	13.04%	0.00%	6.88%	78.99%	14.13%
	Autumn	6.87%	0.00%	83.79%	9.34%	43.41%	0.00%	42.31%	14.29%	3.30%	0.00%	85.71%	10.99%	0.00%	0.00%	90.11%	9.89%
	Winter	0.00%	68.75%	19.57%	11.68%	0.00%	60.05%	26.09%	13.86%	0.00%	69.57%	17.39%	13.04%	0.00%	60.14%	27.17%	12.68%
Policy2	Spring	0.00%	21.67%	71.67%	6.67%	0.00%	75.00%	10.56%	14.44%	0.00%	17.78%	68.89%	13.33%	0.00%	1.11%	88.52%	10.37%
	Summer	0.00%	18.75%	72.55%	8.70%	0.00%	71.47%	14.40%	14.13%	0.00%	19.57%	67.39%	13.04%	0.00%	3.99%	81.88%	14.13%
	Autumn	0.00%	18.68%	73.90%	7.42%	0.00%	43.13%	42.58%	14.29%	0.00%	19.78%	70.33%	9.89%	0.00%	12.09%	76.19%	11.72%
	Winter	0.00%	57.34%	30.43%	12.23%	0.00%	60.33%	24.73%	14.95%	0.00%	53.26%	36.96%	9.78%	0.00%	49.28%	36.59%	14.13%
Policy3	Spring	0.00%	71.11%	24.44%	4.44%	0.00%	72.50%	13.06%	14.44%	0.00%	71.11%	18.89%	10.00%	0.00%	71.11%	23.70%	5.19%
	Summer	0.00%	71.74%	24.73%	3.53%	0.00%	71.74%	14.13%	14.13%	0.00%	71.74%	20.65%	7.61%	0.00%	71.74%	22.10%	6.16%
	Autumn	0.00%	71.43%	25.27%	3.30%	0.00%	71.43%	19.23%	9.34%	0.00%	71.43%	24.18%	4.40%	0.00%	71.43%	23.44%	5.13%
	Winter	0.00%	70.65%	23.91%	5.43%	0.00%	70.65%	22.28%	7.07%	0.00%	70.65%	23.91%	5.43%	0.00%	70.65%	19.57%	9.78%
Policy4	Spring	0.00%	3.89%	90.28%	5.83%	0.00%	71.94%	13.61%	14.44%	0.00%	5.56%	82.22%	12.22%	0.00%	0.00%	94.81%	5.19%
	Summer	0.00%	0.00%	92.39%	7.61%	0.00%	69.84%	16.03%	14.13%	0.00%	0.00%	89.13%	10.87%	0.00%	0.00%	87.32%	12.68%
	Autumn	0.00%	1.10%	91.48%	7.42%	0.00%	34.34%	51.37%	14.29%	0.00%	0.00%	91.21%	8.79%	0.00%	0.00%	87.91%	12.09%
	Winter	0.00%	61.96%	23.91%	14.13%	0.00%	56.52%	29.08%	14.40%	0.00%	58.70%	27.17%	14.13%	0.00%	51.45%	35.51%	13.04%

Figure 6.11: How do different policies use their time to recover from very bad initial conditions?

scenario 1 (see Figure 6.10(a)). Comparing policies 1 to 4 in scenario 1 and 3, we can see that policies 1 and 2 perform better than both of the manual policies 3 and 4 in terms of percentage risk increase. The robustness of both manual policies is considerably worse than that of policies 1 and 2, especially during the autumn period in the first year. Comparing the performance of policies 3 and 4, the fixed schedule strategy (policy 3) is a lot worse than the more flexible strategy (policy 4).

Activity changes during recovery stage To recover from different situations, policies 1 to 4 utilise their time in different ways. Figure 6.11 presents the time usage of each policy during the first year of the recovery stage. Policy 3 has fixed amount of preventative time, about 71%, through all scenarios. However, it still adjusts the remaining 29% of corrective action time to face different types of emerging events (including calls and broken gully-pots). Comparing policies 3 and 4 in scenario 1, 3 and 4, the relatively more flexible policy 4 almost stops its preventative actions except in winter. This flexibility helps policy 4 to recover the system faster in the early stage and results in less total damage during the recovery stage. Interestingly, both policies 3 and 4 take similar amounts of total time to recover the entire system in all scenarios: the rate of recovery for policy 4 slows over time. The predictive policy 2 balances its preventative and corrective time, and is between policies 3 and 4. The balanced strategy results in a steady recovery process; even though when we only do corrective work during autumn period (like policy 1) and has some resting days, the overall performance is not affected.

6.3.4 What-if questions

All the experiments introduced in the previous sections are based on the real-world scenarios. In this section, we test three hypotheses, which uncover potential weaknesses of our PSS (policy 2), and suggest future investment directions to improve maintenance performance.

6.3.4.1 What if we do not have information on risk impact?

Section 4.3.1 describes the method to collect and estimate each gully-pot's risk impact, which estimates the consequences of surface water flooding due to clogged gully-pots. However, not every local council records the information. Figure 6.12 illustrates the performance of policy 2 operating with and without risk impact information (labelled policy 2*). Policy 2* results in a much higher risk than policy 2 (Figure 6.8(a)); in fact, the lack of risk impact data means that policy 2 becomes similar to the policy 4: all gully-pots are serviced relatively evenly.

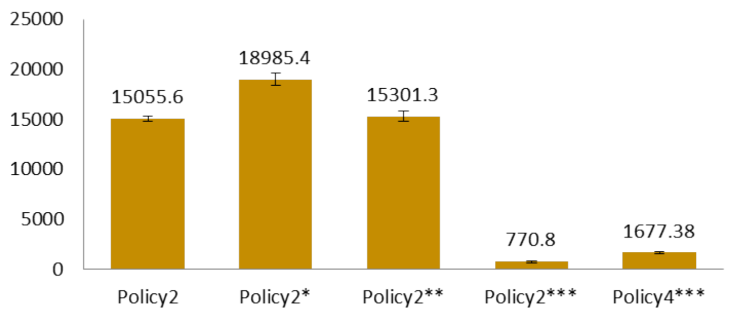


Figure 6.12: The average risk of applying policies 2 and 4 in stable state with 4 assumptions. Error bars show 95% confidence intervals on each mean. Policy 2 is running in the real-world scenario; policy2* assumes we do not have risk impact information; policy2** assumes there are no parking issues during preventative maintenance; policy2*** and policy4*** assume we can know any problematic gully-pot immediately.

6.3.4.2 What if all gullies are accessible – the impact of parking issues?

In Chapter 5, we investigated the effect of parked vehicles on the manual scheduling strategy (policy 4). Here, we explore the impact of inaccessible gully-pots on policy 2 by running simulations with zero inaccessible gully-pots (labelled as policy2**). From Figure 6.12, there is no significant difference between policy 2 and policy 2**. By using the PSS, policy 2 is able to cope with the current parking issues. This is because it flexibly re-schedules preventative maintenance of inaccessible gully-pots.

6.3.4.3 What if we could do condition-based maintenance (CBM)?

Improving low-cost sensor techniques make it potentially feasible to continuously monitor gully-pot condition. This would allow our schedule strategies to be combined with CBM. Chapter 5 discusses the advantages of sensoring technique tested on the current manual scheduling strategy (policy 4). Here, we would like to know whether combining a PSS with the sensoring technique will bring further improvement. In the simulation, we test a scenario in which all problematic gully-pots in the system are known immediately (labelled as policy 2^{***}). From the results in Figure 6.12, comparing policy 2^{***} and policy 4^{***}, we can see a 50% further risk reduction.

6.4 Conclusion

This chapter has solved the gully-pot maintenance in the city of Blackpool, using a PSS. The general aim is to reduce the overall surface water flooding risk caused by clogged gully-pots in continuous time. The PSS runs on a short period rolling planning horizon that is able to automate schedule adaptation to any environment changes.

Due to the dynamic and large-scale features of our problem, we introduce a data structure (see Figure 6.1) to deal with different types of actions. In addition, our objective function is highly sensitive to the gully-pots' changing failure rates. We present a hyper-heuristic framework embedded with a group of route and schedule-related *LLHs*. This structure allows dynamic balancing between route and schedule optimisation.

By adjusting different types of actions in different scenarios, our PSS comfortably outperforms the current real-world gully-pot maintenance approach, which is widely used in the UK, in terms of overall risk management, agility to react to emerging events, and robustness to poor initial states.

This predictive strategy relies significantly on the understanding of asset failure behaviour. Our estimations are based on working with experts in the field to provide the best (limited) data available. We are also working with Gaist solutions Ltd. on a new surveying methodology which will further improve data in the future, but such data will not be available for some time to come.

In Part I, we have fully investigated the large scale gully-pot system maintenance problem from perspectives including modelling (Chapter 4), simulation based issue detection in manual scheduling (Chapter 5), and the maintenance scheduling policy improvement (Chapter 6). In further work we will investigate other investment possibilities. It is worth noting that the work that Gaist Solutions Ltd. has done on road maintenance decision support has resulted in investment worth hundreds of millions of pounds across several UK local councils.

Part II

Heuristic Search Methods with Respect to PVRP and GDMP

Whilst part I of the thesis directly addresses issues of interest to Gaist Solutions Ltd. and its client local authorities, in this part, we focus more on the solution approaches that aim to effectively solve large scale combinatoric optimisation problems. Firstly, we review the fundamental concept of local search methods that are the main components of our search algorithms' design (Chapter 3). We investigate the search from two aspects, including comparison of various high level management strategies over a set of local search operators (Chapter 7), and statistically guiding the local search process (Chapter 8). These heuristic based approaches are tested on both benchmark periodic vehicle routing problems and the real-world geographical distributed maintenance problems.

Chapter 7

Hyperheuristics and local search operators for PVRP

Meta-heuristics and hybrid heuristic approaches have been successfully applied to the periodic vehicle routing problem (PVRP). However, to be competitive, these methods require careful design of PVRP-specific search strategies. In contrast, hyperheuristics use the performance of low-level heuristics (*LLHs*) to automatically select and tailor search strategies. In Chapter 3, we briefly introduced the concept and background knowledge of hyperheuristics. In this chapter, we provide a comprehensive analysis of several hyperheuristics and test them on a number of PVRP instances from the benchmarks and the real world, each with its own different characteristics. The aim is to understand the impact of different mechanisms that each hyperheuristic applies in its search process. We would also like to justify the hyperheuristic approaches that automatically manage *LLHs*, compared to the problem-specific heuristics designed for the PVRP. Finally, we use hyperheuristics as a tool to study the strengths and the weaknesses of *LLHs* designed for PVRPs. The content in this chapter has also been published in Chen et al. (2016d).

7.1 Introduction

The PVRP is the closest classical VRP model to our geographically distributed asset maintenance problem (Chapter 4). From simple heuristic approaches developed from the 1970s to the early 1990s (Beltrami and Bodin (1974); Russell and Igo (1979); Christofides and Beasley (1984); Tan and Beasley (1984); Russell and Gribbin (1991)), to the hybird meta-heuristic approaches appearing recently (e.g. Hemmelmayr et al. (2009); Vidal et al. (2012); Cordeau and Maischberger (2012)), heuristic-based approaches for the standard PVRP have experienced significant improvements. Section 2.2.1 provides a more extensive review of solution approaches for PVRP.

In comparison to meta-heuristics, a hyperheuristic aims to build more general problem-independent search algorithms, which are capable of producing sufficiently good and cheap solutions for different optimisation problems. Theoretically, a hyperheuristic should be able to not only successfully adapt to any hard computational search problem, but also serve as a good tool for studying the strengths and the weaknesses of *LLHs* for a specific type of problems.

In this study, we present a comprehensive analysis of hyperheuristic approaches to solving PVRPs. The performance of hyperheuristics can be compared to the published performance of state-of-the-art meta-heuristics.

The remainder of this chapter is organised as follows: Section 7.2 summarises the general structure and techniques that are often utilised in hyperheuristics; Section 7.3 introduces a number of hyperheuristic approaches to solve the PVRP; and Sections 7.5 presents the experimental design and the analyses of the results.

7.2 Choosing the right hyperheuristic structure

Hyperheuristics introduce various techniques to solve hard computational search problems. However, they all share the common goal of automating the design and adaptation of heuristics. The essential idea is to introduce artificial intelligence to an algorithm design for solving difficult problems.

Burke et al. (2010) classify hyperheuristics from multiple dimensions. Here, we refer to one of their classifications that considers hyperheuristics as either *heuristic generation* or *heuristic selection* methods. The first group of methods, which come under *heuristic generation*, normally apply genetic programming as a hyperheuristic to build new heuristic methods for the problem (e.g. Keller and Poli (2007); Burke et al. (2007a)). Most implementations generate a new heuristic using a training set of problem instances, which is thereafter used on unseen instances of the same problem. In comparison, the second group of methods, which come under *heuristic selection*, are the type of techniques that most other researchers refer to as *hyperheuristics.* In this context, hyperheuristics manage a set of *LLHs.* At each decision point, hyperheuristics should be able to decide which *LLH* is appropriate for usage. In the remainder of this thesis, we only discuss *heuristic selection* methods. This is mostly because it is more appropriate to integrate with existing well-performing local search techniques, which are specifically designed for routing problems.

The general structure of selection hyperheuristics includes a selection and an acceptance mechanism. Selection determines which *LLH* to test at each search stage. The decision making for *LLH* selection could be purely random or arbitrary. More intelligently, hyperheuristics can apply online learning concepts to adapt to decision-making processes with bias towards better performing *LLHs* in earlier iterations. An acceptance mechanism decides whether the current solution is replaced by the new solution. A number of acceptance mechanisms have been examined within hyperheuristics framework, such as only improving (e.g. Cowling et al. (2001)), "exponential Monte Carlo with counter" (e.g. Ayob and Kendall (2003)) and "great deluge" (e.g. Kendall and Mohamad (2004)).

If a hyperheuristic works properly, normally it effectively combines selection-acceptance with the given *LLH* set. Section 3.2.1 introduces a few design concepts of an *LLH* set. Generally speaking, an *LLH* can be designed as a random *move* from a neighbourhood (e.g. Misir et al. (2011); Sabar et al. (2014)), as a *first improvement* (FI) (e.g. Burke et al. (2003)), and as an embedded FI within a local search (e.g. Meignan et al. (2010)), or even as meta-heuristics. If an *LLH* set is only composed of simple *moves*, then the selection-acceptance mechanism plays a vital role to guide the search (e.g. Cowling et al. (2001)). Tested on some simple combinatorial problems, Özcan et al. (2008) concluded that

> *"the acceptance mechanism significantly affects the performance as compared to heuristic selection."*

If an *LLH* set is composed of many independent search strategies (e.g. FI or LS), the algorithm design normally focuses only on the selection strategy (e.g. Burke et al. (2003)).

Based on the study of many modern solvers for VRPs, it seems necessary to apply improvement heuristics to achieve good quality solutions. Therefore, the following study pays more attention to the management of a set of FI *LLHs* in the improvement stage of the search.

7.3 Hyperheuristic methods

Algorithm 7.3.1 presents an overview of our implementations of hyperheuristics. This algorithm combines a mechanism for iterated local search (Lourenço et al. (2010)) and selection hyperheuristics. Many hyperheuristic implementations can be summarised by the Algorithm 7.3.1 (e.g. Özcan et al. (2008); Burke et al. (2010, 2011)), which has been recently named as HyperILS by Ochoa and Burke (2014).

In lines 4 and 5 of the Algorithm 7.3.1, *HyperPerturbation* and *HyperImprovement* normally manage two different sets of *LLHs* for exploration and exploitation purposes during the search. Learning- or non-learning-based *LLH*-selection strategies can be applied in either or both of the perturbation and improvement stages. After an *LLH* is selected in the perturbation stage, we accept the output result from the chosen *LLH* as long as it is a feasible solution. The process of selection-acceptance is repeated until a feasible solution is found. In terms of the acceptance rule in the improvement stage, we apply *only improving* (OI) in our algorithm design.

In the following sections, we introduce the techniques, which have been investigated in the *HyperPerturbation* and *HyperImprovement* stages respectively.

Algorithm 7.3.1 HyperILS (from Ochoa and Burke (2014))

```
1: **Define:** $x_0$ is the initial solution
2: $x^* = HyperImprovement(x_0)$
3: **while** $t < t_{max}$ **do**
4:     $x' = HyperPerturbation(x^*)$
5:     $x^{*'} = HyperImprovement(x')$
6:     **if** $f(x^{*'}) < f(x^*)$ **then**
7:         $x^* = x^{*'}$
8:     **end if**
9: **end while**
```

7.3.1 *LLH* selection in HyperPerturbation

This study investigates two *LLH*-selection strategies in this stage of the search, including *Simple Random* (SR) selection and *Random Permutation* (RP) (Cowling et al. (2001)). SR chooses an *LLH* randomly based on a uniform probability distribution. RP generates a random initial permutation of the *LLHs* and at each step applies the next low-level heuristic in the provided order.

7.3.2 Search in HyperImprovement

In this section, we introduce the three following groups of hyperheuristics that guide the search in the *HyperImprovement* stage: simple hyperheuristics, learning-based hyperheuristics, and variable neighbourhood decent with learning (VND_L). The first two groups are named after the *LLH*-selection mechanism. The VND_L algorithms combine conventional VND (Hansen et al. (2010)) with learning-based selection to choose a subset of *LLHs* at each iteration.

7.3.2.1 Simple hyperheuristics

Simple hyperheuristics apply SR as the *LLH*-selection strategy. After an *LLH* is selected, we either apply it once or apply it repeatedly until no improvement is found, named *Simple Random* (SR) and *Random Descent* (RD) respectively (Cowling et al. (2001)).

7.3.2.2 Learning based hyperheuristics

Learning-based selection strategies consist of *credit assignment* and *selection* stages. *Credit assignment* decides how to reward the *LLHs*, which have just been tested in the current iteration, and updates the evaluation of any or all candidate *LLHs*. Many *credit assignment* strategies have been discussed, such as extreme value (Soria-Alcaraz et al. (2014), "choice function" (Cowling et al. (2001)), "tabu mechanism" (Burke et al. (2003)), and so on. According to the evaluation of each candidate *LLH*, learning-based hyperheuristics select the favourable *LLH(s)* to investigate in the next iteration. Cowling et al. (2001) describe three selection strategies named straight choice, ranked choice, and roulette choice. In later literature, these selection strategies are still the most common methods applied. Straight choice selects the *LLH*, which is evaluated as the best in the candidate set. Ranked choice adapts an *LLH* trial set at each iteration and it can select a fixed proportion of the highest ranking *LLHs* or *LLHs* that are not tabued if a corresponding *credit assignment* strategy is applied. Each tested *LLH* outputs a modified solution and a favourable output replaces the current one. Roulette choice selects an *LLH* in each iteration with a probability, which is proportional to the evaluation value of each *LLH*. Figure 7.1 illustrates a high level concept of using online learning to adapt the selection of *LLHs*.

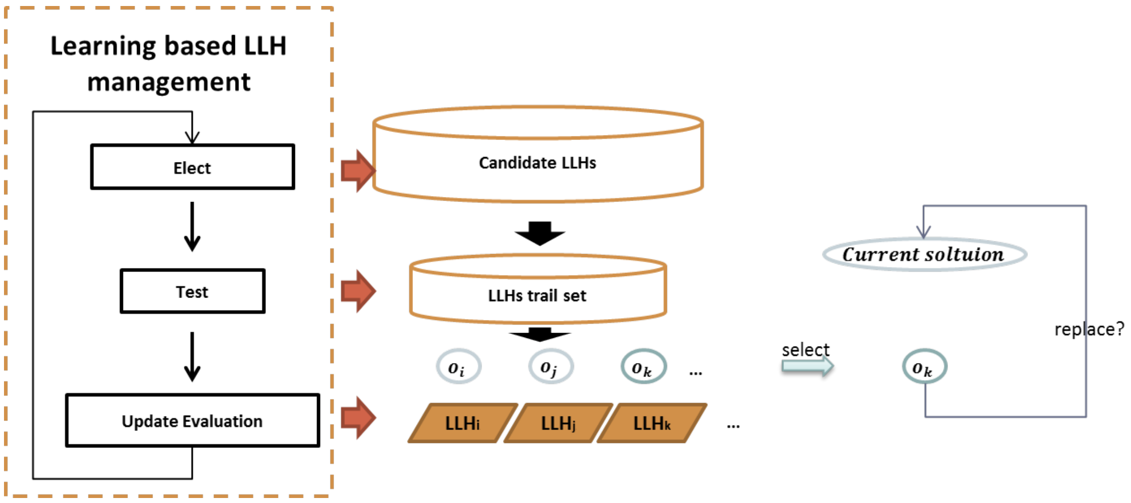


Figure 7.1: Overview of *LLH* management of learning based hyperheuristics. o_i represents the output (a neighbour solution) after apply LLH_i on the current solution.

In this study, the three following well-known learning-based hyperheuristics are tested: binary exponential back off (Remde et al. (2012)), reinforcement learning (Nareyek (2004)) and choice function (Cowling et al. (2001)). Our implementation applies the ranked choice selection and the *LLH* from the trial set that produces the most improved solution is applied. The algorithms here are not novel and we only summarise the difference in terms of the evaluation-adaptation mechanism between these methods. A more detailed description can be found in the papers cited below.

Binary exponential back off (BEBO)

BEBO (Remde et al. (2012)) is a tabu-search-based learning mechanism. A detailed pseudo-code can be found in Algorithm 6.2.2 (Chapter 6). It dynamically adapts tabu tenures, such as $tabu_i$, to control the usage of each *LLH*. If an *LLH* performs poorly in a recent search, it is disabled for a number of iterations. If this *LLH* continues to perform poorly when revisited, the number of forbidden iterations increases. This is controlled by a *backoff* parameter. Here, we summarise the $tabu_i$ adaptation rule as following:

. At each iteration, all LLH_i with $tabu_i = 0$ are selected to form the trial set.

. If LLH_i makes an improvement, $backoff_i$ = minimum value;

. Else $backoff_i = backoff_i * 2$; assign $tabu_i$ with randomly-picked value in $[0, backoff_i]$

. For all LLH_i with tabu tenure $tabu_i > 0$, $tabu_i = tabu_i - 1$

Reinforcement learning (RL)

RL (Nareyek (2004)) rewards (positive reinforcement) good *LLH* choices and punishes (negative reinforcement) bad *LLH* choices. Each *LLH* is given a utility value, which is repeatedly updated based on the *LLH*s performance. Nareyek (2004) tests various positive and negative hyperheuristic reinforcement methods and we apply the one, which is identified as the best, described as follows:

. Choose the $w\%$ of *LLHs*, which have the highest utility to be the trial set.

. For each LLH_i in the trial set, $utility_i = \sqrt{utility_i}$

. If the best performed LLH_i makes an improvement, $utility_i = utility_i^2 + 1$

Choice function (CF)

We consider the CF to be a different utility adaptation scheme (Cowling et al. (2001)). In each iteration, the utility of each LLH_i is updated based on a linear function that considers three factors: 1) the *LLH*'s performance is evaluated by the changes of solution quality and execution time; 2) the ability of the *LLH* in collaboration is evaluated by successively applied pairs of *LLHs*; 3) the CPU time elapsed since the *LLH* was last called. The w% of *LLHs* that have the highest utility are selected to form the trial set.

So far, we have introduced three ways of managing the candidate *LLH* set. In summary, BEBO emphasises that poorly-performing *LLHs* should not be tested in incremental iterations. It is specifically designed for use when a large set of *LLHs* is available. This tabu strategy quickly removes most of the unhelpful *LLHs*. In comparison, the reinforcement rules applied in RL emphasise more on the best-performing *LLH* and all other sub-optimal *LLHs*, in turn, have chances to be selected. Lastly, CF uses a weighted function to balance the usage of the best-performed *LLHs* and the waiting time of sub-optimal *LLHs*. For different problems, the tuning of weight parameters of CF is needed.

7.3.2.3 Variable neighbourhood descent with learning (VND_L)

VND (Mladenović and Hansen (1997); Hansen and Mladenović (2001); Hansen et al. (2010)) differs from the learning-based hyperheuristics in the use of intensive local search, meaning that the selected *LLH* is applied repeatedly until no more improvement is found, rather than applying it once only. VND is a parameter-free approach. *LLH* selection for standard VND uses a pre-ordered *LLH* set, with some researchers addressing VND sensitivity to the order of *LLHs* (e.g. Hemmelmayr et al. (2009)). In order to adapt the application order of *LLHs*, we propose a number of variations to the VND (Hansen et al. (2010)) combined with reinforcement learning techniques.

Algorithm 7.3.2 VND_L(x, *LLHs*)

```
1: Choose a subset $LLHs' \subseteq LLHs$ and order them in a list
2: Define: $k_{max} = |LLHs'|$, the number of heuristics to be tested.
3: Initialise $k = 1$
4: while $k <= k_{max}$ do
5:     $x' = \mathbf{ILS}(x, LLH^k)$, where ILS (iterated local search) applies the selected heuristic,
       $LLH^k$, repeatedly until no further improvement occurs; Within each iteration of ILS,
       $utility_k = \sqrt{utility_k}$; if $LLH^k$ makes an improvement, $utility_k = utility_k^2 + 1$
6:     If $x'$ is better than $x$ then $x = x'$ and $k = 1$, otherwise $k = k + 1$
7: end while
```

In the first step of our VND_L, line 1 in Algorithm 7.3.2, we select the top w% *LLHs* according to the utility value measured for each *LLH* in the candidate set. Then we order the selected *LLHs* using a predefined rule (i.e. random, ascending, or descending, as measured by utility).

In this section, we have introduced various designs of hyperheuristics. Each uses its own mechanism to adapt to the usage of *LLHs* in different search stages. In the following section, we describe the *LLHs* that are specifically designed for PVRP and the algorithm framework of hyperheuristic solvers for PVRP.

7.4 Solve PVRP using hyperheuristics

7.4.1 Solution representation

To solve a standard PVRP, two decisions should be made to construct a complete solution, including customers' assignment and constructed routes. We use a straightforward solution representation that consists of the two corresponding structures as follows:

1. $C = (c_\lambda^1, .., c_\lambda^i, ., c_\lambda^n)$ represents the assignment solution of each customer i to one of this customer's feasible service pattern $\lambda \in \Lambda_i$ (see PVRP description in Section 2.2.1).

2. $R = \{r_1^\vartheta, .., r_j^\vartheta, ...\}$ represents a set of feasible routes that service all customers' required visits with corresponding to the assignment solution, where r_j^ϑ denotes the j'th route in the solution set will be dispatched at ϑ'th day of the planning horizon.

7.4.2 Initialisation

Most researchers use a two-phase approach to derive a solution to a PVRP (Section 2.2.1), either through the process of assigning customers into days and solving the daily VRP, or by assigning customers to each vehicle and solving a PTSP. We use the first type of method as there is no strong relation between vehicles and customers in the standard PVRP problems. In the assignment stage, Cordeau et al. (1997); Cordeau and Maischberger (2012), Hemmelmayr et al. (2009) and Vidal et al. (2012) randomly choose a feasible customer-day pattern for each customer, whereas Chao et al. (1995) and Gulczynski et al. (2011) try to minimise the maximum amount of demand delivered in each day. Additionally, Christofides and Beasley (1984) attempt to minimise the total distance from the depot on each day. To construct routes for each day, the Clarke and Wright algorithm (CW) (Section 2.1.2.1) and GENI insertion heuristic (Gendreau et al. (1992)) are most common.

Our solution uses the same approach as Hemmelmayr et al. (2009): random assignment followed by a CW-routeconstruction process for each day. At the end of initialisation process, we have a complete feasible solution contains both of the assignment C and the route set R.

7.4.3 Low-level heuristics designed for PVRP

After constructing a complete PVRP solution, HyperILS (Section 7.3) is used to manage a given set of *LLHs* to improve the initial solution. *LLHs* are direct operators that work on the solution space. A good set of *LLHs* should be able to reach any solution in a solution space when used in different orders and combinations. Therefore, it is critical to build a good *LLH* repository. In this subsection, we first design three types of *moves* to modify a solution from either or both the daily routes changes and by reassigning customers visit patterns perspective. Then we introduce the *LLH* repository and the corresponding HyperILS structures applied in the experiment section.

Route structure modification Here, we summarise *moves* used to modify single (i.e. intra-route) and multiple routes (i.e. inter-route) in the PVRP literature (cited at the end of each *move* description). These *moves* are widely used in all variations of VRPs. Several books and tutorials describe these *moves* in detail (e.g. Toth and Vigo (2002); Groër et al. (2010)). Many are also described in Section 2.1.2.2. Therefore, we are not going to present the steps of the modification process of each *move* here.

λ-Opt: remove λ [1] edges from a route and replace them with λ new edges (e.g. Lin (1965); Chao et al. (1995); Hemmelmayr et al. (2009); Gulczynski et al. (2011); Vidal et al. (2012)).

Or-opt: remove a string with two to four nodes and insert it into a new position, either in the same route (e.g. Alegre et al. (2007)) or in a different route (e.g. Vidal et al. (2012)).

One point move (1PM): relocate a node to a new position, either in the same route or in a different route (e.g. Chao et al. (1995); Gulczynski et al. (2011)).

Two points swap (2PS): swap two points, either in the same route or between different routes (e.g. Gulczynski et al. (2011)).

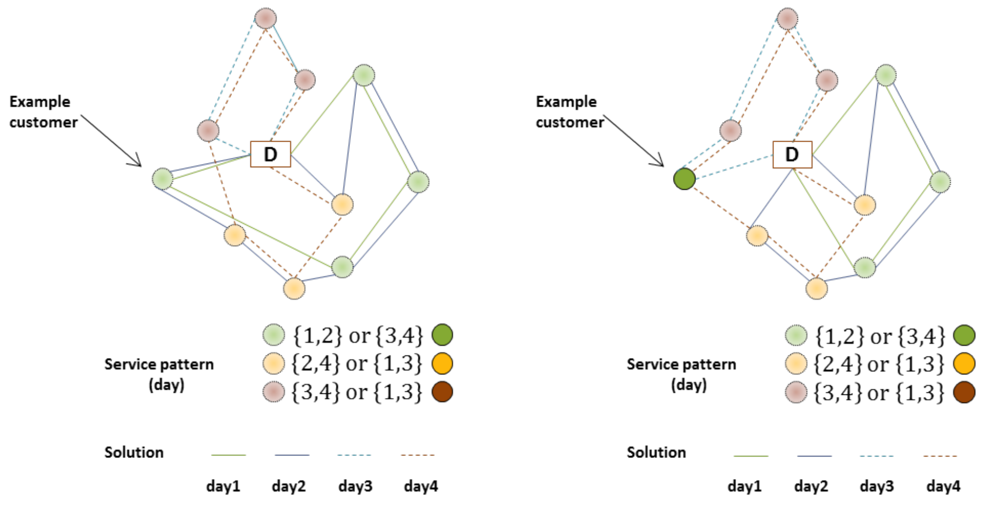
[1] λ-Opt has no relationship to customer service pattern λ above. The overloading is unfortunate, but it is retained as both the uses of λ are conventional here.

k-Relocate: relocate a string of k points to a new position, either in the same route or a different route (e.g. Hemmelmayr et al. (2009); Vidal et al. (2012)). When there is only one node in the string, this *move* is identical to "1PM".

k-Cross: swap two chains of points between two routes. Each chain contains maximum k points (Alegre et al. (2007); Hemmelmayr et al. (2009)).

Our route-based *moves* only modify the partial solution R, after customer visit patterns are assigned. In addition, only if $\vartheta_i = \vartheta_j$ for routes i, j, an inter-route move will be applied.

Service pattern modification Each customer i in a PVRP has a set of valid service patterns Λ_i. We assume a customer i is currently assigned to a pattern λ. Then, a pattern modification chooses a different service pattern $\lambda' \in \Lambda_i$ at a time and reassigns the new pattern to i. Consequently, all visits to customer i in the routes of the current solution are removed, and new visits are inserted with corresponding to the new pattern λ'. For example in Figure 7.2, suppose a customer requires two visits in a four-day period and there are two allowable patterns, pattern-1 (day 1 and day 2) or pattern-2 (day 3 and day 4). The current solution visits the customer in one route from day 1 and one route from day 2. When a pattern *move* operates on this customer, we remove both visits to this customer from day 1 and day 2 and re-insert the customer into the routes from day 3 and day 4.



(a) Current solution (b) After a service pattern move

Figure 7.2: An example of service pattern modification

As described above, our pattern-related *moves* operate on both of the assignment solution part C and the route structure solution part R. After a pattern-related *move* is applied, we always get a complete PVRP solution. We insert each of the new visit at the cheapest feasible position of all routes available in same day.

A random or random permutation reassignment strategy is often used to select a different pattern from the available set (e.g. Hemmelmayr et al. (2009); Vidal et al. (2012)). Another branch of solvers use constraint reassignment to partially examine the available service pattern set. A few integer programming models (e.g. Gulczynski et al. (2011); Crainic et al. (2012)) are proposed to guide the reassignment *moves*.

Here, we design four *moves* to modify customer service patterns.

Random pattern reassign (Pa_RR): Randomly assign new feasible visit patterns to x random customers. x is a user defined parameter. A tabu structure is used to prevent repeated reassignment of the same customer. For example, if the service pattern of customer i is modified, we set an integer tabu value $tabu_i$, which prevents selection of this customer for the Pa_RR *move* for $tabu_i$ iterations.

Score-based reassign (Pa_SR): This is similar to Pa_RR except that the pattern with the highest score $Q(\lambda)$ is chosen for each customer i in x random selected customers. A pattern score $Q(\lambda)$ is associated with each pattern λ of each customer i. It is negatively reinforced every time a customer's pattern is tested and no improvement is found ($Q(\lambda) = \sqrt{Q(\lambda)}$); and gets rewarded if an improvement is found ($Q(\lambda) = Q(\lambda)^2 + 1$).

Pattern reassign first improvement (Pa_FIR): If a customer has an available visit pattern λ, which produces a better solution after remove the customer's current visits and insert each visit in λ using the cheapest insertion, we move this customer.

Two points pattern swap (Pa_2SW): Swaps the visit patterns of two customers i and j, if they have the same visit pattern set and they are not assigned to the same pattern in the current solution ($\Lambda_i = \Lambda_j$ and $c^i_\lambda \neq c^j_\lambda$).

In comparison to the route-based *moves*, the pattern-related *moves* make larger modifications of the solution structure. Consequently, these moves usually take longer computational time and we expect that larger fitness changes will be observed at each utilisation.

Mixed modification In many PVRP instances, there is a large number of "single requirement" customers that have a visit frequency of 1 during the period. This type of customers can be easily moved between days as there is no other affected day within one service pattern. It is helpful to build moves that allow route structures to change between days and to modify the customer service pattern at the same time. Different from service pattern modification, mixed modifications operate on a large and continuous section of routes rather than on individual customers. In our *moves* set, we design two new mixed operators. These operators only consider a chain that includes customers with the same available visit patterns and a visit frequency of 1.

Relocate with Pattern (MRPa): Relocate a string of points from one route to another route in a different day.

Cross with Pattern (MCPa): Swap two chains of points between two routes from different days.

7.4.4 Partially destructive/constructive moves for re-initialisation

The idea of creating a *move*, by partially destroying and reconstructing a current solution, is widely seen in the re-initialisation process of heuristic search. We name this type of *move* as destroy-construct move (DCM). In PVRP literature (e.g. Chao et al. (1995); Cordeau et al. (1997); Gulczynski et al. (2011); Chen et al. (2016d)), a DCM randomly removes up to n customers and then reinserts them into the solution whilst respecting their available valid service patterns.

In PVRP *moves* classification, DCM are akin to service-pattern-modification *moves*, in theory. However, the key difference compared to a local search *move* is the degree of similarity between the solutions before and after a *move* is applied. In the wider literature of heuristic solvers for VRPs, DCM has been successfully integrated into a local search framework, known as *Large Neighbourhood Search* (LNS) (Shaw (1998)). More recently, Ropke and Pisinger (2005) extend the remove/reinsert idea by managing a set of customer removal/reinsertion algorithms instead of only having one removal and one insertion method. The different combinations of removal/reinsertion algorithms enlarge the variety of neighbours of the current solution. In other words, it increases the neighbourhood size of the current solution.

For the PVRP, we design the DCM described in Algorithm 7.4.1, where x_{best} is the best-found solution so far and p_{random} represents the probability of randomly generating a new initial solution. If the current solution has not been improved using *HyperILS* for a certain number of iterations, we assume that the search is stuck in a local optimum from which it cannot escape by a small perturbation. Then a re-initialisation mechanism is applied. The new solution is made feasible by repeatedly removing the customer with the greatest load requirement from any route in the candidate solution, which violates duration or load constraints, and re-inserting it in a route where the constraints are met.

Algorithm 7.4.1 Re-initialisation(x_{best}, p_{random})

```
if random(seed) < $p_{random}$ then
    random assignment and CW daily routes construction (Section 7.4.2).
else
    Destroy $w\%$ of the longest routes in $x_{best}$.
    For each customer in destroyed routes, randomly reassign feasible visit pattern.
    Insert each customer greedily to cheapest position in each day of assigned pattern.
end if
Return the new (re)constructed solution $x$.
```

7.4.5 Summary of *LLH* repository for PVRP

To form our *LLH* repository for PVRP, the hyperheuristic applies the above *moves* as either mutation or first improvement search. As presented in Table 7.1, in the mutation group, we randomly apply a defined *move* to the current solution, whilst in the FI search, we go through each node (or edge) to test the defined *move* until an improvement is found. All of our *LLHs* maintain a feasible solution. If the low-level heuristic cannot make a legal move then the solution is not modified. Route-related *moves* are parametrised by route ID, day, length of chain, and number of nodes changed in one move. This makes it possible for an intelligent hyperheuristic to select an *LLH* specifically related to each sub-problem (e.g. daily VRP or single route optimisation). Pattern-related *moves* reassign the patterns of x customers. We consider $x = 1, 2, ..., 6$ in our experiments. Because of the structure of the *LLH* parameter design, our *LLH* repository contains 70-110 *LLHs*, depending on the problem instance.

Table 7.1: *LLH* Repository used in our PVRP hyperheuristics

Type	Moves	Set ID
Route related: mutation	2 points swap (2PS), Relocate, Cross	Mu_r
Route related: FI	2Opt, 3Opt, 2PS, Relocate,Cross	FI_r
Pattern related: mutation	Random Pattern Reassign (Pa_RR), Score based reassign (Pa_SR), Two points pattern swap (Pa_2SW)	Mu_p
Pattern related: FI	Pattern reassign first improvement (Pa_FIR), Two points pattern swap (Pa_2SW)	FI_p
Mixed: FI	Relocate with Pattern (MRPa), Cross with Pattern (MCPa)	FI_m

7.4.6 Algorithm frameworks

Figure 7.3 illustrates the structure of solvers for PVRP. For the *HyperILS* process, ten algorithms are investigated with different *LLH* set usage and *LLH* selection strategies used at each stage during the search. The differences are summarised in Table 7.2. According to the different *LLH* set management strategies in *HyperILS*, we further categorise these algorithms into three frameworks. In Figure 7.4, framework 1 (FW1) organises all types of *LLH* together, whereas framework 2 (FW2) clearly distinguish the types of *LLH* used in *HyperPerturbation* and improvement stages. In framework 3 (FW3), we embed an iterative local search process within the hyper improvement state.

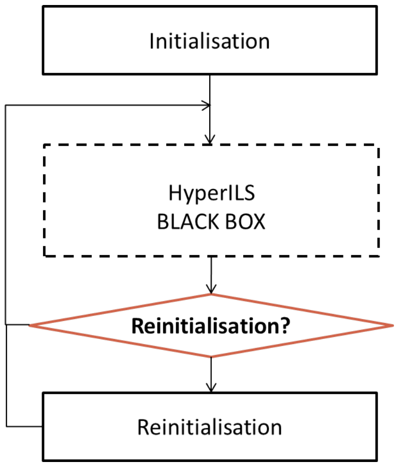
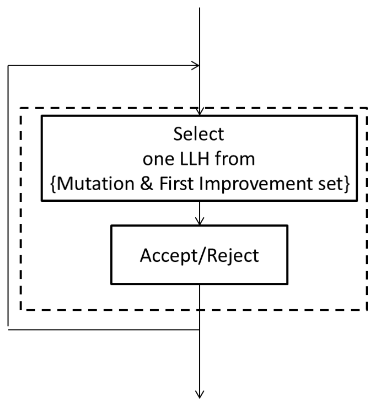


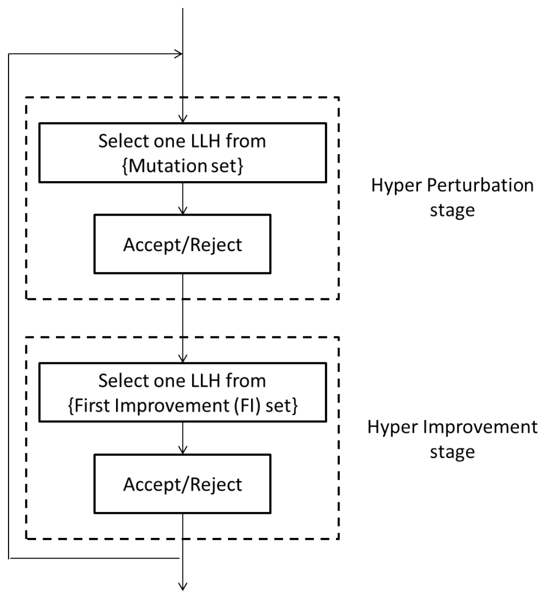
Figure 7.3: Overview of PVRP solvers

Table 7.2: Summary of *HyperILS* implemented. Each HyperILS is embedded with different components at *HyperPerturbation* and *HyperImprovement* stages. Keys for the table: Simple Random (SR); Random Permutation (RP); Reinforcement Learning (RL); Choice Function (CF); Binary Exponential Back Off (BEBO); VND with learning (VND_L). The ID for *LLH* set is shown in Table 7.1 and the framework is shown in Figure 7.4

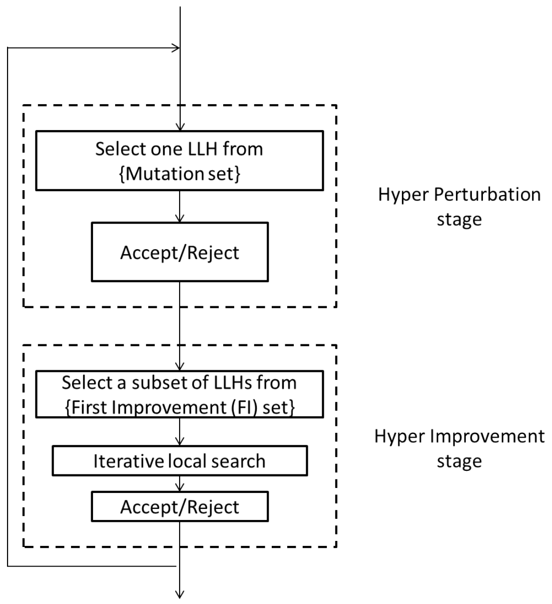
HyperILS	HyperPerturbation		HyperImprovement		Framework
	LLH set	Hyperheuristic	*LLH* set	Hyperheuristic	
SROI	-	-	ALL	SR	FW1
RDOI	-	-	ALL	SR	FW1
RLFW1	-	-	ALL	RL	FW1
RLFW2	Mu_r;Mu_p	SR	$FI_r; FI_p; FI_m$	RL	FW2
CFFW1	-	-	ALL	CF	FW1
CFFW2	$Mu_r; Mu_p$	SR	$FI_r; FI_p; FI_m$	CF	FW2
BeboFW1	-	-	ALL	BEBO	FW1
BeboFW2	$Mu_r; Mu_p$	SR	$FI_r; FI_p; FI_m$	BEBO	FW2
VNS_RL	$Mu_r; Mu_p$	RP	$FI_r; FI_p; FI_m$	VND_L	FW3
VNSr_RL	$Mu_r; Mu_p$	SR	$FI_r; FI_p; FI_m$	VND_L	FW3



(a) Framework 1



(b) Framework 2



(c) Framework 3

Figure 7.4: Framework variations of HyperILS (modified from Özcan et al. (2008))

7.5 Experiments and analysis

We design experiments that allow us to analyse the performance of hyperheuristics from different angles. We use data (benchmark and real) with different spatial characteristics and also compare the performance of hyperheuristics with that of meta-heuristics applied to the benchmark problems.

The experiments are designed to replicate benchmark conditions from Vidal et al. (2012). In particular, the search is always terminated after the fixed amount of CPU time as stated in Vidal et al. (2012). To check the suitability of this time limit for scalability experiments (Section 7.5.4), we run preliminary experiments using twice the CPU time. We find no significant change in the quality of solutions, suggesting that a performance plateau is attained, and the chosen CPU time is appropriate.

All experiments are implemented in C$^\sharp$ and are executed on a cluster composed of eight Windows computers, each with Intel Xeon E3-1230 CPU.

7.5.1 Problem instance

Our data comprises 42 benchmark instances and six instances from a real-world periodic maintenance problem [2]. The detailed information of the data sets can be found in Table A.2 (Appendix A). The 42 benchmark instances are normally organised as a "old data" and a "new data" set by many research works. The "old data" set contains 32 instances collected from early works on PVRP (including works by Christofides and Beasley (1984); Russell and Igo (1979); Russell and Gribbin (1991); Chao et al. (1995)). Cordeau et al. (1997) presented 10 additional PVRP benchmark instances, known as the "new data" set.

Here, we classify the problem instances according to their spatial characteristics (Figure 7.5). Table 7.3 summarises each class. The six real-world instances are all street type. The big random benchmark problems have both a larger number of customers and a greater clustering of data points than the small random class. Our results are compared against the best-found solutions in literature, shown in bold in Table A.3 (Appendix A).

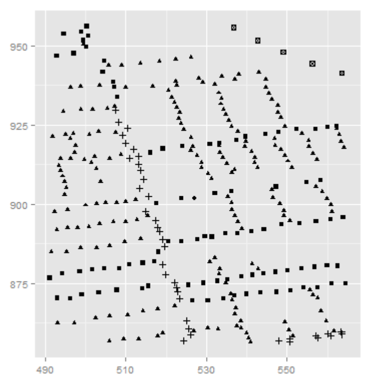
Table 7.3: Information on PVRP instances. n is the number of customers; m is the number of vehicles available; t is the length of planning period in days. Benchmark labelling is taken from Hemmelmayr et al. (2009).

Class	**n**	**m**	**t**	**Average visit frequency (days)**	**Number of problem instances**
Street style	240-324	3-5	6	1.6-2.1	6
Small random	50-100	1-6	2-10	1-2.1	10 (benchmark p01-p10)
Big random	48-417	2-12	4-7	1.1-3	13 (benchmark p11-p13, pr01-pr10)
Symmetrical	20-184	2-9	4-6	1.8-2	19 (benchmark p14-p32)

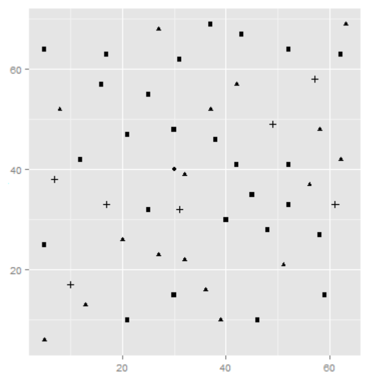
7.5.2 Random vs learning based selection strategies

A hyperheuristic needs an efficient *LLH* selection strategy because it is impractical to test all *LLHs* exhaustively. The first experiment compares the SR selection strategy to the learning-based strategies, which are RL, CF and BEBO. The experiment uses framework 1 (Figure 7.4(a)), where all *LLHs* are managed in one candidate set. For RL1 and CF1, we test using both the best 30% and the best 80% of *LLHs* in each iteration (See Section 7.3.2.2).

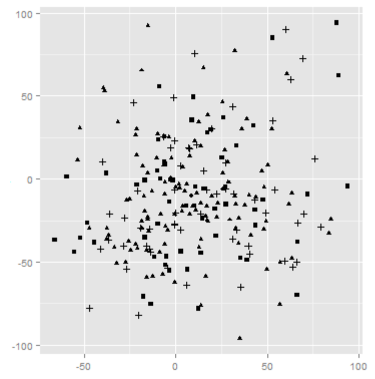
[2]The real-world data and associated best-performance results can be found at `http://yc1005.wixsite.com/yujiechen`



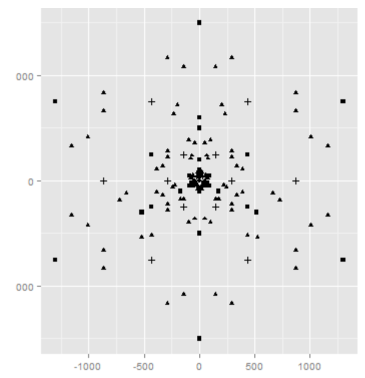
(a) street style example



(b) Small random (from benchmark p02) Christofides and Beasley (1984))



(c) Big random (from benchmark pr05 Cordeau et al. (1997))



(d) Symmetrical problem (From benchmark p32 Chao et al. (1995))

Figure 7.5: Examples of four types of spatial distributions in the PVRP instance set.

Each algorithm runs 20 times on each instance of each of the four classes of problems to give the percentage differences to the best-found benchmark route length of each instance. We then average the results for each class of problems.

The results in Table 7.4 show that, whilst acceptable, none of our solutions match the best-found benchmark solution. Learning-based selection strategies consistently outperform SR *LLH* selection, with BEBO offering the best performance. For both RL and CF, the limited CPU time makes it difficult for the hyperheuristics to produce competitive results for testing 80% of *LLHs* in each iteration. In subsequent experiments we only use the best 30% of *LLHs*.

7.5.3 Impact of algorithm framework

Having shown that learning based selection strategies can manage a large number of *LLHs* in a simple hyperheuristic framework, we now consider the different hyperheuristic frameworks.

Table 7.4: The average percentage difference to the best-found solution over all instances in each group, for simple random (SR) and learning based hyperheuristics, using framework 1 and OI acceptance. Bold numbers represent the best result of each type of the problems.

	SROI	**RLFW1(30%)**	**CFFW1(30%)**	**RLFW1(80%)**	**CFFW1(80%)**	**BeboFW1**
Street style	+8.90	+5.36	+6.08	+6.47	+6.64	**+4.90**
Small Random	+4.67	+2.12	+2.28	+2.15	+2.33	**+2.06**
Big Random	+4.32	+4.14	+4.32	+4.28	+4.35	**+4.13**
Symmetrical	+2.74	**+1.46**	+1.56	+1.55	+1.53	+1.50

In framework 1 (FW1), the only improvement (OI) acceptance rule means that mutation *LLHs* are unlikely to be favoured. In framework 2 (FW2), a mutation operator is randomly selected and applied as long as it generates valid solutions. Thereafter, FI *LLHs* are selected using a learning-based strategy, as shown above. FW2 is similar to FW1 when we use the all-move-accept rule. However, whereas FW1 evaluates the mutation and FI operators together, FW2 allows a separate consideration.

The results in Table 7.5 show that FW2 improves the performance of both RL and CF hyperheuristics for all types of problem instances. However, BEBO does not show any consistent difference between frameworks 1 and 2.

Table 7.5: The average percentage differences to the best found solutions over all instances in each group, for learning based hyperheuristics using frameworks 1 and 2 (FW1, FW2)

Instances	**RL(30%)**		**CF(30%)**		**BEBO**	
	FW1	FW2	FW1	FW2	FW1	FW2
Street style	+5.36	**+5.26**	+6.08	**+5.54**	**+4.90**	+5.31
Small random	+2.19	**+1.88**	+2.28	**+1.90**	+2.06	**+1.65**
Big random	+4.14	**+3.93**	+4.32	**+3.88**	+4.13	**+3.93**
Symmetrical	+1.46	**+1.45**	+1.56	**+1.54**	**+1.50**	+1.56

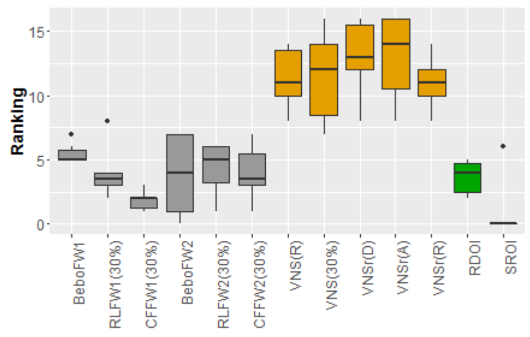
Framework 3 (FW3) describes the algorithms using VND_L hyperheuristic in the *HyperImprovement* stage. The main difference to FW2 lies in the use of *ILS* once a FI *LLH* is selected. Five variations are investigated and the details are described in Table 7.6. The name "VNS_RL" is given because of the similarity between VNS_RL and the standard VNS (Hansen et al. (2010)).

Table 7.6: Variations of VND_L. (The HyperILS labels are described in Table 7.2)

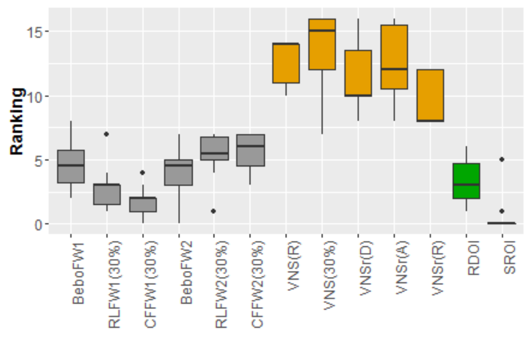
Name	HyperILS	Description of the first step of VND_L
VNS(R)	VNS_RL	Select all *LLH* in the FI-*LLH*-set; Randomly order them
VNS(30%)	VNS_RL	Select top 30% *LLH* in the FI-*LLH*-set ranked by their utility value;Randomly order them
VNSr(R)	VNSr_RL	Select all *LLH* in the FI-*LLH*-set; Randomly order them
VNSr(A)	VNSr_RL	Select all *LLH* in the FI-*LLH*-set; Ascending ordering determined using utility
VNSr(D)	VNSr_RL	Select all *LLH* in the FI-*LLH*-set; Decending ordering determined using utility

We compare performance with the hyperheuristics introduced in Table 7.2. We then rank the performance, awarding 16 points to the best performing hyperheuristic, and then 14, 12, 10, 8, 7, . . . 1, 0 points successively to the worse performing hyperheuristics.

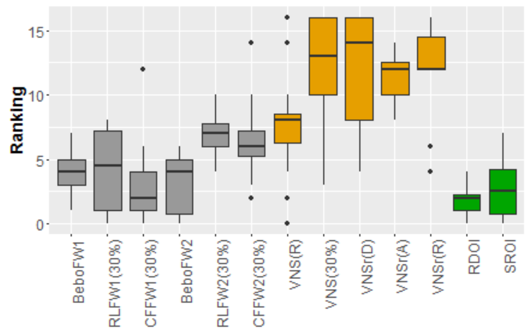
Figure 7.6 shows a small difference in performance between FW1 and FW2. Compared to the FW3 results, they are both generally low-ranking for all cases except the symmetrical benchmark problems. This shows the positive impact of using *ILS*. Among FW3, the five VND_L variations show similar ranking. Random selection of the mutational *LLHs* combined with random-ordered FI *LLHs* (VNSr(R)) is the most robust over all classes of problems, whilst other algorithms in this group perform badly for the symmetrical instances.



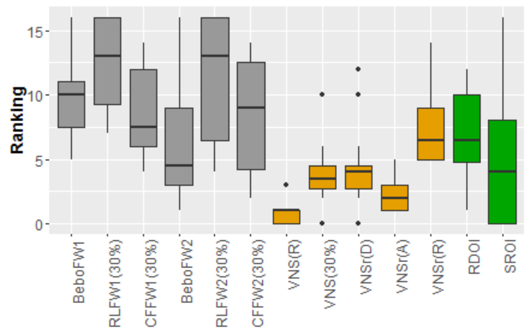
(a) Street style



(b) Small random



(c) big random

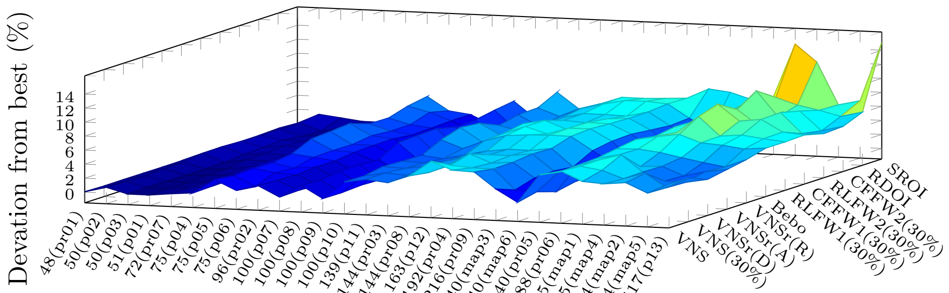


(d) Symmetrical

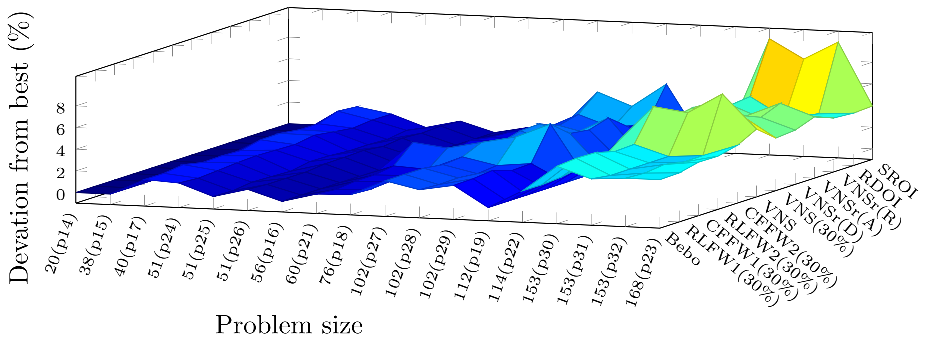
Figure 7.6: Hyperheuristics performance for different types of PVRP. Higher score is better. All tested algorithms stop at the same predefined CPU time.

7.5.4 Scalability

The PVRP is NP-hard (Vidal et al. (2012)). One of its biggest challenges is the rate of growth in complexity with problem size. In preliminary experiments, we determined that the performance of algorithms on symmetrical and non-symmetrical problems is very different. To test the scalability of our hyperheuristics, we first group the problem instances into symmetrical and non-symmetrical problems and then order them by the number of customers. Each method runs 20 times.



(a) Non-symmetric problems



(b) Symmetric problems

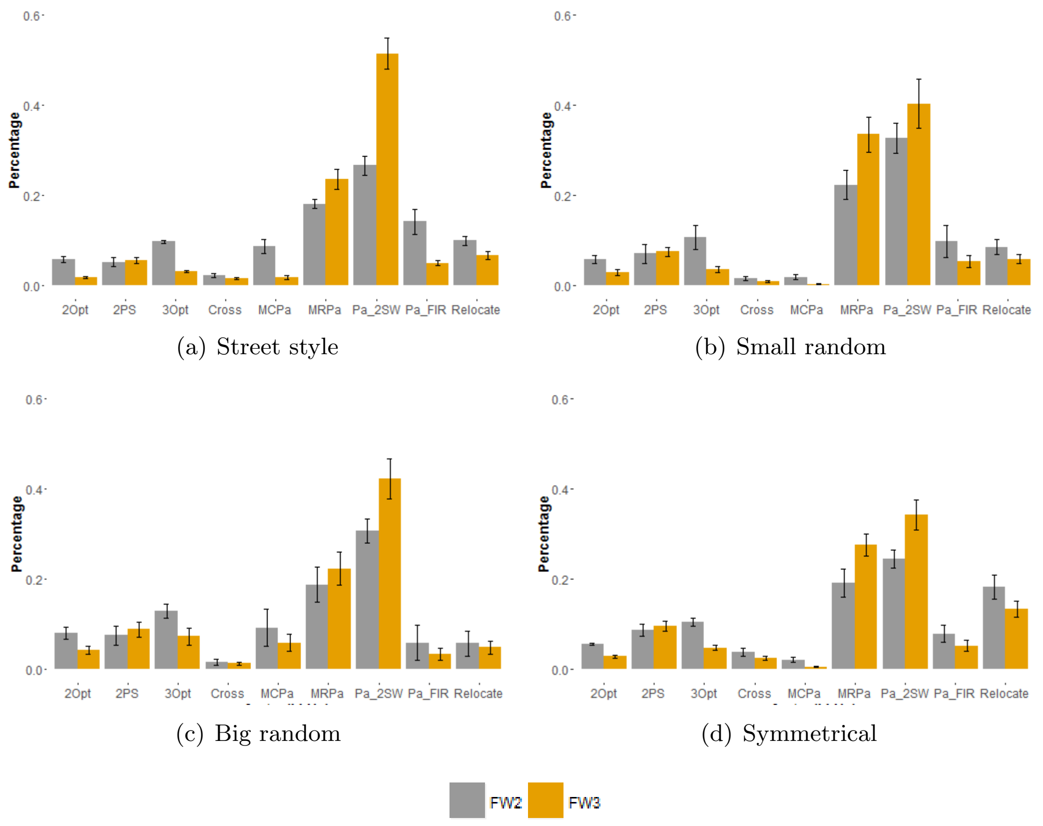
Figure 7.7: Performance of hyperheuristics tested on PVRP with various sizes

Figure 7.7 shows that SROI (Table 7.2) has the worst scalability in both symmetrical and non-symmetrical problems. For the other algorithms, there is little difference in performance for problems with less than 60 customers. Algorithms from FW3 are the most robust across non-symmetrical instances with 150-420 customers. However, performance decreases dramatically for these algorithms when applied to bigger problem instances in the symmetrical data set.

7.5.5 *LLH* usage analysis

Whilst hyperheuristics need little specialised design, the *LLH* repository needs to be built based on problem domain knowledge (e.g. solution structure). In this experiment, we explore the usage of *LLHs* by different hyperheuristics. We use frameworks 2 and 3, which manage the mutation and FI *LLHs* separately. The results focus on the 9 FI *LLHs* since there is no learning in mutational *LLHs* selection.

Figure 7.8 summarises average usage of FI *LLHs* for all learning-based algorithms using FW2 (BeboFW2, RLFW2(30%), CFFW2(30%)) and all VNS-related algorithms using framework 3 (VNS(R), VNS(30%), VNSr(R), VNSr(A), VNDr(D)). First, we can see that"Relocate with pattern" (MRPa) and "two points pattern swap" (Pa_2SW) are the



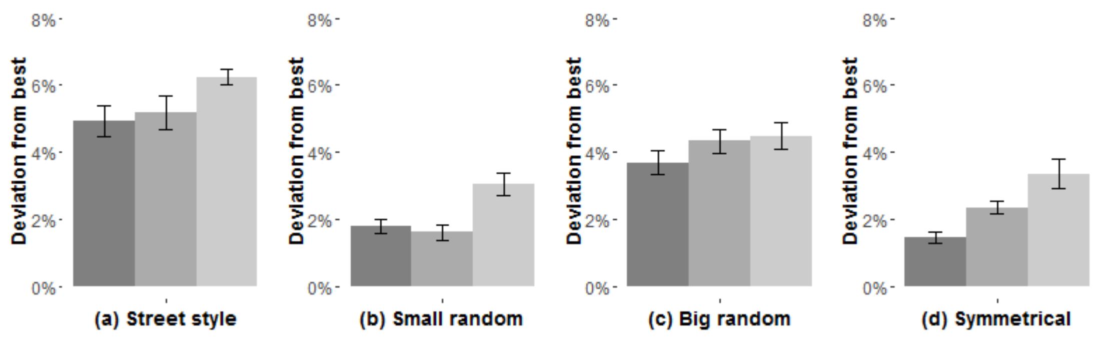
(a) Street style (b) Small random (c) Big random (d) Symmetrical

Figure 7.8: FI *LLHs* usage for different types of problems. Results show the mean value of the percentages of each *LLH* are applied during the search, where 0.1 stands for 10%. Error bars show 95% confidence interval.

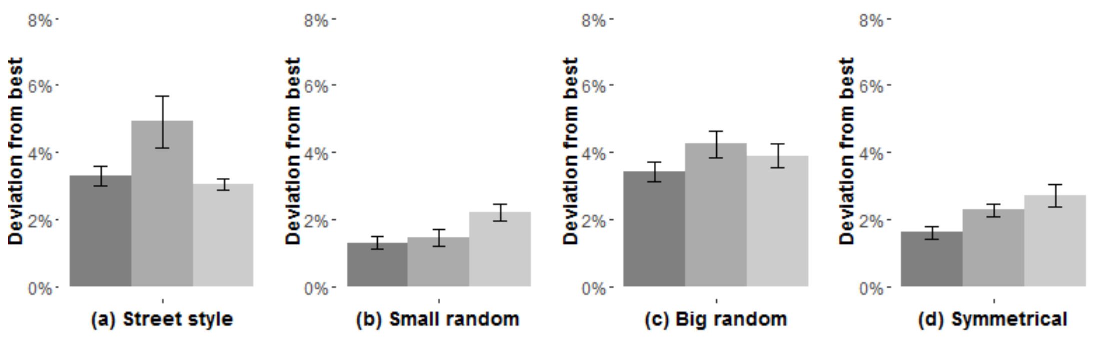
most applied *LLHs* by all hyperheuristics. Since we are using an OI strategy, this implies that they consistently produce improved solutions. Second, algorithms using framework 3 have a stronger usage bias between favoured and unfavoured *LLHs* than framework 2. We think this is because, once a favoured *LLH* is selected in FW3, *ILS* enlarges the usage difference compared to FW2.

To further explore the contribution of specific *LLHs* in improving PVRP solutions, we test the two best-performing hyperheuristics for frameworks 2 and 3 (RLFW2(30%) and VNSr(R)) with different subsets of the original *LLHs*. Each method is run over all problem instances and the results are the average of 20 runs.

Figure 7.9 shows a change in performance after removal of the most-used FI *LLH* (Pa_2SW) and all mutation operators except Pa_RR (Subset 1): the performance of RLFW2(30%) and VNSr(R) decreases dramatically for the "small random" and "symmetrical" problems. However, there is little difference for the big random instances, and we even find a small improvement for VNSr(R) on street style problems. One interpretation of this result is that the strongly-performing FI *LLHs*, which are most effective in small and symmetric prob-



(a) RLFW2(30%)



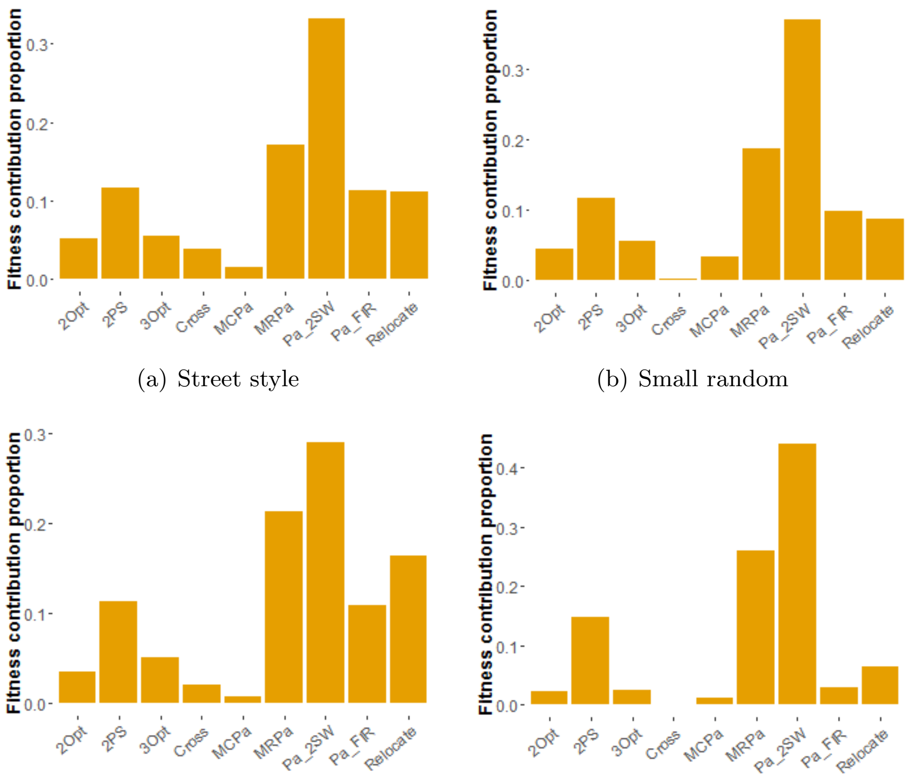
(b) VNSr(R)

Figure 7.9: Performance of RLFW2(30%) and VNSr(R) using different subset of *LLHs*. Error bars show 95% confident interval. The subset1 removes the most used *LLH* (Pa_2SW) and all mutational operators except Pa_RR

lems, tend to become stuck in a local optima in the street style and big random problems.

From Figure 7.8, the Relocate *LLH* is preferred in symmetrical problems but not in others. The importance of this *LLH* is emphasised by the big reduction in performance of solving symmetrical instances when the Relocate *LLH* is removed from the *LLH*-set of RLFW2(30%)(Figure 7.9(a)). No significant differences are seen in the other there types of problem instances.

Interestingly, when we remove the "Relocate" *LLH* from *LLH*-set of VNSr(R), the impact on street style and big random problems is unexpectedly strong and the performance decrease is significantly large (Figure 7.9(b)). Looking at the proportion of fitness value improved by each *LLH* for each type of problem (Figure 7.10), we find that the "Relocate" *LLH* contributes over 10% of the fitness improvement for both street style and big random problems, even with usage at about 5% during the search in both cases. In comparison to symmetrical problem, the "Relocate" seems to make small contributions to fitness improvement through a lot of utilisation. This result suggests that some *LLHs* are rarely applied but produce big improvements, whilst some *LLHs* produce smaller improvements frequently. The RL selection strategy applied in our hyperheuristics only considers whether an *LLH* produces an improvement and misses the information of fitness change. This decision-making strategy may misjudge the "strong" and "weak" *LLHs*. However, there is no evidence showing that the hyperheuristics, which use fitness changes (e.g. CF), perform better. The challenge here is to use this highly problem-related information at a hyper level, which works universally well.



(a) Street style

(b) Small random

(c) Big random

(d) Symmetrical

Figure 7.10: Proportion of fitness value improved by each FI *LLH* for different types of problems, using VNSr(R). Results show the mean value over problem instances in each problem group, where 0.1 stands for 10%.

To explore the robustness of different hyperheuristics when we remove the strongest *LLH* (Pa_2SW), we extend the *LLH* subset experiments to SROI and RDOI. VNSr(R) shows the best robustness (Figure 7.11). Comparing the RLFW2(30%) with VNSr(R) and the SROI with RDOI, the algorithms with *ILS* mechanisms are more robust than the algorithms without *ILS*.

7.5.6 Comparison between hyperheuristics and other meta-heuristics

This section compares the two best-performing hyperheuristics from framework 2 and 3 (RLFW2(30%) and VNSr(R)), with the published meta-heuristics, which have been

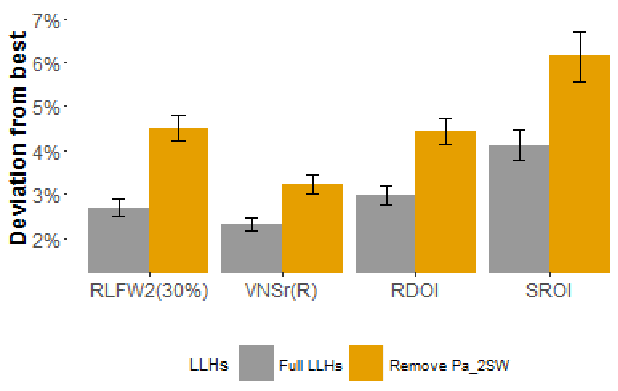


Figure 7.11: Impact of removing Pa_2SW on four hyperheuristics over all problem instances. Error bars show 95% confidence intervals.

designed or tailored for PVRP, including (parallel) tabu search (Cordeau et al. (1997); Cordeau and Maischberger (2012)), scatter search (Alegre et al. (2007)), VNS (Hemmelmayr et al. (2009)), record-to-record ILP (Gulczynski et al. (2011)) and hybrid Genetic Algorithm (GA) (Vidal et al. (2012)).

42 benchmark instances are tested in this experiment. We group our results based on the "old data" and "new data" set classification (Section 7.5.1). Because some research works have not tested both groups. No comparative data exists for our street style data set. Table 7.7 reports the percentage difference in average performance from the best-found solutions (summarised in Vidal et al. (2012)) over these two data sets.

Our hyperheuristics achieve competitive results compared to the tabu search (Cordeau et al. (1997)), scatter search (Alegre et al. (2007)) and VNS (Hemmelmayr et al. (2009)) for the "old data" set. For the larger "new data" set, we achieve close to the best-found solutions in most cases. The hyperheuristic approaches are about 1% worse than these problem-specific algorithms, in terms of total route distance.

Compared to the hybrid-GA (Vidal et al. (2012)), which outperforms all the other algorithms from literature, our hyperheuristics produce routes that are about 2% longer on an average. However, hyperheuristics do not require any direct knowledge of the solution space and require minimal design effort, whereas the meta-heuristics need to be designed for and tailored to each problem.

7.6 Conclusion

The overall goal of this chapter was to investigate hyperheuristics which operate at a higher level of generality than most of the current approaches studied in PVRP. Hyperheuristics aims to build cheap and general algorithms to solve various optimisation problems. The

Table 7.7: Performance on PVRP benchmarks compared with meta-heuristics; tabu search (CGL) (Cordeau et al. (1997)), scatter search (ALP) (Alegre et al. (2007)), VNS (HDH) (Hemmelmayr et al. (2009)), record-to-record ILP (GGW) (Gulczynski et al. (2011)), hybrid-GA (VCGLR) (Vidal et al. (2012)), parallel tabu search (CM) (Cordeau and Maischberger (2012))

	RLFW2(30%)		**VNSr(R)**		**CGL**	**ALP**	**HDH**	**GGW**	**VCGLR**	**CM**
	avg. 20run	avg. (best)	avg. 20run	avg. (best)	1 run	-	avg. 10run	-	avg. 10run	avg. 10run
old data (%)	1.86	1.08	1.77	0.93	1.8	1.57	1.6	1.11	0.032	0.044
new data (%)	3.88	2.40	3.44	2.12	2.82	-	1.86	-	0.071	0.091

PVRP solvers should consider two aspects of optimisation tasks: customer pattern assignment and route optimisation. We would like to test whether a hyperheuristic method is able to choose an appropriate *LLH* at the right time and to solve the PVRP.

According to our experiments, a few tested hyperheuristics find solutions, which are almost as good as those published for meta-heuristics. Since all experiments require limited CPU time, it is possible that this is due to the hyperheuristics' additional overhead in applying search at the *LLH*-selection level. The hyperheuristics are more adaptable to new problems. The competitive results give an evidence that hyperheuristics can efficiently manage a large *LLH* set and automatically select appropriate *LLHs*.

Different hyperheuristics perform differently. Our analysis of hyperheuristics for PVRP shows that both learning-based selection strategy and *ILS* have positive impacts on an algorithm's performance and enhance its scalability. *ILS* also improves the robustness of hyperheuristics when a poor *LLH* set is given because *ILS* concentrates on a neighbourhood structure until it reaches a local optimum, whereas approaches without *ILS* have a wider but shallower exploration within the search space.

The tested hyperheuristics show similar performance on real-world street style problem instances and random instances. However, the symmetrical benchmarks tend to favour different strategies and *LLHs*. This suggests that symmetrical instances are not a good indicator of algorithm performance for real-world PVRP.

Our *LLH* repository of PVRP consists of six well known route-structure operators, a random-pattern-reassign operator, three new pattern-reassign operators and two new mixed operators. Among them, two of the new operators, "Relocate with pattern" and "two points pattern swap" are the most applied *LLHs* across all tested hyperheuristics: these *LLHs* make the most number of times of improvements during the search. In other words, these *LLH* are considered as "strong" choices by a hyperheuristic. However, experiments on *LLH* subsets show that a "strong" *LLH* may lead to premature local optima. Also, the

LLH that is mostly used in our hyperheuristics may not be the *LLH* that makes the most contribution of solution quality improvement. Further work is needed on the effect of ways to measure “strong” *LLHs*.

In the next chapter, we adopt the concept of learning mechanism to a local search procedure. A very different design of credit assignment and selection of neighbourhood (firstly introduced in Section 7.3.2.2) is proposed. We compare the best-performed hyperheuristic (VNSr(R) tested on PVRP) with the new algorithm developed in the next chapter and deliver the analysis based upon the test on both PVRP and GDMP.

Chapter 8

A dynamic multi-arm bandit neighbourhood search

In Chapter 7, we discussed the effect of learning mechanism and algorithm frameworks on hyperheuristic design. In this chapter, we treat the selection of a neighbourhood as a dynamic multi-armed bandit (D-MAB) problem where the learning techniques for solving a D-MAB can be used to guide the local search process. We present a D-MAB neighbourhood search (D-MABNS), which can be embedded within any meta-heuristic framework. Given a set of neighbourhoods, the aim of D-MABNS is to adapt the search sequence, testing promising solutions first. The key difference between the D-MABNS and hyperheuristics is the usage of domain knowledge for neighbourhood management. We demonstrate the effectiveness of D-MABNS on both of GDMP instances (Chapter 4) and the benchmark instances of periodic vehicle routing problem (PVRP) (Section 2.2.1). The content in this chapter is also presented in Chen et al. (2016a) and has been submitted to the Journal of Heuristics.

8.1 Introduction

Local search based meta-heuristics have been a very popular research area in the last 20 years and very impressive results have been obtained in many problem domains including routing and scheduling problems. Meta-heuristics employ different algorithmic schemes for the *exploration* of the search space. Variable neighbourhood search (VNS) (Hansen et al. (2010)), tabu search (Cordeau and Maischberger (2012)), and hybrid genetic algorithm (Vidal et al. (2012)) they all have different exploration behaviours. Section 2.1.2.3 describes some of the popular methods in more detail.

However, there is little work available, which targets the *exploitation* technique of the search. Most approaches apply simple local search (*LS*) methods: starting from a feasible solution and iteratively moving to a better solution by selecting it from a neighbourhood of the current solution (see Section 3.1).

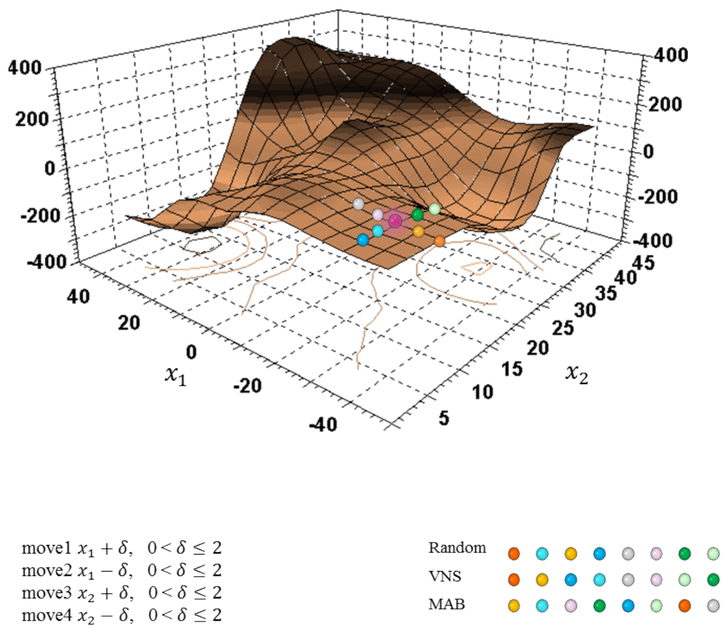


Figure 8.1: Search strategies on an optimisation problem with two variables, $x_1, x_2 \in \mathbb{Z}$

In this chapter, we introduce a machine learning-based *LS*, referred to as the dynamic multi-arm bandit neighbourhood search (D-MABNS). D-MABNS is inspired by well-known decision-making models for the multi-armed-bandit (MAB) problem. The major difference with opting for a traditional *LS* procedure is that instead of pre-specifying the examining order of neighbour solutions (e.g. lexicographic search, Funke et al. (2005)) at each iteration of *LS*, D-MABNS dynamically adapts the search sequence to test promising solutions first. D-MABNS can be embedded in any meta-heuristic or hyperheuristic framework.

Figure 8.1 uses an example to illustrate the inspiration behind the D-MABNS. There are three approaches for searching through the neighbourhood of the current solution, in the search space of a simple discrete optimisation problem that has two variables (x_1, x_2). The purple ball shows the current solution, and we define four types of move that generate eight neighbours of the current solution. The aim is to find an improved solution (light green ball) as efficiently as possible. In Figure 8.1, we see that checking potential moves in a random order finds the improvement after 8 checks. Checking that is performed in a pre-specified order, such as variable neighbourhood decent (VND) (Hansen et al. (2010)), finds the improvement after 7 checks, whereas following a statistically-based sequence, such as in the case of MAB, the improved solution is found after 6 checks.

The remainder of this chapter is organised as follows. Section 8.2 briefly discusses learning techniques to guide heuristic search and MAB problems; D-MABNS is introduced in Section 8.3; In Section 8.4, we demonstrate its use on instances of geographically distributed maintenance problem (GDMP) obtained from our drainage maintenance problem (Chapter 4); Section 8.5 analyses the performance of D-MABNS on periodic vehicle-routing problems (PVRPs) using benchmark instances; and finally the findings are summarised in Section 8.6.

8.2 Background knowledge

Using self-adaptive approaches to guide neighbourhood search combine machine learning and classical heuristic search techniques. Many approaches use the historical performance of operators to adjust future operator utilisation. An issue might be seen here is an exploration vs. exploitation dilemma. The exploitation strategy suggests that an operator that has performed well in the recent tests should certainly be applied again, while exploration suggests to also give a chance to other candidate operators that did not perform so well. In the AOS literature, the key components to handle this dilemma are credit assignment and decision-making (Costa et al. (2008)). The same concept is also applied for *LLH* selection in hyperheuristics (Section 7.3.2.2).

Credit assignment describes a way to evaluate the quality of an operator. Normally, the credit assigned to an operator is based on how often that operator makes a contribution or an improvement (e.g. reinforcement learning hyperheuristic, Section 7.3.2.2), or on the magnitude of the total contribution the operator has made so far (Soria-Alcaraz et al. (2014)). The latter measurement is more sensitive to the fitness landscape of the problem instance, and it is usually combined with a reward rescaling method. To evaluate each option, most approaches consider either the instantaneous reward value after an operator has been applied or the average reward over a sliding-time window. An alternative is to consider an extreme value (Fialho et al. (2009)), on the basis that the generation of rare but highly beneficial improvements matter more than frequent small improvements.

Decision making determines the selection of the next operator based on past credits. Probability matching and adaptive pursuit methods are probably the most widely applied mechanisms (Thierens (2005)). Both methods update the option selection probability according to its evaluation value and these probabilities are then used for selection in a *weighted roulette wheel* like process. An alternative mechanism adds an exploration term to the quality evaluation function and the decision-making process is determined on the evaluation values (e.g. using the UCB1 algorithm, Auer et al. (2002)).

Multi-arm bandit problems (MAB) Auer et al. (2002) define a typical static MAB problem in which the K arms each have an independent reward probability p_i, where $p_i \in [0, 1]$. At each time step t, the player should select an arm j. With probability p_j the arm receives a reward $r_t = 1$, otherwise $r_t = 0$. The decision-making process to guide neighbourhood search has a lot similarity to the MAB problem, in which the goal is to maximise the total rewards collected over time.

To solve a similar problem in a changing environment, Costa et al. (2008) describe a D-MAB in which each arm has a uniform reward distribution, from the interval $[p_{i,t}-1, p_{i,t}+1]$ at time step t. For every T time steps, the mean value of reward distribution of each arm $p_{i,t}$ varies. Further, the reward distribution of all arms change simultaneously.

In the next section, we use the fundamental concepts of D-MAB to address the dilemma of exploration and exploitation during a neighbourhood search process. We introduce techniques such as mapping solution fitness to rewards and dynamic neighbourhood management, to fit the characteristics of the neighbourhood search.

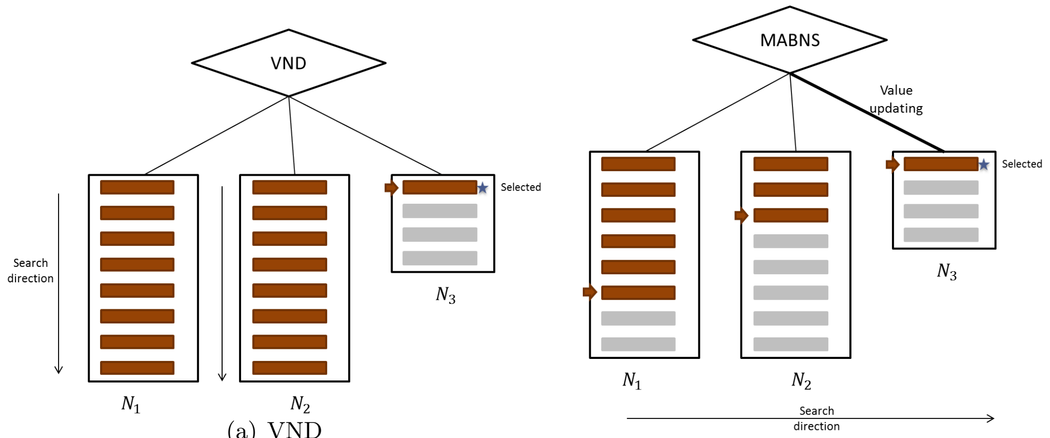
8.3 Dynamic multi-arm bandit neighbourhood search

Our D-MABNS is inspired by many of the techniques reviewed in Chapter 3 and experimental experience in Chapter 7. D-MABNS maintains a search trace within a neighbourhood, as is the case in improvement heuristics, but also introduces selection strategies between the neighbourhoods, like the methods that statistically select *LLH* in hyperheuristics. Furthermore, D-MABNS uses some of the techniques from simple combinatorial search to discard unpromising moves during the search.

8.3.1 D-MABNS overview and framework

The goal of search is to efficiently find an improvement direction in each iteration of the *LS* process so as to reach a local optimum quickly. First, consider a typical VND approach to solving a CVRP (Section 2.1). Neighbourhoods defined by moves, such as *2-opt* and *Cross-exchange*, are searched sequentially until an improved solution is found, as shown in Figure 8.2(a). The search can be made more efficient, for instance, by ordering elements or by discarding unpromising moves (Section 3.1.1). However the search is essentially over a sequence of neighbourhoods.

The D-MABNS design is based on the observation that it can be inefficient to check the entirety of one move's neighbourhood before considering the neighbourhood of the next move. D-MABNS uses a search pointer within each neighbourhood and dynamically decides *when* to examine the next neighbour and from *which* neighbourhood structure. At each decision point, D-MABNS looks at the neighbourhood, which has the best current expectation (Figure 8.2(b)).



(a) VND

(b) D-MABNS

Figure 8.2: The search strategies of VND and of D-MABNS

A neighbourhood $N_k(x)$ represents the set of neighbours that can be obtained by one defined move from the current solution x. For a given problem, there is a finite set of neighbourhood structures $N_k, (k = 1, \ldots, k_{max})$. In D-MABNS (Figure 8.2(b)) each neighbourhood structure N_k is associated with a value v_k, which is used to evaluate the quality of N_k, based on the neighbours seen so far in the neighbourhood. After a selected neighbour $x' \in N_k(x)$ has been tested, a reward (or punishment) r_k is given to $N_k(x)$ and the value v_k is updated. Then, $v_k, (k = 1, ..., k_{max})$ is used to decide which neighbourhood to look at next.

In the following sections, we look in more detail at the D-MABNS decision-making process, based on the MAB model. We also propose dynamic neighbourhood updating, to further improve the search efficiency of *LS*.

8.3.2 Decision making

In traditional static MAB problems, the MAB arms are rewarded either 0 or 1 according to a Bernoulli distribution (Auer et al. (2002)). Applying the MAB model to neighbourhood selection, each neighbourhood is treated as an arm and is characterised by a fitness distribution.

The fitness distribution of each neighbourhood can be approximated by an empirical investigation. Identifying the neighbourhood for the next check is an exploration and exploitation dilemma in the design of D-MABNS. A greedy selection strategy would select the neighbourhood with the current maximum estimated v_k. However we can introduce exploration into the deterministic decision process by using approaches such as the Upper Confidence Bound algorithm (UCB1) (Auer et al. (2002)).

Formally, we denote n_k as the number of neighbours examined so far from the k_{th} neighbourhood, and v_k is the forecast value of the corresponding neighbourhood (measured from rewards collected). The UCB1 algorithm selects the neighbourhood, which maximises the value below (Auer et al. (2002)):

$$v_k + \sqrt{\frac{2log \sum_i^{k_{max}} n_i}{n_k}} \tag{8.1}$$

In Equation 8.1, the square root component represents exploration, thus encouraging the search into less-explored neighbourhoods. The neighbourhood quality estimation, v_k, represents exploitation by preferring the neighbourhoods that have the best expectation value. The UCB1 algorithm ensures that each neighbourhood can be chosen infinite number of times but that the time that elapses between selections for a sub-optimal neighbourhood increases exponentially.

8.3.2.1 Quality value estimation v_k

D-MABNS requires a neighbourhood quality value estimation mechanism to forecast the value of a neighbourhood, v_k. Some existing approaches to credit assignment are outlined in Section 8.2. In our approach, we employ *exponential smoothing* (Gardner (1985)), which is often referred to in the operator-selection literature as additive relaxation (Costa et al. (2008); Fialho et al. (2010a)). Unlike simple average values, exponential smoothing provides a decay mechanism, which becomes necessary later (Section 8.3.3) when D-MABNS is applied to a D-MAB problem, in which new solutions are accepted during search. The value v_k of the selected neighbourhood is updated whenever a reward r_k is received, using Equation 8.2 (Gardner (1985)), such that v_k is the weighted average of the previous smoothed value and the latest reward r_k. A smoothing factor α $(0 < \alpha \leq 1)$ controls the decay rate of historical reward observations.

$$v_k = (1 - \alpha)v_k + \alpha r_k \tag{8.2}$$

8.3.2.2 Reward functions and scaling

In combinatorial search problems, we usually define a fitness function to measure the quality of solutions. Rewards are generally related to the fitness of the solutions found in a neighbourhood. However, directly using the fitness as reward can result in imbalance between the two parts of the Equation 8.1, thus biasing the search either towards exploitation or exploration.

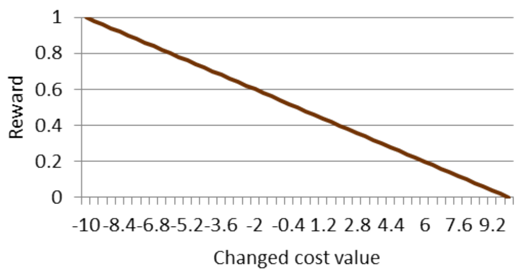
To reduce the bias, a scaling factor can be added either to the exploration or the exploitation parts of Equation 8.1. However, fitness measures and the range of fitness values are domain-dependent, and so, the scaling factor needs to be determined experimentally on a problem-specific basis. Here, we propose an *adaptive reward function*, which scales fitness into the range of $[0, 1]$, and does not need any prior domain knowledge to balance Equation 8.1. We design a reward function RF, which aims to reward a bigger improvement with a score closer to 1 and a worsening solution with a score closer to 0.

Considering a minimisation problem, for a neighbourhood N_k, we select and test a solution x' from the set of neighbours of the current solution, x. The cost of solution x' is returned by the function $f(x')$, and $\delta_f = f(x') - f(x)$ is the difference in cost between the current solution and the tested solution, in which $\delta_f < 0$ indicates an improvement from the current solution. We record the maximum and minimum changes of cost value over the last w tests of neighbours of x, denoted as δ_f^{max} and δ_f^{min}, respectively. We refer to w as a time window parameter, which can be set manually to any suitable value.

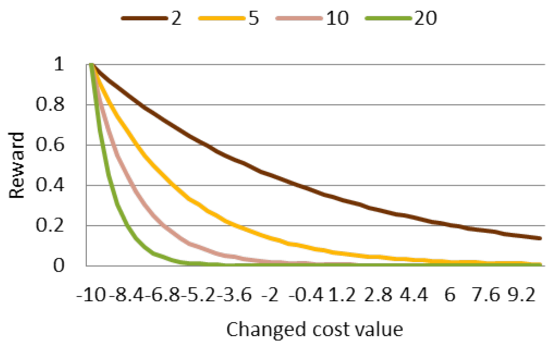
In selecting the reward function, we note that during a neighbourhood search, it is common to have already tested many worse neighbours from one or multiple neighbourhoods before finding any improvement. The cost value of these solutions gives an indication of the quality of a neighbourhood, and so, simply assigning a zero reward to a worse solution is not appropriate. Many reward functions could be applied here. We have tested the linear reward, exponential reward, and sigmoid reward functions. Figure 8.3 shows examples, from a detailed analysis, of how cost value change (x axis) maps to rewards (y axis) for the three functions. A more detailed experimental analysis presented in Section 8.4.4.1.

8.3.3 Dynamic neighbourhood management

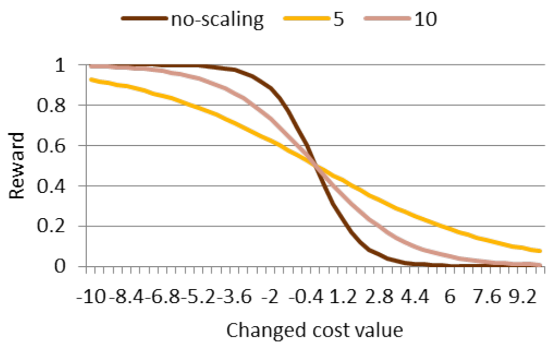
So far, we have only talked about a search process moving from the current solution x towards x', where $x' \in N_k(x)$. Solution x' is one *move* from x, and thus, it typically has a similar solution structure. If the same move were to operate on solutions x and x', it would also generate solutions with similar structures. Furthermore, depending on the problem



(a) Linear reward function: $RF = (-\delta_f + \delta_f^{max})/|\delta_f^{max} - \delta_f^{min}|$



(b) Exponential reward function: $RF = exp(-a(\delta_f - \delta_f^{min})/|\delta_f^{max} - \delta_f^{min}|)$, where $a = 2, 5, 10, 20$



(c) Sigmoid reward function: 1) $RF = 1/(1 + exp(\delta_f))$ (no scaling); 2) $RF = 1/(1 + exp(a(\delta_f - \delta_f^{min})/|\delta_f^{max} - \delta_f^{min}| - a/2)))$, where $a = 5, 10$

Figure 8.3: Examples of different reward functions, calculated from example cost values in the range of -10 to 10, using different scaling factors, a. Note that the above examples illustrate reward functions for minimisation problems, where $\delta_f < 0$ indicates an improvement.

and neighbourhood structure design, $N_k(x)$ and $N_k(x')$ may share many neighbours. If we denote the fitness distribution of $N_k(x)$ as $F(N_k)$ and of $N_k(x')$ as $F(N'_k)$, then $F(N_k) \simeq F(N'_k)$. There is an obvious benefit, which the dynamic model offers, in that we do not need to build the understanding of each neighbourhood structure from scratch at each LS

iteration.

Where the fitness distribution of each neighbourhood structure gradually changes as the search progresses, we have a D-MAB. Costa et al. (2008) apply a hybridisation of the UCB1 algorithm (Section 8.3.2) to D-MAB, which uses a Page-Hinkley test (PH, see Section 8.3.3.1) to detect abrupt changes in the environment. In our D-MABNS, "environment" refers to the reward signals generated from the fitness changes of the tested neighbour solutions. Algorithm 8.3.1 presents D-MABNS, which is a proposal to adopt the concept of D-MAB solvers for neighbourhood search problems.

Algorithm 8.3.1 D-MABNS (x)

```
1: Define:
2:     $N_k$ as a set of neighbourhood structures ($k = 1, ..., k_{max}$).
3:     UCB$_k$ (Function 8.1) is the UCB value of each neighbourhood structure $N_k$.
4:     $I_k$ is a pointer that indicates the next solution $x' \in N_k(x)$ will be checked.
5:     $v_{best}$ is the best forecasting value among all neighbourhoods.
6: Assign initial UCB value UCB$_k$=1 to each $N_k$
7: Assign $I_k = 0, (k = 1, ..., k_{max})$
8: Assign $v_{best} = 0$
9: while $\exists I_k$ do
10:    i.e. not reached the end of $N_k(x)$
11:    $k^* = argmax_{k \in \{k=1,...,k_{max}\}}$UCB$_k$
12:    $x' \leftarrow$ the solution indicated by $I_{k^*}$.
13:    $I_{k^*}$ move to the next neighbour of $N_k$
14:    if $f(x')$ is better than $f(x)$ then
15:        $x \leftarrow x'$
16:        Update neighbourhoods (Section 8.3.3.2)
17:    end if
18:    Calculate reward $r_{k^*}$ based on changed fitness (Section 8.3.2.2)
19:    if Environment change (Algorithm 8.3.2, Section 8.3.3.1) then
20:        Reset all values used to calculating UCBs
21:        $v_{best} = 0$
22:        $\forall I_k = 0, (k = 1, ..., k_{max})$
23:    end if
24:    Update all values used to calculate UCB
25:    Update UCB values for all neighbourhoods
26:    If $v_{k^*}$better than$v_{best}$, then $v_{best} = v_{k^*}$
27:    Pruning($I_{k^*}$,$v_{k^*}$,$v_{best}$) (Algorithm 8.3.3, Section 8.3.3.3)
28: end while
29: return $x$
```

8.3.3.1 Environment Change detection

We use the Page-Hinkley (PH) statistics (Page (1954); Hinkley (1971)), as proposed by Costa et al. (2008), to detect significant changes in the environment. For a given set of reward observations over time $\{r_{k,1}, ..., r_{k,t}, r_{k,t+1}...\}$, PH detects a reward $r_{k,t+1}$, which does not come from the same statistical distribution as the previous observations.

Formally, we use $\overline{r_k}$ to represent the average value of rewards observed so far for neighbourhood N_k; $\overline{r_k}$ is updated every time a new reward is received. We define $\epsilon_{k,t} = r_k - \overline{r_k} + \theta$ to represent the difference between the reward r_k from the current iteration and the average reward value, where θ is a tolerance parameter that is used to enhance the robustness of the PH test in a slowly changing environment. For simplicity, we set $\theta = 0$. The PH statistic calculates a variable $m_{k,t}$ as the sum of $\epsilon_{k,t_1}, ..., \epsilon_{k,t_{max}}$, and a variable $M_{k,t}$ which is the maximum value of $m_{k,1}, ..., m_{k,t}$.

Algorithm 8.3.2 is used to update information about the tested neighbourhood and to detect environment change. The parameter λ is a user-defined value, which controls the trade-off between false positive and false negative detection errors, just like θ, λ controls the sensitivity of the change detection. In Section 8.4.4.4, we present experimental tests on a set of λ values to analyse the impact of environment change detection on algorithm performance.

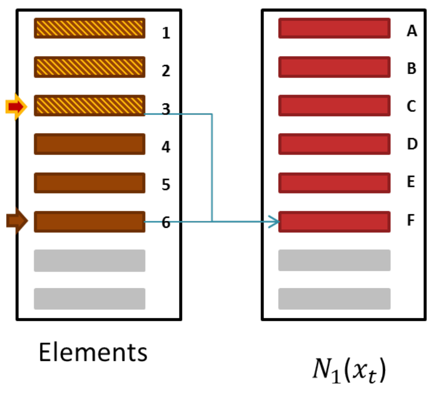
Algorithm 8.3.2 Environment change (Costa et al. (2008))

```
1: $\overline{r_k} \Leftarrow \frac{1}{n_k+1}(n_k\overline{r_k} + r_{k+1})$
2: $n_k \Leftarrow n_k + 1$
3: $m_k \Leftarrow m_k + (\bar{r_k} - r_k + \theta)$
4: $M_k = max(M_k, m_k)$
5: if $M_k - m_k > \lambda$ then
6:     return True //environment is changed
7: else
8:     return False
9: end if
```

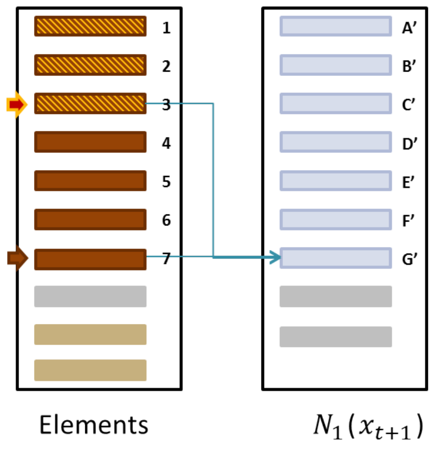
8.3.3.2 Feature Sequential Search and Neighbourhood Updating

Within the neighbourhood search, D-MABNS applies Feature Sequential Search (FSS) (see Section 3.1.1). FSS identifies a set of changing elements of a move according to the cost of changing elements. The elements represent some basic units of our solution structure, such that each next element of a neighbourhood should get generated in a constant time. For example, for a *2-opt* move, elements might be edges.

FSS usually considers a candidate element list sorted on the basis of intuition by features (e.g. in *2-opt* moves, length of edges). However, sorting is not essential. Figure 8.4(a) shows construction of a complete move, involving two changing elements labelled 3 and 6, which produce a neighbour solution from N_1, using a nested loop to search through the elements list.



(a) Neighbourhood search in $N_1(x_t)$ at LS iteration t



(b) Neighbourhood search in $N_1(x_{t+1})$ at LS iteration $t + 1$

Figure 8.4: Neighbourhood Updating

In Figure 8.4(b), solution x_{t+1} is derived from solution x_t in Figure 8.4(a). The two solutions share mostly the same structure and many joint neighbours, and so, we do not need to check the neighbours that have already been checked in the previous LS iteration. In practice, it is also unlikely that a move which was checked in iteration t and did not make an improvement would produce a better solution in iteration $t + 1$, even when the move leads to different solutions in iteration $t + 1$.

After making a move from x_t to x_{t+1}, where $x_{t+1} \in N_1(x_t)$, we continue to check other elements that still exist in the new solution x_{t+1}, checking the new elements last. The previously checked elements are reconsidered only when the algorithm detects a change (Algorithm 8.3.1, line 15). When the pointer I_k (Algorithm 8.3.1) reaches the end of a neighbourhood $N_k(x_t)$, a large negative value is assigned to the neighbourhood to ensure that it cannot be chosen again until PH signals a change.

8.3.3.3 Neighbourhood structure pruning

For each neighbourhood structure N_k, the forecasting value v_k is used to keep track of quality. v_k can also be used as a prompter to prune a bad neighbourhood. As shown in Algorithm 8.3.3, when the value (v_{k^*}) of the current selected neighbourhood k^* is set to a large negative value (line 3), the neighbourhood structure N_k will not be selected and updated until there is a change in fitness distribution. This strategy is especially useful in conjunction with FSS, where a promising area of a neighbourhood is explored early in the search. The user-defined parameter γ in Algorithm 8.3.3 adjusts the tolerance of accepting a neighbourhood that is worse than the best evaluated one. Setting $\gamma = 0$ turns off the pruning function.

Algorithm 8.3.3 Pruning(I_{k^*},v_{k^*},v_{best})

```
1: if $v_{k^*} < \gamma\ v_{best}$ then
2:     set $I_{k^*}$ to the end of $N_k$
3:     set $v_{k^*}$ to a big negative value
4: end if
```

This section has introduced the general concept of D-MABNS, as well as the techniques that it employs. The following sections apply D-MABNS on two combinatorial optimisation problems, using a real-world and a set of benchmark problems.

8.4 An application to maintenance scheduling

8.4.1 Problem description

We firstly test our D-MABNS on instances of GDMP, which are obtained from the real-world gully-pot system maintenance problem (Equation (4.1) – Equation (4.8), Section 4.2.2). For convenience, we briefly recall the high level objective (Equation (4.1)), which is to select a judicious subset of gullies from N assets and assign them to days of the following maintenance period in order to minimise the risk in the period:

$$\sum_{d \in W} \sum_{i \in N} r_i P_i(d)$$

To test the performance of our D-MABNS algorithm, we generate five problem instances of various sizes, by randomly selecting 10%, 25%, 50%, 75% and 100% of the gullies from the system. Each gully pot is associated with location, risk impact r_i, the number of days since its last service, and two parameters for the failure probability estimation function $P_i(d)$ [1]. The planning horizon in each instance is set to 7 days ($|W| = 7$) and only one vehicle is available to deliver the service.

8.4.2 GDMP solution approach

In Chapter 6, we use BEBO hyperheuristic to schedule maintenance actions, and we use the same solution framework for the D-MABNS experiments. We briefly outline the process as following:

[1]A Weibull distribution is used to estimate the lifetime of a gully pot. The data set, can be found online, at `http://yc1005.wixsite.com/yujiechen`

1. **Preparation**: Generate a candidate route set S_{all}, which ignores all risk impact and failure rate information; optimise for distance using a standard CVRP heuristic approach.

2. **Initialisation**: Generate a schedule for W days by selecting the routes $s \in S_{all}$ with the highest risk, measured by $\sum_{i \in s} r_i P_i(d_1)$, where d_1 represents the first day of W period.

3. **Optimisation** The optimisation steps repeat for a fixed amount of CPU time.

   Improvement: Apply a heuristic approach to improve the solution, evaluated by the objective function in Equation 4.1.

   Re-initialisation: Randomly destroy a few days' schedule from the output of the improvement stage; rebuild the destroyed routes from scratch by considering assets with the highest risk estimation and a few randomly selected asset points; and assign these routes randomly to the days.

BEBO hyperheuristic is employed in the improvement stage (above) and we now replace BEBO by D-MABNS.

8.4.3 Local search moves

We use the same six moves for D-MABNS and BEBO (Chapter 6). Our implementation of BEBO applies a first-improvement heuristic (using lexicographic search) for each move as a low-level heuristic (denoted as LLH_1 - LLH_6, Chapter 6). Differently, D-MABNS governs the search between all available moves directly (Figure 8.2(b), Section 8.3.1). For clarity, we list the six moves (denoted as *move1* -*move6*) as follows.

move1 Delete two edges each from two routes and reconstruct to generate two feasible new routes. This move is the same as chain cross exchange, where each chain contains a maximum of k points, $1 \leq k \leq 5$.

move2 Insert k points that do not appear in the schedule plan, using the cheapest insertion heuristic with a relaxed route duration constraint. If, as a result of the planned insertion, any target route exceeds the duration constraint, repeatedly remove the best-condition point from the route until it becomes feasible, $5 \leq k \leq 20$.

move3 Replace the last n days' schedule with n other routes from the candidate set S_{all}, $1 \leq n \leq |W|$.

move4 Same as move3, except that we choose the schedule of n days to replace uniformly at random, instead of the last n days' schedule, $1 \leq n \leq |W| - 1$.

move5 Switch the schedules of two days with each other

move6 Move one day's schedule to an earlier day.

8.4.3.1 Element sorting by features

It is not essential to sort elements, however, we do so whenever we have suitable domain information. GDMP is a risk-minimisation problem, and so, many elements can be sorted by their current risk estimation. In our implementation, *move1* uses an edge list that is sorted by the length of the edges whereas *move2* uses an asset point list that is sorted by the current risk estimation of each point. Due to the large number of asset points we have in an instance, a single search loop that selects the next k points is used instead of a nested loop. *move3* and *move4* use a route information list, which is sorted by the sum of the current risk of all points in each route. *move5* and *move6* do not sort their elements by any features.

8.4.4 Computational results for GDMP

This section first reports a series of sensitivity analyses on the parameter settings of D-MABNS. We then compare D-MABNS with two algorithms based on the MAB. The algorithms tested apply different decision-making strategies (probability matching (Goldberg (1990)) and random selection) allowing for a comparison to be drawn with the UCB1 algorithm (Section 8.3.2). Section 8.4.6 compares D-MABNS to BEBO hyperheuristics (Chapter 6) and a modified VNS (denoted as VNSr(R), Chapter 7). All algorithms are implemented in C$^\sharp$ and executed on a cluster composed of 8 Windows computers with 8 core Intel Xeon E3-1230 CPUs, which have 16GB RAM. All tested parameter settings and algorithms start with the same initial solution obtained by the *Preparation* and *Initialisation* steps in section 8.4.2. For each of the problem instances, each algorithm is run 30 times. The stopping condition for all algorithms is the same CPU time, as shown in Table 8.1.

8.4.4.1 Sensitivity analysis

Finding the best setting of parameter values is a non-trivial and time consuming task, which often requires considerable expertise and experience. Table 8.2 summaries the parameters

Table 8.1: Data set information and CPU time allowed for experiments

Name of GDMP instances	number of assets	CPU allowed
mi01	2815	120s
mi02	7037	240s
mi03	14074	900s
mi04	21111	1800s
mi05	28149	3600s

used by the D-MABNS algorithm, including the time window parameter w and the reward function RF (Section 8.3.2.2), the smoothing factor α (Section 8.3.2.1), the environment change detection λ (Section 8.3.3.1) and the pruning rate γ (Section 8.3.3.3). For each parameter in Table 8.2, we sample values from a given range and run each parameter-set 30 times for the predefined CPU times, as shown in Table 8.1. The parameter-set that has the best average performance over all instances is used for comparison with other heuristic methods.

Small values of w, the time window parameter, mean that only recent fitness impacts decision making, resulting in a more dynamic reward process over time. Consequently, if recent fitness observations are not good, even a slightly better fitness may result in a big reward. For the smoothing factor α, larger values mean that historic rewards are forgotten more quickly.

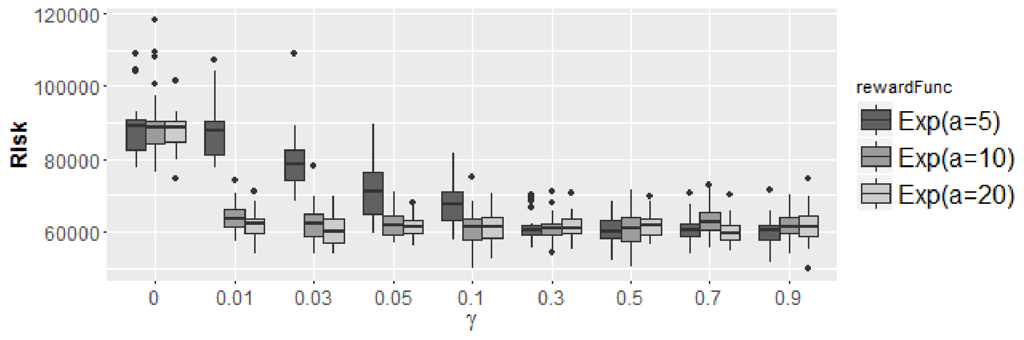
Our preliminary tests show a strong impact on performance from the environment change detection parameter λ and the neighbourhood pruning parameter γ. Some interesting behaviours have also been observed in reward function usage and search within neighbourhoods. Therefore, we design further experiments and explore these parameters in more detail.

Table 8.2: Parameter settings

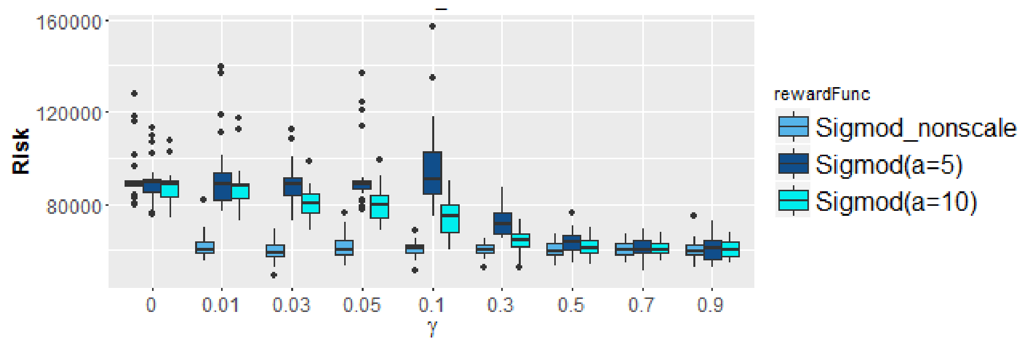
Parameter	Tested range	Value applied
Time window w	$[100, 200, 400, 600, 1000, 3000, 5000]$	400
Reward function RF	linear, sigmoid, exponential	exponential (a=5)
smoothing factor α	$[0.2, 0.4, 0.6, 0.8, 1.0]$	0.8
Environment change λ	$[0.01, 0.05, 0.15, .., 0.3, 0.5, .., 1.0, 2.0, .., 4.0]$	0.05
Pruning rate γ	$[0.01, 0.03, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9]$	0.7

8.4.4.2 Reward function and Neighbourhood pruning

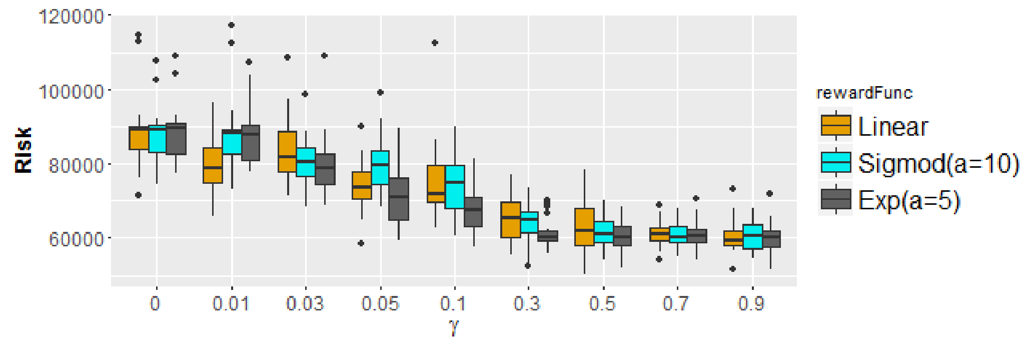
In Section 8.3.2.2, a number of functions that map the changes in fitness to rewards are discussed. The reward function, along with the pruning parameter (Section 8.3.3.3), are essential parts of the D-MABNS algorithm design. We test three reward functions and different settings of the neighbourhood pruning parameter γ. The other parameters (Table 8.1) are fixed as $\{w = 400, \alpha = 0.8, \lambda = 0.05\}$. Each setting runs 30 times.



(a) $RF = exp(-a(\delta_f - \delta_f^{min})/|\delta_f^{max} - \delta_f^{min}|)$



(b) $RF = 1/(1 + exp(\delta_f))$ (no scaling); $RF = 1/(1 + exp(a(\delta_f - \delta_f^{min})/|\delta_f^{max} - \delta_f^{min}| - a/2)))$;



(c) Linear function: $RF = (-\delta_f + \delta_f^{max})/|\delta_f^{max} - \delta_f^{min}|$, compared with the other two

Figure 8.5: Effect of reward function with different neighbourhood pruning parameter settings γ, illustrated for mi02. We measure the algorithm performance using the objective function of risk minimisation.

We run the experiment on all five GDMP instances, and observe similar results. In all plots, we can see that the pruning parameter γ is important to algorithm performance. Figure 8.5 presents the results for the mi02 instance as box plots of risk (y axis, the objective) against pruning parameter settings (x axis).

Figure 8.5(a) presents the results for the exponential reward function. The function uses a positive constant a to map the raw risk value change δ_f to the range $[-1, a - 1]$, such that when $\delta_f > 0$, $\lim_{a \to \infty} r_k = 0$. Bigger values of a emphasise the importance of the greatest improvements. From Figure 8.5, we can see that when $\gamma = 0$, there is no significant difference across all setting of a. However, when γ is increased to values in the range 0.01-0.05, bigger a values achieve better results. This is because a bigger a value emphasises the difference between results of improving and worsening moves. Combining the effect of the pruning mechanism and the scaling factor a for the exponential reward function, bigger a values result in an earlier or more harsh neighbourhood pruning policy, which seems to especially benefit our problem instances. However, this advantage fades as γ increases. The best-performing combination here measured by the mean fitness value is $\{\gamma = 0.5, a = 5\}$.

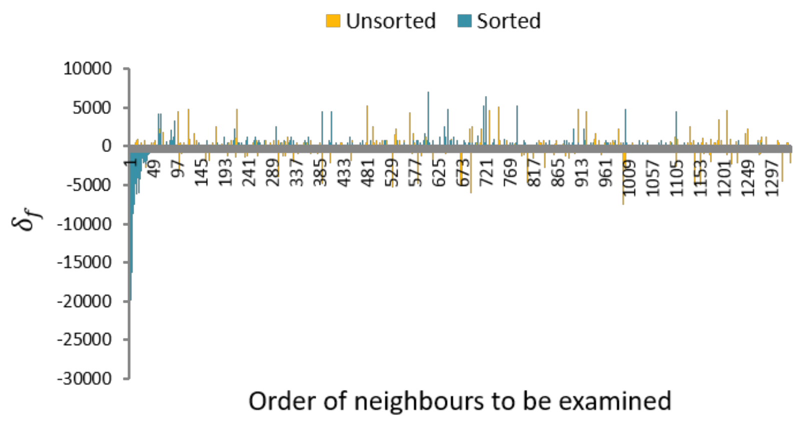
The results for the sigmoid reward functions (Figure 8.5(b)) follow similar patterns to those of the exponential reward functions. Comparing the best setting of sigmoid reward function $\{\gamma = 0.7, a = 10\}$ with the best setting of exponential reward function and linear function (Figure 8.5(c)), no significant difference can be seen in terms of solution quality.

These results are common to all our GDMP instances. However, because of the different fitness landscapes and neighbourhood structure design, a suitable reward function must be chosen for each new problem domain. The interaction of reward function and other design elements, such as pruning, may also be domain specific. For the linear reward function, which simply squashes the δ_f in to the range of [0,1] with respect to the minimum and maximum value seen within time window w, too small w leads to a random reward system. In comparison, the exponential and sigmoid function are less sensitive to w. The exponential function especially amplifies small differences in δ_f and needs an extra parameter a to adjust the shape of reward function. This may affect the overall performance of algorithms. Increasing a leads to a smaller reward range of worsening fitness, which dilutes information from most tested neighbours. We suggest assigning a to a value smaller than 10 to map the worst found fitness to values about 10^{-5}. The sigmoid function has a natural advantage of producing values in the range [0,1] (Figure 8.3(c)). The calculation does not need assistance from the maximum and minimum records within the sliding time window w. A smaller number of parameters is good, but the function shape may not generate an efficient mechanism.

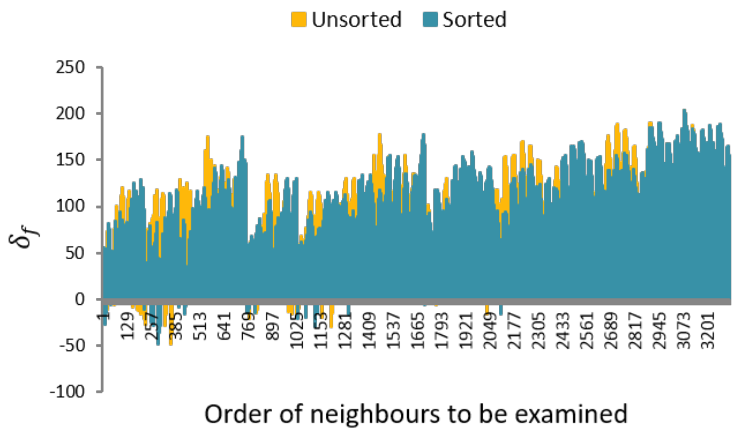
8.4.4.3 Neighbourhood sorting and pruning

Our proposed D-MABNS uses FSS (Section 8.3.3.2) to determine the order in which neighbours are checked within one neighbourhood. To verify the importance of the FSS strategy, we capture a few neighbourhood structures for a current solution x, and plot the risk value changes δ_f.

Figure 8.6(a) shows one example structure of the neighbourhood $N_{move2}(x)$. Because the search using *move2* applies a single search loop that picks every next k elements (Section 8.4.3), only a sub-area of $N_{move2}(x)$ is checked. In this case, the sorted features seem especially helpful in forming improved solutions early in the search stage and pruning the neighbourhood after about the 50th examination would most likely produce savings in CPU time without missing good moves.



(a) Neighbourhood structure using *move2*



(b) Neighbourhood structure using *move1*

Figure 8.6: A snapshot of two neighbourhood structures using or not using FSS, illustrated for the mi02 instance. Note that this is a risk minimisation problem, in which $\delta_f < 0$ implies an improvement.

Comparing the search with unsorted and sorted elements for *move1*, the feature sorting strategy shows no significant contribution to the search in $N_{move1}(x)$ (Figure 8.6(b)). In this case, the pruning strategy may not be beneficial as improvements could be found throughout the neighbourhood. Recall that the elements in *move1* are sorted by the length of edges, whereas our objective function is risk minimisation. This result reveals the importance of problem domain knowledge in FSS design. If the fitness distribution of a neighbourhood shows strong orderliness on one or a few features, FSS significantly improves the search efficiency. In other situations, the sorting process may not be worth the effort.

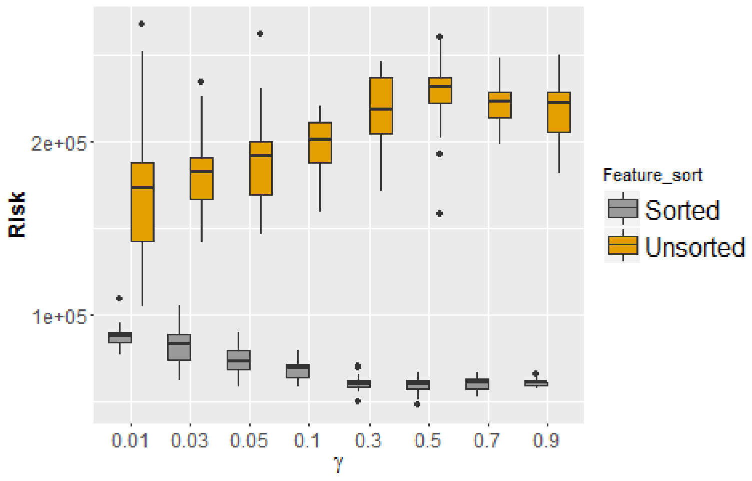


Figure 8.7: Impact of sorted features with different pruning parameter γ, on the mi02 instance. Parameter settings: $\{w = 400, RF = exp(a = 5), \alpha = 0.8, \lambda = 0.05\}$

An example of another, more direct, way to illustrate the impact of FSS and pruning is shown in Figure 8.7. We repeat each parameter setting 30 times with the pre-defined CPU time and record the solution quality. The same experiment has been tested for our five instances (see Appendix, Figure A.1) and similar effects have been observed across all of them. Overall, the results clearly show the positive impact of FSS on algorithm performance. Feature sorting guides the search to promising moves in the early stages. Combined with the pruning strategy, FSS significantly improves the search efficiency. On the other hand, when not using feature sorting, pruning reduces the chance of finding good neighbours too early into the search. As we can see from Figure 8.6, when a neighbourhood is unsorted and its fitness landscape is chaotic, the pruning decision (see Algorithm 8.3.3) becomes less meaningful.

8.4.4.4 Environment change detection

D-MABNS resets the evaluation of all of the neighbourhood arms when the PH statistic signals an environment change (Section 8.3.3.1). The PH statistic is widely used in D-MAB, but no analysis is presented (Fialho et al. (2009); Sabar et al. (2014)). In order to understand the contribution of the PH statistic and the impact of the parameter λ, we test different λ values with other given parameters $\{w = 400, RF = exp(a = 5), \alpha = 0.8, \gamma = 0.05\}$. A small γ value is used for this experiment as preliminary tests showed that a large

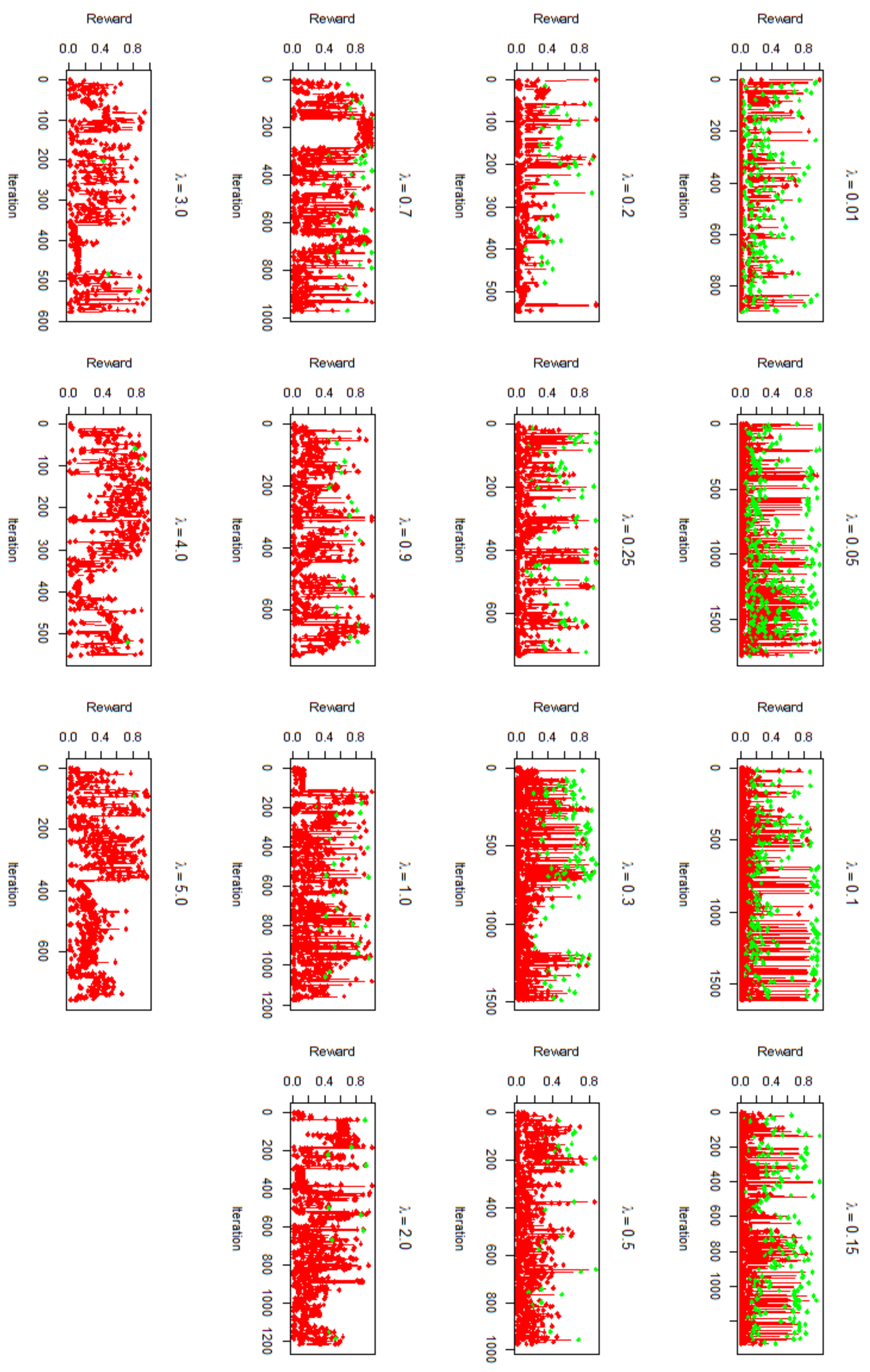
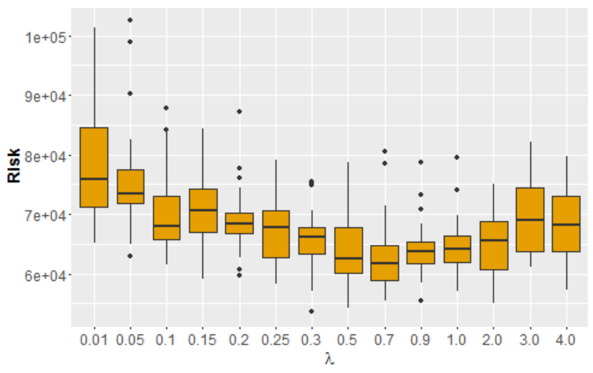
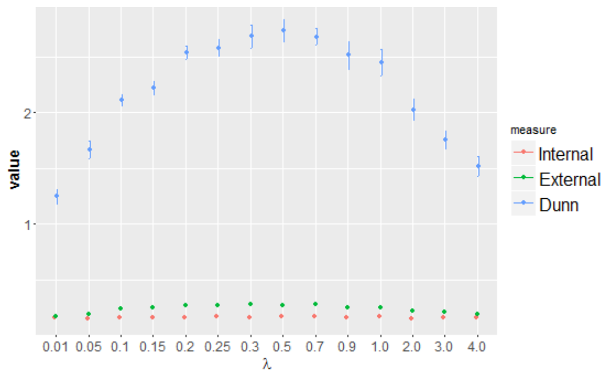


Figure 8.8: The effect of parameter λ on environment change detection. The green points indicate a change leading to a resetting of arms and the red points represent the normal situation. Other parameter settings are presented in Table 8.2

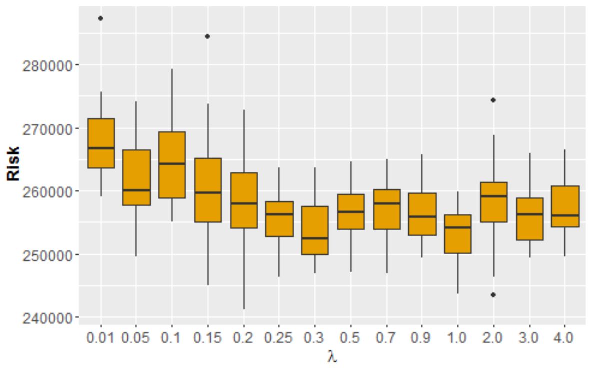
γ diminishes the impact of λ.



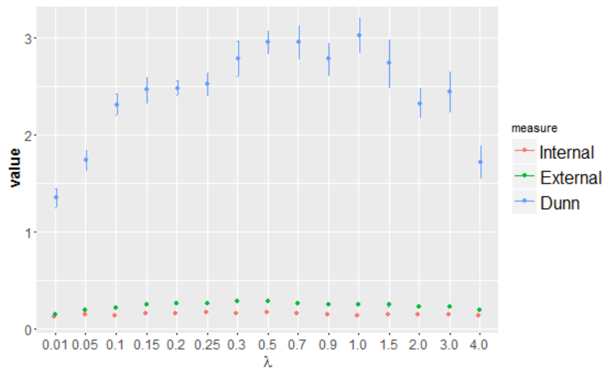
(a) mi02: Performance with different lambda setting



(b) mi02: Cluster validation index



(c) mi04: Performance with different lambda setting



(d) mi04: Cluster validation index

Figure 8.9: The effect of parameter λ on algorithm performance. The Dunn index is calculated as follows: $Dunn = \frac{1}{K}\sum_{i=1}^{i=K} Dunn_i$, where K is the number of neighbourhoods; $Dunn_i = \frac{\delta(C_{true}, C_{false})}{max\ \Delta_C}$, where $\delta(C_{true}, C_{false})$ measures the Euclidean distance between the centres of two clusters (denoted as external distance), and $\Delta_C = \frac{1}{|C|}\sum d(s, v(C))$ measures the average distance between all samples $s \in C$ and the cluster's centre $v(C)$ (denoted as internal distance). For more information about cluster validation index (see Rendón et al. (2011)).

We evaluate the PH statistic via algorithm performance in terms of final solution quality. As before, each parameter setting is repeated 30 times with the pre-defined CPU time (Table 8.1). Figure 8.8 illustrates an example of rewards collected by *move2(k* $= 10$*)* in the different λ settings given. As λ gets bigger, the PH statistic becomes more tolerant of reward variations. Figure 8.9(a) shows that D-MABNS achieves worse results when λ is too small, because there is too much noise in the PH alarm signals.

To further analyse the relation between solution quality and environment change detection, we measure the distance between PH signals for environment change and PH for the normal situation. We repeat each parameter setting 5 times for 180 CPU seconds, and each run records the rewards collected over time by each neighbourhood arm. Each data point is labelled *True* or *False* depending on whether it is a changing point or not. We then use a Dunn index (Rendón et al. (2011)) to evaluate the decision quality of PH by measuring the distance within and between environment changing and non-changing points. The intuition is that lower the noise in the detection, the better is the solution quality that an algorithm can achieve. Figure 8.9(b) includes the average value of the Dunn index of each neighbourhood detection result. We can see that as the Dunn index gradually gets bigger, the solution quality in Figure 8.9(a) also gets better. Note that the Dunn index could thus be used to efficiently tune the environment threshold parameter.

8.4.4.5 Environment change detection and Neighbourhood pruning

In the previous experiments, we observe the strong positive impact of the pruning parameter γ on the algorithm performance of our GDMP instances. As γ increases, the algorithm performance also improves. When γ is bigger than 0.3, sensitivity to other parameters is reduced.

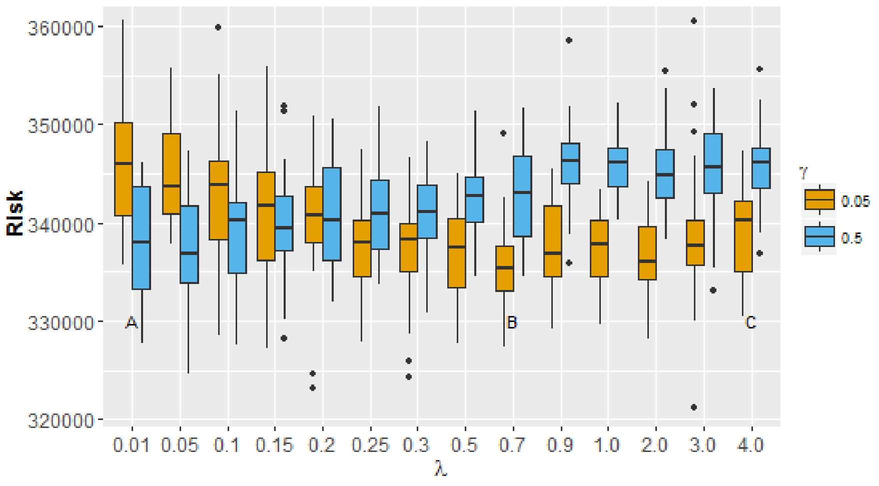


Figure 8.10: The effect of parameter λ on the algorithm performance with two different pruning parameter setting γ. The other parameter settings are $\{w = 400, RF = exp(a = 5), \alpha = 0.8\}$. Tested on mi05.

Our experiments on the combined impact of environment detection λ and pruning parameter γ give a rather different view. For example, Figure 8.10 shows that, when the environment detection parameter λ is bigger than 0.25, the bigger pruning parameter γ starts to show its negative impact. To compare the search strategies, three box plots, A, B, and C, are picked out in Figure 8.10. Parameter settings at A ($\lambda = 0.01, \gamma = 0.5$) emphasise wide exploration, whilst those at C ($\lambda = 4.0, \gamma = 0.05$) concentrate on exploitation. During exploitation, historical reward information becomes useful to guide the search. Smaller λ values give poorer results because the MAB arms are reset too frequently, resulting in loss of historical information. Parameter setting B is the best performing one in this experiment, as it achieves a balance between exploration and exploitation. Results for the other GDMP instances are presented in the Appendix Figure (A.2).

8.4.5 Other search strategies between neighbourhoods

Apart from the UCB1 algorithm, many other decision making strategies are proposed in the existing literature (Section 8.2). These methods introduce strategies from different perspectives to tackle the exploration and exploitation dilemma of MABPs. In this section, we compare the UCB1 method to probability matching (PM) (Goldberg (1990); Thierens (2005)) and a simple random (Ran) selection strategy.

8.4.5.1 Probability matching (PM)

PM is widely used for operator selection (Fialho et al. (2010a)). Similar to many genetic algorithms (Dianati et al. (2002)), it applies a roulette wheel selection. At each decision point, the probability p of examining a neighbourhood is proportional to its forecast quality. The detail of PM has been described in many papers (Goldberg (1990); Thierens (2005)).

Compared to UCB1, PM implements exploration by adding randomness to the selection process instead of using a statistical equation. At each decision point, the probability p of examining a neighbourhood is proportional to its forecast quality. The benefit is obvious: there is no need to rescale fitness to tune the statistic.

Our implementation of PM follows the pseudo code provided by Thierens (2005). We use an exponential reward function, as shown in Function 8.3, for which no scaling factor a is needed. Other parameters for PM are: $\{p_{min} = 0.01, \alpha = 0.8, \gamma = 0.94\}$, where p_{min} is the minimal probability value to ensure that none of the arms are ignored. α and γ are the same parameters as for the UCB1 algorithm, with values selected through preliminary experiments.

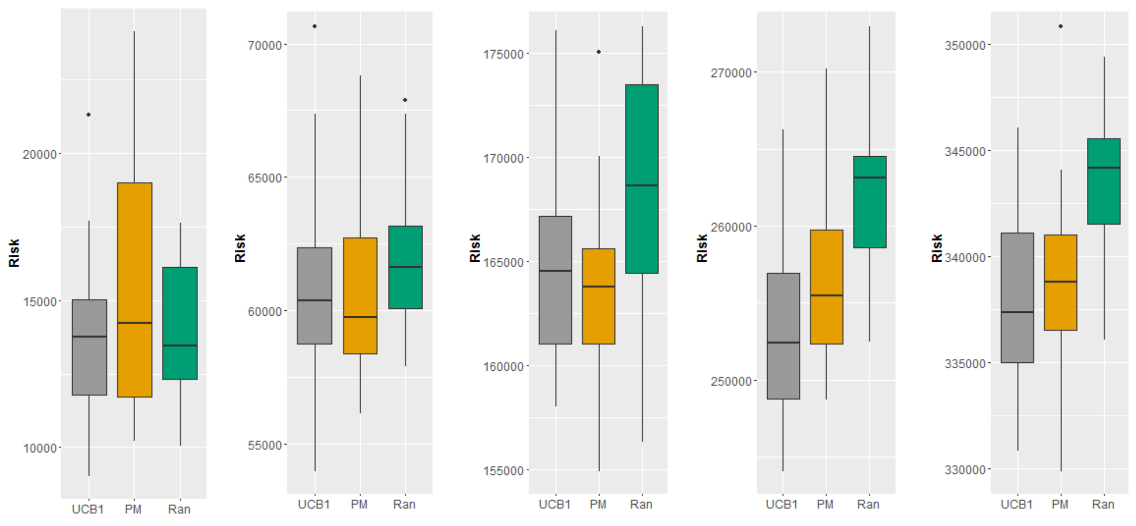
$$RF = exp(-\delta_f / f(x)) \tag{8.3}$$

8.4.5.2 Random selection

Random selection (Ran) randomly selects a neighbourhood N_k from the available candidate set following a uniform distribution. Compared to UCB1 and PM, Ran does not need any statistical technique to evaluate arms. To apply a neighbourhood pruning strategy, we simply cut off a neighbourhood if no improvement is found through the last m tries of that neighbourhood. We use $m = 50$, as the value that produces the best average results across our five GDMP instances.

8.4.5.3 Comparing the UCB1, PM and Ran strategies

We compare UCB1 (parameter settings in Table 8.2), PM and Ran with each other. Each algorithm is repeated 30 times and the results are shown in Figure 8.11. For small problem instances (instance mi01, mi02), there is no obvious difference between the three tested methods. However, for larger problem instances, the performance of Ran is much worse than the other two methods. This suggests that there is some useful information in the fitness distribution and landscape and that we can statistically capture this information via the reward function, to guide the search in the correct direction.



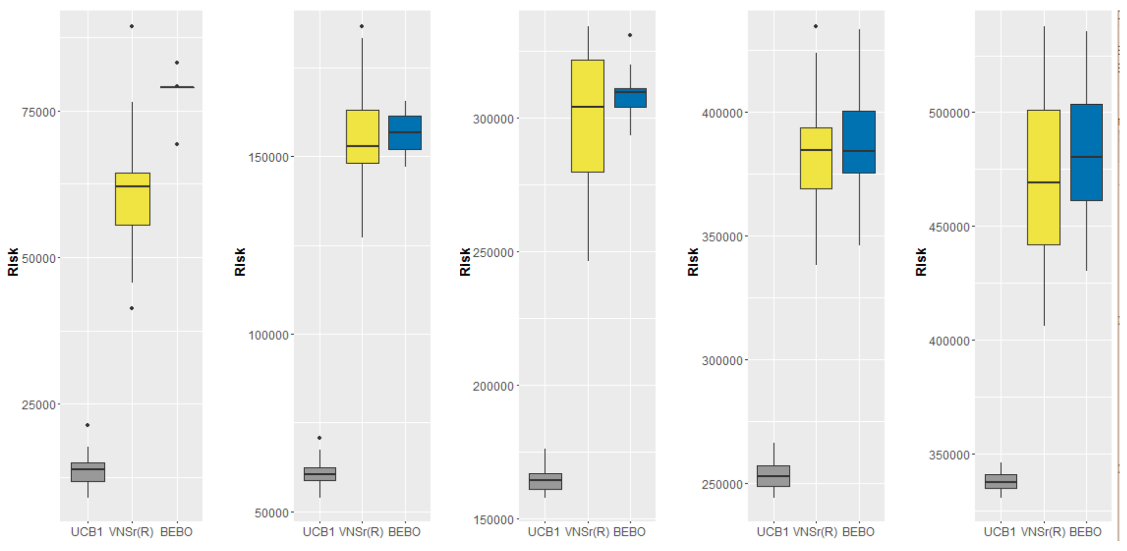
(a) mi01 (b) mi02 (c) mi03 (d) mi04 (e) mi05

Figure 8.11: Performance comparison of different decision making strategies: UCB1, PM and Ran

8.4.6 Comparison to traditional hyperheuristic

In this section, we compare D-MABNS with two hyperheuristic algorithms, which are BEBO (Chapter 6) and the VNSr(R) (Chapter 7). As Figure 8.12 shows, D-MABNS improves the solution quality for all tested instances.

Comparing the architecture of the three algorithms, D-MABNS (Figure 8.2) is more like a breadth first search, whereas BEBO and VNSr(R) use first improvement (FI) low-level heuristics to repeatedly examine one type of moves until an improvement is found, D-MABNS propels the neighbourhood search in a promising direction, which tends to improve the algorithm efficiency. At a higher level, in comparison to VNSr(R), D-MABNS



(a) mi01 (b) mi02 (c) mi03 (d) mi04 (e) mi05

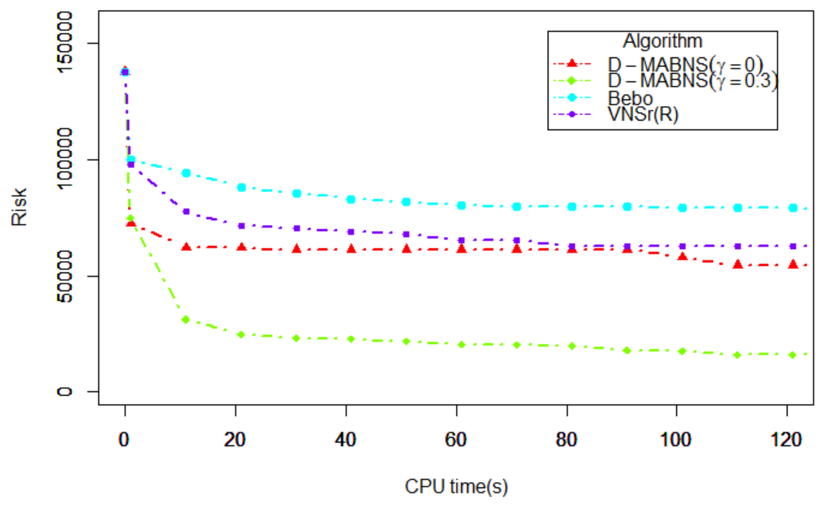
Figure 8.12: Performance of D-MABNS (labelled as UCB1), VNSr(R) and BEBO

builds a descent search path with a more flexible combination of moves. Furthermore, D-MABNS introduces many tricks, such as dynamic neighbourhood updating to avoid repeatedly checking the same elements as well as pruning to discard unpromising areas of the neighbourhood.

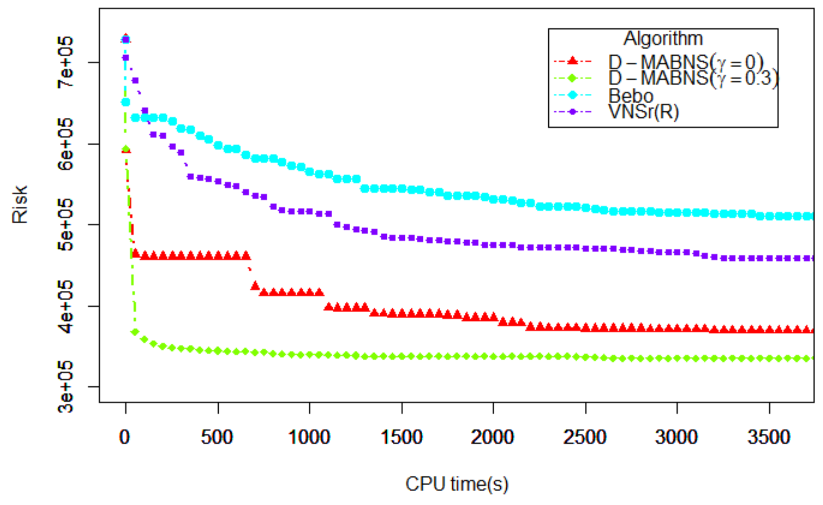
Figure 8.13 plots an example of the risk value (minimisation objective) changing over time using BEBO, VNSr(R), D-MABNS without pruning ($\gamma = 0$) and D-MABNS with pruning ($\gamma = 0.3$). Pruning reduces unnecessary search space, thus allowing the algorithm to converge earlier. Even without pruning, the D-MABNS algorithm has the advantage over the two hyperheuristics, especially for larger problem instances. We attribute this achievement to the D-MABNS algorithm's breadth first style of search and dynamic neighbourhood updating.

8.5 An application to PVRP

So far, we have tested the D-MABNS on five instances of GDMP and shown that it significantly outperforms other hyperheuristic approaches. In Chapter 4, we have made the point that our risk driven GDMP has a number of features, which make the model different from the standard PVRP studied in the available literature. We think it would be interesting to also test our algorithms on the PVRP instances to compare with a wide range of heuristic approaches. Compared to GDMP, PVRP has tighter constraints on service pattern requirements. Consequently, the search space of feasible solutions for visiting pattern assignment



(a) mi01



(b) mi05

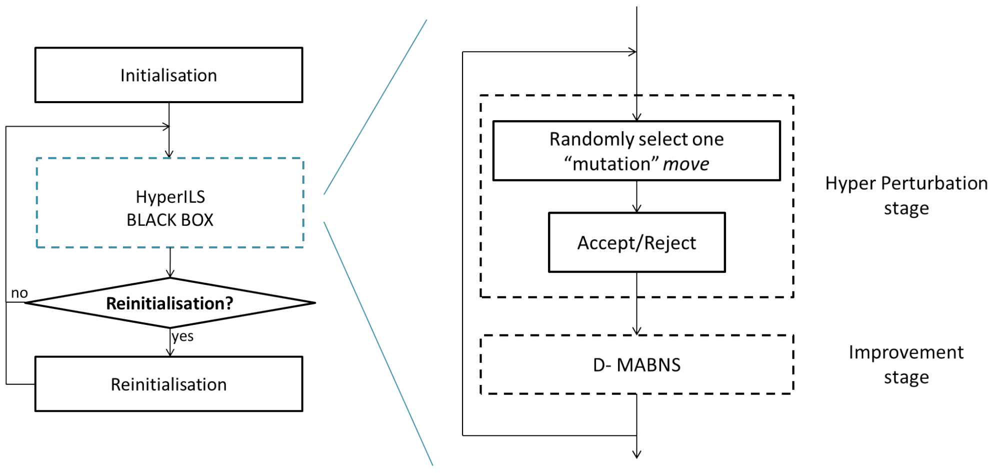
Figure 8.13: Comparing single runs of D-MABNS, VNSr(R) and BEBO.

is smaller.

8.5.1 Using D-MABNS as an improvement heuristic in a hyperheuristic

To evaluate D-MABNS performance on PVRP benchmark problems, we embed the D-MABNS algorithm in the improvement stage of the HyperILS framework (Ochoa and Burke (2014); **?**), as shown in Figure 8.14. The work presented here extends from Chapter 7, and it uses the same (re)initialisation and mutation moves in the *HyperPerturbation* stage.

To build the neighbourhoods for D-MABNS, we consider the same three following types of moves as introduced in Chapter 7: route modification, customer service pattern modification and mixed operators. Unlike the FI low-level heuristics employed by hyperheuristics developed in Chapter 7, all local moves are directly managed by D-MABNS. FSS is used,



(a) PVRP solution (b) HyperILS

Figure 8.14: Algorithm framework for solving the PVRP

as described in Table 8.3.

Table 8.3: Summary of local search moves for PVRP

Type	**Moves**	**Element sorting (FSS)**
Route related	2Opt, 3Opt, 2PS, Relocate, Cross	Edges sorted by length
Pattern related	Customer pattern reassign, Two customer pattern swap	Customers sorted by adjacent edge lengths
Mixed	Relocate with pattern, Cross with pattern	Edges sorted by length

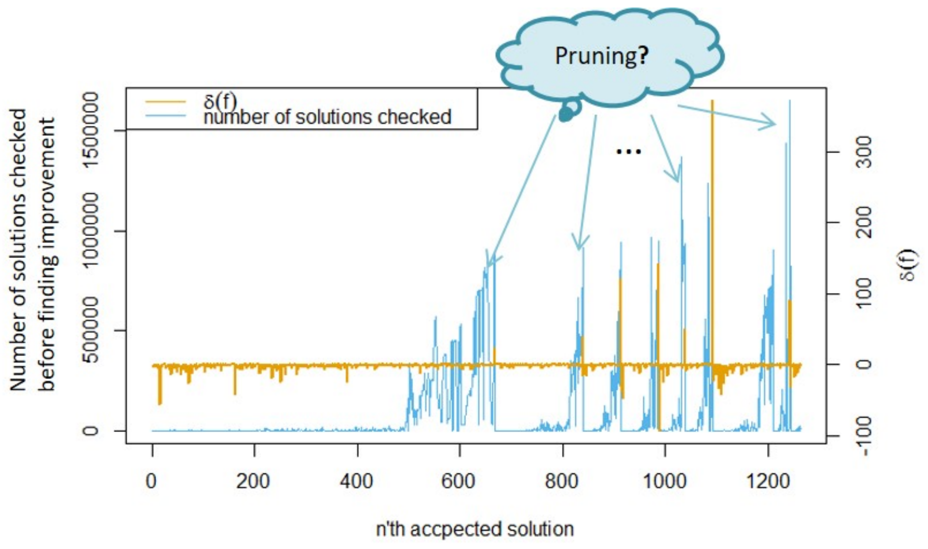
8.5.2 Computational results for PVRP

In this section, we discuss the behaviour of D-MABNS on PVRP, and compare results to D-MABNS on GDMP. In addition, we compare D-MABNS with the state-of-the-art meta-heuristics for PVRP. The same 42 PVRP benchmark instances tested in Chapter 7 are used to conduct the following experiments and analysis. A detailed description about the benchmark instances is given in Section 7.5.1. Also, we give a download link of the instance source files in the Appendix (Table A.2).

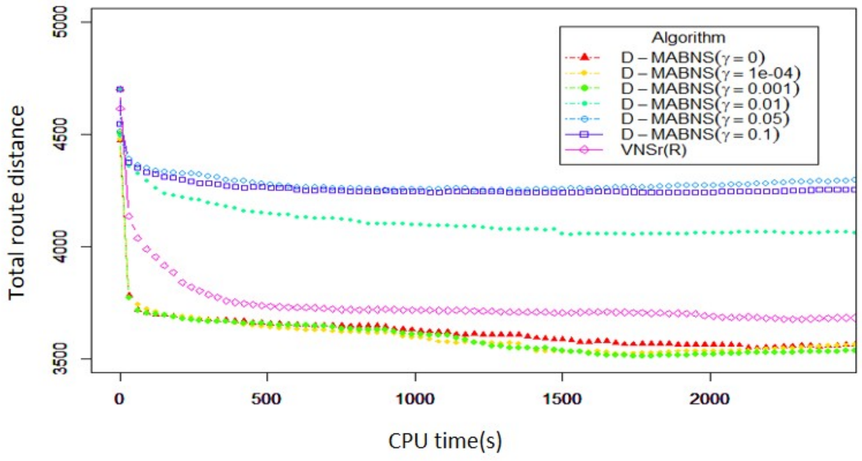
8.5.2.1 Sensitivity analysis

We conduct parameter sensitivity experiments, similar to those in Section 8.4.4.1. The best performing parameter settings for D-MABNS are used for performance comparison with other methods.

Compared to solving GDMP, neighbourhood pruning and sorting exhibit some different



(a)

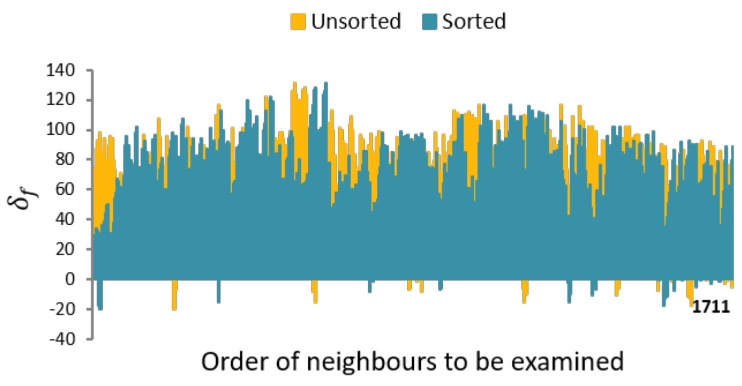


(b) D-MABNS with different γ

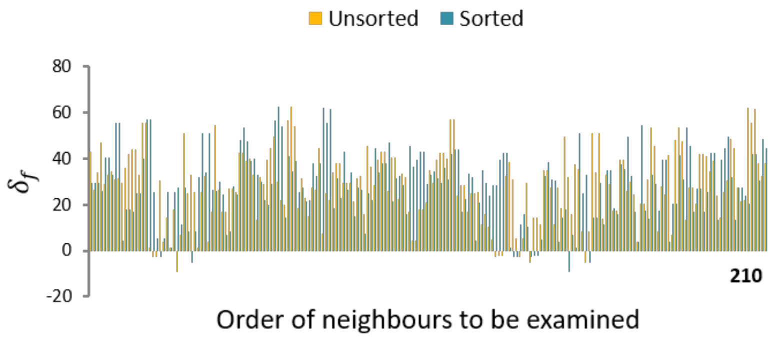
Figure 8.15: Impact of neighbourhood pruning on PVRP benchmark *p13*. Graph (a) records the changed fitness $\delta(f)$, and the number of solution examined before accepting the next solution, for D-MABNS ($\gamma = 0$).

behaviours. Figure 8.15(a) shows an example tested on PVRP instance *p13*, the largest benchmark containing 417 customers, of recording the number of solutions that D-MABNS ($\gamma = 0$) examines before it moves to the next solution. Every interval between two worsening solutions (shown as positive $\delta(f)$) is an *LS* process. We can see that, at a late stage of each *LS*, the cost to find an improvement increases significantly. When is the optimal time to prune the search within a neighbourhood? Tested on *p13* (Figure 8.15(b)), we can see the algorithm performance decreases significantly when the pruning parameter γ is bigger than 0.01. Because of early pruning, the algorithm cannot reach local optima and the final solution is of poor quality.

Similarly, sorting strategies do not give any obvious advantage in the PVRP solver. In



(a) Relocated neighbourhood



(b) 3Opt neighbourhood

Figure 8.16: PVRP sorting effects: examples of current-solution neighbourhoods

contrast to GDMP, the PVRP fitness landscape does not show obvious orderliness on the features we tried. Figure 8.16 illustrates two examples, from *Relocate* and *3-Opt* neighbourhoods.

Our experiments raise an important question: when is the appropriate moment to stop the *LS* and restart the journey somewhere else in the solution space? From our experience, we summarise the guideline rules for the same as follows.

1. For small problem instances, such as the PVRP benchmarks (between about 50 and 200 customers), the total time needed for each *LS* procedure is short. In this situation, no pruning strategy is needed.

2. In any problem where a sorting strategy does not help to guide the *LS* within neighbourhood structures, a pruning strategy is also unhelpful.

3. When the solution space is very big but a sorting strategy fails, memory techniques plus adaptive pruning rate can be applied to control exploration and exploitation in the early and the later stages of the search process. The memory technique needs to

record potentially promising areas of the solution space, which have been cut off in the early search.

4. When the solution space is big and the fitness distribution of many neighbourhood structures shows strong orderliness on some features, neighbourhood sorting and pruning bring big benefits to the search process.

8.5.2.2 Comparing to other meta-heuristics

We compare our results with the state-of-the-art meta-heuristics and the best results were achieved using VNSr(R) from Chapter 7[2]. We test MAB based algorithms including the Ran and D-MABNS with parameter settings $\{w = 400, RF = exp(a = 5), \alpha = 0.8, \lambda = 0.7, \gamma = 0.001\}$. The experiments are designed to replicate benchmark conditions from Vidal et al. (2012). In particular, the search is always terminated after a fixed amount of CPU time (shown in Table A.3, Appendix). We record the average results of 10 repeats of each instance and compare them against the best-found solutions in literature. The results are presented into two groups as some research works have only tested the instances from the "old data set". The average gap between our solution quality to the best-found solutions are summarised in Table 8.4 and are presented in more detail in the Appendix (Table A.3).

From Table 8.4, hybrid-GA (Vidal et al. (2012)), which combines the population based recombination operators and *LS* techniques, achieves the best performance among all algorithms. In addition, the specific design of infeasible solution management, population diversity and elite individual management have all made great contributions to locating the promising start points of successive *LS* procedures. Another successful development using hybridisation concept is the hybrid record-to-record method (Gulczynski et al. (2011)). The authors iteratively solve a customer reassignment problem using integer programming and improve the route length using record-to-record heuristic. However, compared to other recent approaches, the hybrid record-to-record takes almost double computational time. Parallel tabu (Cordeau and Maischberger (2012)) takes the advantage of parallel computing and enhance its performance significantly compared to its original development (Cordeau et al. (1997)).

In comparison, our two MAB-based algorithms achieve competitive results. On average, D-MABNS produces routes that are 1.7% longer than the best-known solutions. D-MABNS

[2]Compared to the version of VNSr(R) tested in Chapter 7, we improve its code implementation and rerun the experiments with 10 repeats.

has found 10 best solutions out of 42 tested instances and a new best solution for instance *p04*. The routes are shown in the Appendix (Table A.1).

Table 8.4: Summary of algorithm performance compared to the state-of-the-art meta-heuristics. Since not every algorithm is tested on all instances, results are grouped into old and new data set (see more detail in Section 7.5.6)

Method	**Author**	**Average CPU**	**gap (all instances)**	**gap (old data set)**	**gap (new data set)**
Record-to-record	Chao et al. (1995)	20.33min	-	2.72%	-
Tabu search	Cordeau et al. (1997)	4.28min	1.71%	1.59%	2.08%
Scatter search	Alegre et al. (2007)	39.93min	-	1.38%	
VNS	Hemmelmayr et al. (2009)	3.34min	1.34%	1.40%	1.14%
Hybrid-record-to-record	Gulczynski et al. (2011)	10.36min	-	0.91%	-
Hybrid-GA	Vidal et al. (2012)	5.56min	0.09%	0.12%	0.0%
Parallel tabu search	Cordeau and Maischberger (2012)	3.55min	0.23%	0.24%	0.20%
VNSr(R)	Chapter 7	5.56min	1.72%	1.53%	2.40%
Ran	Section 8.4.5.2	5.56min	1.56%	1.26%	2.63%
D-MABNS	Section 8.3	5.56min	1.70%	1.37%	2.90%

Comparing the average results of D-MABNS with *Ran*, in 29 out of 41 tested instances, *Ran* outperforms D-MABNS. Further analysis suggests that *Ran* reaches different local optima more than D-MABNS within given CPU times. Consequently, it increases the chances of finding better solutions. For GDMP, we found that the advantage of the pure random strategy was lost for larger solution spaces (Section 8.4.5), and we might predict a similar result for larger PVRP.

8.6 Conclusion

In this chapter, we introduce a new D-MABNS algorithm for our drainage system maintenance problem. To test D-MABNS, we introduce 5 GDMP instances that were obtained from the data of the real-world drainage system in Blackpool (Chapter 4). In order to solve this problem more effectively, the D-MABNS utilises the orderliness property of the neighbourhood structures and applies techniques, which focus the search in the promising areas of the solution space, such as dynamic neighbourhood updating, feature sequential search and neighbourhood pruning. We perform a comprehensive sensitivity analysis and gain insights into the relationship between FSS and neighbourhood pruning, showing that pruning is reliant on good sorting to gain big advantages. Compared to the two hyperheuristic approaches, BEBO (Chapter 6) and VNSr(R) (Chapter 7), D-MABNS achieves significantly better results in all the tested instances.

We then test our algorithm on 42 PVRP benchmark instances to compare its performance with other heuristic approaches found in recent literature. Unlike GDMP, the PVRP neighbourhood structures show no orderliness on the features that have been tested, which reduces the impact of D-MABNS's techniques during the search. However, D-MABNS still achieves very competitive results. On average, D-MABNS achieves solutions within 1.7% of the best-known solutions. After reviewing the techniques applied in the better performing algorithms, we would like to improve our implementation of D-MABNS from the following perspectives in future works: 1) allowing search to accept infeasible solutions; 2) enhance reinitialisation using recombination operators with the concept of elite solution management; 3) optimise code implementation.

In part II, a variety of techniques have been investigated to enhance the heuristic search algorithms for solving our large-scale combinatorial optimisation problem. We review existing efficient search strategies (Chapter 3); deliver experimental analysis of various hyperheuristic approaches (Chapter 7); propose a learning-based *LS* method (Chapter 8). According to the experiments from both Chapters 7 and 8, we observe the preference of algorithms depending on the problems and attempt to explain some of the behaviours. The major contribution is building the understanding of relationship between corresponding problem characteristics and search methods. Other researchers could use much of our advice presented in this part, when they design heuristic methods for their problems. Indeed, to comprehensively understand the relation, further systematic experiments should be designed using a lager variety of problems with more distinctive characteristics.

Part III

Large-scale Road Inspection Problem

Having considered modelling of the geographically distributed asset maintenance problem (GDMP) and having investigated various GDMP solvers, this part of the thesis focuses on a different problem, which is of another specific interest to Gaist Solutions Ltd. – the problem of road inspection.

Road inspection plays an important role in road management programmes. It provides the up-to-date key information of road conditions, which helps decision making on maintenance actions. Usually, road inspection is cyclically scheduled and done at a sufficient frequency. In order to improve work efficiency, two branches of research have proved fruitful. The first one is to optimise inspection frequency and policies based on the asset deterioration process (e.g. Madanat and Ben-Akiva (1994); Smilowitz and Madanat (2000); Kallen and Van Noortwijk (2006); Maji and Jha (2007)), and the second one is to optimise daily inspection routes with time constraints (e.g. Jha et al. (2008, 2010)).

In this study, we consider a third avenue for optimisation by comparing two road inspection strategic plans. Our business partner Gaist Solutions Ltd. plans to carry out a national-scale road inspection. Vehicles capable of high-quality video recording will be used for road inspection work. Two potential road inspection methods may be applied. First, using vehicles that can only get video data from one side of a road so that every road needs to be visited bi-directionally. Second, vehicles with more advanced equipment, which travel on one side of the road but monitor both sides at the same time in one-pass. Our task is to solve both routing problems in order to determine how much distance can be saved using one-pass inspection strategy in comparison to a bi-directional approach, and in turn determine whether the additional cost of one-pass equipment is justified.

It is worth noting here that there is little academic link to the work in parts I and II. According to the problem analysis and theoretical proof, we model the road inspection problem as an arc-routing problem the Chinese postman problem (CPP). The main contribution here is to answer the above question based on thoughtful theoretical analysis and calculation. A well-known exact solver (Edmonds and Johnson (1973)) for CPP is applied, which means an accurate analysis can be provided at the stage. To solve the large-scale CPP, we propose a pre-processing method (graph reduction strategy), which significantly reduces the computation time and allows analysis on previously-unsolved large graphs.

Chapter 9

A comparison of one-pass and bi-directional approaches applied to large-scale road inspection

9.1 Introduction

In the UK, local government agencies have a duty to maintain roads for public utility and safety. Road inspection is cyclically scheduled and done at a defined frequency, to support decision making on road repair scheduling. Modern inspection uses vehicles equipped with high quality video recording devices. The quality of recording is affected by obstructions such as parked cars, and complex post-processing is required to extract suitable data on road condition. With 240,000 kilometres of road in the UK, efficiency of the inspection process has a high priority.

We attempt to improve operational efficiency via the road inspection strategies. We investigate two strategies: a one-pass inspection, in which an inspection vehicle can monitor both sides of a road in one traversal, and the more traditional bi-directional inspection, in which the vehicle monitors only its near-side carriageway. Finding a suitable route is computationally expensive for real-world road networks: Gaist's network data for the UK cities of Blackpool, Southend, Manchester, Stockport, Halton, Warrington, and for the rural county of Norfolk, gives total road lengths of 515, 508, 1315, 945, 619, 879 and 26243 kilometres, respectively.

9.1.1 Road networks and representations

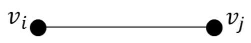
In the UK, local authorities are responsible for local road networks (excluding major trunk roads and motorways). The local authority road networks are mostly designated as 1-lane, 2-lane, 3-lane and 4-lane single carriageways or 2-lane dual carriageways (with a central reservation).

We represent the road network as an undirected graph $G(V, E)$. The vertices V represent junctions, dead ends, bends and any data collection point identified in the original data. The edges E represent roads that link the vertices. As Figures 9.1 and 9.2 show, 1-lane, 2-lane and 3-lane single carriageways can be represented by a single undirected edge. In the 4-lane single carriageway and dual carriageway situations, we transform the road into two parallel undirected edges. A crescent road is transformed into a loop (see Figure 9.3) and a cul-de-sac is represented as an one-degree vertex (see Figure 9.4).

(a) 1-lane single carriageway

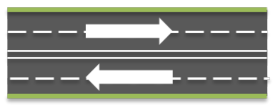
(b) 2-lane single carriageway

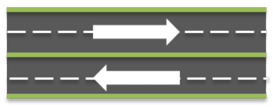
(c) 3-lane single carriageway

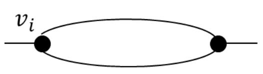


(d) Graph representation

Figure 9.1: Common road types in urban areas and the corresponding graph representation


(a) 4-lane single carriageway

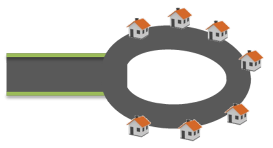

(b) 2-lane dual carriageway

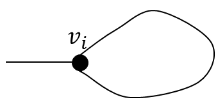


(c) Graph representation

Figure 9.2: 4-lane single carriageway, 2-lane dual carriageway and the corresponding graph representation

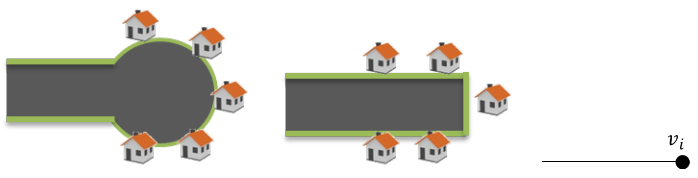
Our data comprises basic road information from seven UK local councils that are Gaist's


(a) Crescents



(b) Graph representation

Figure 9.3: Crescents and the corresponding graph representation



(a) Bulb (b) Dead-end (c) Representation

Figure 9.4: Cul-de-sac and the corresponding graph representation

clients. Because the data was not collected for network analysis, there are some accidental omissions and other issues to be addressed. To clean the data, all intersections have to be explicitly labelled as vertices.

The pre-processing required for any inspection route analysis is as follows. From consideration of the original data, we define $\varepsilon = 3$ metres.

- The first change made to the data is to remove 2-degree vertices, since these represent a bend in a road, rather than an intersection, and thus have no impact on the construction or distance of inspection tours. For all two-degree vertices, v_k, $e(v_i, v_k)$ and $e(v_k, v_j)$ are replaced by a single edge, $e(v_i, v_j)$, with length $l((v_i, v_j)) = l((v_i, v_k)) + l((v_k, v_j))$.
- We assume that an intersection has been omitted if the data indicates that two roads terminate close together. Thus, where the distance between two one-degree vertices is smaller than some ε, the vertices are merged.
- Similarly, we assume that a road that terminates very close to another road is an omitted intersection. Thus, where the distance from a one-degree vertex v_k to an edge $e(v_i, v_j)$ is less than ε, the edge $e(v_i, v_j)$ is replaced by two new edges $e(v_i, v_k)$ and $e(v_k, v_j)$.

Table 9.1 summarises the vertex degree-distribution of the seven datasets available, after data cleaning. As we can see, the majority of vertices in road maps have a degree of two, so their removal significantly reduces the size of the graph for each network.

9.1.2 Analysis of bi-directional and one-pass approaches

The inspection route for a one-sided inspection vehicle – which must pass along every road twice to monitor both sides – comprises a graph in which every edge is replaced by two arcs, so that all the vertices in the resulting digraph have equal in-degree and out-degree. The optimum route for inspection is an Euler tour of the resulting digraph and the optimum

Table 9.1: Distribution of vertex degrees, and total number of vertices before and after removal of 2-degree vertices for seven local authority road networks in the UK. Road network from left to right– B: Blackpool; SO: Southend; M: Manchester; ST:Stockport; H:Halton; W:Warrington; N: Norfolk

	B	**SO**	**M**	**ST**	**H**	**W**	**N**
Vertex total	26302	22864	45408	44470	29610	24518	549345
Degree							
1	3.40%	5.60%	8.32%	6.12%	6.08%	10.68%	2.62%
2	80.60%	80.00%	70.63%	80.51%	83.80%	69.53%	91.82%
3	13.10%	12.97%	19.75%	12.31%	9.72%	19.61%	5.33%
4	2.80%	1.42%	1.26%	1.03%	0.36%	0.18%	0.22%
5	0.03%	0.02%	0.04%	0.03%	0.017%	-	-
6	0.01%	-	-	-	-	-	-
less degree-2s	5103	4571	13337	8665	4796	7471	44912

distance is given by equation 9.1 (Even (2011)), in which $l(e)$ is the travelled length of the road segment represented by edge e.

$$l(\textit{bi-directional}) = 2\sum_{e\in E} l(e) \tag{9.1}$$

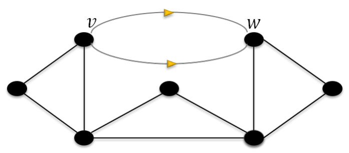
In the one-pass road inspection case, each edge in the graph has to be visited at least once. This problem can be modelled as the Chinese postman problem (CPP) and the total travel distance is the length of Chinese Postman Tour (CPT) (Guan (1962); Thimbleby (2003)).

$$l(\textit{one-pass}) = l_{CPT} \tag{9.2}$$

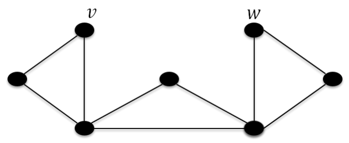
Lemma 9.1. *If a graph $G(V,E)$ is Eulerian and edge $\{v,w\}$ appears twice in E, then there is an Euler tour of G where $\{v,w\}$ is travelled in both directions $\{v,w\}$ and $\{w,v\}$.*

Since we generate an Eulerian graph from a given road network, Lemma 9.1 says that we can always find a CPT that passes complementary directions of parallel edges, and which is thus valid for 4-lane single and dual carriageways (e.g. Figure 9.2).

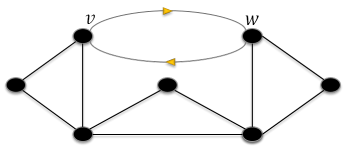
One-way streets make up a very small proportion of the total road distance in our networks, so we make the assumption that inspection vehicles can traverse roads in either direction, and that the effect of data-cleaning or data errors is similar in both the one-pass and bi-directional approaches, and thus has minimal effect on our analysis.



(a) Suppose that there is an Euler tour A=$v-w-x_1-x_2-...-x_n-v-w-y_1-y_2-...-y_n-v$, where edge (v,w) is travelled in the same direction both times.



(b) Then tour B=$v-y_n-y_{n-1}-...-y_1-w-x_1-x_2-...x_n-v$ is an Euler tour in the remaining Eulerian graph after remove the parallel edges $\{v,w\}$ and $\{w,v\}$.



(c) By adding a travelling path $v-w-v$ to the beginning of tour B, we have get another Euler tour $v-w-v-y_n-y_{n-1}-...-y_1-w-x_1-x_2-...x_n-v$, where edge (v,w) is travelled in both directions in the original graph.

Figure 9.5: Proof of Lemma 9.1

9.1.3 Solution of the Chinese Postman Problem and the challenge of large-scale problems

The optimal solution to the one-pass inspection problem is a CPT, in which a vehicle must visit every edge at least once whilst travelling the least overall distance (Guan (1962); Thimbleby (2003)).

For a general undirected graph, a CPT is derivable by adding the smallest possible number of edges to construct an Eulerian graph and finding an Eulerian tour based on it.

Edmonds and Johnson (1973) provide a widely-used exact CPP solution that is polynomial on the number of vertices and edges, as follows:

1. From an undirected graph $G(V,E)$, find the shortest path between all pairs of odd-degree vertices.

2. Find the minimum-cost perfect matching, M, of odd-degree vertices using the blossom algorithm (Edmonds (1965); Edmonds and Johnson (1973)).

3. Add extra edges that connect all the matched pairs of vertices through the shortest path in G.

4. Find an Eulerian tour in the resulting Eulerian graph.

Then, the length, l_{CPT}, of the identified CPT is:

$$l_{CPT} = \sum_{e \in E} l(e) + l(M) \tag{9.3}$$

The approach of Edmonds and Johnson (1973) does not scale well to large graphs, such as our road network representations. The first step of the approach requires calculation of the shortest path in G between every pair of odd-degree vertices. Floyd (1962) and others developed an algorithm of complexity $O(n^3)$, where n is the number of vertices, now known as the Floyd-Warshall algorithm (FW), whereas the most efficient implementations of Dijkstra's algorithm (Dijkstra (1959)), a single-source shortest-path algorithm, can achieve $O(m + n \log n)$, where m is the number of edges (Fredman and Tarjan (1987)). There is ongoing debate over the most efficient way to find the all-pairs shortest path in large-scale sparse graphs (Solomonik et al. (2013)); tests on our graphs show that FW systematically outperforms the other algorithms.

The second step of the approach by Edmonds and Johnson (1973), minimum-cost perfect matching, is also computationally expensive. The best-known implementation of the blossom algorithm achieves $O(n(m+n \log n))$ by Gabow (1990). More recently, Kolmogorov (2009) published an executable implementation which achieves time complexity of $O(n^2 m)$ – again, the complexity of the algorithm is dependent on the number of edges and vertices in the graph. This matching strategy is also used by Christofides (1976b) for the travelling salesman problem giving a worst-case ratio of 3/2 of the optimum tour length.

There are several efficient approaches for identification of an Euler tour, required for both one-pass and two-pass inspection routing. Fleury (1883) proposes the best-known algorithm, of order $O(m^2)$. However, we use another algorithm of order $O(m)$, proposed by Hierholzer and Wiener (1873).

Apart from Edmonds' CPP solution, Laporte (1997) introduces methods of transforming an arc routing problem into an equivalent TSP. This idea is also shown by Irnich (2008) to solve a large-scale real-world postman problem with complex constraints. Heuristics for the TSP can then be used to solve the transformed CPP problems. As a result, no optimal result is guaranteed to be found.

To analyse our large-scale real-world road inspection networks, we firstly propose a novel graph reduction process before finding the CPT (Section 9.2). Section 9.3 justifies the contribution of our graph reduction pre-processing and compares one-pass and bi-directional inspection strategies for the seven local authority road networks. Section 9.4

further tests our approach on three groups of simulated scenarios. The estimation of using these two inspection strategies on the entire UK road networks is presented in Section 9.5. Section 9.6 summarises the contributions of this chapter.

9.2 Finding optimal inspection routes

In this section, we describe how we apply the 4-step approach outlined above to our road network graphs. However, our first step is to reduce the graph, to make it more amenable to the computation of the CPT.

9.2.1 Graph reduction

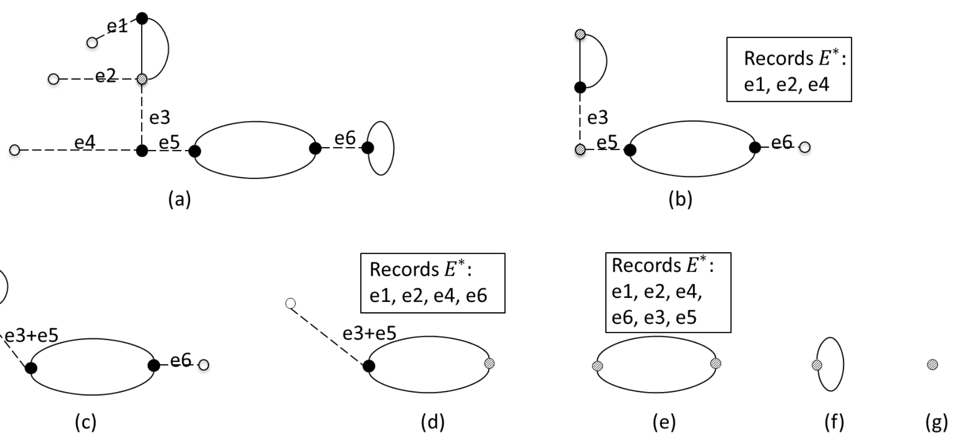

e1
e2
e3
e4
e5
e6
(a)
Records E^*: e1, e2, e4
e3
e5
e6
(b)
e3+e5
e6
(c)
Records E^*: e1, e2, e4, e6
e3+e5
(d)
Records E^*: e1, e2, e4, e6, e3, e5
(e)
(f)
(g)

Figure 9.6: Systematic reduction of an undirected graph. (a) the graph after the removal of degree-2 vertices, with the matching M_{odd} shown in dashed lines. (b) the graph after removal of degree-1 vertices, with their originally connected edges recorded in E^*. (c)-(g) the results of repeating these steps – here, the result is a null graph. White nodes have degree 1; striped nodes have even degree and black nodes have odd degree.

Our graph reduction applies graph contraction techniques as used in graph minor theory (Chartrand and Oellermann (1993); Lovász (2006)). Edge contraction is a fundamental operation in graph minors which deletes an edge from a graph G and merges the two end points. Here, we propose a novel graph reduction method to decrease the calculations time whilst maintaining the necessary characteristics of the original graph to reconstruct a CPT. After the data preparation described in the previous section, each road network is represented as a finite undirected graph which contains parallel edges and self-loops. All degree-2 vertices have been removed.

Let V_{even} and V_{odd} describe the even-degree vertex set and odd-degree vertex set, respectively, of our graph $G(V, E)$. $l(v_i, v_j)$ represents the length of the shortest edge between vertices v_i and v_j. If there is no direct connection between v_i and v_j, $l(v_i, v_j) = \infty$. For all vertices v_i, $l(v_i, v_i) = 0$. The shortest path between v_i and v_j in the graph G is represented as $p(v_i, v_j)$.

Our approach deletes vertices systematically, but records the length and location of removed edges in a structure, E^*. Figure 9.6 shows how this works on a stylised representation of a road network graph.

We make the following observations.

1. Deleting a self-loop from a graph does not change the parity of a vertex's degree.
2. The shortest path $p(v_i, v_j)$ between vertex v_i and vertex v_j does not include any self-loops.
3. The paths P of the minimum cost matching $M(V_{odd})$ include all the edges connected to degree one vertices. In other words, if you reach a dead end, then you have to get out the same way.
4. If a shortest path between vertex v_i and v_j is $p(v_i, v_j) = (v_i, v_{i+1}, v_{i+2}, ..., v_j)$, then the shortest path between v_{i+1} and v_j is $p(v_{i+1}, v_j) = (v_{i+1}, v_{i+2}, ..., v_j)$.
5. Deleting a degree-1 vertex and its adjacent edge, the total number of odd degree vertices is either unchanged or reduced by 2.

From these observations, the following two deductions can be made.

Deduction 1: Deleting a self-loop (v_i, v_i) from a graph G will not change the paths in the minimum perfect matching $M(V_{odd})$ of an undirected graph.

Deduction 2: There is a path set P of the matching $M(V_{odd})$ of the original graph G (as shown by dashed lines in panel (a) of Figure 9.6) that equals the deleted edges E^* connected to one-degree vertices (as shown by the E^* in Figure 9.6(b)), plus a path set P' in the new matching $M'(V_{odd})$ of the simplified graph G'(shown in Figure 9.6(b) as dashed lines), such that, $P = E^* \cup P'$.

Having reduced our road network graphs, we then apply the 4-step process, as follows.

Step 1: Finding the shortest distance between all odd-degree vertices

Our graph reduction process results in a simplified graph G', and a record of all the deleted edges which were connected to one-degree vertices in the reduction process, E^*.

To find the shortest distance between all pairs of odd-degree vertices of graph G', we use Floyd-Warshall (FW) algorithm (Floyd (1962)). The FW algorithm's complexity is worst-case $O(|V|^3)$, so reducing the number of vertices in the graph is advantageous. We find that, after applying the graph reduction process above, there are still many even-degree vertices in G', and the matching process does not need these vertices. Therefore, before calculating the shortest path, we can consider deleting these even-degree vertices.

There are many approaches to even-degree vertex deletion. Our preferred approach is Algorithm 9.2.1, which detects and deletes all even-degree vertices in G' without affecting the shortest connection and distance between other vertices. Deleting vertices with degree bigger than three may increase calculation complexity and the total number of edges in the graph. However, even where the number of edges increases, the total number of vertices is reduced. The time complexity of deleting each even-degree vertex $v_i \in V_{even}$ is $((m(v_i)(m(v_i) - 1)) / 2)$, where $m(v_i)$ is the number of edges connected to vertex v_i.

Having performed the additional reductions using Algorithm 9.2.1, we use the FW to calculate the shortest path between all vertices remaining in the graph.

Algorithm 9.2.1 Even-degree vertex selection and deletion. V and E are the sets of all vertices and all edges in a given graph G.

```
for each vertex $v_i \in V_{even}$ do
  for each pair of edges $e_p(v_k, v_i) \in E$ and $e_q(v_m, v_i) \in E$ do
    if $l(e_p) + l(e_q) < l(v_k, v_m)$ then
      generate a new edge $e'(v_k, v_m)$ with cost $l(e') = l(e_p) + l(e_q)$
      if $e(v_k, v_m) \in E$ then
        replace the edge between $v_k$ and $v_m$ with the new edge $e'(v_k, v_m)$
      else
        add the edge $e'(v_k, v_m)$ between $v_k$ and $v_m$
      end if
    end if
    delete edges $e_p(v_k, v_i)$ and $e_q(v_m, v_i)$
  end for
  delete vertex $v_i$
end for
```

Step 2: Minimum-cost perfect matching

The standard blossom algorithm (Edmonds and Johnson (1973)) finds the minimum-cost perfect matching, $M(V'_{odd})$ of graph G'.

The length of the minimum perfect matchings is represented as $l_{M(V'_{odd})}$. According to

Deduction 2 and Equation 9.3, the length l_{CPT} of the CPT of the original graph G is:

$$l_{CPT} = \sum_{e \in E} l(e) + \sum_{e \in E^*} l(e) + l_{M(V'_{odd})} \tag{9.4}$$

Step 3: Construct the Eulerian graph

Using Deduction 2 to construct an Eulerian graph from the original graph G, we only need to add edges recorded in E^* and $M(V'_{odd})$ to the original graph G.

Step 4: Finding the CPT in original graph G

From the graph produced in step 3, the Euler tour can be found by applying the algorithm proposed by Hierholzer and Wiener (1873).

To generate a CPT in a real world situation, when Hierholzer's algorithm meets a vertex connected to 4-lane single carriageway or dual carriageway edges, priority is given to the edge whose underlying direction is away from this vertex.

9.3 Experimental set-up and results of real-world data set

In order to justify the impact of our graph reduction process, we firstly introduce three heuristic methods to solve the CPP. We then run all three approaches plus the 4-step approach of Edmonds and Johnson (1973) on the road network graphs, firstly without our graph reduction, then on the graphs after our graph reduction method (Section 9.2.1), and finally on the further reduced graph with all even-degree vertices removed before the shortest distance calculation.

Our heuristic approaches focus on the matching process (*Step2*) and retain all the other CPP solving steps of Edmonds' method. Blossom and the first two heuristics labelled *greedy* and GLS (*greedy method with local search*) require the FW calculation of shortest paths between odd-degree vertex pairs. The final heuristic is a breadth-first-search (labelled BFS) for matching, and does not need FW to run first. The combinations of CPP solvers and graphs are labelled $m1 \ldots m12$, as shown in table 9.2.

The base-case minimum-cost matching is calculated using Kolmogorov (2009)'s implementation of the blossom algorithm. We now introduce the three heuristics that are proposed in place of the blossom matching algorithm.

Table 9.2: Summary of the three graphs and four methods applied. Cells (m1 - m12) provide the key to labelling of the later results and graphs. FW = Floyd-Warshall algorithm to find shortest distance between vertex pairs, step 1. blossom, greedy, GLS and BFS are tested in the matching process, step 2. GR = Graph reconstruction, step 3. HA = Hierholzer Algorithm to find the CPT, step 4.

pre-processing	**Steps 1-4**			
Graph	FW, *blossom*, GR, HA	FW, *Greedy*, GR, HA	FW, *GLS*, GR, HA	*BFS*, GR, HA
Road Network with 2-degree vertices removed	m1	m2	m3	m4
.. and reduction applied (Section 9.2.1)	m5	m6	m7	m8
.. and all even degree vertices removed	m9	m10	m11	m12

Greedy method (greedy): The greedy method is a heuristic that systematically constructs a matching where shortest distances between pairs of vertices are known, as described in algorithm 9.3.1. The algorithm is based on those by Kurtzberg (1962) and Reingold and Tarjan (1981).

Greedy method + local search (GLS): GLS attempts to improve the result of the greedy method by following algorithm 9.3.1 with a greedy first improvement heuristic, shown in algorithm 9.3.2.

Algorithm 9.3.1 Greedy construction for matching

```
$VL$ is a list contains all the vertices, $v_i$ in a given graph
while $VL$ contains at least two vertices do
    Choose the pair of vertices with shortest distance $(v_i, v_j) \in VL$
    Add $(v_i, v_j)$ to the matching $M$
    delete $v_i, v_j$ from $VL$
end while
Return $M$
```

Breadth first search (BFS): BFS is a basic search: each unmatched odd-degree vertex v_i in graph G is the root point of a BFS to find the next unmatched odd-degree vertex v_j. Then two vertices v_j and v_i are a matching pair and the path from v_i to v_j in the BFS tree is the matching path. The BFS terminates when there are no unmatched odd-degree vertices left.

Algorithm 9.3.2 Greedy improvement algorithm, used to improve the matching result of the construction approach algorithm 9.3.1. $l(m_i)$ is the distance between the two vertices in the matching m_i.

```
improved=true;
while improved do
  improve=false;
  for Every pair of matchings $m_i,\ m_j\ \in\ M$ do
    Generate new matchings $m_k, m_l$ by exchanging vertices between $m_i$ and $m_j$
    if $l(m_k) + l(m_l) < l(m_i) + l(m_j)$ then
      Replace $m_i$ and $m_j$ by $m_k$ and $m_l$;
      improved = true;
      break;
    end if
  end for
end while
Return $M$
```

9.3.1 Results comparison

Our experiments allow us to address two questions:

- is there a computationally-efficient (in terms of CPU time) solution to the CPP on large scale general graphs?
- how much distance can be saved using one-pass inspection strategy in comparison to a bi-directional approach?

The second question is of particular importance to the local authorities concerns.

To compare the computation time required to identify a CPT (inspection route) for each of the different graphs and variant approaches, the experiments were each run on a standard desktop PC: Intel i7-3770 CPU at 3.40 GHz with 24GB memory under the Windows 7 operation system.

The exact methods using blossom algorithm in the matching process achieve the optimal CPP tour, whereas heuristic based methods can only find near-optimal solutions. Therefore, we analyse question two using the results generated by the exact methods. Equation 9.5 is applied to normalise the difference and express it as a percentage distance saving:

$$saving = \frac{l(\textit{bi-directional}) - l(\textit{one-pass})}{l(\textit{bi-directional})} * 100\% \tag{9.5}$$

The 12 experimental set-ups were run for each of the local authority road network graphs except Norfolk. The graph representation of the Norfolk road network is too large to process without our novel graph reduction step.

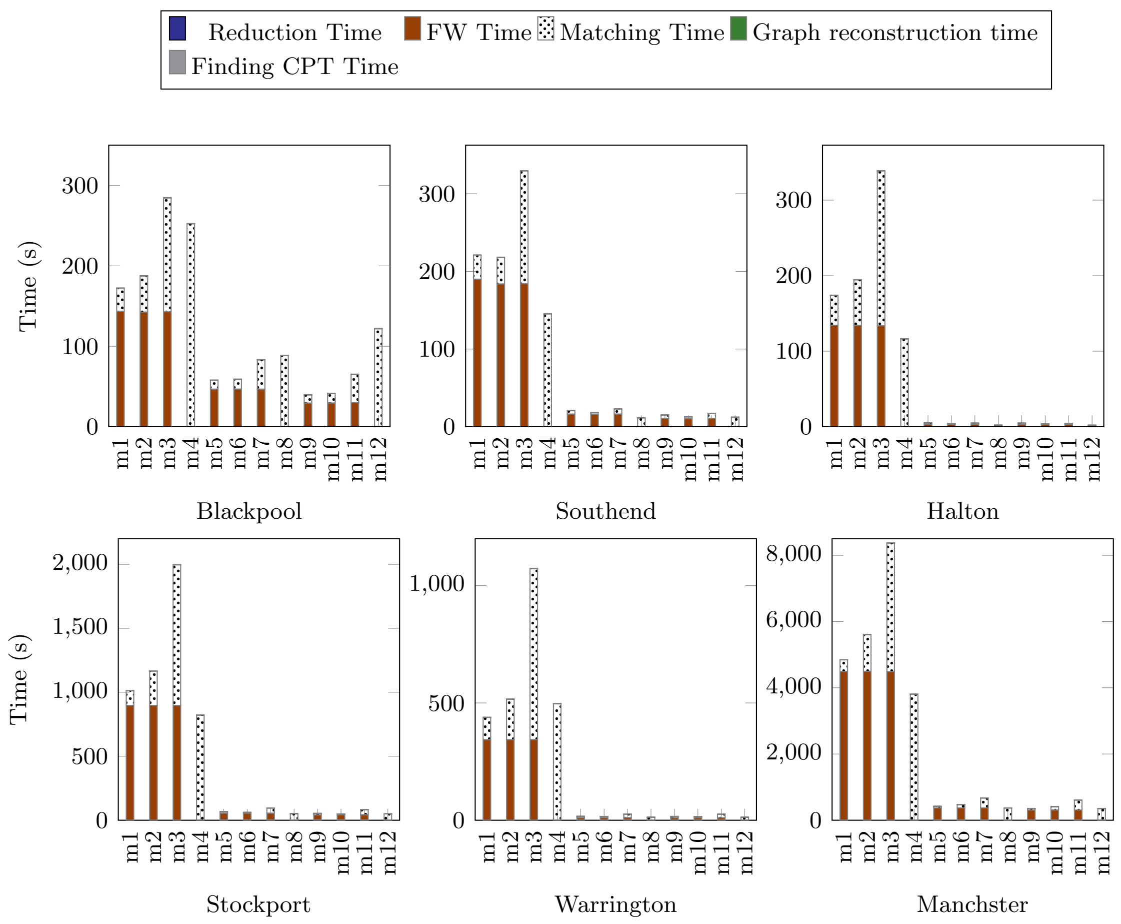


Figure 9.7: Results of the 12 methods for the six road networks. For each experiment, the bar shows the CPU time taken, with shading showing the CPU usage of each algorithm (graph reduction and graph reconstruction take negligible time and are not visible in the plots). m1 to m4 are tested on the original graphs; m5-m8 are tested on the reduced graphs; m9-m12 are tested on further reduced graphs that without any even degree vertices.

9.3.1.1 CPU time comparisons

Figure 9.7 plots the CPU time taken. For each road network graph, the graph reduction and graph reconstruction times are negligible, and do not show up on this scale.

In all road networks, the methods applying graph reduction (m5 to m12) shows much lower overall CPU time, which demonstrates the importance of our graph reduction approach on graphs of this scale. After graph reduction, the exact methods can produce the CPT faster than any tested methods directly running on the original graph in all cases.

Table 9.3 gives numerical results for the blossom algorithm experiments on the three forms of graphs for all seven local authority road networks (except for the un-reduced Norfolk graph). The numerical results again emphasise the reductions in CPU time due to graph reduction.

Some explanation is needed of an apparent anomaly in the Blackpool data. In the top left panel of Table 9.3, the number of edges (column m) is significantly higher for the fully-reduced graph (third row) than for the unreduced or partially reduced graphs. This arises from the removal of all even vertices in this network graph. In the Blackpool network, almost 3% of vertices have degree 4 or above, whereas no other network has more than 1.5% (Table 9.1). Whilst removal of degree-one and degree-two vertices always decreases graph complexity in terms of both the number of vertices and edges, removal of higher-degree vertices reintroduces significantly more edges. Fortunately, the CPU-hungry algorithms depend more strongly on the number of vertices.

Table 9.3: CPU time taken to calculate the CPT of each road network using the blossom algorithm for matching. The lower right panel gives the total CPU time for each road network for each experiment. The columns are as follows: n is the number of vertices (also shown in Table 9.1), and m the number of edges in the graphs after data cleaning, etc. RT is the CPU time for graph reduction: no reduction, reduction as described in section 2.1, and the additional reduction in Algorithm 9.2.1, respectively. FW is CPU time for the Floyd-Warshall algorithm. MT is the CPU time for the blossom matching algorithm. CT is the CPU time to construct the final graph. CPT is the CPU time to extract the final inspection route using the Hierholzer algorithm.

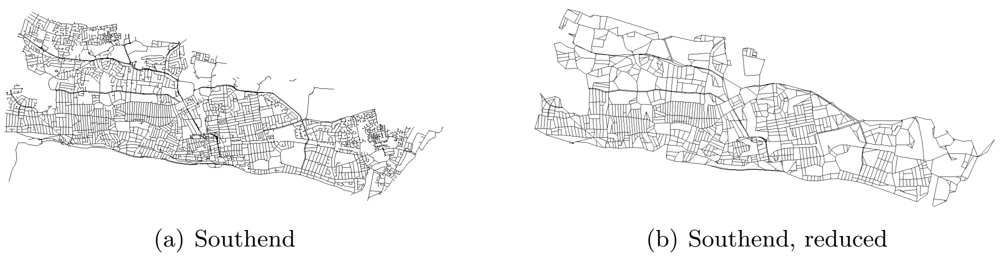
	Blackpool (B)							**Southend** (SO)						
	n	m	CPU (s)					n	m	CPU (s)				
			RT	FW	MT	CT	CPT			RT	FW	MT	CT	CPT
m1	5103	7124	0	143.32	28.97	0.016	0.187	4571	5738	0	189.84	31.14	0.001	0.202
m5	3398	5419	0.14	45.54	11.11	0.016	0.234	2016	3162	0.14	16.05	4.26	0.001	0.203
m9	2772	10803	0.51	28.84	10.34	0.016	0.016	1746	5922	0.28	10.51	4.25	0.001	0.202
	Halton (H)							**Stockport** (ST)						
	n	m	CPU (s)					n	m	CPU (s)				
			RT	FW	MT	CT	CPT			RT	FW	MT	CT	CPT
m1	4796	5430	0	134.38	39.44	0.002	0.187	8665	10482	0	910.62	114.21	0.000	0.671
m5	1211	1852	0.15	2.96	1.97	0.001	0.218	3277	5063	0.45	55.91	12.46	0.006	0.826
m9	1148	1970	0.16	2.53	1.96	0.002	0.218	3004	5662	0.53	40.86	11.79	0.003	0.858
	Warrington (W)							**Manchester** (M)						
	n	m	CPU (s)							CPU (s)				
			RT	FW	MT	CT	CPT	n	m	RT	FW	MT	CT	CPT
m1	7471	8556	0	343.18	95.32	0.001	0.499	13337	16500	0	4438.04	355.83	0.013	1.58
m5	2136	3218	0.29	10.31	5.84	0.000	0.561	5842	8949	1.02	369.71	47.75	0.015	1.79
m9	2108	3265	0.31	9.78	5.83	0.001	0.561	5486	9622	1.12	306.29	46.71	0.015	1.67
	Norfolk (N)							Summary						
	n	m	CPU (s)					Total CPU(s)						
			RT	FW	MT	CT	CPT	B	SO	H	ST	W	M	N
m1	44912	53482	0	-	-	-	-	172	221	174	1025	439	4795	-
m5	15647	23904	18.27	4977.19	5499.49	0.171	34.523	57	21	5	70	17	420	10530
m9	14796	26487	19.66	4227.01	4759.01	0.182	34.881	49	15	5	54	16	356	9041

9.3.1.2 Distance comparisons

To compare totally distance vehicles should travel using one-pass versus bi-directional road inspection approach, table 9.4 presents details of distance savings in these experiments. The distance saving for the one-pass inspection is between 26% and 35%.

Table 9.4: Distance and distance saving of the CPT one-pass route, compared to the bi-directional route monitoring approach. Results are generated using the exact solver (the 4-step approach of Edmonds and Johnson (1973)).

	Bi-directional (Euler Tour)	**One-pass (CPT)**	**Saving distance & percentage (equation 9.5)**		**Average degree**
Blackpool	1031km	671km	360km	34.86%	2.79
Southend	1016km	676km	340km	33.46%	2.51
Manchester	2631km	1839km	792km	30.10%	2.47
Stockport	1891km	1369km	522km	27.57%	2.42
Norfolk	26234km	18268km	7966km	30.36%	2.39
Halton	1239km	917km	322km	26.03%	2.26
Warrington	1758km	1300km	458km	26.00%	2.29


(a) Southend (b) Southend, reduced

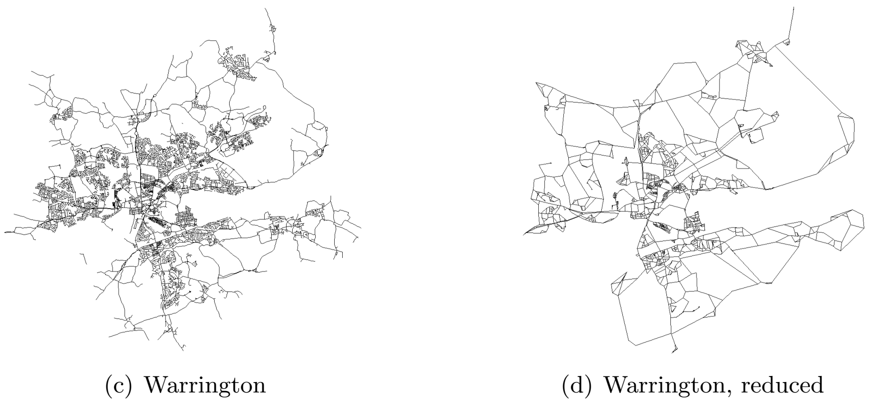

(c) Warrington (d) Warrington, reduced

Figure 9.8: Original maps and reduced graphs for Southend and Warrington road networks, illustrating grid-like and tree-like road networks.

Investigation of the differences in distance saved shows that, in addition to the added complexity of networks with high-degree vertices (noted above, and shown in the final column of Table 9.4), these reflect different road network topologies. Blackpool, Southend, Manchester and Norfolk, can be characterised as having predominantly grid-structured networks, which are conducive to efficient one-pass monitoring. In contrast, Warrington and Halton have predominantly tree-structured road networks which inevitably leads to visiting more streets twice, even in the one-pass case. To illustrate this, Figure 9.8 gives the original and reduced-graph networks for Southend and Warrington.

9.4 Experiments of simulated data set

Our analyses of graphs representing real road network data lead to striking conclusions about the distance saving of one-pass, as compared to bi-directional, road inspection strategies. Although savings vary by about 10 percentage points across different road networks, the distance savings were consistently above 26%.

To explore the interaction between graph layout and distance savings, we conducted a set of experiments on randomly generated graphs using the blossom-based approach that was shown to be optimal above.

Table 9.5: Vertex degree distributions for generated graphs. All generated graphs have 1000 vertices and edges of length 1 only.

	parameters						
	vertex degree					**average degree**	
Graph structures	1	2	3	4	5	initial	no d-2
degree distribution match:							
g1	15%	70%	12%	3%	-	2.03	2.1
g2	10%	70%	15%	5%	-	2.15	2.5
g3	5%	80%	10%	5%	-	2.15	2.75
degree distribution with no degree-2 vertices:							
g4	35%	-	60%	5%	-	2.35	2.35
g5	35%	-	50%	15%	-	2.45	2.45
g6	35%	-	40%	25%	-	2.55	2.55
g7	35%	-	40%	10%	15%	2.7	2.7
g8	35%	-	40%	5%	20%	2.75	2.75
dominated by high-degree :							
g9	20%	-	15%	60%	5%	3.3	3.3
g10	20%	-	15%	5%	60%	3.85	3.85

Random graphs were created using an algorithm proposed by Blitzstein and Diaconis (2011). Graphs are created with a fixed number of vertices (we choose 1000) to specified vertex-degree distributions. Our vertex-degree distribution parameters are shown in Table 9.5, which also summarises the average vertex degree of the generated graphs, before and after the initial removal of degree-two vertices. Three parameter settings of random graphs (g1 – g3) are generated with similar vertex-degree distribution to the graphs representing our real-world road networks. A further five parameter settings of random graphs (g4 – g8) have similar vertex-degree distribution to the graphs of real-world road networks after cleaning to remove degree-two vertices. There are also two parameter settings of random graphs that are dominated by degree-four (g9) and degree-five (g10) vertices. In the random generation, all edge lengths are set to one, and no attempt is made to generate graphs with particular structural characteristic (grid-like, tree-like). Our experiments focus on the generalisation of the influence of vertex degree, only.

For each group of graphs, we run these three sets of experiments with progressively greater reductions, dictated by our data cleaning and graph reduction approach to the road network graphs: 1) data cleaning to remove degree-two vertices, 2) graph reduction to remove all degree-one and degree-two vertices, and 3) further reduction to remove all even-degree vertices. For each experiment, we report the average of 30 runs, a number selected to give acceptable total run time but a suitably-low statistical error. Here, the approach that applies blossom algorithm in the matching process is used.

9.4.1 CPU time results

Table 9.6 presents details of the CPU time taken (using the computational platform described in Section 9.3.1) for the overall CPT identification process (vertex-pair distances using FW, blossom matching, graph reconstruction and CPT identification), and then for the three levels of data cleaning and graph reduction, on each of the 10 types of graph.

Each graph reduction makes a big contribution to CPU time reduction. The results for the first group of graphs, those generated to match the vertex degree distribution of the complete road network graphs, shows the importance of removing degree-two vertices in this respect. A further large time saving occurs when removing degree-one vertices – from Table 9.6 we can see that these make up 50% of the vertices in the example g1 graph after removal of the degree-two vertices. Comparing the improvement in CPU times between the full graph reduction (last column of Table 9.6) and the data-cleaning reduction of removing only degree two-vertices (middle column total), the results for this first group of graphs show CPU time savings of 95.5%, 90.3% and 78.5%, respectively.

The graphs in group g4 – g8 (and, indeed, g9 and g10) are generated without degree-two vertices. For these graphs, the graph reduction steps also show a very large CPU time reduction over the un-reduced time.

For the graphs dominated by higher-degree vertices, g9 and g10, the results again show that graph reduction leads to time reductions, but, for g9 which is dominated by degree-four vertices (60% of all vertices), CPT extraction with removal of all even vertices is almost three times the CPU time for just removing degree-one vertices, and somewhat greater than the CPU time for cleaned graphs with neither subsequent reduction step. In this

Table 9.6: CPU time for CPT tour extraction on generated random graphs g1 – g10, showing the effect of graph reductions. The notation "reduction" represents the CPU time used to reduce a graph and "total" denotes for the overall CPU time taken to produce the CPT of an original graph. Lowest overall CPU times for each graph type are in emboldened.

	CPU(ms)						
Graph structure	**FW, blossom GR, HA**	**removing degree 2**		**and removing degree 1**		**and removing all even degree**	
	total	reduction	total	reduction	total	reduction	total
g1	19691.82	10.66	577.56	11.61	29.49	12.07	**25.91**
g2	19616.67	10.21	568.72	11.36	86.71	12.93	**54.86**
g3	19522.89	11.30	183.39	11.75	60.01	15.18	**39.37**
g4	20464.62	-	-	6.34	1831.37	7.76	**1436.3**
g5	20312.60	-	-	6.22	2086.82	14.64	**1062.11**
g6	20582.34	-	-	5.55	2528.90	863.95	**1666.97**
g7	20433.91	-	-	4.88	2780.43	13.07	**1414.97**
g8	20359.70	-	-	4.87	2810.04	10.99	**1622.76**
g9	20339.90	-	-	3.88	**8468.91**	248918.2	249369.4
g10	20407.10	-	-	3.68	9209.61	17.08	**4992.87**

case, the last column of Table 9.6 shows that there is a very large CPU time overhead for deleting the significant number of degree-four vertices and any other higher even-degree vertices. In contrast, the graphs dominated by degree-five vertices, g10, show a pattern that is consistent with the graphs that are similar to the real road network graphs.

Further analysis of specific graph results shows how these CPU time savings arise, since the FW and the blossom matching process are the most CPU-intensive parts of the CPT extraction.

Table 9.7: Degree distribution after each graph reduction for a typical g1 graph

	Vertex degree				**Total vertices**
	1	**2**	**3**	**4**	
Original graph	15%	70%	12%	3%	1000
Delete degree 2	50%	-	40%	10%	300
Delete degree 1,2	-	-	78%	22%	84
Delete degree 1,even	-	-	-	-	66

The complexity of FW is dependent on the number of vertices in the graph. Table 9.7 shows that degree-two removal (data cleaning) removes 70% of vertices, and subsequent graph reduction reduces 300 vertices successively to 84 then 66 vertices on which to run FW. It is the odd-degree vertices that influence the CPU time of blossom algorithm matching (step 2) – there are 270 odd-degree vertices in this example of a g1 graph, but we have the much smaller number, 66, of odd-degree vertices after removal of degree-one and all even-degree vertices. Thus we can conclude that the graph reduction contributes both to the reduced running time of the FW, shortest-path-between-pairs calculation, and to the reduced running time of matching process.

9.4.2 Distance saving results

Figure 9.9 shows the average distance savings (calculated using equation 9.5) and variances about the mean from our 30-run sets. As in the road network graphs, the greatest distance savings are associated with higher average degree graphs. The pattern also shows within groups of graphs with similar degree distribution (identified by shading in Figure 9.9).

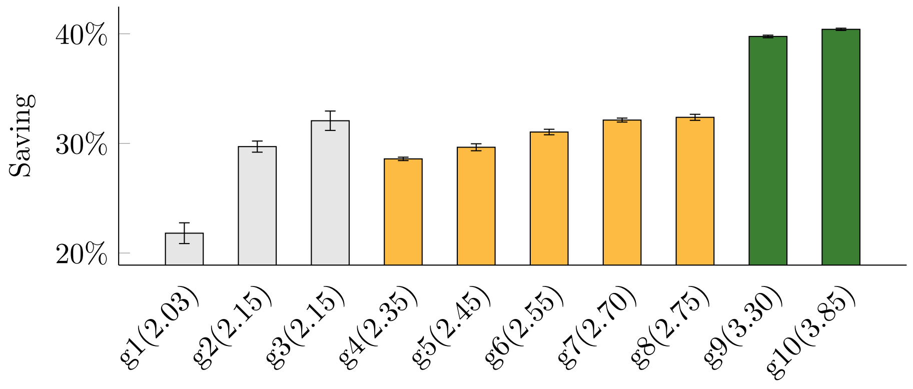


Figure 9.9: Distance saving (equation 9.5) for the one-pass route compared to bi-directional routing on the randomly generated graphs g1 – g12. For graph characteristics, see Table 9.5. Numbers in brackets are the average vertex degree for that graph. Error bars show one standard deviation of distance saving results.

These results on randomly generated graphs support our general observation of significant distance savings for a one-pass CPT over a bi-directional inspection tour, even though our generated graphs ignore the impact of different graph topologies and edge lengths. The only distance saving that is worse than the 26%-plus savings on the road network graphs is for g1, those graphs that have an average degree of close to two before and after removal of degree-two nodes. Comparing to other graphs, g1 graphs also have the highest proportions of degree-one vertices after data cleaning (removing degree-two vertices) – dead-ends that have to be traversed twice in order to continue a tour.

9.5 Estimated cost savings of one-pass road monitoring

In co-operation with our business partner, Gaist Solutions Ltd., we can estimate the national savings of the one-pass strategy. Across all the results for the seven UK local authority road networks, we find a distance-saving of roughly 30%. If we assume that the seven case studies are typical of the UK, then we can use this to estimate the total annual saving for UK local authority spending on road inspection using a one-pass strategy. We use published data, rounded to avoid a false appearance of precision in the estimates.

The whole UK road network under local authority control (i.e. without trunk roads and motorways) is roughly 240,000 kilometres (Murphy (2014)), broken down to 30,000 km of major roads and 210,000 km of minor roads. The UK highway management authority states that road inspection should cover all major and about one-third of minor roads each year (Department for Transport (2005)).

Extrapolating from our seven examples, we can estimate that a bi-directional strategy requires annual inspection route distances totalling roughly 60,000 km of major roads (twice the total distance) and 140,000 km minor roads (twice the total distance of one-third of the routes each year).

If a one-pass strategy results in a 30% distance saving, then the inspection route distance reduction would be roughly 18,000 km for major roads and 42,000 km for minor roads. According to the cost information provided by our industry partner, Gaist, we can estimate a cost saving using a linear function that includes labour, petrol, equipment and vehicle maintenance costs. This gives an approximate cost of £30 per km of major road and £90 per km of minor road. The cost is higher for minor roads because running speed is lower, and there are more likely to be obstructions and blockages. We can therefore tentatively suggest an annual saving across all UK local authorities of $30 * 18000 + 90 * 42000 = \pounds 4.32$ million.

9.6 Conclusion

This chapter explores the potential benefits for road inspection of using inspection vehicles that can collect both side of road-lane information of a single carriageway in one traversal, as opposed to more traditional single-sided monitoring that has to use a bi-directional strategy to monitor every road.

By making various assumptions, we can systematically clean the map-based route data to replace omitted intersections. We create road network graphs from the seven UK local authority road network data sets, cleaning this data to remove degree-two vertices that simply record bends in the road. A novel contribution of our work is to introduce a graph reduction technique. It is notably helpful on sparse graph like real-world road networks; even the graph of a UK county road network is amenable to CPT route calculation on a standard PC within reasonable CPU time. For road networks in residential areas, the reduction helps to manage the many branch roads, close road and culde-sacs which lead to complex graph structures.

Similar to the drainage system maintenance study (Part I), once again, the real-world problem study shows the necessity of data pre-processing for the use of real data, to fit specific analysis purpose. Furthermore, to deal with large scale problems, experiences from solving the drainage system maintenance and the road inspection demonstrate that it is worth while to design data simplification and problem size reduction before solving the problem.

Comparing to heuristic based solvers studied in previous chapters and applied to the large scale drainage system maintenance problem (Chapter 6), it is worth noting that an exact CPP solver is applied in this chapter to deliver various analysis. The standard CPP is solvable in polynomial time and there is an efficient exact solver (Edmonds' solution) for problems at the scale that we are dealing with. Our experimental results also support the effectiveness of Edmonds' solution compared to other simple heuristics.

We have also presented initial results on randomly generated graphs that allow us to identify what sorts of road network graphs generate most savings – CPU time use is related to the number of vertices in graphs, but the cost of graph reduction exceeds that saved on computation in the pathological case where most vertices are of degree-four. In this situation, we can apply only the process of degree-1 and degree-2 removing (Section 9.2.1).

In general, road network results show that the greater average degree a graph has, the greater the route distance savings that can be generated. The results also show some influence from graph structure, with better improvements in grid-like structures than tree-like structures.

We present conclusive evidence that the one-pass approach offers significant savings over the bi-directional approach to road inspection. This assumes that the effects of data cleaning (which may wrongly insert new intersections, and may not be able to identify some missing road sections) are similar on both the one-pass and bi-directional tour calculations.

In terms of the answer to the question from Gaist, at this stage, we have an overview of how much cost saving could be made by using a one-pass inspection strategy instead of a bi-directional strategy. Gaist is now using our approach, including graph reduction in planning its national scale road inspection programme. However, there are more practical issues to address in future works, such as the number of one-pass inspection vehicles required to inspect the network, and identification of optimal starting points for each vehicle tour.

Chapter 10

Thesis conclusions and future work

This thesis discusses real-world asset maintenance scheduling problems arising from highway management. Three parts are included. In Part I, in co-operation with Gaist Solutions Ltd., we specifically consider a drainage system maintenance problem and deliver an automated scheduling process to replace the current manual approach. Due to the high computational complexity in solving the large-scale optimisation problem, we focus on the solution approach improvement in Part II. Part III talks about local authority road inspection, which is another problem that is specifically relevant to the interests of Gaist Solutions Ltd.. This chapter summarises each of the parts mentioned above.

Part I: Geographically distributed asset maintenance

The research in part I is motivated by a real-world large-scale drainage system maintenance scheduling problem. Based on problem analysis and assistance from our business partner, Gaist Solutions Ltd., we built our understanding of geographically distributed maintenance problems (GDMP) step-by-step, and we realised that many public asset maintenance problems show similar characteristics. At a high level, a GDMP has a set of geographically distributed assets that require maintenance service for a long period or over a continuous period of time to perform their designed functions. Each asset gradually degrades over time and the objective is to make decisions regarding how to deliver maintenance service optimally to such a system while using limited resource.

Further analysis leads the study to focus on subjects within vehicle routing problems, periodic vehicle routing problem (PVRP) in particular. PVRP is the closest standard model that captures some of the properties of our GDMP, including the geographically distributed points and the specified service pattern for each point within the planning horizon. However, we do not always have clear service date information in all cases due to reasons, such as:

1. Large-scale system and different degradation processes of each of the assets in the system (e.g. affected by local environments). It is too hard to specify the service pattern for each individual asset in the system.
2. Dynamic degradation process in which factors such as season and weather changes may affect the assets' degradation process.

In these situations, the question is to find the optimal date to service each asset and the optimal daily service routes. More complex situations, such as dealing with multiple service types (i.e. preventative, corrective maintenance) and sudden disasters are involved in real-world scenarios. To solve this problem, we propose the following:

1. We propose a predictive scheduling strategy operating on a rolling planning horizon, and transform the research question to find the optimal date to deliver service and to identify the assets that need maintenance services the most in the short planning horizon in the near future.
2. We apply a function-based asset lifetime estimation method, Weibull distribution (Weibull (1950)), to estimate the condition of each gully pot in the drainage system at any day. Local environment and seasonal information is considered in the lifetime estimation. A further discussion is given regarding other asset lifetime estimation approaches, which can be utilised in other asset maintenance scenarios.
3. We employ a risk driven model in the predictive scheduling strategy to guide the service despatching towards assets, which need services soon.
4. Within the predictive scheduling solver, the service routes are also optimised with respect to the total travel and service time.

According to our simulation-based analysis, we suggest a large reduction in the daily surface water flooding risk, by about 17% in Blackpool's drainage system, if we replace the current manual scheduling strategy with the predictive scheduling strategy.

The simulation-based analysis has also helped to investigate the potential effect of two investments to improve the manual maintenance quality, which are"banning parking" policy and improving the timeliness of information, as might arise from the use of a low-cost wireless sensor technique. Using simulation-based analysis, we see that a "banning parking" policy might improve maintenance quality to some extent, and that "untimely information" is a significant factor in lowering the efficiency of maintenance. There are still many challenges for the sort of sensor technique (See et al. (2012)), which is needed to provide timely information in practice. Our simulation results suggest that a full-sensoring drainage system can bring about a large risk reduction by about 92% and cope with up to 30% false positive and false negative error information.

To set up the simulations, we closely worked with Gaist Ltd. and local councils in the UK to provide the best data. However, even then limited data was available. Even though our analysis uses simulations, which may not be accurate for realistic situations, the proposed maintenance strategy and the risk analysis provide insights into potential ways to improve the current practices on drainage system maintenance in the UK. The contribution of this thesis is to increase our ability to model and analyse GDMPs.

Discussion and future work In this research, we aim to link drainage maintenance with city flooding risk management. Surface water flooding (SWF) normally occurs when intense rainfall is unable to enter a drainage system. This may be because of drainage blockages, breaks or if the draining capacity has been exceeded. In the first two cases, good drainage maintenance should make a big contribution to the SWF risk management.

Reviewing the risk driven model applied to the drainage maintenance problem, in order to evaluate the daily SWF risk caused by drainage failure, we analyse the historical flooding frequency in the Blackpool area to derive the *average daily* risk impact information r_i for each gully pot i in the system. Then the risk estimation ($r_i P_i(d)$) is used to produce optimised maintenance scheduling. ($P_i(d)$ represents the probability of gully i being blocked on day d). This approach uses historical information, which may lead to the risk estimation for future days to become less accurate. In addition, for high risk flooding areas with natural water resources (i.e. river, coast), it is important to integrate additional information (e.g. water level) to adjust emerging maintenance actions if necessary.

To improve the adoption of the risk driven model on the drainage maintenance problem, instead of using historical information, we could utilise weather forecast information to assist in forming more accurate risk estimation. In more detail, the flooding risk caused by each gully pot i is the product of its direct risk impact q_i and the probability of flooding happens there due to gully failures, denoted as F_i. Here, direct risk impact measures the estimation of total value loss if floods happens. Further work is required to come up with a function $F_i(P_i(d), l_{rain}, b_{risk})$, which defines the flooding probability based on the information of the probability of a gully pot blocked $P_i(d)$, the level of rainfall l_{rain} estimated in future days, and the flooding risk base level, which grades the gullies based on its historical likelihood of general flooding (e.g. distance to a natural water source).

As time passes and knowledge gain increases, we can improve the quality of our risk estimates and thus our predictive scheduling quality.

We give two further suggestions when adopting the predictive scheduling strategy and similar analysis, when it comes to general real-world applications. They are:

- For any data-supported decision-making process, it is worth noting that we cannot spend too much effort to complete an accurate survey and collect more data resource. Accurate and sufficient data is critical for effective and efficient maintenance scheduling.

- When dealing with large-scale problems like the drainage system maintenance case study from Gaist, problem simplification techniques, such as grouping gullies and pre-preparing candidate routes are always worth considering.

Part II: Heuristic search methods

To be able to solve large-scale combinatorial optimisation problems, (meta-)heuristic methods are preferred. Due to the high computational complexity involved in solving these problems, developing efficient algorithms is equally important to deliver useful solutions to real-world problems.

In part II, our focus is on the designing of efficient heuristic based methods. The aim is not only building efficient solvers for the GDMP, but also more generally understanding the behaviour of each algorithm while solving problems with different characteristics. With the successful experience of applying a BEBO hyperheuristic to solve our drainage system maintenance problem in Chapter 6, we continue our investigation on various hyperheuristic frameworks, and test them on the benchmark PVRP instances, which allows us to compare the algorithm performance with the state-of-art meta-heuristics developed by other authors.

The spirit of hyperheuristic is to introduce artificial intelligence to algorithm self-design and self-adaptation for solving different difficult problems. The algorithms that apply simple random decision making process (e.g. low-level-heuristic selection) still count as hyperheuristic, even though they may not looks smart. According to our experiments tested on small benchmark PVRP instances (Chapter 7) and experience from other authors (e.g. Gulczynski et al. (2011); Vidal et al. (2012)), there are many advantages to involving random decision making within the algorithm design:

1. Memory efficient. Complex decision making always needs additional information stored (e.g. historical search trajectory).

2. Computationally efficient. A random strategy requires minimal calculation to make a search decision. An additional benefit of this method is to complete more iterations when search through the solution space, which may explain why this method out performs other sophisticated algorithms on some problems.

However, when applying the random decision-making approach to bigger problems, we can see that the random strategy gradually becomes less competitive than learning-based decision-making approaches. Learning-based hyperheuristics introduce mechanisms that statistically evaluate the performance of low-level-heuristics and make a choice based on the evaluations. The evaluation method usually defines the name of learning-based hyperheuristics. According to our experience, it is difficult to conclude that one hyperheuristics always outperforms others for different types of problems. Further work could investigate ways to evaluate "strong" low-level heuristics and the reasons of preference of problem type to hyperheuristics.

To successfully apply a hyperheuristic method to problems, as well as the hyperheuristic design itself, it is important to have a good set of low-level-heuristics. The set of low-level-heuristics can be considered as a tool box, whilst a hyperheuristic acts as an intelligent agent to choose the right tool at the right time. Therefore, a good set of low-level-heuristics should be able to reach any solution in a solution space, when used in different orders and combinations.

In Chapter 8, we address designing of an efficient local search (LS) process. A novel dynamic multi-arm-bandit neighbourhood search (D-MABNS) is proposed, which aims to quickly find a descent direction during each iteration of LS and reach the local optimum effectively. The D-MABNS design is based on the observation that it can be inefficient to check the entirety of one move's neighbourhood before considering others (Figure 8.2, Chapter 8). We perform a comprehensive analysis and gain insights into the relationship between neighbourhood structure and search strategies. To effectively solve our large scale GDMP, D-MABNS utilises the orderliness property of its neighbourhood structures and applies techniques that focus the search in promising areas of the solution space. Compared to the BEBO hyperheuristic developed in the earlier work, D-MABNS achieves significant better results within the same calculation time.

The D-MABNS does not show much advantages when tested on the benchmark PVRP instances. The PVRP neighbourhood structures show little or no orderliness in features that have already been used. Also, the tested PVRP instances are of much smaller size compared to the drainage maintenance problem. Therefore, it is relatively fast to search through an entire neighbourhood in these cases. These may be the reasons that reduce the impact of D-MABNS's tricks during the search.

Discussion and future work In the developing of our heuristic search methods, we mostly focus on *LS*-based approaches (see review in Section 2.1.2.3). In recent years, many hybrid-heuristics that combine the techniques from population-based meta-heuristic and LS have achieved significant success in solving difficult combinatorial problems (e.g. Nagata et al. (2010); Vidal et al. (2012)). Compared to the multi-restart technique mostly used in this thesis, when a search gets stuck in a local optimum, we think it is worth investigating some recombination operators, which are normally introduced in population search methods. By introducing recombination between good solutions, we may have a bigger chance of starting the following *LS* process within promising areas.

Another thought when adopting our algorithms in real-world scenarios is parallelisation. For the learning based hyperheuristic (Chapter 7), the process of calling each selected low-level-heuristic can certainly be parallelised. Similarly, D-MABNS (Chapter 8) could also examine several neighbours each from a different neighbourhood simultaneously.

In Chapter 8, experiments show that the D-MABNS outperforms the BEBO hyper-heuristic in solving Gaist Solutions Ltd.'s drainage system maintenance scheduling problem in a short period. In the future work, we would like to apply D-MABNS on the rolling horizon scheduling framework (Chapter 6) and deliver the long period risk analysis of adopting D-MABNS as the schedule solver. We are hopeful that a larger risk reduction can also be achieved in the long period as its success has been shown in the short-period experiments.

Part III: Large-scale road inspection

Part III focuses on a separate highway maintenance issue, which again is of interest to our partner, Gaist Solutions Ltd. As it deals with the road inspection problem. The aim here is to make a strategic decision in OR to choose a cost-effective method of delivering the road inspection on a national scale. Specifically, we explore the potential benefits of using inspection vehicles, which can collect road-lane information for both sides of a single carriageway in one traversal, as opposed to more common single-sided monitoring, which has to use a bi-directional strategy to monitor every road.

The interest of this study is not only in the question itself, but also in its large scale. Three achievements of this work are addressed. First, according to our data analysis and theoretical proof, we model the real-world road inspection problem as a CPP, which allows us to accurately estimate the total travelling distance needed in a national-scale road inspection. Second, to solve large-scale problems, we propose a graph-reduction strategy, which significantly speeds up the Chinese Postman Tour (CPT) (Guan (1962)) calculation time. Third, based on the study of seven UK road networks, including the ones in both urban and rural areas, the total savings from road-inspection expenditure for UK government is estimated to be up to £4.32 million, when the more efficient one-pass road inspection strategy is applied.

Discussion and future work The experiments in this study show that significant performance improvements follow if data simplification is performed before actually solving the problem. This idea is also employed in solving the drainage system maintenance problem (Part I). The problem reduction techniques introduced in Chapter 6, including the point grouping and routes preparing, essentially transform the original routing and scheduling problem into a much smaller combinatorial problem. These techniques earn the advantages of both computational and memory efficiency.

In this study, the proposed that graph reduction approach can only be applied to undirected graphs. In practice, there are many one-way systems in our road networks. It would be more useful and accurate if a similar approach would be applied in a mixed or directed graph. In the design of one-way road system, we could always find a set of minimum distance tours, which include the one-way edges at least once. From this perspective, many of the one-way road system could be (partially) pre-digested as giant nodes in a directed graph. We think that further effort is worthy of being spent on the extension of the current graph-reduction approach to deal with mixed and directed graphs.

To complete the road inspection task in practice, as well as the analysis from a strategic decision, further work should be considered at the operational decision making level. Example questions to answer include optimally dividing the country into sub-areas, finding the optimal starting points for each vehicle tour and designing optimal survey routes for each vehicle with consideration of constraints (e.g. traffic in urban area, one way road).

Last words

In conclusion, this research has increased our knowledge of using the risk management concept (from a high-level strategical planning) to automatically deliver detailed operational actions (i.e. scheduling and routing) for large-scale GDMPs. The work in this thesis makes contributions toward this goal from several aspects, including problem analysis, modelling, and solution approach development. We hope that the scope for further research discussed in each part will enhance the generality and robustness of solutions, which can be applied to wider problem domains and more complex scenarios.

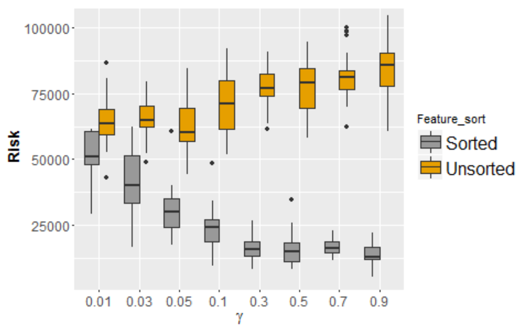
Appendix A

Table A.1: Best found solution of PVRP instance (p04)

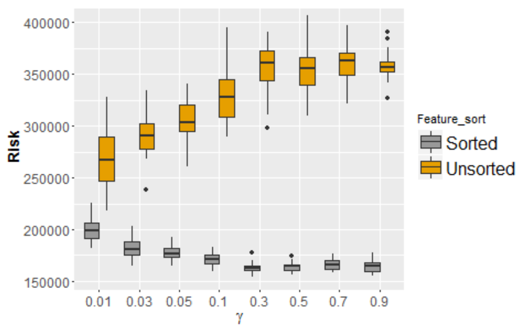
p04 (total length of routes: 835.3)
day1:
Route0: 0,67,46,34,4,75,0,
Route1: 0,6,33,63,23,56,24,49,16,0,
Route2: 0,27,37,20,70,60,71,69,36,47,48,0,
Route3: 0,62,22,64,42,41,43,1,73,51,0,
Route4: 0,30,74,21,61,28,2,68,0,
day2:
Route5: 0,17,40,9,39,12,26,0,
Route6: 0,38,65,66,59,14,7,0,
Route7: 0,52,19,54,13,57,15,5,29,45,0,
Route8: 0,8,35,53,11,10,58,0,
Route9: 0,3,44,32,50,18,55,25,31,72,0,

Table A.2: Information of PVRP street-style and benchmark instances. (Our real-world PVRP instances are named from map1 to map6. There are two data sets available in literature, where instances p01–p32 compose the old data set and instances pr01–pr10 compose the new data set. *n* is the number of customer; *m* is the number of vehicle available; *t* is the number of days in the planning horizon. The problem instances can be download from `http://yc1005.wixsite.com/yujiechen`)

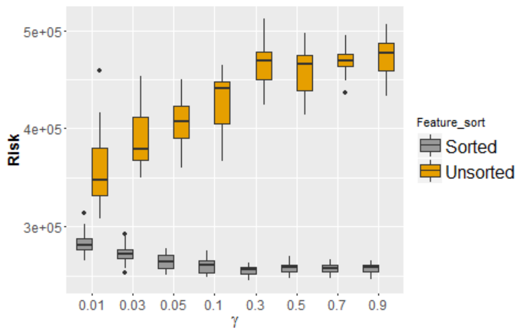
	n	**m**	**t**	**Average service frequency**	**Proposed by**
				Street style instance	
map1	315	3	6	2.057142857	Chen et al. (2016d)
map2	324	3	6	1.586419753	
map3	240	3	6	1.820833333	
map4	315	5	6	2.057142857	
map5	324	5	6	1.586419753	
map6	240	5	6	1.820833333	
				Benchmark instance	
p01	51	3	2	1	Christofides and Beasley (1984)
p02	50	3	5	2.08	
p03	50	1	5	1	
p04	75	5	2	1	
p05	75	6	5	2.04	
p06	75	1	10	1	
p07	100	4	2	1	
p08	100	5	5	2.02	
p09	100	1	8	1	
p10	100	4	5	1.74	
p11	139	4	5	1.381294964	Russell and Igo (1979)
p12	163	3	5	1.134969325	Russell and Gribbin (1991)
p13	417	9	7	1.095923261	
p14	20	2	4	2	Chao et al. (1995)
p15	38	2	4	1.894736842	
p16	56	2	4	1.857142857	
p17	40	4	4	2	
p18	76	4	4	1.894736842	
p19	112	4	4	1.857142857	
p20	184	4	4	1.826086957	
p21	60	6	4	2	
p22	114	6	4	1.894736842	
p23	168	6	4	1.857142857	
p24	51	3	6	1.764705882	
p25	51	3	6	1.764705882	
p26	51	3	6	1.764705882	
p27	102	6	6	1.764705882	
p28	102	6	6	1.764705882	
p29	102	6	6	1.764705882	
p30	153	9	6	1.764705882	
p31	153	9	6	1.764705882	
p32	153	9	6	1.764705882	
pr01	48	2	4	2	Cordeau et al. (1997)
pr02	96	4	4	2	
pr03	144	6	4	2	
pr04	192	8	4	2	
pr05	240	10	4	2	
pr06	288	12	4	2	
pr07	72	3	6	3	
pr08	144	6	6	3	
pr09	216	9	6	3	
pr10	288	12	6	3	



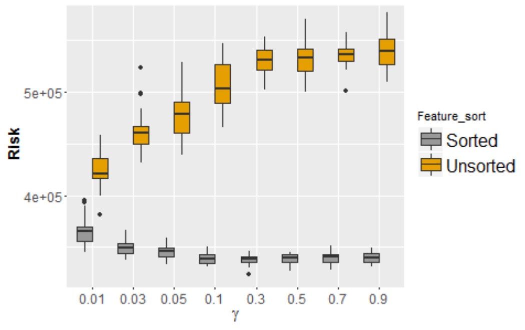
(a) mi01: 2815 vertices



(b) mi03: 14074 vertices

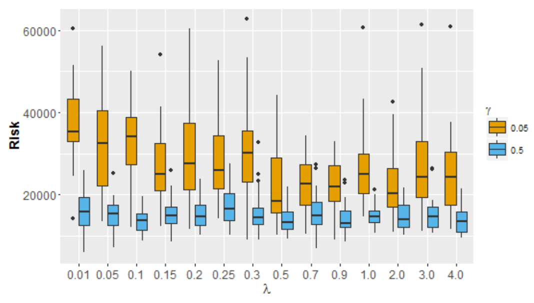


(c) mi04: 21111 vertices

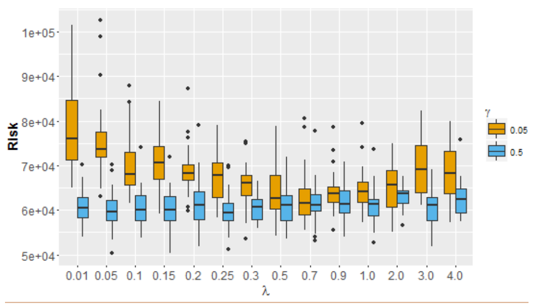


(d) mi05: 28149 vertices

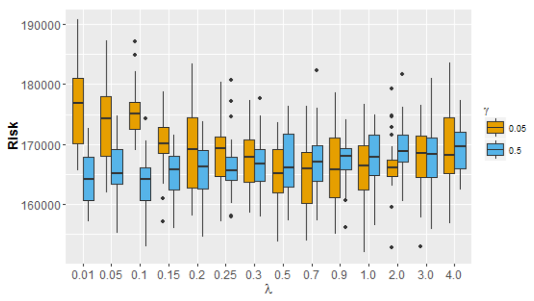
Figure A.1: Effect of feature sorting strategy with different neighbourhoods prune rate γ to different sizes of GDMP instances



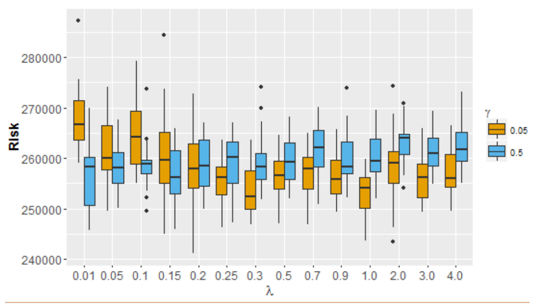
(a) mi01: 2815 vertices



(b) mi02: 7037 vertices



(c) mi03: 14074 vertices



(d) mi04: 21111 vertices

Figure A.2: The effect of parameter λ on the algorithm performance with two different pruning parameter setting. The other parameter settings are $\{w = 400, RF = exp(a = 5), \alpha = 0.8\}$. Tested on different sizes of GDMP instances

Table A.3: Performance on PVRP benchmarks compared with meta-heuristics; record-to-record heuristic (Chao) Chao et al. (1995), tabu search (CGL) Cordeau et al. (1997), scatter search (ALP)Alegre et al. (2007), VNS (HDH)Hemmelmayr et al. (2009), record-to-record ILP (GGW) Gulczynski et al. (2011), hybrid-GA (VCGLR)Vidal et al. (2012), parallel tabu search (CM) Cordeau and Maischberger (2012). The first column shows the number of customers in each instance.

		Chao	CGL	ALP	HDH	GGW	VCGLR	CGL2	VNSr(R)		Ran		D-MABNS		
			Tabu	scatter search	VNS	IPH	Hybird GA	Tabu	mean	best	mean	best	mean	best	CPU(s)
Size	Instance		1 run	best	10run	-	10 run	10 run	10 run		10 run		10run		
51	p01	**524.6**	**524.6**	531	**524.6**	**524.6**	**524.6**	**524.6**	**524.6**	**524.6**	**524.6**	**524.6**	524.7	**524.6**	13.2
50	p02	1337.2	1330.1	1324.7	1332	1335.1	**1322.9**	1328.7	1333.2	1325.1	1336.1	1324.2	1336.2	**1322.9**	26.4
50	p03	**524.6**	**524.6**	537.4	529	**524.6**	**524.6**	**524.6**	**524.6**	**524.6**	524.7	**524.6**	524.7	**524.6**	10.8
75	p04	860.9	837.9	846	847.5	838.5	836.6	836	841.2	835.8	843.5	836.8	842	**835.3**	63
75	p05	2089	2061.4	2043.8	2059.7	2052.8	**2033.7**	2034.7	2077	2062.6	2068	2048.5	2069	2047	136.2
75	p06	881.1	840.3	840.1	884.7	838.5	842.5	**836.4**	851.7	841.8	861.7	845.1	856.3	843.4	53.4
100	p07	832	829.4	829.7	829.9	827.4	827	826.8	829.2	**826.1**	831.2	827.8	831.9	827.5	52.8
100	p08	2075.1	2054.9	2052.2	2058.4	2054.3	**2022.9**	2044.3	2069.2	2053.7	2072.3	2054.1	2075.1	2050.2	152.4
100	p09	829.9	829.5	829.7	834.9	827.4	826.9	826.6	831	827.4	833.3	829.5	833.8	**826.1**	60.6
100	p10	1633.2	1630	1621.2	1629.8	1645.4	1605.2	**1600.9**	1637.4	1617.5	1651.1	1634.2	1649.1	1618.6	108
139	p11	791.3	817.6	782.2	791.2	781.7	**775.8**	780.6	792.2	785.2	792.8	786.9	795.4	785.7	276
163	p12	1237.4	1239.6	1231	1258.5	1225.8	**1195.3**	1196.8	1249.8	1220.3	1243.6	1217.9	1246.4	1218	320.4
417	p13	3629.8	3602.8		3835.9	3624.8	3599.9	**3518.7**	3662.8	3585.6	3581.2	3548.7	3607.6	3541.8	2400
20	p14	**954.8**	**954.8**	**954.8**	**954.8**	**954.8**	**954.8**	**954.8**	**954.8**	**954.8**	**954.8**	**954.8**	**954.8**	**954.8**	4.8
38	p15	**1862.6**	**1862.6**	**1862.6**	**1862.6**	**1862.6**	**1862.6**	**1862.6**	**1862.6**	**1862.6**	**1862.6**	**1862.6**	**1862.6**	**1862.6**	10.2
56	p16	**2875.2**	**2875.2**	**2875.2**	**2875.2**	**2875.2**	**2875.2**	**2875.2**	**2875.2**	**2875.2**	**2875.2**	**2875.2**	**2875.2**	**2875.2**	19.2
40	p17	1614.4	1597.8	1597.8	1601.8	**1597.7**	1597.8	1597.8	1621.8	1597.7	1627	1597.7	1620.8	1597.7	16.2
76	p18	3217.7	3159.2	3157	3147.9	3215.4	**3131.1**	3155.2	3159	3152.4	3157.1	3150.2	3158.4	3151.6	53.4
112	p19	4846.5	4902.6	4846.5	4851.4	4846	**4834.5**	4843	4846.5	4846.5	4846.5	4846.5	4846.5	4846.5	135.6
184	p20	**8367.4**	**8367.4**	8412	**8367.4**	8369.7	**8367.4**	**8367.4**	**8367.4**	**8367.4**	**8367.4**	**8367.4**	**8367.4**	**8367.4**	240.6
60	p21	2216.1	2184	2173.6	2180.3	2189	**2170.6**	2184.1	2187.6	2183.5	2184.4	2182.6	2189.4	2182.6	54
114	p22	4436.4	4307.2	4330.6	4218.5	4317.2	**4194.2**	4213.7	4282.3	4229	4288.6	4235.6	4279.2	4232.5	256.2
168	p23	6769	6620.5	6813.5	6644.9	6685.1	**6434.1**	6575.5	6674.8	6609.2	6633.5	6589	6704.5	6572.7	257.4
51	p24	3773	3704.1	3702	3704.6	3742	**3687.5**	3695.2	3731	3693.5	3725.6	3693.5	3734	3693.5	19.2
51	p25	3826	3781.4	3781.4	3781.4	3817.1	**3777.2**	3780.2	3785.6	3781.4	3781.4	3781.4	3783.6	3781.4	35.4
51	p26	3834	3795.3	3795.3	3795.3	**3795**	3795.3	3795.3	3831.8	3795.3	3834	3834	3833.5	3815.3	19.8
102	p27	23401.6	23017.5	22561.3	22153.3	21946.9	21885.7	**21877.5**	22293	22196.3	22136.7	22071.4	22158.7	22057.7	211.2
102	p28	23105.1	22569.4	22562.4	22418.5	22384	22272.6	**22271.4**	22559	22444.7	22455.8	22360.7	22471.4	22391.4	280.2
102	p29	24248.2	24012.9	23752.2	22864.2	22629	**22564.1**	22586.6	23103	22775.2	22749.8	22698.8	22825.4	22663.6	231.6
153	p30	80982.1	77179.3	76794	75579.2	75003.9	**74534.4**	74547.5	77525.9	76776.7	75903.5	75215.7	76187.1	75421	599.4
153	p31	80279.1	79382.4	77944.8	77459.1	76842.3	**76686.7**	76883.2	78650.4	78128.5	77663	77346.3	77916.9	77373.2	600
153	p32	83838.7	80909	81055.5	79488	78650.8	**78168.8**	78366.2	80832	79647.3	79404.6	78907.4	79652.9	78927.3	600
48	pr01		2234.2		2209.1		**2209**	2209.9	2209	2209	2210.6	2209	2211.7	2209	17.4
72	pr02		3836.5		3787.5		**3768.9**	3779.2	3842.2	3822	3849.8	3825.6	3851.7	3817.6	149.4
96	pr03		5277.6		5243.1		**5174.8**	5206.6	5303.3	5254.7	5324.3	5255.6	5342.3	5259.1	439.2
144	pr04		6072.7		6011.4		**5936.2**	5947.9	6109.3	6056.6	6132.9	6065.7	6150	6070.2	600
144	pr05		6769.8		6778		**6651.8**	6664.8	6873.1	6810.2	6873.4	6826.3	6885	6792.8	1200
192	pr06		8462.4		8461.5		**8284.9**	8316.4	8619.2	8569.9	8667.6	8560.9	8735.3	8597.8	1200
216	pr07		5000.9		5007		**4996.1**	4996.1	5013.8	5000.6	5022.3	5012.3	5024.2	5000.3	89.4
240	pr08		7183.4		7119.6		**7035.5**	7041.1	7240.6	7193.6	7246.8	7190.7	7276.6	7174.7	600
288	pr09		10507.3		10259.1		**10162.2**	10174	10527.4	10397	10541.4	10484.7	10578.7	10445.4	1200
	pr10		13629.3		13342.4		**13091**	13105.9							
							summary								**average CPU(s)**
0-100	gap	0.015014	0.006321	0.005836	0.00853	0.006956	0.00062	0.001999	0.009529	0.004327	0.011071	0.005371	0.011078	0.004115	68.86
101-300	gap	-	0.028973	-	0.018712	-	0.00118	0.002645	0.026084	0.016554	0.020872	0.013821	0.023947	0.013163	594.63
All	gap	-	0.017107	-	0.013379	-	0.00089	0.002307	0.017201	0.009993	0.015613	0.009287	0.017041	0.008308	312.51

List of Abbreviations

BEBO	Binary Exponential Back Off (for LLH selection)	Chapter 7
BI	Best Improvement	Chapter 3
CBM	Condition Based Maintenance	Chapter 4
CF	Choice Function (for LLH selection)	Chapter 7
CPP	Chinese Postman Problem	Chapter 9
CPT	Chinese Postman Tour	Chapter 9
CW	Clark-Wright algorithm	Chapter 2
D-MAB	Dynamic Multi Arm Bandit	Chapter 8
D-MABNS	DMAB Neighbourhood Search	Chapter 8
FI	First Improvement	Chapter 3
FSS	Feature Sequential Search	Chapter 3
FW	Floyd-Warshall algorithm	Chapter 9
GDMP	Geographically Disturbed asst Maintenance Problem	Chapter 1
HA	Hierholzer Algorithm	Chapter 9
ILS	Iterated Local Search	Chapter 7
LLH	Low Level Heuristics	Chapter 3
LS	Local Search	Chapter 3
MAB	Multi Arm Bandit	Chapter 8
PSS	Predictive Scheduling Strategy	Chapter 6
PVRP	Periodic Vehicle Routing Problem	Chapter 2
RF	Reward Function	Chapter 8
RL	Reinforcement Learning (for LLH selection)	Chapter 7
SR	Simple Random (for LLH selection)	Chapter 7
TBM	Time Based Maintenance	Chapter 4
TOP	Team Orienteering Problem	Chapter 2
VND	Variable Neighbourhood Descent (search)	Chapter 7
VNS	Variable Neighbourhood Search	Chapter 2
VRP	Vehicle Routing Problem	Chapter 2

Bibliography

Ahmad, R. and Kamaruddin, S. (2012). An overview of time-based and condition-based maintenance in industrial application. *Computers & Industrial Engineering* 63(1): 135–149.

Alegre, J., Laguna, M. and Pacheco, J. (2007). Optimizing the periodic pick-up of raw materials for a manufacturer of auto parts. *European Journal of Operational Research* 179(3): 736–746.

Alfa, A., Heragu, S. and Chen, M. (1991). A 3-opt based simulated annealing algorithm for vehicle routing problems. *Computers & Industrial Engineering* 21(4): 635–639.

Alonso, F., Alvarez, M. J. and Beasley, J. E. (2007). A tabu search algorithm for the periodic vehicle routing problem with multiple vehicle trips and accessibility restrictions. *Journal of the Operational Research Society* 59(7): 963–976.

Altnel, . K. and Öncan, T. (2005). A new enhancement of the Clarke and Wright savings heuristic for the capacitated vehicle routing problem. *Journal of the Operational Research Society* 56(8): 954–961.

Alvarenga, G. B., Mateus, G. R. and de Tomi, G. (2007). A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows. *Computers and Operations Research* 34(6 SPEC. ISS.): 1561–1584.

Amorim, P., Parragh, S. N., Sperandio, F. and Almada-Lobo, B. (2014). A rich vehicle routing problem dealing with perishable food: A case study. *TOP* 22(2): 489–508.

An, Y. J., Kim, Y. D., Jeong, B. J. and Kim, S. D. (2012). Scheduling healthcare services in a home healthcare system. *Journal of the Operational Research Society* 63(11): 1589–1599.

Angelelli, E., Bianchessi, N., Mansini, R. and Speranza, M. (2009). Short Term Strategies for a Dynamic Multi-Period Routing Problem. *Transportation Research Part C: Emerging Technologies* 17(2): 106–119.

Angelelli, E., Speranza, M. G. and Savelsbergh, M. (2007). Competitive analysis for dynamic multiperiod uncapacitated routing problems. *Networks* 49(4): 308–317.

Archetti, C., Hertz, A. and Speranza, M. G. (2007). Metaheuristics for the team orienteering problem. *Journal of Heuristics* 13(1): 49–76.

Archetti, C., Speranza, M. and Vigo, D. (2014). Vehicle Routing Problems with Profits. In P. Toth and D. Vigo, eds., *Vehicle Routing: Problems, Methods, and Applications*. Philadelphia: SIAM: 273–298.

Asphalt Industry Alliance (2010). The economic impact of local road condition. Date accessed: 2015-12-05. http://www.asphaltuk.org/images/library/files/AIA_YouGov_Report_final.pdf. Technical report: 1–28.

Auer, P., Cesa-Bianchi, N. and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47(2): 235–256.

Ayob, M. and Kendall, G. (2003). A Monte Carlo Hyper-Heuristic To Optimise Component Placement Sequencing For Multi Head Placement Machine. In *International Conference on Intelligent Technologies*: 132–141.

Bai, R., Blazewicz, J., Burke, E. K., Kendall, G. and McCollum, B. (2012). A simulated annealing hyper-heuristic methodology for flexible decision support. *4OR* 10(1): 43–66.

Bai, R. and Kendall, G. (2005). An investigation of automated planograms using a simulated annealing based hyper-heuristics. In T. Ibaraki, K. Nonobe and M. Yagiura, eds., *Metaheuristics: Progress as Real Problem Solvers*. Boston, MA: Springer US, 32: 87–108.

Baik, H.-S., Jeong, H. S. D. and Abraham, D. M. (2006). Estimating Transition Probabilities in Markov Chain-Based Deterioration Models for Management of Wastewater Systems. *Journal of Water Resources Planning and Management* 132(1): 15–24.

Balas, E. (2007). The prize collecting Traveling Salesman Problem and its Applications. In G. Gutin and A. P. Punnen, eds., *The traveling salesman problem and its variations*. Boston, MA: Springer US, 12: 663–695.

Baldacci, R., Bartolini, E., Mingozzi, a. and Valletta, a. (2011). An Exact Algorithm for the Period Routing Problem. *Operations Research* 59(1): 228–241.

Baldacci, R., Battarra, M. and Vigo, D. (2008). Routing a Heterogeneous Fleet of Vehicles. In B. Golden, S. Raghavan and E. Wasil, eds., *The vehicle routing problem: latest advances and new challenges*. Boston, MA: Springer US, 43: 3–27.

Baldacci, R., Mingozzi, A. and Roberti, R. (2012). Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research* 218(1): 1–6.

Baldacci, R., Toth, P. and Vigo, D. (2007). Recent advances in vehicle routing exact algorithms. *4OR* 5(4): 269–298.

Balinski, M. L. and Quandt, R. E. (1964). On an integer program for a delivery problem*. *Operations Research* 12(2): 300–304.

Baptista, S., Oliveira, R. C. and Zúquete, E. (2002). A periodic vehicle routing case study. *European Journal of Operational Research* 139(2): 220–229.

Barceló, J., Grzybowska, H. and Pardo, S. (2007). Vehicle routing and scheduling models, simulation and city logistics. In V. Zeimpekis, C. D. Tarantilis, G. M. Giaglis and I. Minis, eds., *Dynamic Fleet Management: Concepts, Systems, Algorithms & Case Studies*. Boston, MA: Springer US, 38: 163–195.

BBC (2011). Flooding affects Pembrokeshire residents and businesses. Date accessed: 2015-09-04. http://www.bbc.co.uk/news/uk-wales-15441912.

BBC (2012). Floods: North Wales Police travel warning after rain. Date accessed: 2015-09-05. http://www.bbc.co.uk/news/uk-wales-19702806.

Bell, J. E. and McMullen, P. R. (2004). Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics* 18(1): 41–48.

Beltrami, E. and Bodin, L. (1974). Networks and vehicle routing for municipal waste collection. *Networks* 4(1): 65–94.

Bentley, J. J. (1992). Fast algorithms for geometric traveling salesman problems. *Orsa Journal on Computing* 4(4): 387–411.

Besnard, F., Fischer, K. and Bertling, L. (2010). Reliability-Centred Asset Maintenance-A step towards enhanced reliability, availability, and profitability of wind power plants. In *Innovative Smart Grid Technologies Conference Europe. IEEE*: 1–8.

Bilgin, B., Özcan, E. and Korkmaz, E. (2007). An experimental study on hyper-heuristics and exam timetabling. In E. K. Burke and H. Rudova, eds., *Practice and Theory of Automated Timetabling VI*. Berlin, Heidelberg: Springer Berlin Heidelberg, 3867: 394–412.

Blackpool (2009). Blackpool strategic flood risk assessment. Date accessed: 2015-07-05. https://www.blackpool.gov.uk/Residents/Planning-environment-and-community/Documents/Blackpool-Strategic-Flood-Risk-Assessment.pdf. Technical Report June.

Blakeley, F., Bozkaya, B., Cao, B., Hall, W. and Knolmajer, J. (2003). Optimizing periodic maintenance operations for Schindler Elevator Corporation. *Interfaces* 33(1): 67–79.

Blasum, U. and Hochstättler, W. (2002). Application of the Branch and Cut Method to the Vehicle Routing Problem. Technical report: 1–22.

Blitzstein, J. and Diaconis, P. (2011). A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet Mathematics* 6(4): 489–522.

Brandão, J. (2011). A tabu search algorithm for the heterogeneous fixed fleet vehicle routing problem. *Computers & Operations Research* 38(1): 140–151.

Brandão, J. and Mercer, A. (1997). A tabu search algorithm for the multi-trip vehicle routing and scheduling problem. *European Journal of Operational Research* 100(1): 180 – 191.

Brouwer, R. and Ek, V. R. (2004). Integrated ecological, economic and social impact assessment of alternative flood control policies in the Netherlands. *Ecological Economics* 50(1-2): 1–21.

Brysy, O. and Gendreau, M. (2005). Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms. *Transportation Science* 39(1): 104–118.

Burke, E. K., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., Vázquez-Rodríguez, J. and Gendreau, M. (2010). Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. In *Evolutionary Computation IEEE*. 1210: 1–8.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E. and Qu, R. (2013). Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society* 64(12): 1695–1724.

Burke, E. K., Gendreau, M., Ochoa, G. and Walker, J. D. (2011). Adaptive iterated local search for cross-domain optimisation. In *Genetic and Evolutionary Computation Conference, GECCO'11*. Dublin, Ireland, 162: 1987–1994.

Burke, E. K., Hyde, M. R., Kendall, G. and Woodward, J. (2007a). Automatic Heuristic Generation with Genetic Programming: Evolving a Jack-of-all-trades or a Master of One. In *Genetic and Evolutionary Computation Conference, GECCO 2007*. London, England, Uk: 1559–1565.

Burke, E. K., Kendall, G. and Soubeiga, E. (2003). A Tabu-Search Hyperheuristic for Timetabling and Rostering. *Journal of Heuristics* 9(6): 451–470.

Burke, E. K., McCloumn, B., Meisels, A., Petrovic, S. and Qu, R. (2007b). A Graph-Based Hyper-Heuristic for Educational Timetabling Problems. *European Journal Of Operational Research* 176(1): 177–192.

Butler, D., Xiao, Y. and Karunaratne, S. (1995). The gully pot as a physical, chemical and biological reactor. *Water Science and Technology* 31(7): 219–228.

Caceres-cruz, J., Riera, D. and Juan, A. A. (2014). Rich Vehicle Routing Problem : Survey. *ACM Computing Surveys* 47(2): 1–28.

Campbell, A. M. and Savelsbergh, M. (2004). Efficient Insertion Heuristics for Vehicle Routing and Scheduling Problems. *Transportation Science* 38(3): 369–378.

Campbell, A. M. and Wilson, J. H. (2014). Forty years of periodic vehicle routing. *Networks* 63(1): 2–15.

Campos, J. (2009). Development in the application of ICT in condition monitoring and maintenance. *Computers in Industry* 60(1): 1–20.

Cappanera, P., Gouveia, L. and Scutellà, M. (2011). The skill vehicle routing problem. In J. Pahl, T. Reiners and S. Voss, eds., *Network Optimization: 5th International Conference, INOC 2011*. Berlin, Heidelberg: Springer Berlin Heidelberg, 6701: 354–364.

Caramia, M., Italiano, G., Oriolo, G., Pacifici, A. and Perugia, A. (2002). Routing a Fleet of Vehicles for Dynamic Combined Pick-up and Deliveries Services. In P. Chamoni, R. Leisten, A. Martin, J. Minnemann and H. Stadtler, eds., *Operations Research Proceedings 2001*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001: 3–8.

Carnero Moya, M. C. (2004). The control of the setting up of a predictive maintenance programme using a system of indicators. *Omega* 32(1): 57–75.

Chakhlevitch, K. and Cowling, P. (2008). Hyperheuristics: Recent developments. In C. Cotta, M. Sevaux and K. Sorensen, eds., *Adaptive and multilevel metaheuristics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 136: 3–29.

Changnon, S. a. (1999). Record Flood-Producing Rainstorms of 1718 July 1996 in the Chicago Metropolitan Area. Part III: Impacts and Responses to the Flash Flooding. *Journal of Applied Meteorology* 38(3): 273–280.

Chao, I. and Liou, T. (2005). A New Tabu Search Heuristic for the Site-Dependent Vehicle Routing Problem. In B. Golden, S. Raghavan and E. Wasil, eds., *The Next Wave in Computing, Optimization, and Decision Technologies*. Boston, MA: Springer US, 29: 107–119.

Chao, I.-M., Golden, B. L. and Wasil, E. (1995). An improved heuristic for the period vehicle routing problem. *Networks* 26(6): 25–44.

Chao, I.-M., Golden, B. L. and Wasil, E. A. (1996a). A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research* 88(3): 475–489.

Chao, I.-M., Golden, B. L. and Wasil, E. a. (1996b). The team orienteering problem. *European Journal of Operational Research* 88(3): 464–474.

Chartrand, G. and Oellermann, O. (1993). *Graph Minors*. McGraw-Hill: 277–281.

Chen, Y., Cowling, P., Polack, F. and Mourdjis, P. (2016a). A multi-arm bandit neighbourhood search for routing and scheduling problems. *Journal of Heuristics (submitted)*
.

Chen, Y., Cowling, P., Polack, F., Remde, S. and Mourdjis, P. (2016b). Dynamic optimization of preventative and corrective maintenance schedules for a large scale urban drainage system. *European journal of operational research* 257(2): 494–510.

Chen, Y., Cowling, P. and Remde, S. (2014). Dynamic Period Routing for a Complex Real-World System : A Case Study in Storm Drain Maintenance. In *Evolutionary Computation in Combinatorial Optimisation*. 8600: 109–120.

Chen, Y., Cowling, P., Remde, S. and Polack, F. (2016c). Efficient Large-scale Road Inspection Routing. In *Proceedings of 5th the International Conference on Operations Research and Enterprise Systems*: 304–312.

Chen, Y., Mourdjis, P., Polack, F., Cowling, P. and Remde, S. (2016d). Evaluating Hyper-heuristics and Local Search Operators for Periodic Routing Problems. In *Evolutionary Computation in Combinatorial Optimisation*: 104–120.

Chen, Y., Polack, F., Cowling, P., Mourdjis, P. and Remde, S. (2016e). Exploring Techniques to Improve Large-Scale Drainage System Maintenance Scheduling Using a Risk Driven Model. In *Communications in Computer and Information Science (submitted)*.

Chen, Y., Polack, F., Cowling, P., Mourdjis, P. and Remde, S. (2016f). Risk Driven Analysis of Maintenance for a Large-scale Drainage System. In *Proceedings of 5th the International Conference on Operations Research and Enterprise Systems*: 296–303.

Chen, Y., Polack, F., Cowling, P. and Remde, S. (2016g). A comparison of one-pass and bi-directional approaches applied to large-scale road inspection. In *Communications in Computer and Information Science (submitted)*.

Choi, E. and Tcha, D. W. (2007). A column generation approach to the heterogeneous fleet vehicle routing problem. *Computers and Operations Research* 34(7): 2080–2095.

Christofides, N. (1976a). The vehicle routing problem. *R.A.I.R.O. Recherche opérationnelle* 10(2): 55–70.

Christofides, N. (1976b). Worst-case analysis of a new heuristic for the travelling salesman problem. In *Symposium on new directions and recent results in algorithms and complexity*.

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

Claassen, G. D. H. and Hendriks, T. H. B. (2007). An application of Special Ordered Sets to a periodic milk collection problem. *European Journal of Operational Research* 180(2): 754–769.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research* 12(4): 568–582.

Constantine, G., Darroch, J. and Miller, R. (1996). Predicting underground pipeline failure. *Water-melbourne then artarmon* 23(2): 9–10.

Cordeau, J. F., Gendreau, M., Hertz, A., Laporte, G. and Sormany, J. S. (2005). New heuristics for the vehicle routing problem. In A. Langevin, and D. Riopel, eds., *Logistics Systems: Design and Optimization*. Boston, MA: Springer US, 30: 279–297.

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

Cordeau, J.-F. and Laporte, G. (2001). A tabu search algorithm for the site dependent vehicle routing problem with time windows. *INFOR: Information Systems and Operational Research* 39(3): 292–298.

Cordeau, J.-F., Laporte, G., Savelsbergh, M. W. P. and Vigo, D. (2007). Vehicle Routing. In *Handbooks in Operations Research and Management Science*. 14: 367–428.

Cordeau, J.-F. and Maischberger, M. (2012). A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research* 39(9): 2033–2050.

Costa, L. D., Fialho, A., Schoenauer, M. and Sebag, M. (2008). Adaptive Operator Selection with Dynamic Multi-Armed Bandits. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation GECCO 08*: 913–920.

Cowling, P., Kendall, G. and Soubeiga, E. (2001). A hyperheuristic approach to scheduling a sales summit. *Practice and Theory of Automated Timetabling* 2079: 176–190.

Crainic, T. G., Vahed, A. R., Gendreau, M. and Rei, W. (2012). Fleet-sizing for multi-depot and periodic vehicle routing problems using a modular heuristic algorithm. *Computers & Operations Research* 53(53): 9–23.

Cutter, S. L., Boruff, B. J. and Shirley, W. L. (2003). Social vulnerability to environmental hazards. *Social Science Quarterly* 84(2): 242–261.

Dantzing, G. B. and Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science* 6(1): 80–91.

Delgado, C., Laguna, M. and Pacheco, J. (2005). Minimizing labor requirements in a periodic vehicle loading problem. *Computational optimization and applications* 32(3): 299–320.

Dell'Amico, M., Righini, G. and Salani, M. (2006). A Branch-and-Price Approach to the Vehicle Routing Problem with Simultaneous Distribution and Collection. *Transportation Science* 40(2): 235–247.

Department for Transport (2005). Well-maintained Highways: Code of Practice for Highway Maintenance Management. Technical report.

Department for Transport (2012). Guidance on the Management of Highway Drainage Assets. Technical report.

Department for Transport (2013). Highway Infrastructure asset management. Technical report.

Desrosiers, J., Sournis, F. and Desrochers, M. (1984). Routing with Time Windows by Column Generation*. *Networks* 14(4): 54–565.

Dianati, M., Song, I. and Treiber, M. (2002). An introduction to genetic algorithms and evolution strategies. *Sadhana* 24(4-5): 293–315.

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

Dror, M. and Levy, L. (1986). A vehicle routing improvement algorithm comparison of a greedy and a matching implementation for inventory routing. *Computers & Operations Research* 13(1): 33–45.

Duffuaa, S., Ben-Daya, M., Al-Sultan, K. and a.a. Andijani (2001). A generic conceptual simulation model for maintenance systems. *Journal of Quality in Maintenance Engineering* 7(3): 207–219.

Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of mathematics* 17(3): 449–467.

Edmonds, J. and Johnson, E. L. (1973). Matching, Euler tours and the Chinese postman. *Mathematical programming* 5(1): 88–124.

Even, S. (2011). Paths in graphs. In G. Even, ed., *Graph Algorithms*. Cambridge University Press: 1–28.

Faccio, M., Persona, A. and Zanin, G. (2011). Waste collection multi objective model with real time traceability data. *Waste Management* 31(12): 2391–2405.

Fahrion, R. and Wrede, M. (1990). On a Principle of Chain-exchange for Vehicle-routeing Problems (-VRP). *The Journal of the Operational Research Society* 41(9): 821–827.

Fialho, Á., Da Costa, L., Schoenauer, M. and Sebag, M. (2009). Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5851 LNCS: 176–190.

Fialho, Á., da Costa, L., Schoenauer, M. and Sebag, M. (2010a). Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence* 60(1): 25–64.

Fialho, Á., Schoenauer, M. and Sebag, M. (2010b). Toward comparison-based adaptive operator selection. *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10* : 767.

Fleszar, K., Osman, I. H. and Hindi, K. S. (2009). A variable neighbourhood search algorithm for the open vehicle routing problem. *European Journal of Operational Research* 195(3): 803–809.

Fleury, M. (1883). Deux problemes de geometrie de situation. *Journal de mathematiques elementaires* 2(2): 257–261.

Floyd, R. W. (1962). Algorithm 97: Shortest Path. *Communications of the ACM* 5(6): 345.

Fredman, M. L. and Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM* 34(3): 596–615.

Fukasawa, R., Longo, H., Lysgaard, J., De Aragao, M. P., Reis, M., Uchoa, E. and Werneck, R. F. (2006). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming* 106(3): 491–511.

Funke, B., Grunert, T. and Irnich, S. (2005). Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of Heuristics* 11(4): 267–306.

Gabow, H. N. (1990). Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*. San Francisco: Society for Industrial and Applied Mathematics: 434–443.

García, I., Pacheco, J. and Alvarez, A. (2013). Optimizing routes and stock. *Journal of Heuristics* 19(2): 157–177.

Gardner, E. S. (1985). Exponential smoothing: The state of the art. *Journal of Forecasting* 4(1): 1–28.

Garrido, P. and Riff, M. C. (2010). DVRP: A hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. *Journal of Heuristics* 16(6): 795–834.

Garvin, W. W., Crandall, H. W., John, J. B. and Spellman, R. A. (1957). Applications of Linear Programming in the Oil Industry. *Management Science* 3(4): 407–430.

Gaskell, T. J. (1967). Bases for vehicle fleet scheduling. *Journal of the Operational Research Society* 18(3): 281–295.

Gaur, V. and Fisher, M. L. (2004). A Periodic Inventory Routing Problem at a Supermarket Chain. *Operations Research* 52(6): 813–822.

Gendreau, M., Hertz, A. and Laporte, G. (1992). New Insertion and Post Optimization Procedures for the Traveling Salesman Problem. *Operations Research* 40(6): 1086–1095.

Gendreau, M., Laporte, G. and Séguin, R. (1996). Stochastic vehicle routing. *European Journal of Operational Research* 88(1): 3–12.

Gendreau, M., Laporte, G. and Semet, F. (1998a). A branch-and-cut algorithm for the undirected selective traveling salesman problem. *Networks* 32(4): 263–273.

Gendreau, M., Laporte, G. and Semet, F. (1998b). A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research* 106(2-3): 539–545.

Giaglis, G., Minis, I., Tatarakis, A. and Zeimpekis, V. (2004). Minimizing logistics risk through real-time vehicle routing and mobile technologies: Research to date and future trends. *International Journal of Physical Distribution & Logistics Management* 34(9): 749–764.

Gillett, B. and Miller, L. (1974). A Heuristic Algorithm for the Vehicle Dispatch Problem. *Operations Research* 22(2): 340–349.

Glover, F. (1990a). Artificial intelligence, heuristic frameworks and tabu search. *Managerial and Decision Economics* 11(5): 365–375.

Glover, F. (1990b). Artificial intelligence, heuristic frameworks and tabu search. *Managerial and Decision Economics* 11(5): 365–375.

Glover, F. and Laguna, M. (2013). Tabu Search. In P. M. Pardalos, D.-Z. Du and R. L. Graham, eds., *Handbook of Combinatorial Optimization*: 3261–3362.

Goldberg, D. E. (1990). Probability Matching, the Magnitude of Reinforcement, and Classifier System Bidding. *Machine Learning* 5(4): 407–425.

Golden, B. and Wasil, E. (1987). OR PracticeComputerized Vehicle Routing in the Soft Drink Industry. *Informs* 35(1): 6–17.

Golden, B. L., Wasil, E. A., Kelly, J. P. and Chao, I.-M. (1998). The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In *Fleet Management and Logistics*: 33–56.

Goncalves, L., Ochi, L. and Martins, S. (2005). A GRASP with Adaptive Memory for a Period Vehicle Routing Problem. *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce* 1: 721–727.

Groër, C., Golden, B. and Wasil, E. (2010). A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation* 2(2): 79–101.

Guan, M. (1962). Graphic programming using odd or even points. *Chinese Math* 1(273-277): 110.

Gulczynski, D., Golden, B. and Wasil, E. (2011). The period vehicle routing problem: New heuristics and real-world variants. *Transportation Research Part E: Logistics and Transportation Review* 47(5): 648–668.

Hamphshire (2016). Hampshire ' s Highway Asset Management S trategy. Technical Report January.

Hansen, P. and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European journal of operational research* 130(3): 449–467.

Hansen, P., Mladenović, N. and Moreno Pérez, J. A. (2010). Variable neighbourhood search: Methods and applications. *Annals of Operations Research* 175(1): 367–407.

Helsgaun, K. (2000). An effective implementation of the LinKernighan traveling salesman heuristic. *European Journal of Operational Research* 126(1): 106–130.

Hemmelmayr, V. C., Doerner, K. F. and Hartl, R. F. (2009). A variable neighborhood search heuristic for periodic routing problems. *European Journal of Operational Research* 195(3): 791–802.

Hierholzer, C. and Wiener, C. (1873). Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen* 6(1): 30–32.

Hinkley, D. v. (1971). Inference about the change-point from cumulative sum tests. *Biometrika* 58(3): 509–523.

Holland, J. H. (1962). Outline for a Logical Theory of Adaptive Systems. *Journal of the ACM* 9(3): 297–314.

Hosny, M. (2011). Heuristic Techniques for Solving the Vehicle Routing Problem with Time Windows. In *International Conference on Future Informaion Technology*. 13: 19–23.

Irnich, S. (2008). Solution of real-world postman problems. *European Journal of Operational Research* 190(1): 52–67.

Jang, W., Lim, H. H., Crowe, T. J., Raskin, G. and Perkins, T. E. (2006). The missouri lottery optimizes its scheduling and routing to improve efficiency and balance. *Interfaces* 36(4): 302–313.

Jha, M. K., Udenta, F., Chacha, S. and Abdullah, J. (2010). Formulation and solution algorithms for highway infrastructure maintenance optimisation with work-shift and overtime limit constraints. *Procedia-Social and Behavioral Sciences* 2(3): 6323–6331.

Jha, M. K., Udenta, F., Chacha, S. and Karri, G. (2008). A modified arc routing problem for highway feature inspection considering work-shift and overtime limit constraints. In *WSEAS International Conference on Urban and Planning and Transportation*. Heraklion: 105–109.

Joubert, J. and Claasen, S. (2006). A sequential insertion heuristic for the initial solution to a constrained vehicle routing problem. *Orion* 22(1): 105–116.

Kallen, M. and Van Noortwijk, J. (2006). Optimal periodic inspection of a deterioration process with sequential condition states. *International Journal of Pressure Vessels and Piping* 83(4): 249–255.

Karlsson, K. and Viklander, M. (2008). Polycyclic aromatic hydrocarbons (PAH) in water and sediment from gully pots. *Water, Air, and Soil Pollution* 188(1-4): 271–282.

Kataoka, S. and Morito, S. (1988). An Algorithm for Single Constraint Maximum Collection Problem. *Journal of the Operations Research Society of Japan* 31(4): 515–531.

Keller, R. E. and Poli, R. (2007). Cost-Benefit Investigation of a Genetic-Programming Hyperheuristic. In *Artificial Evolution, International Conference, Evolution Artificielle*. Tours, France: Springer Berlin Heidelberg, 4926: 13–24.

Kendall, G. and Mohamad, M. (2004). Channel assignment in cellular communication using a great deluge hyper-heuristic. *12th IEEE International Conference on Networks* 2(1): 769–773.

Kenne, J. P. and Nkeungoue, L. J. (2008). Simultaneous control of production, preventive and corrective maintenance rates of a failure-prone manufacturing system. *Applied Numerical Mathematics* 58(2): 180–194.

Kirkpatrick, S. (1984). Optimization by simulated annealing: Quantitative studies. *Journal of statistical physics* 34(5): 975–986.

Kleiner, Y. and Rajani, B. (2001). Comprehensive review of structural deterioration of water mains: statistical models. *Urban Water* 3(3): 131–150.

Kolmogorov, V. (2009). Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation* 1(1): 43–67.

Kurtzberg, J. M. (1962). On approximation methods for the assignment problem. *Journal of the ACM (JACM)* 9(4): 419–439.

Laguna, M. and Marti, R. (2012). *Scatter search: methodology and implementations in C*. Springer Science & Business Media.

Lahyani, R., Khemakhem, M. and Semet, F. (2015). Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research* 241(1): 1–14.

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

Laporte, G. (1997). Modeling and solving several classes of arc routing problems as traveling salesman problems. *Computers & operations research* 24(11): 1057–1061.

Laporte, G., Gendreau, M. and Potvin, J.-y. (2000). Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research* 7(4-5): 285–300.

Laporte, G. and Martello, S. (1990). The selective travelling salesman problem. *Discrete Applied Mathematics* 26(2-3): 193–207.

Laporte, G. and Nobert, Y. (1987). Exact algorithms for the vehicle routing problem. In S. Martello, G. Laporte, M. Minoux and C. Ribeiro, eds., *Surveys in Combinatorial Optimization*. Amsterdam: North-Holland: 147–184.

Laporte, G., Nobert, Y. and Desrochers, M. (1985). Optimal Routing under Capacity and Distance Restrictions. *Operations Research* 33(5): 1050–1073.

Larsen, A., Madsen, O. B. . and Solomon, M. M. (2008). Recent Developments in Dynamic Vehicle Routing Systems. In *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer US, 43: 199–218.

Le Gat, Y. and Eisenbeis, P. (2000). Using maintenance records to forecast failures in water networks. *Urban Water* 2(3): 173–181.

Leylandguardian (2015). Blocked drain causes flooding danger on Lancashire road. Date accessed: 2015-08-24. http://www.leylandguardian.co.uk/news/blocked-drain-causes-flooding-danger-on-lancashire-road-1-7425326.

Li, F., Golden, B. and Wasil, E. (2007). A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem. *Computers and Operations Research* 34(9): 2734–2742.

Li, X., Tian, P. and Leung, S. C. H. (2010). Vehicle routing problems with time windows and stochastic travel and service times: Models and algorithm. *International Journal of Production Economics* 125(1): 137–145.

Liang, Y. C., Kulturel-Konak, S. and Smith, A. E. (2002). Meta heuristics for the orienteering problem. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on. IEEE*. 1: 384–389.

Lin, S. (1965). Computer Solutions of the Traveling Salesman Problem. *Bell System Technical Journal* 44(10): 2245–2269.

Lin, S. and Kernighan, B. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research* 21(2): 493–516.

Liu, F. H. and Shen, S. Y. (1999). The Fleet Size and Mix Vehicle Routing Problem with Time Windows on JSTOR. *The Journal of the Operational Research Society* 50(7): 721–732.

Lourenço, H. R., Martin, O. C. and Stützle, T. (2010). Iterated Local Search: Framework and Applications. In *Handbook of Metaheuristics*. 146: 363–397.

Lovász, L. (2006). Graph minor theory. *Bulletin of the American Mathematical Society* 43(1): 75–86.

Lysgaard, J., Letchford, A. N. and Eglese, R. W. (2004). A New Branch-and-Cut Algorithm for the Capacitated Vehicle Routing Problem. *Mathematical Programming* 100(2): 423–445.

Madanat, S. and Ben-Akiva, M. (1994). Optimal inspection and repair policies for infrastructure facilities. *Transportation science* 28(1): 55–62.

Madanat, S. and Ibrahim, W. H. W. (1995). Poisson regression models of infrastructure transition probabilities. *Journal of Transportation Engineering* 121(3): 267–272.

Magnanti, T. L. (1981). Combinatorial optimization and vehicle fleet planning: Perspectives and prospects. *Networks* 11(2): 179–213.

Maji, A. and Jha, M. (2007). Modeling highway infrastructure maintenance schedules with budget constraints. *Transportation Research Record: Journal of the Transportation Research Board* 1991(1): 19–26.

Matos, a. and Oliveira, R. (2004). An experimental study of the ant colony system for the period vehicle routing problem. In *Ant Colony Optimization and Swarm Intelligence*. 3172: 286–293.

Maya, P., Sörensen, K. and Goos, P. (2012). A metaheuristic for a teaching assistant assignment-routing problem. *Computers and Operations Research* 39(2): 249–258.

Meidan, L., Yehua, S., Jing, W. and Feifei, Z. (2010). Insertion heuristic algorithm for dynamic vehicle routing problem with time window. *Information Science and Engineering (ICISE), 2010 2nd International Conference* : 3789–3792.

Meignan, D., Koukam, A. and Créput, J. C. (2010). Coalition-based metaheuristic: A self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics* 16(6): 859–879.

Merz, B., Kreibich, H., Thieken, a. and Schmidtke, R. (2004). Estimation uncertainty of direct monetary flood damage to buildings. *Natural Hazards and Earth System Science* 4(1): 153–163.

Misir, M., Vancroonenburg, W., Verbeeck, K. and Vanden-berghe, G. (2011). A selection hyper-heuristic for scheduling deliveries of ready-mixed concrete. In *Proceedings of the 9th Metaheuristic International Conference*: 289–298.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research* 24(11): 1097–1100.

Mole, R. and Jameson, S. (1976). A sequential route-building algorithm employing a generalised savings criterion. *Operational Research Quarterly* 27(2): 503–511.

Morcous, G., Rivard, H. and Hanna, A. M. (2002). Modeling bridge deterioration using case-based reasoning. *Journal of Infrastructure Systems* 8(3): 86–95.

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical report.

Murphy, A. (2014). Road Lengths in Great Britain: 2013. Technical Report June: 5–8.

Nagata, Y., Bräysy, O. and Dullaert, W. (2010). A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers and Operations Research* 37(4): 724–737.

Nareyek, A. (2004). Choosing search heuristics by non-stationary reinforcement learning. In M. G. C. Resende, J. P. de Sousa and A. Viana, eds., *Metaheuristics*. Norwell, MA, USA: Kluwer Academic Publishers, 86: 523—-544.

Nuortio, T., Kytöjoki, J., Niska, H. and Bräysy, O. (2006). Improved route planning and scheduling of waste collection and transport. *Expert Systems with Applications* 30(2): 223–232.

Ochoa, G. and Burke, E. K. (2014). HyperILS : An Effective Iterated Local Search Hyper-heuristic for Combinatorial Optimisation. In *International Conference of the Practice and Theory of Automated Timetabling*: 26–29.

O'Connor, P. D. T. (1997). An Introduction to Reliability and Maintainbility. *Quality & Reliability Engineering International* 14(4): 295–295.

Or, I. (1976). *Traveling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking*. Ph.D. thesis, Northwestern University.

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research* 41(4): 421–451.

Özcan, E., Bilgin, B. and Korkmaz, E. (2008). A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis* 12(1): 3–23.

Paessens, H. (1988). The savings algorithm for the vehicle routing problem. *European Journal of Operational Research* 34(3): 336–344.

Page, E. S. (1954). Continuous inspection schemes. *Biometrika* 41(1): 100–115.

Pecin, D., Pessoa, A., Poggi, M. and Uchoa, E. (2016). Improved Branch-Cut-and-Price for capacitated vehicle routing. In *Mathematical Programming Computation*. 8494: 1–40.

Penna, P. H. V., Subramanian, A. and Ochi, L. S. (2013). An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics* 19(2): 201–232.

Pillac, V., Gendreau, M., Guéret, C. and Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research* 225(1): 1–11.

Pirkwieser, S. and Raidl, G. R. (2010). Multilevel variable neighborhood search for periodic routing problems. In P. Cowling and P. Merz, eds., *Evolutionary Computation in Combinatorial Optimization*. Springer Berlin Heidelberg, 6022: 226–238.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research* 34(8): 2403–2435.

Poggi, M. and Uchoa, E. (2014). New Exact Algorithms for the Capacitated Vehicle Routing Problem. In P. Toth and D. Vigo, eds., *Vehicle Routing Problems. Methods, and Application*, chapter 3. Siam, 2 : 59–86.

Reingold, E. M. and Tarjan, R. E. (1981). On a greedy heuristic for complete matching. *SIAM Journal on Computing* 10(4): 676–681.

Remde, S., Cowling, P., Dahal, K., Colledge, N. and Selensky, E. (2012). An empirical study of hyperheuristics for managing very large sets of low level heuristics. *Journal of the Operational Research Society* 63(3): 392–405.

Remde, S., Dahal, K., Cowling, P. and Colledge, N. (2009). Binary exponential back off for tabu tenure in hyperheuristics. In C. Cotta and P. Cowling, eds., *Evolutionary Computation in Combinatorial Optimization,, European Conference, Evocop 2009*. Springer Berlin Heidelberg, 5482: 109–120.

Rendón, E., Abundez, I., Arizmendi, A. and Quiroz, E. M. (2011). Internal versus External cluster validation indexes. *International Journal of Computers and Communications* 5(1): 27–34.

Robuste, F., Daganzo, C. F. and Souleyrette, R. R. (1990). Implementing vehicle routing models. *Transportation Research Part B: Methodological* 24(4): 263–286.

Ropke, S. (2012). Branching decisions in branch- and-cut-and-price algorithms for vehicle routing problems The Solomon instances are solved ! *Presentation in Column Generation* .

Ropke, S. and Pisinger, D. (2005). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science* 40(4): 455–472.

Ross, P., Marin-Blazquez, J. G., Schulenburg, S. and Hart, E. (2003). Learning a procedure that can solve hard bin-packing problems: A new GA-based approach to hyper-heuristics. In *Genetic and Evolutionary Computation - Gecco 2003*. Springer Berlin Heidelberg, 2724: 1295–1306.

Russell, R. and Igo, W. (1979). An assignment routing problem. *Networks* 9(1): 1–17.

Russell, R. A. and Gribbin, D. (1991). A multiphase approach to the period routing problem. *Networks* 21(7): 747–765.

Sabar, N. R., Ayob, M., Kendall, G. and Qu, R. (2014). A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Cybernetics* 45(2): 217–228.

Salari, M., Toth, P. and Tramontani, A. (2010). An ILP improvement procedure for the Open Vehicle Routing Problem. *Computers and Operations Research* 37(12): 2106–2120.

Salhi, S. and Nagy, G. (1999). A Cluster Insertion Heuristic for Single and Multiple Depot Vehicle Routing Problems with Backhauling. *The Journal of the Operational Research Society* 50(10): 1034–1042.

Savelsbergh, M. W. P. (1991). The Vehicle Routing Problem with Time Windows: Minimizing Route Duration. *Informs Journal on Computing* 4(2): 146–154.

Scarf, P. a. and Cavalcante, C. a. V. (2010). Hybrid block replacement and inspection policies for a multi-component system with heterogeneous component lives. *European Journal of Operational Research* 206(2): 384–394.

Scott, K. (2012). *Investigating Sustainable Solutions for Roadside Gully Pot Management*. Ph.D. thesis.

See, C. H., Horoshenkov, K. V., Abd-alhameed, R. A., Hu, Y. F. and Tait, S. J. (2012). A Low Power Wireless Sensor Network for Gully Pot Monitoring in Urban Catchments. *IEEE Sensors Journal* 12(5): 1545–1553.

Shamir, U. and Howard, C. D. (1979). An analytic approach to scheduling pipe replacement. *Technical Physics* 71(5): 248–258.

Shaw, P. (1998). Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In M. Maher and J.-F. Puget, eds., *Principles and Practice of Constraint Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1520: 417–431.

Shieldsgazette (2012). Flooding hell caused by blocked gully'. Date accessed: 2015-08-24. http://www.shieldsgazette.com/news/local-news/flooding-hell-caused-by-blocked-gully-1-4761698.

Shih, L. H. and Chang, H. C. (2001). A routing and scheduling system for infectious waste collection. *Environmental Modeling and Assessment* 6(4): 261–269.

Smilowitz, K. and Madanat, S. (2000). Optimal inspection and maintenance policies for infrastructure networks. *ComputerAided Civil and Infrastructure Engineering* 15(1): 5–13.

Solomonik, E., Buluç, A. and Demmel, J. (2013). Minimizing communication in all-pairs shortest paths. In *2013 IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE: 548–559.

Soria-Alcaraz, J. A., Ochoa, G., Swan, J., Carpio, M., Puga, H. and Burke, E. K. (2014). Effective learning hyper-heuristics for the course timetabling problem. *European Journal of Operational Research* 238(1): 77–86.

Tagmouti, M., Gendreau, M. and Potvin, J. Y. (2011). A dynamic capacitated arc routing problem with time-dependent service costs. *Transportation Research Part C: Emerging Technologies* 19(1): 20–28.

Taillard, É. (1993). Parallel iterative search methods for vehicle routing problems. *Networks* 23(8): 661–673.

Talbi, E. G. (2009). Metaheuristics: From Design to Implementation. *Proceedings of SPIE - The International Society for Optical Engineering* 42(4): 497–541.

Tan, C. and Beasley, J. (1984). A Heuristic Algorithm for the Period Vehicle Routing Problem. *OMEGA Int. J. Of Mgmt Sci. Pergamon Press.* 12(5): 497–504.

Tang, H., Miller-Hooks, E. and Tomastik, R. (2007). Scheduling technicians for planned maintenance of geographically distributed equipment. *Transportation Research Part E: Logistics and Transportation Review* 43(5): 591–609.

Tarantilis, C. D., Kiranoudis, C. T. and Vassiliadis, V. S. (2004). A threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem. *European Journal of Operational Research* 152(1): 148–158.

Teixeira, J., Antunes, A. P. and de Sousa, J. P. (2004). Recyclable waste collection planninga case study. *European Journal of Operational Research* 158(3): 543–554.

Thieken, A., Ackermann, V., Elmer, F., Kreibich, H., Kuhlman, B., Kunert, U., Maiwald, H., Merz, B., Muller, M., Piroth, K., Schwarz, J., Schwarze, R., Seifert, I. and Seifert, J. (2008). Methods for the evaluation of direct and indirect flood losses. In *4th International Symposium on Flood Defence*: 1–10.

Thierens, D. (2005). An adaptive pursuit strategy for allocating operator probabilities. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. Washington DC, USA: ACM: 1539–1546.

Thimbleby, H. (2003). The directed Chinese Postman Problem. *Software Practice and Experience* 33(11): 1081–1096.

Toth, P. and Vigo, D. (2002). *The Vehicle Routing Problem*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Toth, P. and Vigo, D. (2003). The Granular Tabu Search and Its Application to the Vehicle-Routing Problem. *INFORMS Journal on Computing* 15(4): 333–346.

Transport For London (2007). Highway asset management plan. Technical report.

Tsang, A. H. (1995). Condition-based maintenance: tools and decision making. *Journal of Quality in Maintenance Engineering* 1(3): 3–17.

Uchoa, E., Pecin, D., Pessoa, A. and Poggi, M. (2016). New Benchmark Instances for the Capacitated Vehicle Routing Problem. *European Journal of Operational Research* 257(3): 845–858.

UK GOV (2015). Price Paid Data 2015. Date accessed: 2015-09-09. https://www.gov.uk/government/statistical-data-sets/price-paid-data-downloads.

Vansteenwegen, P. and Oudheusden, D. V. (2007). The mobile tourist guide: an OR opportunity. *OR insight* 20(3): 21–27.

Vansteenwegen, P., Souffriau, W., Berghe, G. V. and Oudheusden, D. V. (2009a). A guided local search metaheuristic for the team orienteering problem. *European Journal of Operational Research* 196(1): 118–127.

Vansteenwegen, P., Souffriau, W. and Oudheusden, D. V. (2011). The orienteering problem: A survey. *European Journal of Operational Research* 209(1): 1–10.

Vansteenwegen, P., Souffriau, W., Vanden Berghe, G. and Van Oudheusden, D. (2009b). Iterated local search for the team orienteering problem with time windows. *Computers and Operations Research* 36(12): 3281–3290.

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

Vidal, T., Crainic, T. G., Gendreau, M. and Prins, C. (2013). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research* 231(1): 1–21.

Voudouris, C., Tsang, E. P. K. and Alsheddy, A. (2010). Guided Local Search. *Wiley Encyclopedia of Operations Research and Management Science* 16(3): 185–218.

Walker, J. D., Ochoa, G., Gendreau, M. and Burke, E. K. (2012). Vehicle routing and adaptive iterated local search within the HyFlex hyper-heuristic framework. In Y. Hamadi, and M. Schoenauer, eds., *Learning and Intelligent Optimization: 6th International Conference*. Berlin, Heidelberg: Springer Berlin Heidelberg, 7219: 265–276.

Weibull, W. (1950). A statistical distribution function of wide applicability. *Journal of applied mechanics* 13(2): 293–297.

Wen, M., Cordeau, J.-F., Laporte, G. and Larsen, J. (2010). The dynamic multi-period vehicle routing problem. *Computers & Operations Research* 37(9): 1615–1623.

Wen, M., Stidsen, T. K., Krapper, E. and Larsen, J. (2011). A multi-level variable neighborhood search heuristic for a practical vehicle routing and driver scheduling problem vehicle routing and driver scheduling problem. *Networks* 58(4): 311–322.

Wu, J., Adam Ng, T. S., Xie, M. and Huang, H. Z. (2010). Analysis of maintenance policies for finite life-cycle multi-state systems. *Computers and Industrial Engineering* 59(4): 638–646.

Xu, J. and Kelly, J. (1996). A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Science* 30(4): 379–393.

Yellow, P. (1970). A computational modification to the savings method of vehicle scheduling. *Journal of the Operational Research Society* 21(2): 281–283.

Yick, J., Mukherjee, B. and Ghosal, D. (2008). Wireless sensor network survey. *Computer Networks* 52(12): 2292–2330.

Yorkpress (2016). York drains crisis: 150 gullies found to be blocked. Data accessed: 2016-08-04. http://www.yorkpress.co.uk/news/14637520.York_drains_crisis__150_gullies_found_to_be_blocked/?ref=mr&lp=15.

Yu, B. and Yang, Z. Z. (2011). An ant colony optimization model: The period vehicle routing problem with time windows. *Transportation Research Part E: Logistics and Transportation Review* 47(2): 166–181.

Zeimpekis, V., Tarantilis, C. D., Giaglis, G. M. and Minis, I. E. (2007). *Dynamic fleet management: concepts, systems, algorithms & case studies*. New York: Springer Science Business Media, LLC.

Zhang, Z., Che, O., Cheang, B., Lim, A. and Qin, H. (2013). A memetic algorithm for the multiperiod vehicle routing problem with profit. *European Journal of Operational Research* 229(3): 573–584.