UNIVERSITY OF SOUTHAMPTON

FACULTY OF SOCIAL, HUMAN AND MATHEMATICAL SCIENCES

Mathematical Sciences

The vehicle routing problem with release and due dates: Formulations, heuristics and lower bounds

Benjamin C Shelbourne MSc

Thesis submitted for the degree of Doctor of Philosophy in Mathematics (Operational Research)

August 2016

Declaration of Authorship

I, Benjamin C Shelbourne, declare that this thesis, titled 'The vehicle routing problem with release and due dates: Formulations, heuristics and lower bounds', and the work presented in it are my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work has been done wholly or mainly while in candidature for a research degree at this University.
2. Where any part of this thesis has been previously submitted for a degree or any other qualification, at this University or any other institution, it is always clearly stated.
3. Where I have consulted the published work of other, it is always clearly attributed.
4. I have acknowledged all main sources of help.
5. Where the thesis is based on work done by myself jointly with others, I have made it exactly clear what has been done by others and what I have contributed myself.
6. None of this work has been published before submission.

Signed: _____

Date: _____

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF SOCIAL, HUMAN AND MATHEMATICAL SCIENCES
Mathematical Sciences

Doctor of Philosophy

THE VEHICLE ROUTING PROBLEM WITH RELEASE AND DUE DATES: FORMULATIONS, HEURISTICS AND LOWER BOUNDS

Benjamin C. Shelbourne

A novel extension of the classical vehicle routing and scheduling problem is proposed that integrates aspects of machine scheduling into vehicle routing. Associated to each customer is a release date that defines the earliest time that the order is available to leave the depot for delivery, and a due date that indicates the time by which the order should ideally be delivered to the customer. The objective is to minimise a convex combination of the operational costs and customer service level, measured as total distance travelled and total weighted tardiness, respectively. A formal definition of the problem is presented, and a variety of benchmark instances are generated to analyse the problem experimentally, and evaluate the performance of any solution approaches developed. Both experimental and theoretical contributions are made in this thesis, and these arise from the development of mixed integer linear programming (MIP) formulations, efficient heuristics, and a Dantzig-Wolfe decomposition and associated column generation algorithm.

The MIP formulations extend commodity flow formulations of the capacitated vehicle routing problem, and are generally related by aggregation or disaggregation of the variables. Although a set of constraints is presented that is only valid for m-commodity flow formulations. A path-relinking algorithm (PRA) is proposed that exploits the efficiency and aggressive improvement of neighbourhood search, but relies on a new path-relinking procedure and advanced population management strategies to navigate the search space effectively. To provide a comparator algorithm to the PRA, we embed the neighbourhood search into a standard iterated local search algorithm. The Dantzig-Wolfe decomposition of the problem yields a traditional set-partitioning formulation, where the pricing problem (PP) is an elementary shortest path problem with resource constraints and weighted tardiness. Two dynamic programming (DP) formulations of the PP are presented, modelling the weighted tardiness of customers in a path as a pointwise function of the release dates, or decomposing the states over the release dates. The CG algorithm relies on a multi-phase pricing algorithm that utilises DP heuristics, and a decremental state-space relaxation algorithm that solves an ng-route relaxation at each iteration.

Extensive computational experiments on the benchmark instances show that the newly defined features have a significant and varied impact on the problem. As a result, finding tight lower bounds and eventually optimal solutions is highly complex, but tight upper bounds can be found efficiently using advanced heuristics.

Acknowledgements

The most acknowledgement for the degree should go to the patience and tolerance of Danielle and Ruth. Especially Danielle, I cannot verbalise my appreciation that you accepted and supported my decision to undertake such an all-consuming task.

I would like to thank my supervisors, Prof Chris Potts and Dr Maria Battarra, for their continued patience, assistance and advice. The many meandering discussions you have humoured during my research, and the relative freedom you allowed, have enabled me to gain a broader view on combinatorial optimisation and a variety of solution techniques. I feel it has been a successful apprenticeship and I am ready to begin my adventure as journeyman.

There have been two particularly important visitors to the University during my research, and both have had an impact on my research in different ways. A special thanks is due to Matteo Salani, for easing my introduction into the area of column generation for vehicle routing problems, especially concerning the implementation. Matteo has continued to support this research with informed advice, and will be a co-author of the resulting article. I would also like to mention Thibaut Vidal, and thank him for interesting discussions and insights about heuristics for vehicle routing problems, particularly during the early stages of my research.

A number of students, staff and visitors have made my time at the University of Southampton both memorable and easier, and I would like to thank them all. I also acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton.

Contents

List of Figures

List of Tables

Dedicated to Danielle, together through thick and thin.

Chapter 1

Introduction

The vehicle routing problem (VRP) is a classical NP-hard combinatorial optimization problem. Starting with the paper of Dantzig and Ramser (1959), it has been studied widely for more than 50 years. The class of VRPs traditionally involve minimizing the total distance travelled by a number of vehicles to visit a set of customers. Commonly, the vehicles are assumed to operate from a single depot, and each customer's demand must be satisfied through a single visit from one of the vehicles, although various studies propose a significant variety of additional features and constraints. There are direct applications in transportation and logistics, and others in a variety of areas including manufacturing, communications, and the military. A more detailed discussion of application areas is given in the book of Toth and Vigo (2002), and more recently by Hoff et al. (2010).

The transportation of goods and services in the supply-chain is an important activity for many businesses. It ensures that resources are in the necessary locations and often connects the business with suppliers and customers. In both the vehicle routing and production scheduling literature, a trend toward simultaneously addressing a greater proportion of the operational supply-chain is observed. Some notable examples include supply chain scheduling problems (Hall and Potts 2003), production-routing problems (Boudia et al. 2007), location-routing problems (Nagy and Salhi 2007), and inventory-routing problems (Coelho et al. 2014). Many of these problems consider integrating operational decisions about distribution with others from different elements of the supply-chain. A problem class of particular interest combines machine scheduling to process customer orders and vehicle routing to deliver the processed orders. A variety of these integrated problems are proposed in the literature, and some examples are described by Chang and Lee (2004), Chen and Vairaktarakis (2005), and Ullrich (2013). Reviews of these problems, and the wider class of integrated machine scheduling and distribution problems, are given by Chen (2010) and Ullrich (2013).

The problems discussed above motivate us to define a VRP where orders become available for delivery to the customers at different times, which we call release dates. Delivery of the orders therefore requires a traditional VRP to be solved, except that the departure time of each vehicle from the depot is dependent on the release dates of the orders on its route. Moreover, the objective is to minimize a convex combination of transportation cost and customer service level, measured as the total distance travelled and total weighted tardiness associated with delivery, respectively. We refer to this problem

as the VRP with release and due dates (VRPRDD). Situations in which release dates can

arise include processing or production of the orders as described above, and more generally any situation where the orders are not immediately available.

As an example, consider a facility that produces personalised goods, such as own or store brand products, or printing services. The facility has a fleet of vehicles and offers a delivery service to its clients. Orders are received and produced continuously, and therefore any orders that are produced before the start of the planning horizon are available immediately, but other orders may not have been completed. Each day, the company must decide how to deliver the required orders to minimise the delivery cost and any penalties for lateness. Notice that other than the products being produced or processed by the company, the problem also contains situations in which products are continuously delivered to the facility.

At the polar extremes of the objective, the VRPRDD models the classical capacitated vehicle routing problem (CVRP), and a parallel machine total weighted tardiness problem with sequence-dependent set-up times (PWTPSST) and additional constraints. Although this generalization of the CVRP and the PWTPSST is of high practical and theoretical importance, to our knowledge it has not been addressed in the literature. A similar problem is being studied in parallel by Johar (2014), but no results have been published. Independently of this study, some preliminary results are presented by Cattaruzza et al. (2014) for a multi-trip VRP with hard time windows and release dates. The latter authors consider batched release dates, a purely transportation cost objective, and vehicles that can perform multiple routes.

1.1 Contributions

Alongside introducing the new problem, this thesis makes a number of contributions. To explore the complexity and tractability of finding optimal solutions for instances of the VRPRDD, we present and compare a number of mixed integer programming formulations. These are commodity flow formulations, and are related by aggregation (respectively disaggregation) of the variables. The comparative quality of the lower bounds produced by the linear relaxations of the formulations, and their effectiveness for different instances, is not immediately clear. To explore these effects, we present an extensive comparison of the performance of the branch and cut algorithm of CPLEX for the different formulations across the benchmark instances. Note that to achieve interesting results in acceptable computation time it is necessary to provide an upper bound to the branch and cut algorithm.

We propose a novel path-relinking algorithm (PRA) to obtain upper bounds for the VRPRDD. This is based on a hybrid evolutionary algorithmic framework rooted in scatter search and concepts proposed by Glover (1989) to enhance tabu search. Our PRA utilizes the exploratory potential and adaptive memory of evolutionary algorithms, but relies on more intelligent solution recombination and population (diversity) management. A new relinking procedure is described to enable efficient exploration of the search space between the solutions in the population, and which introduces some controlled randomization. To intensify the search and improve the convergence of the algorithm, an efficient and powerful neighbourhood search (NS) is applied to some solutions resulting from relinking. With the aim of assessing the performance of our proposed PRA against an alternative method, we extend the NS from the PRA to produce an iterated local search (ILS) algorithm. An ILS algorithm iteratively

applies NS followed by a kick to escape the resulting local optima. Examples for a broad range of problem types show that ILS achieves competitive results (see Lourenço et al. 2010).

Given the complexity of solving the VRPRDD to optimality, we describe a Dantzig-Wolfe decomposition of the problem. This yields a set-covering formulation (SC), where the set of variables corresponds to the set of feasible routes. To manage the exponential number of variables, the linear relaxation of the SC formulation is solved using a (delayed) column generation (CG) algorithm (see, e.g. Lübbecke and Desrosiers 2005). The pricing problem (PP) is a novel elementary shortest path problem with resource constraints and release and due dates, which is unary NP-hard. Two DP formulations of the PP are proposed, the first models the weighted tardiness of a path as a function of the departure time from the depot, and other relies the observation that this function decomposes over the release dates. A multi-phase pricing algorithm is suggested, which utilises a parameterised DP heuristic to generate columns efficiently, and a decremental state space relaxation (DSSR) algorithm based on the ng-route relaxation.

1.2 Outline of the thesis

The remainder of the thesis is organized as follows. Chapter 2 introduces the theoretical concepts and algorithmic solution approaches considered in the thesis. In Chapter 3, the VRPRDD is formally introduced and a proposed set of benchmark instances is described. In light of the problem definition, a thorough review of the literature for VRPs similar to the VRPRDD is presented in Chapter 4. Chapter 5 presents mathematical programming formulations of the VRPRDD, and compares the resulting performance of the CPLEX branch and cut algorithm. Our PRA is described in detail in Chapter 6 and computational results are reported. Chapter 7 illustrates the Dantzig-Wolfe decomposition, describes the CG and DP algorithms in detail, and reports computational experience. Lastly, Chapter 8 summarises and concludes our findings, and discusses avenues for further research.

Bibliography

Boudia, M., Louly, M., and Prins, C. (2007). A reactive grasp and path relinking for a combined production-distribution problem. *Computers & Operations Research*, 34(11):3402–3419.

Cattaruzza, D., Absi, N., and Feillet, D. (2014). The multi-trip vehicle routing problem with time windows and release dates. Working Paper EMSE CMP:SFL 2014/1.

Chang, Y.-C. and Lee, C.-Y. (2004). Machine scheduling with job delivery coordination. *European Journal of Operational Research*, 158(2):470–487.

Chen, Z.-L. (2010). Integrated production and outbound distribution scheduling: Review and extensions. *Operations Research*, 58(1):130–148.

Chen, Z.-L. and Vairaktarakis, G. L. (2005). Integrated scheduling of production and distribution operations. *Management Science*, 51(4):614–628.

Coelho, L. C., Cordeau, J.-F., and Laporte, G. (2014). Thirty years of inventory-routing. *Transportation Science*, 48:1–19.

Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, 6(1):80–91.

Glover, F. (1989). Tabu search - Part 1. *ORSA Journal on Computing*, 1(2):190–206.

Hall, N. G. and Potts, C. N. (2003). Supply chain scheduling: Batching and delivery. *Operations Research*, 51(4):566–584.

Hoff, A., Andersson, H., Christiansen, M., Hasle, G., and Løkketangen, A. (2010). Industrial aspects and literature survey: Fleet composition and routing. *Computers & Operations Research*, 37(12):2041–2061.

Johar, F. (2014). PhD progress report, University of Southampton.

Lourenço, H. R., Martin, O. C., and St utzle, T. (2010). Iterated local search: Framework and applications. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 363–397. Springer, New York, USA.

Lübbecke, M. E. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, 53(6):1007–1023.

Nagy, G. and Salhi, S. (2007). Location-routing: Issues, models and methods. *European Journal of Operational Research*, 177(2):649 – 672.

Toth, P. and Vigo, D. (2002). *The vehicle routing problem*. SIAM, Philadelphia, USA.

Ullrich, C. A. (2013). Integrated machine scheduling and vehicle routing with time windows. *European Journal of Operational Research*, 227(1):152 – 165.

Chapter 2

Methodology

Combinatorial optimisation (CO) is concerned with solving *CO problems* (COPs), or finding an optimal object from a finite set. These objects or solutions have a discrete or combinatorial structure, for example, a partition or permutation of a set, or a type of graph. In this chapter, we give a brief introduction to COPs and algorithmic approaches to address these problems. More detailed introductions to the area are provided by Steiglitz and Papadimitriou (1982), Nemhauser and Wolsey (1988), Schrijver (2003), and more recently by Cook et al. (2011). These books have also influenced the presentation of some of the material in this chapter.

COPs historically arose in economics and the management of resources and operations, and it is now clear that COPs are found in a significant variety of applications. This is highlighted by examples such as: production, inventory, and transportation planning; network design, planning and analysis; resource allocation; health-care and medicine; biology; and investment and portfolio planning. This list is not meant to be exhaustive and examples or applications of COPs are continually discovered.

The availability and accuracy of problem information over time leads to a variety of problem types, ranging from perfect information at the start of the planning horizon in static and deterministic problems, to no warning until the time of the event in online and real-time problems. Between these extremes are stochastic and robust problems in which some distribution or interval is known for the information, respectively. In this thesis, we will only consider static and deterministic COPs, but the interested reader is referred to Kleywegt et al. (2002) and Hentenryck and Bent (2009) for more on problems with unknown or uncertain information.

The remainder of the chapter is organised as follows. § 2.1 formally defines a general COP and introduces some examples. § 2.2 describes a classification of types of algorithms and the general concepts behind computational complexity theory. § 2.3 introduces a variety of basic heuristics and presents some examples of metaheuristic frameworks. § 2.4 introduces techniques that can solve a COP in finite time by implicitly enumerating the set of feasible solutions, and § 2.5 briefly introduces linear programming and its combinatorial aspects. This is developed in § 2.6, where integer programming is presented, and a number of techniques frequently used to solve these models are explained using examples.

2.1 Combinatorial optimisation problems

A definition of a more general optimisation problem that includes COPs, can be stated as follows.

Definition 2.1. An *optimisation problem* is a set of problem instances.

A problem is a potentially infinite collection of instances, and each of these instances represents a specific example of the problem.

Definition 2.2. An *instance* of an optimisation problem is a pair (S, f), where S is the set of *feasible solutions*, or the domain, and $f : S \to \mathbb{R}$ is the *objective function*. An *optimal solution* $x^* \in S$ for a minimisation (maximisation) problem is such that $f(x^*) \leq f(x) : x \in S$ $\big(f(x^*) \geq f(x) : x \in S\big)$.

Without loss of generality, we consider minimisation problem only and for maximisation problems it is equivalent to minimise the negative of the objective function. In the following, we will state a general COP as $P = \{\min f(x) : x \in S\}$, and only the case for minimisation problems is presented unless it is not trivial to adapt to maximisation.

Generally, an instance is not given explicitly, but is represented by a set of *problem parameters*, and a statement of the properties that a feasible solution must satisfy, which are often called *constraints*. In this case, an instance is defined by a set of values for the problem parameters. *Solutions* are usually represented by a set of bounded *decision variables*, and an assignment of values to these variables represents an individual solution.

Definition 2.3 (*0-1 knapsack problem* (KP))**.** Given a capacity $W \in \mathbb{Z}^+$ and a set of n items V, where each item $i \in V$ has weight $w_i \in \mathbb{Z}^+$ and value $v_i \in \mathbb{Z}^+$. Find a subset of the items with maximum total value such that the total weight is less than W.

For the KP, we can choose $S = \{X \subseteq V : \sum_{i \in X} w_i \leq W\}$, where $|S| = O(2^n)$, and $f(X) = \sum_{i \in X} v_i$.

Definition 2.4 (*travelling salesman problem* (TSP))**.** Given a set of n cities V and a symmetric $n \times n$ distance matrix d_{ij}, where $d_{ij} \in \mathbb{Z}^+$ and represents the distance between cities i and j. Find the shortest tour that visits every city once.

Note that a *tour* is a closed path or cycle, and because d_{ij} is symmetric, then $d_{ij} = d_{ji}$ for $i, j \in V$. For the TSP, if we define σ as a cyclic permutation of V, where $\sigma(i)$ is the customer visited after $i \in V$, then S can be chosen as all cyclic permutations of V and $f(\sigma) = \sum_{i \in V} d_{i\sigma(i)}$. Notice that there are $|S| = ((n-1)/2)!$ unique solutions in S, where the division is due to symmetry in d_{ij}, and S represents all Hamiltonian cycles in the connected graph K_n where the set of vertices is V.

Definition 2.5 (*weighted completion time scheduling problem* (WCSP))**.** Given a set V of n jobs, where each job $i \in V$ has processing time $p_i \in \mathbb{Z}^+$ and weight $w_i \in \mathbb{Z}^+$. Find a schedule of the jobs which minimises the total of the individual job completion times multiplied by their weights.

For the WCSP, if we define π as a permutation of V, where $\pi(i)$ is the i^{th} job. Then S can be chosen as all permutations of V and

$$f(\pi) = \sum_{i=1}^{n} w_{\pi(i)} \sum_{j=1}^{i} p_{\pi(j)}. \tag{2.1}$$

Now $|S| = n!$, because the permutation π is not a cycle and there is no symmetry between solutions.

2.2 Algorithms and computational complexity

To find a solution to an instance of a given COP we can apply an algorithm. An *algorithm* is formally defined using a model of computation, such as a Turing machine, but for our purposes it will be sufficient to define an algorithm as a general sequence of instructions, which will take some input and return some output in a finite number of steps.

Definition 2.6. An algorithm is said to *solve* a problem, or is an *exact algorithm* for a problem, if it is guaranteed to return an optimal solution when the input is any instance of the problem.

2.2.1 Computational complexity

Given two exact algorithms for a problem, we are then concerned with identifying the most "efficient". We can measure the efficiency of an algorithm using the memory (or space) and time requirements. The time required is often the most critical resource and this is usually the primary concern. For many problems, a "sufficiently" fast algorithm has yet to be identified, and some theories exist which try to explain this phenomenon.

There are a variety of factors which can make a COP difficult or impossible to solve in acceptable time. For example, the size of the problem, the complexity of the constraints, and the behaviour of the objective function. One of the most important concepts in understanding this boundary is the theory of NP-completeness. We will give a brief outline of the theory and ramifications, but more rigorous and detailed treatments are given by Garey and Johnson (1979) and Papadimitriou (1994), and more recently by Arora and Barak (2009).

The time requirements of an algorithm when applied to an instance of a problem are measured as a function of the *instance size*, or the number of bits needed to store the instance. Formally, to find the instance size we must decide on an encoding, find the cardinality of the set of symbols needed to describe an instance, and choose a model of computation. Although this is frequently measured informally, and the broad results in the theory are robust to this. For example, in the TSP, KP and WCSP, the instance size is informally taken to be n. The *time complexity* function of an algorithm is an upper bound on the time requirements of the algorithm for an instance of a problem as a function of the instance size.

Definition 2.7. We say that a function $f(n) = O(g(n))$, if there exists a constant c such that $f(n) \leq cg(n)$ for $n \geq 0$.

A simple yet key distinction is made by Cobham (1965), between algorithms with time requirements bounded by a polynomial function, and those with time requirements which cannot be bounded by a polynomial function.

Definition 2.8. For a problem where the instance size is given by n, an algorithm is a *polynomial-time algorithm* for the problem, or has *polynomial complexity*, if the time complexity function is $O(p(n))$ and p is a polynomial function.

Definition 2.9. If the time complexity function of an algorithm for a problem cannot be bounded by a polynomial function of the instance size, it is an *exponential-time algorithm* for the problem, or has *exponential complexity*.

Note that the definition of exponential-time algorithms includes super-polynomial functions which are not traditionally described as exponential. The position taken by Cobham, amongst others, is that exponential-time algorithms are not computationally practical.

For any COP, a brute-force approach that simply enumerates the feasible solutions is an exact algorithm with at least exponential complexity. To achieve better results, the specific structure of the COP considered must be exploited. For example, the *shortest weighted processing time rule* of Smith (1956) proves that an optimal solution to the WCSP has jobs $i \in V$ ordered by non-increasing weighted processing times p_i/w_i. We can calculate the p_i/w_i values in $O(n)$ and sort V using a standard sorting algorithm like *merge sort* in $O(n \log n)$. This yields a polynomial-time exact algorithm for the WCSP.

Analysis of the time complexity function and the distinction between polynomial- and exponential-time algorithms is widely accepted, but there are a number of caveats. An upper bound on the time required by an algorithm is a *worst-case* analysis, and therefore defines the complexity of an algorithm for the most complex instances. The time complexity is also an asymptotic measure, and therefore constant factors and lower order terms are ignored. For this reason, the accuracy of the time complexity function increases as the instance size tends to infinity, and it may not always be accurate for instances with "typical" or practical size.

A classification of the "hardness" of COPs follows from the time complexity of the most efficient algorithm that can solve a problem. This requires the definition of a decision problem, as a problem with a binary solution yes or no. Computational models, such as the Turing machine, can be used to classify decision problems that can be solved in polynomial time using a deterministic computer or using a non-deterministic computer. A very informal description of a non-deterministic computer is that it performs all operations simultaneously (or guesses the correct operation) at each step, and therefore performs an exponential number of operations in a polynomial number of steps and polynomial time. These ideas lead to the following classes of problems.

Definition 2.10. Decision problems that can be solved in polynomial time on a *deterministic computer* are in the class $\mathcal{P}$, and those which can be solved in polynomial time on a *non-deterministic computer* are in the class $\mathcal{NP}$.

From this definition, it is clear that $\mathcal{P} \subseteq \mathcal{NP}$. It is widely conjectured that $\mathcal{P} \neq \mathcal{NP}$ and therefore $\mathcal{P} \subset \mathcal{NP}$, but no valid proof or disproof has been published. This is despite many attempts and a one million dollar prize from the Clay Mathematics Institute

The conjecture and the foundations of the theory of NP-completeness are established in the seminal work by Cook (1971). He highlights the importance of *polynomial reducibility*, describing problems which can be reduced to other problems using a mapping which requires

only polynomial time. The critical importance of this type of reducibility is due to the following definition.

Definition 2.11. Given a polynomial time algorithm A for a problem P, if a problem Q is *polynomially reducible* to P then A is also a polynomial time algorithm for Q.

Further, this reveals the importance of the *NP-complete* problems in $\mathcal{NP}$, which have polynomial reductions to all other problems in $\mathcal{NP}$. Therefore any polynomial time algorithm for an NP-complete problem will solve all problems in $\mathcal{NP}$ in polynomial time, and this leads some to describe these as the "hardest" problems in $\mathcal{NP}$. As a key example, Cook proves that the *satisfiability problem* is NP-complete.

A problem P is described as *NP-hard*, if it is at least as hard as the problems in $\mathcal{NP}$. In this sense, a problem P is NP-hard if there is a polynomially reduction from at least one NP-complete or NP-hard problem. Notice that this definition allows for problems that are not necessarily decision problems or in $\mathcal{NP}$, but a polynomial-time algorithm for an NP-hard problem will still solve all problems in $\mathcal{NP}$ in polynomial time.

The relationship between optimisation problems and decision problems can be defined as follows.

Definition 2.12. Given a COP, $P = \{\min f(x) : x \in S\}$ and some number L, the *recognition version* R can be stated as "is there a solution $x \in S : f(x) \leq L$?".

There are polynomial reductions between P and R, defined as follows. The reduction from P to R is trivial because if we solve P and $f(x^*) \leq L$ then the answer to R is yes and otherwise is no. In the opposite direction, if we can bound the optimal objective value in polynomial time to take v different values, then a dichotomous search on f using R will require polynomial time if $\log v = O(p(n))$ and p is a polynomial function of the instance size n. Notice that the recognition version is easier than the optimisation problem, and this leads to the following definition.

Definition 2.13. An optimisation problem is *NP-hard* if the recognition version is NP-complete or NP-hard, or there is a polynomial reduction from another NP-hard problem.

This theory is enriched significantly by Karp (1972), who proves that the recognition versions (or closely related decision problems) of a range of 21 COPs are NP-complete. A significant number of problems are now known to be NP-complete or NP-hard, and an exposition is found in Garey and Johnson (1979). Notice that exhibiting a polynomial-time exact algorithm for a problem, such as shown for the WCSP, proves that this problem is in $\mathcal{P}$. Although this does not extend to the higher-order complexity classes, because demonstrating an exponential-time exact algorithm does not prove a problem is NP-complete or NP-hard.

A further classification amongst NP-hard problems can be made if we consider the choice of encoding. A unary encoding uses numbers of base one, and therefore a number is stored as a string of ones with length equal to value, but if a higher base is used then less symbols are required to describe a number. This leads to the definition of *binary* NP-hard problems, or problems that are NP-hard *in the ordinary sense*. These problems are NP-hard using a binary or higher base encoding but are polynomially solvable using a unary encoding. The most efficient algorithms for these problems need *pseudo-polynomial* time using a binary or

higher encoding. An example of this is demonstrated for the KP in § 2.4, where a dynamic programming algorithm for the KP is shown to require $O(nW)$ time. For this algorithm to be polynomial using a binary encoding then it must be polynomial in n and $\log_2 W$ rather than n and W.

2.2.2 Upper and lower bounds

Assuming $\mathcal{P} \neq \mathcal{NP}$, then an optimal solution cannot be found in polynomial time for every instance of an NP-hard problem. Either the optimality or generality requirement must be relaxed to find efficient algorithms for NP-hard problems, but without significant motivation it is often not practical to consider finding efficient algorithms to solve specially defined subsets of instances. For this reason, it is usual to consider retaining instance generality and efficiency but relaxing the optimality requirement to near-optimal solutions.

Definition 2.14. A *heuristic* for a problem is an algorithm which attempts to find a feasible solution x', and for a minimisation (maximisation) problem x' provides an *upper bound* (*lower bound*) on the optimal objective value, $f(x') \geq f(x^*)$ ($f(x') \leq f(x^*)$).

For some problems, heuristics can be designed which guarantee the worst-case performance compared to the optimal objective value.

Definition 2.15. For $\alpha > 1$ ($0 \leq \alpha < 1$), an *α-approximation algorithm* for a minimisation (maximisation) problem is guaranteed to find a solution $x' \in S$ in polynomial time, and $f(x') \leq \alpha f(x^*)$ ($f(x') \geq \alpha f(x^*)$) for any instance of the problem.

For an α-approximation, the inequality must be true for all instances. It must also be *tight* and therefore equal for at least one instance or for a series of instances must tend to equality.

Some COPs cannot be approximated within a constant factor unless $\mathcal{P} = \mathcal{NP}$, but for others we have good approximations.

Definition 2.16. A *polynomial time approximation scheme* for a minimisation (maximisation) problem is a family of algorithms, which for $\epsilon \geq 0$ ($0 < \epsilon \leq 1$) provide a $(1+\epsilon)$-approximation ($(1-\epsilon)$-approximation) algorithm for any instance of the problem.

Definition 2.17. A *fully polynomial time approximation scheme* for a minimisation (maximisation) problem is a family of algorithms, which for $\epsilon > 0$ ($0 < \epsilon \leq 1$) provide a $(1+\epsilon)$-approximation ($(1-\epsilon)$-approximation) algorithm for any instance of the problem with time requirements bounded by a polynomial in both the instance size and $1/\epsilon$.

Notice that the hardness of COPs can be further classified by considering approximability and the strength of approximation, problems that are easier to solve admit tighter approximation algorithms.

It is also possible to approach the optimal objective value from the opposite direction. For a minimisation (maximisation) problem, a *lower bound* (*upper bound*) attempts to find a solution x', which is infeasible unless it is the optimal solution, and x' provides a lower (upper) bound on the optimal objective value, $f(x') \leq f(x^*)$ ($f(x') \geq f(x^*)$). These bounds are often obtained by solving *relaxations* of the original problem, where different approaches

are applied to relax the constraints. If non-redundant constraints are relaxed, the feasible set of solutions for the relaxed problem contains the solutions for the original problem but also allows infeasible solutions which the constraints prevented. Lower (upper) bounds are sometimes termed dual-heuristics and heuristics are then termed primal-heuristics. Similar to approximations algorithms, the difference between the objective value of the optimal solution and the bound can be guaranteed for some problems.

Finding tight upper and lower bounds is important for problems where an optimal solution cannot be found efficiently for all instances, because upper and lower bounds must be compared to judge their quality. As we will also come to see, both upper and lower bounds are important in many approaches to solving NP-hard COPs in acceptable time.

2.3 Heuristics

In this section, we will describe some key categorisations and design templates for heuristics. In the latter parts of the section, we briefly introduce an area of research which is attempting to theoretically analyse heuristics and their performance on different problems.

The design and engineering of many heuristics is empirical and can be considered an art form. Even with worst-case bounds, a clear appreciation of the performance of a heuristic requires experimentation. This consists of extensive computational testing on a representative set of problem instances. Statistics can then be used to compare the results obtained and identify significant differences in performance between heuristics or problem variants. This can also lead to the development of empirical bounds on the objective values produced by heuristics. An interesting discussion of the many facets of experimental analysis for heuristics and some guidance on good and bad practice is given by Johnson (2002).

2.3.1 Constructive heuristics

A natural approach to find a feasible solution to a COP is to build a solution step-by-step. This idea is encapsulated by *constructive heuristics*, which sequentially extend a partial solution to construct a complete solution. Depending on the problem, these heuristics may require some initialisation or start with an empty solution, and heuristic criteria are used to decide on the extension at each stage. Extensions are definitive and therefore once an extension is chosen it cannot be changed, but it may be influenced by latter extensions. The time complexity of constructive heuristics is usually lower than most other algorithms, but the solutions produced are often far from optimal.

A COP can be represented by defining a set of n elements E, and a solution is then a feasible subset of E. A skeleton of a constructive heuristic using this representation is given in Algorithm 2.1, where $X \subseteq E$ is a partial solution until the last iteration and $R = E \backslash X$ is the set of remaining elements. The function Select(X, R) returns an element $i \in R$ to include in X according to some criteria.

For the KP, an example of a constructive heuristic is to start from an empty solution, and select at each extension the item i from the remaining items which minimises v_i/w_i. This extension is feasible if w_i does not exceed the remaining capacity. If every item has been considered, or the remaining capacity is lower than the item remaining with smallest weight, then the solution is complete. If we consider each item at each stage of the construction then

Algorithm 2.1 Outline of constructive heuristics

```
1: Set R = E and initialise X to ∅ or initial element
2: while X not complete
3:     i = Select(X, R)
4:     if X ∪ {i} is feasible then
5:         Include i in X and remove from R
6: return X
```

the time complexity is $O(n^2)$. If we first sort the items by non-decreasing v_i/w_i in $O(n \log n)$ time, then the construction takes $O(n)$ time, and the time complexity is therefore $O(n \log n)$. We will see in § 2.6 that this constructive approach is optimal for the linear relaxation of the KP.

For some problems the criteria used to select the next extension can be defined exactly, and an optimal solution is produced by the heuristic. For the WCSP, if we start from an empty solution then it is optimal at each stage to select the job i from the remaining jobs with lowest p_i/w_i. This not true for most COPs, and it not true of the heuristic for the KP presented above. For further details and a discussion of the mathematical structure of problems with this property, the reader is referred to Korte and Lovász (1981).

2.3.2 Neighbourhood search

The decisions made in constructive heuristics are definite, and therefore changes cannot be made to the solution. *Local improvement* or *neighbourhood (descent) search* (NS) heuristics remedy this issue by applying small modifications to a complete solution. Given an initial solution, these heuristics search neighbourhoods of the current solution and move to the best neighbour. The neighbours of a solution are the set of solutions reachable by applying a simple modification or *move*.

A defining feature of these heuristics is the exploration approach, if a neighbour with better objective value than the current solution is found, then this is taken to be the new current solution. This process is repeated until no better neighbours are found. Despite the relative simplicity of this "trial and error" approach, NS is an important heuristic for many COPs. In the following sections, it will become clear that neighbourhoods and NS are important concepts for many more advanced heuristics.

We formally define a neighbourhood as follows.

Definition 2.18. Let $P = \{\min f(x) \colon x \in S\}$ be an optimisation problem, a *neighbourhood* is a mapping $N : S \to 2^S$. For each solution $x \in S$, the mapping defines a set of neighbours $N(x) \subseteq S \backslash \{x\}$.

A neighbourhood is *symmetric* if for any two solutions $x, y \in S$, $x \in N(y)$ and $y \in N(x)$. A neighbourhood N connects a feasible solution $x \in S$ to the set of solutions $N(x)$, and these mappings induce a graph or search-space on the set of feasible solutions.

Definition 2.19. Let $P = \{\min f(x) \colon x \in S\}$ be a problem and N be a neighbourhood. The *search space of P with respect to N*, or *search-space of P* if N is clear from the context,

is the graph $G_s(S, A)$, where $A = \{(x, y) : x \in S, y \in N(x)\}$. A *search-trajectory* is a (not necessarily simple) path in G_s.

An outline of NS is given in Algorithm 2.2, where x_c is the current solution, and N is a neighbourhood. The procedure $Select(N(x))$ returns a neighbour $x' \in N(x)$ of $x \in S$, such that $f(x') < f(x)$, and if no such neighbours exist then it returns $\emptyset$. The search-space is explored starting from the initial solution by iteratively selecting a neighbour of the current solution with better objective value and moving to this solution. This process continues until no such neighbours can be found.

Algorithm 2.2 Outline of NS

1: Generate initial solution x_c
2: **while** Select$(N(x_c)) \neq \emptyset$
3: $\quad x_c = \text{Select}(N(x_c))$
4: **return** x_c

Definition 2.20. Let $P = \{\min f(x) : x \in S\}$ be a COP and N a neighbourhood. A solution $\bar{x} \in S$ is *locally optimal with respect to* N if $f(\bar{x}) \leq f(x) : x \in N(\bar{x})$, or simply *locally optimal* if N is clear from the context.

Notice that a locally optimal solution for a neighbourhood is not necessarily locally optimal for other neighbourhoods. Some neighbourhoods can guarantee that a locally optimal solution is the optimal solution, and these are defined as follows.

Definition 2.21. Let P be a COP and N a neighbourhood. N is an *exact neighbourhood* for P if any locally optimal solution is guaranteed to be an optimal solution.

Some exact neighbourhoods can be searched efficiently, although for most COPs the size of the neighbourhoods is prohibitive and searching requires exponential time. For example, Papadimitriou and Steiglitz (1977) show that searching an exact neighbourhood for the TSP cannot be performed in polynomial time unless $\mathcal{P} = \mathcal{NP}$. These concepts point to an important trade-off in NS, between the size of the neighbourhood and therefore the time taken to search it, and the increased chance of finding good quality neighbours in larger neighbourhoods.

An example of a neighbourhood for the TSP is 2-*Opt* neighbourhood, proposed by Lin (1965). Let (i, j) represent an edge between cities i and j in a solution, 2-Opt removes edges (i, j) and (u, v) from the tour and introduces (v, j) and (u, i). The size of the neighbourhood is $n(n-3)/2 = O(n^2)$. Figure 2.1 shows two tours for an instance of TSP with 6 cities, and the tours are neighbours in the 2-Opt neighbourhood. The neighbourhood can be generalised to k-*Opt* by removing and introducing k arcs, although the size of the neighbourhood is then $O(n^k)$. Notice that k-Opt is exact for $k = n$, but has exponential size.

A neighbourhood for the KP, which we will call 1-*replace*, can be defined as removing an item $i \in X$ and replacing it with an item $j \in V \backslash X$. A 1-replace move is feasible if the new total weight does not exceed the capacity W. Let $\bar{n} = |X|$ be the cardinality of the current solution $X \subseteq V$, then the size of this neighbourhood is $\bar{n}(n - \bar{n}) = O(n^2)$. We can generalise

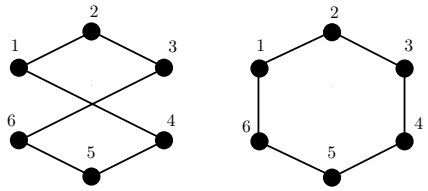

Figure 2.1: An instance of TSP, and two tours reachable from each other using a 2-Opt move

this neighbourhood to *k-replace*, where we remove k items from X and replace these with k items from $V \backslash X$. The size of the neighbourhood is now $O(n^{2k})$. The neighbourhood is exact if k is the cardinality of the optimal solution, but to find this requires solving the problem.

Given the difficulty of searching exact and generally exponential neighbourhoods in many COPs, Kernighan and Lin (1970) propose a method called *variable depth search*. This can be viewed as searching for a sequence of moves to construct a higher order compound move. In the TSP for example, any k-Opt move can be achieved through a combination of 2-Opt moves.

Ideally, a compound neighbourhood can be searched in polynomial time, either heuristically, or optimally through *dynasearch* as proposed by Congram et al. (2002). A number of other polynomially searchable exponential neighbourhoods are proposed for different COPs, and often these exploit the special structure of the problem. Although for many of the hardest or harder COPs, no polynomially solvable exponential neighbourhoods have been found. A review of these concepts, grouped under the term *very large-scale neighbourhood search*, is given by Altner et al. (2014).

A solution may have a number of neighbours with better objective value, and a strategy to choose a neighbour from these is called a *pivoting rule*. A number of pivoting rules are traditionally used in NS, and two extremes are represented by first-improvement and best-improvement. *First-improvement* selects the first neighbour encountered with better objective value than the current solution, and *best-improvement* selects the neighbour with best objective value from those with better objective value than the current solution.

Commonly in first-improvement, the neighbours are randomly ordered before evaluating their objective values. In this case, application of a first-improvement NS to the same initial solution may produce different local optima, whereas for best-improvement it will be the same local optimum, unless ties are broken with some randomisation.

Neither of these pivoting rules dominates the other in all applications. Each iteration of first-improvement is faster, because the full neighbourhood is only searched when the current solution is locally optimal. In any other iteration, less neighbours may be evaluated before one is found with better objective value. Although best-improvement searches the full neighbourhood in every iteration, the improvement in objective value may frequently be larger.

A single type of neighbourhoods is used in NS traditionally. More recently, a number of different neighbourhoods are used in combination. Two extreme approaches to searching multiple neighbourhoods are first-improvement or partial search on the union of the neighbourhoods, or considering the individual neighbourhoods in some order.

Using more than one neighbourhood decreases the number of local optima, and also increases the connectivity of the search-space. Therefore, NS with multiple neighbourhoods

has greater potential to explore the search-space, and we can expect locally optimal solutions with better objective values in general. The total computation time of a NS may also be improved through using multiple neighbourhoods.

An example of a static neighbourhood ordering, termed *variable neighbourhood descent* by Mladenović and Hansen (1997), is to order the neighbourhoods by non-decreasing complexity. When a neighbour with better objective value cannot be found then the next neighbourhood is searched, and if a new current solution is found the search returns to the smallest neighbourhood.

A simple dynamic ordering of the neighbourhoods would be to randomly order the neighbourhoods each time a new current solution is found. A completely random ordering may be improved by using the past performance of neighbourhoods in the search to encourage certain orderings, and this adaptive principle to select neighbourhoods will be seen in more detail in a following section.

An important consideration when applying NS is how to obtain the initial solution. Constructive heuristics are often used but usually some level of randomisation is introduced to the construction process. Constructive heuristics will often produce solutions with acceptable objective value and that are close to the boundary between feasibility and infeasibility (the constraints are tight). As a result, this often permits less locally optimal solutions to be sampled when applying NS to these solutions. Introducing randomisation into the initial solution permits a greater number of initial solutions. This effect can be particularly important for *multi-start NS* heuristics, which perform a series of independent executions of NS and return the best solution found in all executions.

A NS heuristic repeatedly searches neighbourhoods to find an improving neighbour, and this requires evaluating a large number of solutions. This is a computational bottleneck in the process, and many different techniques are proposed to reduce this. These techniques mainly focus on defining small but effective neighbourhoods or equivalently neighbourhood restrictions, and reducing the complexity of evaluating a neighbour.

We define a *solution attribute* as an element of a solution, for example this could be an edge used in a solution of the TSP. The size of a neighbourhood can be reduced by restricting our attention to neighbours which involve attributes that we consider likely to feature in the optimal solution. In the TSP, if a list of the δ closest cities for each city is stored, then the neighbourhoods can be restricted to solutions where these cities are adjacent. This approach reduces the size of most neighbourhoods for the TSP by a factor of $O(n - \delta)$, and often has a negligible effect on the final objective function value (Johnson and McGeoch 1997). For example, if we consider removing a city and inserting it in another position in the tour, this requires $O(n^2)$ evaluations for the whole neighbourhood. If we only consider inserting the city after one of the δ closest cities, there are $O(n\delta)$ evaluations.

We can evaluate a solution to the TSP by considering each city in turn, which requires $O(n)$ time. If we consider applying a 2-Opt move, it is more efficient to calculate the improvement in the objective value of the current solution due to the move. If a 2-Opt move removes arcs (i, j) and (u, v) and introduces (v, j) and (u, i), the improvement is calculated as $d_{ij} + d_{uv} - d_{vj} - d_{ui}$ and requires $O(1)$ time. Notice that memory structures can also be used to store the evaluation of moves or solutions in a table or map. For problems with complex evaluation of solutions this can often allow considerable reductions in computation time.

2.3.3 Metaheuristics

In this section, we will give descriptions of a variety of popular metaheuristic frameworks. Metaheuristics have received considerable attention as a technique to find solutions to combinatorial optimisation problems in acceptable computational time, as evidenced in the surveys of Osman and Laporte (1996), Blum and Roli (2003), Gendreau and Potvin (2005), and a number of books including those by Talbi (2009) and Gendreau and Potvin (2010). We also refer the reader to these surveys and books for further details and other frameworks.

Metaheuristics are problem-independent heuristic frameworks that act to guide and develop problem-specific heuristics. Although there is no guarantee on optimality or approximation, heuristics based on these frameworks often achieve "good" objective values in "acceptable" time. To avoid degeneracies and escape difficult parts of the domain, one of the key features of metaheuristics is introducing randomisation and diversification, and the balance with intensification. Blum and Roli (2003) give a conceptual comparison of metaheuristics, and explore the balance between intensification and diversification further.

The definition of metaheuristics suggests hybridisation, and clearly in hybrid genetic algorithms (HGAs) and path-relinking algorithms (PRAs) it is part of the framework. A classification of the approaches to hybridise metaheuristics clarifies many of the design choices and evaluation of strategies which perform well. A number of authors have attempted to classify the broad scope of hybrid methods, including Talbi (2002), Raidl (2006), Raidl et al. (2010), and Blum et al. (2011). A clear description of a number of hybrid metaheuristics is given in the tutorial by Blum (2012).

If, at either level of the framework, an algorithm is used that solves the respective subproblem to optimality, this is commonly termed a *matheuristic*. Some reviews of the approaches to hybridising exact algorithms and metaheuristics are given by Maniezzo et al. (2009). The book by Alba (2005) describes parallel metaheuristics, which can be categorised as *cooperative* if the metaheuristics exchange information during the search, and otherwise as *competitive*. These approaches are outside the scope of the thesis, but we refer to papers by Crainic and Toulouse (2010) and Alba et al. (2013) for further reviews.

Tabu search

Tabu search (TS) is an early metaheuristic proposed by Glover (1986). The basic approach searches neighbourhoods like NS, but also forbids certain solutions and therefore accepts the best possible solution in the restricted neighbourhood. A *short term memory* or *tabu list* is maintained to dynamically restrict the neighbourhoods, forbidding certain solutions being visited for a number of iterations.

The principles of the method are also proposed independently by Hansen (1986), under the name *steepest ascent mildest descent* (referring to a maximisation problem). In the past, many implementations of TS for different COPs have been competitive. More recently, this framework appears less popular and this may be attributable to the lack of randomisation.

The tabu list can be composed of the forbidden solutions explicitly, although for many COPs this is computationally expensive and instead characterisations of moves are stored. If we define an *attribute* as a feature present in a solution, then a move can be characterised by a set of attributes. Usually the *tabu list* prevents the reversal of recently applied moves

or types of moves by storing a list of attributes involved in the τ most recent moves. The parameter τ is termed the *tabu tenure*.

Consider the KP and the neighbourhood 1-replace. We define attributes as pairs (i, j), if item i is replaced with item j in the current solution, then (i, j) is introduced to the tabu list. This forbids the reversal of this move in the next τ iterations. An alternative and more restrictive definition of attributes would be as items i. If item i is involved in a move it is introduced to the tabu list, preventing this item from being moved if it has been moved in the last τ iterations.

If a solution is forbidden by the tabu list, it may still be accepted if it satisfies an *aspiration criteria*. A commonly used aspiration criterion accepts a forbidden solution if it has better objective value than the best solution found. The algorithm terminates once the termination criteria are satisfied. Examples of termination criteria include the time elapsed, or the number of iterations without improving the current solution or iterations without improving the best solution found.

Longer term memory is also incorporated in some implementations of TS, and commonly relies on the same definition of attributes. *Long-term memory* acts to introduce more intensification by encouraging moves introducing attributes present in a number of good solutions found during the search, and more diversification by encouraging moves introducing attributes which have been appeared less frequently in solutions. The reader is referred to the book by Glover and Laguna (1998) for further details on the types of memory used in TS, and the components of TS more generally.

An outline of TS is given in Algorithm 2.3, where T is the tabu list, x_c is the current solution, x' is the best solution found, A is a set of solutions. Each iteration, Test$(N(x), T)$ is used to find the subset of neighbours in $N(x)$ that are not forbidden or satisfy the aspiration criteria. Select(A) is then used to find the best solution from the set A, and Update(T) introduces the attributes from the move just applied to T and manages the size.

Algorithm 2.3 Outline of TS

```
Initialise parameters
Generate initial solution x_c, and set x' = x_c, A = ∅, T = ∅ and initialise long-term
    memory structures
while termination criteria not satisfied
    A = Test(N(x_c), T)
    x_c = Select(A)
    T = Update(T) and update long-term memory structures
    if f(x_c) < f(x') then
        x' = x_c
return x'
```

An important concept in TS is *strategic oscillation*, which describes how the objective function value of the current solution oscillates in a strategic way due to the tabu list. This is also extended to include oscillating between feasibility and infeasibility. Infeasible solutions are permitted by relaxing some of the constraints from the problem, and a penalty on the level of infeasibility is introduced to the objective function to balance the level of infeasibility.

For example, in the KP we can define the level of infeasibility of a solution $X \subset V$ with

respect to the capacity W as

$$h(X) = \max\{0, \sum_{i \in X} w_i - W\}. \tag{2.2}$$

If β is the infeasibility penalty, then the penalised objective function is stated as follows.

$$f^p(X) = \sum_{i \in X} v_i - \beta h(X). \tag{2.3}$$

An optimal value for β may be difficult to find and may change at different stages in the search. For these reasons, β is often adaptively updated to balance the infeasibility with the greater potential for exploration.

Relaxing constraints, the search-space is enlarged but may also be more connected, and if the search-space is more connected then it may enable faster progression to high quality solutions using NS. Glover and Hao (2011) conjecture that allowing infeasible solution in heuristics enables transition between regions of the search-space which may otherwise be distant or even unreachable. This concept is frequently used in many other metaheuristics, particularly for COPs with complex constraints.

Iterated local search

It is clear that NS is limited by its acceptance criterion, and becoming stuck at locally optimal solution is a key issue. A natural approach to over come this may be to restart the NS from different initial solutions and this is the principle of multi-start NS (MNS). Although it is important to note that the number of locally optimal solutions is proportional to the size of the instance. As a result, MNS heuristics may return local optima with poor objective values, because the search will frequently become stuck after short search-trajectories from the initial solution. Another extension of NS is *iterated local search* (ILS), which applies a *kick* to locally optimal solutions, such as a number of random moves, before applying NS again. The kick tries to disturb a locally optimal solution enough to yield a different locally optimal solution when NS is applied, whilst preserving much of the information captured in the solution.

For a neghbourhood N and a solution x, the set of solutions that can reach x through improving N moves comprise the *basin of attraction* of x (with respect to N). Using this terminology, the kick attempts to find a solution outside the basin of attraction of the current locally optimal solution, but not too distant in the search-space. Similarly to NS, ILS is relatively simplistic but represents an important technique for many COPs. Especially considering implementations of ILS are amongst the best performing heuristics for the TSP (Johnson and McGeoch 1997) and other similar sequencing problems.

The kick must be chosen relative to the neighbourhoods used in the NS, and is usually a random move from a larger or different neighbourhood. For more complex problems, some authors incorporate information from the search history into the kick, similar to the long term memory in TS.

The final component in ILS is the *acceptance criteria*, which is used to decide whether a new solution is accepted as the current solution. The most basic of these is identical to NS, and only accepts solutions with better objective value than the current solution. A prominent

example of a more complex acceptance is using a cooling function, or *simulated annealing* acceptance criterion. A *cooling function* is usually defined such that solutions with worse objective value than the current solution are initially accepted with high probability, and this is progressively reduced until only solutions with better objective value are accepted. This idea is taken from simulated annealing, proposed by Kirkpatrick et al. (1983), which is a NS using a cooling function for acceptance criteria.

An outline of ILS is given in Algorithm 2.4, where x_c, x_s and x' are the current, intermediary and best solutions respectively. The function $NS(x)$ applies a NS to a solution x and returns the resulting locally optimal solution, Kick(x_c, *history*) is used to kick x_c and returns the resulting solution, and Accept(x_c, x_s) represents the acceptance criteria and returns x_s if this solution is accepted and x_c otherwise. A comprehensive survey of ILS and its applications can be found in Lourenço et al. (2010).

Algorithm 2.4 Outline of ILS

```
1: Initialise parameters
2: Generate initial solution x_c
3: x_c = NS(x_c), x' = x_c
4: while termination criteria not satisfied
5:     x_s = Kick(x_c, history)
6:     x_s = NS(x_s)
7:     x_c = Accept(x_c, x_s)
8:     if f(x_s) < f(x') then
9:         x' = x_s
10: return x'
```

Adaptive large neighbourhood search

A limitation of constructive heuristics is the inability to change decisions once a greater part of the solution is known. *Large neighbourhood search* using *ruin-and-recreate* neighbourhoods is proposed by Shaw (1998), and addresses this limitation by alternating between *destroy* and *repair* heuristics, which either destroy or repair the solution. The heuristics may include some randomisation to promote diversification in the search. For the TSP, a destroy heuristic would remove a number of cities from the tour, and a repair heuristic would re-insert these cities.

The overall process can be viewed as searching a large neighbourhood, where a neighbour is sampled by the application of the destroy and repair heuristics. The size of this neighbourhood is heavily dependent on the extent to which the solution is destroyed and then needs to be repaired. Differently to the neighbourhoods presented previously, the neighbourhood is not defined explicitly, but implicitly as the composition of all solutions reachable through the application of the pair of heuristics. Notice that the neighbourhood is also sampled rather than searched, where the complexity of the pair of constructive heuristics applied is the complexity of sampling. This complexity will usually be low, and the premise is that this method of sampling from a large neighbourhood is more effective than random sampling. If we consider the destroy heuristic as a kick and the repair heuristic as a NS, there are also

parallels with ILS.

The general framework is extended by Pisinger and Ropke (2007), who introduce the use of multiple destroy and repair heuristics, and an adaptive mechanism to choose the heuristics applied at each iteration based on the past performance. This is termed *adaptive large neighbourhood*. A weight is assigned to each heuristic which is then used to select the heuristics at each iteration. A roulette wheel type selection approach is then used based on the weights, to select either individual heuristics or pairs of heuristics to apply. The probability of a heuristic or equivalently a pair of heuristics being selected at any iteration is the respective weight divided by the total weights.

A key feature of the adaptive mechanism is the idea of *segments*. A segment is a fixed number of iterations, and during these iterations information about the performance of the heuristics is collect,. At the end of each segment, this information is used to update the weights. There are a variety of other considerations and design choices related to weighting the heuristics and updating the weights, and many of these are discussed by Pisinger and Ropke (2007).

An outline of ALNS is presented in Algorithm 2.5, where x_c, x_s and x' are the current, intermediary and best solutions, respectively, Ω is set of weights of the constructive heuristics, and Φ represents information about the performance of the destroy and repair heuristics in the current segment. The sets H^- and H^+ represent the sets of destroy and repair heuristics, respectively, and H_d and H_r are single destroy and repair heuristics, respectively. The function Choose(H^-, H^+, Ω) returns both a destroy and a repair heuristic from the sets H^- and H^+, the choice of heuristics is subject to the weights in Ω. The function Adjust(Ω, Φ) is used to update Ω considering Φ, and returns the updated weights. The function Accept(x_c, x_s) represents the acceptance criteria, and returns the x_s if it is accepted and otherwise x_s.

Algorithm 2.5 Outline of ALNS

```
Initialise parameters
Generate initial solution x_c and initialise Φ and Ω
while termination criteria not satisfied
    (H_d, H_r) = Choose(H^-, H^+, Ω)
    x_s = H_r(H_d(x_c))
    Update Φ
    x_c = Accept(x_c, x_s)
    if end of segment then
        Ω = Adjust(Ω, Φ)
        Reset Φ
    if f(x_s) < f(x') then
        x' = x_s
return x'
```

Hybrid genetic algorithms and path relinking

The group of metaheuristics called *evolutionary algorithms* (EAs) simulate evolutionary systems and natural adaptation to solve optimisation problems. These are some of the earliest

examples of metaheuristics, and are implemented as early as the 1970s in the work of Rechenberg (1973) and others. A good general introduction to EAs is given in the book by Eiben and Smith (2003).

A key example of EAs is *genetic algorithms* (GAs), which are popularised by the book of Holland (1975). GAs maintain a population of solutions, rather than a single solution, This operates as a type of adaptive memory, and may allow the heuristic a better appreciation of the search-space than is captured by a single solution. Solutions are traditionally encoded using a binary representation, however for many COPs this encoding abstracts the solution structure and feasibility, and therefore a natural encoding is used.

GAs evolve a population of solutions by iterative solution recombination, mutation and population management. A *generation* or iteration of a GA is usually defined by a parameter representing the number of offspring solutions generated in each generation. A generation consists of recombining solutions in the population to produce a set of offspring solutions, then probabilistically applying a random mutation to each offspring solution, and finally the population of the next generation is selected.

To produce an offspring solution, two *parent solutions* are selected from the population and an operator called the *crossover* is used to recombine the parent solutions. The crossover features some randomisation, and usually offspring solutions inherit varying proportions of attributes from the parent solutions. The premise behind the crossover is that combining parts of good parent solutions can make better offspring solutions.

The crossover departs noticeably from the ideas of defining neighbourhoods and constructive heuristic for a single solution. Now the neighbourhood is defined by a pair of solutions, and is composed of the possible offspring solutions resulting from a crossover. As is clear from the definition, these neighbourhood are usually large, and therefore the crossover typically returns a random neighbour. The mutation usually makes small random changes to parts of the solution, and this is similar to the kick used in ILS.

For the TSP, we can encode solutions as a permutation of the cities and then an example of a crossover is the *order based crossover*. Given two parent solutions p_1 and p_2, this crossover selects two cities $i, j \in V$, where $i < j$, at random and copies the sequence of cities between i and j from p_1 to the same position in the offspring solution, and the remaining positions are filled with the remaining cities in the order that they feature in p_2. An example of the order based crossover applied to two tours of a 6 city instance of the TSP can be seen in Figure 2.2. An example of a mutation for the TSP may be to change the position of each city in the permutation with a certain probability.

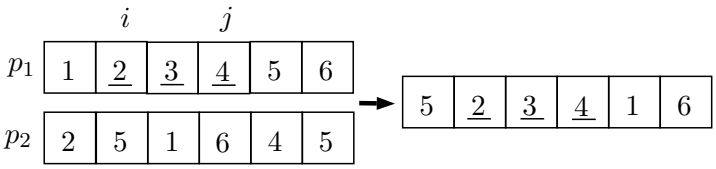

Figure 2.2: Two tours of an instance of the TSP, and an offspring resulting from an order-based crossover.

Following a mutation, the solutions that survive to the next generation are selected. Often these are selected from the union of the current population and the offspring solutions, but sometimes only the offspring solutions are considered. A GAs is termed *elitist*, if the next

generation is formed of the solutions with best objective value in the current population and offspring. Using such a myopic strategy can cause the population to *prematurely converge*, and this is much the same as a locally optimal solution being reached in NS. If this happens too early in the search then the performance of the GA will be poor as its exploration potential is restricted.

GAs appear to have potential to navigate the search-space of a COP to identify interesting and diverse regions, however, a key issue with the traditional operators is intensifying the search in these regions using the traditional operators. This causes the convergence of the population to solutions with objective values close to optimal to be slow. *Hybrid GAs* (HGAs) are proposed to address this, and hybridise GAs with NS to achieve greater intensification. NS is most frequently used to improve offspring solutions before considering selection of the next generation, and mutation is rarely used. The population is then composed of locally optimal solutions.

An outline of a general HGA is given in Algorithm 2.6, where P is the population and O the set of offspring solutions. The function Generate(P) returns a set of offspring solutions produced from P, Improve(O) applies a NS to each solution in O and returns the updated solutions, and Select(P, O) returns a set of solutions for the next generation from both P and O, or simply O.

Algorithm 2.6 Pseudo-code for a hybrid genetic algorithm

```
1: Initialise parameters
2: Generate initial population P, O = ∅
3: while stopping criteria not satisifed
4:     O = Generate(P)
5:     O = Improve(O)
6:     P = Select(P, O)
7: return Best solution found
```

NS improves solutions aggressively and although this can allow faster improvements in the best objective value found, it can encourage the population to converge prematurely. If diversity amongst solutions in the population falls too low then the population will stagnate. To balance the increased intensification potential, the diversity of population must be carefully managed. A dispersal rule is proposed by Prins (2004), where a threshold is defined on the minimum difference in objective value to other solutions in the population. Another approach is suggest by Repoussis et al. (2009), in which the objective value and diversity are considered lexicographically. Offspring solutions may replace any solution that has at least the same objective value, and less average difference in objective value to the whole population than the solution or compared to the median difference.

In contrast, Sörensen and Sevaux (2006) propose a threshold on a measure of diversity of a solution, or distance between solutions, compared to the current population. This is developed by Vidal et al. (2012), who consider both a measure of the diversity of a solution and its objective value when managing the population. This diversity measure is defined as the average distance to a fixed number of the closest solutions in the population, and if a solution has low average distance to the closest solutions in the population then it is

essentially penalised.

Scatter search (SS) and *path-relinking algorithms* (PRAs) are EAs, but in contrast to HGAs are not inspired by natural phenomenon. These frameworks have many similarities to HGAs, and are formalised by Glover et al. (2000). For example, NS is used to improve offspring solutions, parent solutions are recombined, and more complex population management strategies are preferred over random mutations. The main difference is the method of recombining solutions. SS does this by generating linear combinations of the solutions and is therefore usually less easily applicable to COPs, and PRAs achieve this by generating a search-trajectory connecting the solutions or *path-relinking*.

We will give a general description of path-relinking, but the interested reader is referred to Resende et al. (2010) for greater detail on the variety of design choices. To construct a search-trajectory between two solutions, or a *relinking-trajectory*, the solutions are assigned to be either the *initial solution* x_I or the *guiding solution* x_G. Then x_I is transformed into x_G by applying neighbourhood moves.

Figure 2.3: A relinking trajectory between two solutions.

For a chosen solution attribute, let $A(x_I, x_G)$ be the set of attributes present in x_I and not x_G. The cardinality of this set represents a distance measure $\text{dist}(x, x')$, which is an upper bound on the number of neighbourhood moves needed to transform x into x'. x_I is transformed into x_G by applying moves introducing the attributes in $A(x_I, x_G)$, and not removing attributes shared by x_I and x_G. Notice that the second requirement ensures the length of the relinking-trajectory is bounded by the distance between the solutions. For simplicity, a single solution is considered for x_G, but following the same principles a set of solutions can be used.

The relinking-trajectory is a sequence of solutions $(x_I, x_1, \cdots, x_k, x_G)$, where the distance to x_G is strictly decreasing and $k = O(\text{dist}(x_I, x_G))$. For further clarity, an illustration is given in Figure 2.3. If x_c is the current solution in the relinking-trajectory, there may be a number of neighbourhood moves which introduce at least one attribute from $A(x_c, x_G)$ and do not remove a shared attribute. Therefore, a policy must be set on which move to select, and this is frequently the move which results in the solution with best objective value. Notice that solutions in the relinking-trajectory can be infeasible, because the primary objective is to reduce $\text{dist}(x_c, x_G)$. Although the trajectory is guaranteed to reach the level of feasibility of x_G by the end.

2.3.4 Landscape analysis

The (search) landscape is a topology that arises if we consider the objective function as a height on solutions in the search-space. Understanding the landscapes induced by different problems and neighbourhoods is perceived to be critical in achieving a mathematical analysis of the performance of NS and different neighbourhoods, and heuristics more widely. Although

for many problems and neighbourhoods a useful description of the landscape may be difficult to find.

Definition 2.22. For $P = \{\min f(x) \colon x \in S\}$ and a neighbourhood N, the *landscape of P with respect to N*, or *landscape of P* when N is clear from the context, is the search-space where the solutions are weighted by their objective values.

An introduction to landscape analysis in different disciplines and a rigorous mathematical description of landscapes is given by Reidys and Stadler (2002). Analysis of the 2-Opt neighbourhood for the TSP with symmetric and asymmetric distance matrix is provided, which motivates its success and failure for each problem respectively. More recently, Merz (2012) give a detailed discussion of the motivation for landscapes to analyse metaheuristics, and present two case studies.

2.4 Implicit enumeration

In the previous sections, the focus is on techniques that sacrifice optimality for greater efficiency. In the following sections, paradigms for exact algorithms are described that are popular for solving NP-hard problems relatively efficiently. This section introduces techniques for implicitly enumerating the feasible solutions of a COP.

2.4.1 Branch and bound

Branch and bound (BB) is a general technique for efficiently and implicitly enumerating the solutions to a COP. The general idea and proofs are given by Markowitz and Manne (1957), and an early algorithmic approach is applied to COPs such as the TSP in the PhD Thesis of Eastman (1958). Although the popularisation of the method, and the first application using *linear relaxation* to obtain the lower bounds and branching on integrality of variables, is attributed to the paper by Land and Doig (1960). We will expand further on BB using *linear relaxation* in § 2.6, but in this section we consider the technique more generally.

BB is characterised by repeatedly splitting a problem into smaller subproblems. This is achieved by fixing elements of the solution, and therefore reducing the size of the feasible set of solutions in these subproblems. Tight upper and lower bounds are used to decide whether to continue solving a subproblem and the subproblems resulting from it, or if the associated set of feasible solutions can be excluded from further consideration.

Graph terminology is usually associated with BB, and the process is viewed as exploring a tree. The *search tree* is rooted at the complete problem, or the *root node*, and the remaining *nodes* are the subproblems. The process of splitting a problem is described as *branching*, and the outgoing arcs of a node connect it to the descendent nodes that represent the subproblems created by branching. The sub-tree rooted at a node is called a *branch*, and if further branching on a node is proven not to lead to a uniquely optimal solution, the branch rooted at the node does not need to be investigated and is said to be *fathomed*.

Given a problem $P = \{\min f(x) : x \in S\}$, a BB algorithm begins by finding upper and lower bounds, and using this information *branches*, or fixes some variables, at the root node to produce two or more nodes. Next, an untreated node is chosen and a lower bound is calculated for this node. There are then four possible scenarios, the lower bound of a subproblem is not

less than the best upper bound and the subproblem is fathomed; the subproblem is infeasible and it will not lead to a feasible solution; the solution associated to the lower bound is feasible for P and has no descendants, and if it has a lower objective value than the upper bound then the upper bound is updated; and otherwise the subproblem is branched. The process then repeats, creating subproblems of decreasing size until all the necessary nodes have been treated and the optimal solution is the best upper bound found.

From the basic description, it is clear that BB algorithm is finite, but its efficiency relies on the efficiency and tightness of the lower and upper bounds, and the branching scheme. Notice that in the worse-case a BB algorithm can have exponential time-complexity, because no solutions can be fathomed and the full enumeration is required. Fathoming branches early in the search is most critical because this causes the greatest reduction in the size of the search tree. It is also critical that the branching scheme partitions the search-space and produces a small number of tightly constrained nodes. It is also possible to apply heuristics to a node to attempt to tighten the upper bound, and therefore fathom more solution. These are the main concerns when applying BB to a COP, but much of the research effort is focused on developing tighter lower bounds and algorithms to calculate these more efficiently.

There are a variety of techniques for exploring the search tree, or equivalently policies for selecting the next node to treat. Two extremes are represented by *depth-first* and *breath-first*, which choose from the most or least recently generated nodes, respectively. The advantages of depth-first are low memory requirements and the opportunity to find upper bounds quickly, but the disadvantage is that we may make bad decisions early in the search and these must be exhausted before backtracking. In contrast, breath-first can require significantly more memory to store the large number of untreated nodes, although this enables us to keep the search tree as small as possible. Considering these ideas, often a more hybrid approach is employed that applies depth-first search until a branch is exhausted, and then uses breadth-first search to backtrack.

The following BB formulation for the TSP is originally proposed by Eastman (1958), who suggests using the lower bound of the assignment problem relaxation of the TSP. To describe this relaxation, we will define the linear assignment problem as follows.

Definition 2.23 (*linear assignment problem* (AP))**.** Given a set of V of n resources, a set U of n tasks, and an $n \times n$ matrix c_{ij}, where $c_{ij} \in \mathbb{Z}^+$ and represents the cost of assigning a task $i \in U$ to resource $j \in V$. Find the least cost one-to-one assignment of tasks to resources.

For the AP, the set of feasible solutions S is the $O(n!)$ possible one-to-one assignments. If we define $x_{ij} \in \{0, 1\}$ as an $n \times n$ permutation matrix that represents whether or not a task i is assigned to a resource j, then $f(x) = \sum_{i \in U, j \in V} c_{ij} x_{ij}$. For the *assignment problem relaxation* of the TSP (APR), we take x_{ij} to represent whether or not an edge (i, j) between cities i and j is used in the solution, $c_{ij} = d_{ij}$ for $i, j \in V$ such that $i \neq j$, and $c_{ii} = \infty$ for $i \in V$. Notice that there is roughly one feasible tour for every $O(n)$ feasible assignments.

The assignment problem, and therefore the assignment problem relaxation of the TSP, can be solved in $O(n^3)$ time using the Hungarian method proposed by Kuhn (1955). The solution to the relaxation is a union of simple cycles, and this is only feasible for the TSP if it is a single Hamiltonian cycle, rather than a set of distinct *subtours*. For a solution x to the APR, let $A_s = \{(i_1, i_2), \cdots, (i_k, i_1)\}$ be the set of arcs associated to the subtour of minimum cardinality. To prevent this subtour in the APR, then one of the k arcs must be

excluded from the solution. The branching scheme uses this idea to create k subproblems, each restricting a different arc in A_s from being used. Notice that defining A_s for the subtour of minimum cardinality controls the number of subproblems created in branching.

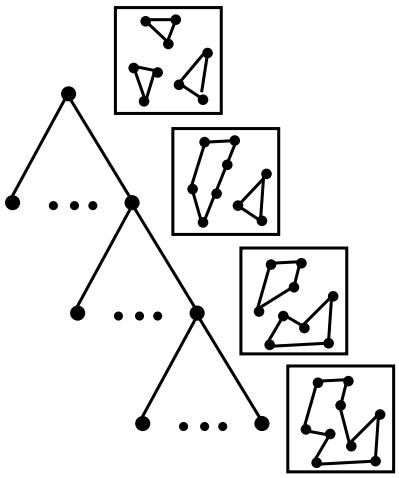

Figure 2.4: An instance of the TSP, and some of the BB search tree.

It is important to note that this branching scheme does not partition the set of feasible solutions, and therefore the k subproblems and their descendent subproblems can share solutions. This is not the most efficient approach to branching, and a method to address this is described in Bellmore and Malone (1971). The authors show that the branching can be strengthened if for each subproblem, rather than simply excluding the a^{th} arc in A_s, we also require the $a-1$ previous arcs are included.

Another technique to reduce the size of the search tree is domination, which is defined as follows.

Definition 2.24. A node x is *dominated* by a node y, if y is guaranteed to produce a solution that is at least as good as the best solution that can be produced by x.

In the BB algorithm for the TSP described above, we define E_k and I_k as the sets of excluded and included arcs respectively in node k. Then a node x dominates a node y if

$$E_x \cup I_x \subset E_y \cup I_y, \tag{2.4}$$

$$\sum_{(i,j)\in I_x} c_{ij} \leq \sum_{(i,j)\in I_y} c_{ij}. \tag{2.5}$$

Using dominance in a BB algorithm may enable less nodes to be treated because some branches will not need to be investigated if the node is dominated.

2.4.2 Dynamic programming

Dynamic programming (DP) is a general technique for solving optimisation problems. The approach is formalised by Bellman (1957), for application to *discrete-time sequential decision problems*, or problems in which a sequence of decisions must be taken at discrete time points or stages. DP decomposes the problem into simpler subproblems similarly to BB. To apply DP to a problem, it must decomposable into a sequence of overlapping subproblems. The definition of these subproblems allows an optimal solution to the problem to be constructed

sequentially through inductive principles. In contrast to constructive heuristics, all possible extensions of the partial solutions are considered implicitly in the induction.

A problem is often modelled in stages, where each *stage* is a subproblem, although the stages are not always explicitly defined. At each stage, there are a set of possible *states*, and each is described by an assignment of values to a set of *state variables*. A requirement of DP is that the optimal solution to the subproblem at each stage must be dependent on the states at that stage, and independent of the solutions to the subproblems at previous stages. Alternatively, the state variables must capture all the information required to evaluate the feasibility and objective values of the solution represented by the state or any extension of the state. This is the inductive principle of DP, and is described as the *principle of optimality* by Bellman.

The last feature of a DP approach is the Bellman equation, which is a recurrence relation that defines the transitions between states at different stages. This enables us to find the solution one stage at a time, by solving the subproblem at each stage until the all stages are considered. The initial or final states are known, and either a forward or backward induction is used to enumerate the states for each stage, respectively. A backward induction begins at the possible states in the final stage and works backwards to the initial state, whereas forward induction begins at the initial state and works forwards to the states at the final stage.

The definition of states is often one of the main difficulties when developing an efficient DP formulation, and often requires artful exploitation of special properties of the problem. This because number of state variables and the dimension of these variables are critical to the number of possible states, and therefore the size of the resulting DP and time-complexity. The number of possible states can often result in a computationally intractable formulation, unless the number of states generated by a DP algorithm is carefully controlled.

Notice that the states and state-transitions form an acyclic digraph, usually called the *state-space graph*. If the arcs in this graph are weighted by the difference in objective value of the states connected by the arc, then solving any DP formulation can be viewed as finding the shortest path in the state-space graph.

The following DP formulation of the TSP is originally described independently by Held and Karp (1962) and Bellman (1962). We introduce the city $n+1$ to V as a copy of city 1, and find the shortest path between 1 and $n+1$. States are defined as (L, i), and represent an elementary path starting at city 1 and visiting the set of cities $L \subseteq V$ such that i is visited last. The state space graph $G_1(\Sigma, \Gamma)$ is defined as follows.

$$\Sigma_1 = \{(L, i) : L \backslash \{i\} \subseteq V, \forall\, i \in L\}, \tag{2.6}$$

$$\Gamma_1 = \{((L \backslash \{i\}, j), (L, i)) : j \in L \backslash \{i\}, \forall (L, i) \in \Sigma_1\}. \tag{2.7}$$

The recursion can then be stated as follows.

$$f(L, i) = \min_{j \in L \backslash i} \{f(L \backslash \{i\}, j) + d_{ji}\}, \forall (L, i) \in \Sigma_1. \tag{2.8}$$

This is initialised by $f(\emptyset, 1) = 0$, and the cost of the optimal solution can be found as $f(V, n+1)$. There are $O(n2^n)$ states and each takes $O(n)$ time to compute, therefore the DP has complexity $O(2^n n^2)$. This is clearly exponential in n, but better than the $O(n!)$ time taken to explicitly enumerate all tours. Notice that the stages in this formulation can be

defined as the cardinality of the path.

For the KP, states can be defined as (w, i), representing a subset of items with indices less than or equal to i and combined weight of w. The state space graph $G_2(\Sigma_2, \Gamma_2)$ is defined by

$$\Sigma_2 = \{(w, i) : w \leq W, 0 \leq i \leq n\}, \tag{2.9}$$

$$\Gamma_2 = \{\big((w - w_i, i - 1), (w, i)\big) : \ \forall (w, i) \in \Sigma_2\} \cup \{\big((w, i - 1), (w, i)\big) : \ \forall (w, i) \in \Sigma_2\}, \tag{2.10}$$

where the recursion is defined as

$$f(w, i) = \min\{f(w - w_i, i - 1) - v_i, f(w, i - 1)\}, \forall (w, i) \in \Sigma_2. \tag{2.11}$$

This is initialised by $f(w, 0) = 0$, $w \leq W$, and the cost of the optimal solution can be found as $f(w, n)$. There are $O(nW)$ states and each takes $O(1)$ time to compute, therefore the complexity of the DP is $O(nW)$. This is polynomial in n but pseudo-polynomial in W. The stages for this formulation can be defined as number of items considered.

Dominance conditions can also be defined for DPs, and a state s dominates another state s' if it is guaranteed to extend to states that are at least as feasible and optimal as the best state that extends from s'. For example, in the KP a state $s = (w, i)$ dominates a state $s' = (w', i)$ if the following conditions are true.

$$w \leq w', \tag{2.12}$$

$$f(w, i) \geq f(w', i). \tag{2.13}$$

This is criteria are verified by observing that any feasible extension of s' is feasible for s and results in at least equal objective value. The concepts of fathoming also translate to DP, where a state can be fathomed if a lower bound on the best solution it leads to is greater than the best upper bound found.

2.5 Linear programming

Linear programming is a technique for modelling and solving optimisation problems with a linear objective function and linear constraints, and therefore continuous variables. The development of linear programming is attributed to Leonid Kantorovich in 1939, but widespread knowledge of the approach was initiated in 1947 by the invention of the *simplex method* to solve linear programs by George B. Dantzig. The theory of duality for linear programming was conjectured by John von Neumann soon after, and the first proof is independently published by Gale et al. (1951). This section describes the general linear programming problem and introduces duality and some of the implications of the duality theories, but for an exposition of the subject see the book by Dantzig (1998) originally printed in 1963.

A linear programming problem is defined by an objective function, a set of inequalities or constraints, and a set of variables. An assignment of values to the variables represents an individual solution. The objective function and constraints must be linear in the variables, and a solution is feasible if the variables satisfy the set of constraints. This is formally defined as follows.

Definition 2.25 (*linear programming problem* (LP))**.** Given an $m \times n$ integer matrix A with rows $\mathbf{a}_i$ representing the i^{th} of the m constraints, an integer column vector $\mathbf{b}$ with m elements b_i representing the right hand side of the i^{th} constraint, and a vector $\mathbf{c}$ with n elements c_i representing the unit cost of the i^{th} variable. Let M be the set of rows indices of A that represent equality constraints, and M' be the set of remaining row indices that represent the inequality constraints. A solution is represented by a real vector $\mathbf{x}$ with n elements x_j representing the variables. Let N be the set of indices representing non-negative variables, and N' be the set of remaining indices representing free variables. The *general form of a linear program* can then be stated as follows.

$$\min \mathbf{c}^T\mathbf{x} \tag{2.14}$$

$$\text{s.t. } \mathbf{a}_i\mathbf{x} = b_i, \qquad \forall i \in M, \tag{2.15}$$

$$\mathbf{a}_i\mathbf{x} \geq b_i, \qquad \forall i \in M', \tag{2.16}$$

$$x_j \geq 0, \qquad \forall j \in N. \tag{2.17}$$

The definition of the LP above represents a continuous optimisation problem, and does not seem to have the finite set of solutions required in a COP. However, this is not true, and in fact the simplex algorithm can be viewed as solving a combinatorial version of the LP. A *convex polyhedron* is the intersection of finitely many linear inequalities. A polyhedron can also be defined for a set by the convex hull, where the *convex hull* of a set is the smallest subset such that any element of the set can be found as a convex combination of elements in the subset. Each element of the convex hull is found at an intersection of the linear inequalities describing the polyhedron.

In the LP, we wish to minimise some linear objective function of the variables, where the variables are bounded by the polyhedron Q_c defined by constraints (2.15)-(2.17). It can be proven that any optimal solution to the LP is an element of the convex hull of the set of feasible solutions, or equivalently an *extreme point* of Q_c. The number of extreme points of Q_c is finite, although it can be exponential in the instance size. The LP can then be solved as a COP, where we must find the optimal extreme point of Q_c. For an expansive survey and a rigorous treatment of polyhedral combinatorics, see the book by Schrijver (2003).

Considering the discussion above, the simplex algorithm can be described as a NS that uses an exact neighbourhood to search the extreme points of Q_c. This is searched in a very efficient and intuitive way, although the complexity can be exponential in the worse-case, as shown by Klee and Minty (2002). The general LP is proven to be in $\mathcal{P}$ by the ellipsoid algorithm of Khachiyan (1977), which is adapted from work for non-linear problems by Shor (1970). In practise, however, the ellipsoid method is not efficient, and the simplex algorithm and the polynomial-time interior point method of Karmarkar (1984) are the two competitors.

Many interesting results concerning the LP relate to the theory of duality. We will call the LP in general form the *primal* problem, and then the corresponding *dual* problem is defined as follows.

Definition 2.26. For the LP, let $\mathbf{y}$ be a dual solution with m elements y_i, and $\mathbf{a}'_j$ represent

the columns of matrix A. The general form of the dual LP is then stated as follows.

$$\begin{aligned}
&\min \mathbf{c}^T\mathbf{x} && && \min\ -\mathbf{b}^T\mathbf{y} && (2.18)\\
&\text{s.t. } \mathbf{a}_i\mathbf{x} = b_i, && \forall i \in M, && && (2.19)\\
&\quad \mathbf{a}_i\mathbf{x} \geq b_i, && \forall\, i \in M', && y_i \geq 0 && (2.20)\\
&\quad x_j \geq 0, && \forall\, j \in N, && \mathbf{a}'_j\mathbf{y} \leq c_j && (2.21)\\
& && \forall\, j \in N' && \mathbf{a}'_j\mathbf{y} = c_j. && (2.22)
\end{aligned}$$

The dual of the dual is the primal and this can be easily verified. The *weak duality theorem* establishes that $\mathbf{b}^T\mathbf{y} \leq \mathbf{c}^T\mathbf{x}$, where $\mathbf{y}$ is any dual-feasible solution and $\mathbf{x}$ is any primal-feasible solution. Furthermore, the *strong duality theorem* establishes that $\mathbf{b}^T\mathbf{y}^* = \mathbf{c}^T\mathbf{x}^*$, where $\mathbf{y}^*$ is any dual-optimal solution and $\mathbf{x}^*$ is any primal-optimal solution.

2.6 Integer programming

If all or some of the variables in the LP are required to only take integer values, then the problem becomes an integer linear programming or mixed integer linear programming problem, respectively.

Definition 2.27 (*mixed integer linear programming problem* (MIP))**.** Adapting the LP, a feasible solution $\mathbf{x}$ now has $n' \leq n$ integer elements with indices $I \subseteq \{N \cup N'\}$. The MIP in *general form* is then stated as follows.

$$\begin{aligned}
&z(\text{MIP}) = \min \mathbf{c}^T\mathbf{x} && && && (2.23)\\
&\quad \text{s.t. } \mathbf{a}_j\mathbf{x} = b_j, && \forall\, j \in M, && && (2.24)\\
&\quad \mathbf{a}_j\mathbf{x} \geq b_j, && \forall\, j \in M', && && (2.25)\\
&\quad x_i \geq 0, && \forall\, i \in N, && && (2.26)\\
&\quad x_i \in \mathbb{Z}, && \forall\, i \in I. && && (2.27)
\end{aligned}$$

Other than the integrality constraints, all the constraints and the objective function must be linear. There is significant interest in MIPs because most COPs can be transformed into an MIP, although this also reveals the complexity of the MIP in general, which is clearly NP-hard. It is also possible to model many types of non-linearities using integer variables that are not possible in the LP, but if the objective function or constraints are non-linear then the problem is a mixed integer non-linear programming problem.

For the MIP, the weak duality theorem still holds but the strong duality theorem does not, and a non-negative difference, or duality gap, exists between the objective values of the primal and dual optimal solutions. The *linear relaxation* of the MIP is obtained by relaxing constraints (2.27) to be upper and lower bounds, and provides a lower bound on the optimal objective value. The solution produced will usually have fractional values, and otherwise it is optimal for the original problem. The linear relaxation of different MIP formulations of a COP have different strengths, or tightness of lower bound, and this often creates a trade-off with computational time required to solve the linear relaxation. Additional constraints may be found that can exclude fractional solutions but not integer solutions, and this idea is

described in the following section.

Interestingly, for a number of COP, including the AP, the linear relaxation of the natural MIP formulation yields integer solutions, and this property is captured in the theory of unimodularity. For a given LP, if the constraint matrix A is total unimodular and the right hand side vector b is integer, then the solution is integer. For further details, we refer the reader to the relevant section in the book by Schrijver (2003). Unfortunately, these properties are only satisfied by a small number of COPs, and more complex techniques are required to solve the majority.

For example, the MIP formulations of the KP requires $O(n)$ binary variables x_i, equal to one if item $i \in V$ is included in the solution. The formulation can then be stated as follows.

$$z(\text{KP}) = \max \sum_{i \in V} v_i x_i, \tag{2.28}$$

$$\text{s.t.} \sum_{i \in V} w_i x_i \leq W, \tag{2.29}$$

$$x_i \in \{0, 1\}, \qquad \forall\, i \in V. \tag{2.30}$$

This formulation can be viewed as one of the most basic classes of the MIP, because there is a single constraint and n binary variables. An interesting property of the problem is that an optimal solution to the linear relaxation can be obtained using a constructive algorithm. Each item is included in the solution in order of non-increasing v_i/w_i until the capacity is exceeded, and then the proportion of the last item that exceeds the capacity can be removed.

An MIP formulation of the TSP requires $O(n^2)$ binary variables x_{ij}, equal to one if $(i, j) \in A$ is used in the solution. The formulation can then be stated as follows.

$$\min z = \sum_{i,j \in V : i \neq j} d_{ij} x_{ij}, \tag{2.31}$$

$$\text{s.t.} \sum_{j \in V} x_{ij} = 1, \qquad \forall\, i \in V, \tag{2.32}$$

$$\sum_{j \in V :} x_{ji} = 1, \qquad \forall\, i \in V, \tag{2.33}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 2 \qquad S \subset V, \tag{2.34}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall\, i, j \in V. \tag{2.35}$$

Constraints (2.32) and (2.33) are load balancing and ensure that each city is entered and exited once. The remaining constraints are first proposed by Dantzig et al. (1954), and eliminate any subtours. This is achieved by ensuring that any subset of vertices has one less connected arc used in the solution than its cardinality and therefore cannot form a subtour. Notice that there are an exponential number of subsets $S \subset V$ and therefore an exponential number of subtour elimination constraints. Miller et al. (1960) propose a polynomial number of constraints that eliminate subtours, but these have been shown to be weaker in the linear relaxation and are rarely used.

One approach to solving the MIP is using BB, where the lower bound is provided by the linear relaxation of the IP formulation, and branching is done on the integrality of a variable with fractional value. This is the approach suggested by Land and Doig (1960), as mentioned

previously, and as we will see is a critical component in many algorithms for solving NP-hard COPs. An illustration of the search tree and some solutions for such a BB algorithm applied to an instance of the TSP is given in Figure 2.5, where arcs with a fractional value in the solution are shown by broken lines.

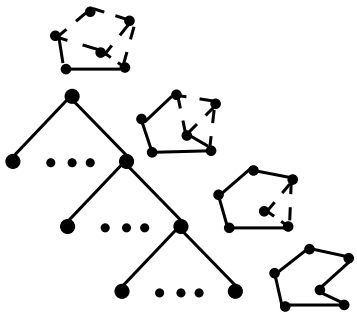

Figure 2.5: An instance of the TSP, and a partial BB search tree.

2.6.1 Cutting planes

A constraint for a linear relaxation of an MIP formulation is a *valid inequality* if it is satisfied by all feasible integer solutions but not all feasible fractional solutions. A valid inequality is in turn a *cutting-plane*, or *cut*, if it is violated by the optimal solution to the linear relaxation. Let Q_I be the polyhedron defined by the convex hull of the feasible integer solutions of an MIP formulation, a valid inequality is a *facet* (of the formulation) if it features in the most compact representation the polyhedron Q_I as a set of linear inequalities.

Clearly Q_I is contained in the polyhedron Q_c, and from polyhedral theory, we know Q_I can be described using finitely many linear inequalities. The most compact definition of these inequalities is a family of facets that give a complete representation of Q_I. Unfortunately, for most problems only some of facets for Q_I are known, and the number of facets is often exponential for NP-hard problems.

At each iteration, a cutting-planes algorithm searches for at least one cut to be added and then solves the linear relaxation of the resulting formulation. The search for a cut can be characterised by an optimisation problem that is described as the separation problem. An important but unfortunate result concerning the separation problem is that it is polynomial time equivalent to the original problem. Although for NP-hard problems heuristics are often used to find cuts in acceptable time. For further details on the theory and mathematics, we refer the reader to the relevant sections in the book by Schrijver (2003).

A general valid inequality for the MIP is proposed by Gomory (1958), and these are termed Gomory cuts. Gomory proposed repeatedly solving the linear relaxation of an IP, each time identifying the Gomory cuts violated by the optimal solution and introducing these to the model. Eventually, this will result in the optimal solution to the original IP, and this technique is known as a *cutting- plane algorithm*. The seminal paper by Dantzig et al. (1954) that solves a 48 city instance of the TSP for the first time, can be viewed as a cutting-planes algorithm that separates the subtour elimination constraints.

Crowder and Padberg (1980) suggest combining BB and cutting-plane algorithms for the TSP, and this is usually described as *branch and cut* (BC). At each subproblem in the search tree of BB, we consider introducing violated cuts, and together branching, bounding and

cutting reduces the total number of subproblems which must be solved. BC algorithms are demonstrated to perform well on a significant number of NP-hard COPs, and the algorithm of Applegate et al. (2006) can solve instances of the TSP with thousands of cities.

2.6.2 Dantzig-Wolfe reformulation and column generation

Dantzig-Wolfe reformulation is a general technique for reformulating and solving certain LPs, and it has been found to be particularly important for achieving tight linear relaxations for complex integer programming problems. A general set-covering (SC) MIP formulation that can be stated as follows.

$$z(\text{SC}) = \min c(\mathbf{x}), \tag{2.36}$$
$$\text{s.t.,} A\mathbf{x} \geq \mathbf{b}, \tag{2.37}$$
$$D\mathbf{x} \geq \mathbf{e}, \tag{2.38}$$
$$\mathbf{x} \in \mathbb{Z}^n. \tag{2.39}$$

Let us assume that constraints (2.37) are complicating or linking constraints, such that relaxing these constraints in SC makes the formulation computationally easier to solve.

Dantzig-Wolfe reformulation relies on the observation that the set $X = \{\mathbf{x} \in \mathbb{Z}^n \colon D\mathbf{x} \geq \mathbf{e}\}$ is a finite set of vectors and can be represented by its convex hull $\text{conv}(X)$. If we assume X is bounded, each $\mathbf{x} \in X$ can be expressed as a convex combination of the extreme points of $\text{conv}(X)$, denoted by $\mathbf{x}_r$ for $r \in \Omega$. This is expressed as follows.

$$\mathbf{x} = \sum_{r \in \Omega} \mathbf{x}_r \lambda_r, \tag{2.40}$$
$$\sum_{r \in \Omega} \lambda_r = 1, \tag{2.41}$$
$$\lambda_r \in \mathbb{R}^+, \qquad \forall\, r \in \Omega. \tag{2.42}$$

Further details about the situation when X is unbounded are given by Barnhart et al. (1998).

Let $c_r = c(\mathbf{x}_r)$ and $\mathbf{a}_r = A\mathbf{x}_r$ for $r \in \Omega$. If $\mathbf{x}$ is substituted into SC, then we arrive at the following *extended formulation*.

$$z(\text{EF}) = \min \sum_{r \in \Omega} c_r \lambda_r, \tag{2.43}$$
$$\text{s.t.,} \sum_{r \in \Omega} \mathbf{a}_r \lambda_r \geq \mathbf{b}, \tag{2.44}$$
$$\sum_{r \in \Omega} \mathbf{x}_r \lambda_r = \mathbf{x}, \tag{2.45}$$
$$\sum_{r \in \Omega} \lambda_r = 1, \tag{2.46}$$
$$\boldsymbol{\lambda} \geq \mathbf{0}, \tag{2.47}$$
$$\mathbf{x} \in \mathbb{Z}^n. \tag{2.48}$$

If the integrality conditions on $\mathbf{x}$ are relaxed, these variables and the associated linking constraints (2.45) become redundant. Then the reformulation is complete, and the resulting

formulation is described as the *master problem*. Notice that the function c is not required to be linear to solve the master problem as an MIP.

Let γ_0 and $\boldsymbol{\gamma}$ be a free dual-variable and non-negative dual-variables associated to constraints (2.46) and (2.44), respectively. Observe that the dual of the master problem can then be stated as follows.

$$z(DM) = \max \mathbf{b}^T\boldsymbol{\gamma} + \gamma_0, \tag{2.49}$$

$$\text{s.t.} A^T\boldsymbol{\gamma} + \gamma_0 \leq \mathbf{c}, \tag{2.50}$$

$$\boldsymbol{\gamma} \geq \mathbf{0}. \tag{2.51}$$

The reduced cost of a variable λ_r for $r \in \Omega$ is therefore defined by $\tilde{c}_r = c_r - \boldsymbol{\gamma}^T\mathbf{a}_r - \gamma_0$.

Both the master problem and its dual are intractable for most COPs, because the size of the set Ω, or the number of variables or constraints, respectively, is exponential. In the simplex method, the reduced cost of all non-basic variables is calculated and usually the variable with least reduced cost is chosen to enter the basis. Clearly if the number of variables is exponential then this approach is intractable, and this leads to the definition of a *restricted master problem* (RMP) that uses only a subset of the variables $\Omega' \subset \Omega$.

The master problem can then be solved using a (delayed) column generation (CG) algorithm, which iterates between solving the RMP and generating columns with negative reduced cost to be added to the current set of columns. If no such a column exists, then the complimentary slackness conditions of LPs ensure that the solution is optimal. Finding a column to add to the RMP, or at least proving that one does not exist, requires minimising the reduced costs over all $x \in X$, and this is described as the *pricing problem* (PP). Notice that the CG algorithm can be cast as a cutting-planes algorithm for the dual, and the PP is then the separation problem. It is also interesting to note that in the Dantzig-Wolfe reformulation of an integer programming problem such as SC, the master problem is an LP formulation and the PP is an MIP formulation.

Geoffrion (1974) highlights the equivalence between the lower bounds provided by the master problem and the Lagrangian dual of SC with optimal multipliers, where constraints (2.37) are dualised. This also leads to the following lower bound $z(\text{RMP}) + z(\text{PP})$, where $z(P)$ is the optimal objective value of problem P. Furthermore, Geoffrion (1974) shows that if the PP possess the integrality property then the lower bounds from the master problem and the linear relaxation of SC will be equal.

If the PP decomposed into independent subproblems, or equivalently X decomposes into subsets X_k for $k \in K$, then each of these subproblems can be solved independently and any columns with negative reduced cost can be added to the RMP. If these subproblems are identical, for example, if X has block-diagonal structure, then only one pricing problem must be solved and constraint (2.46) is aggregated across the subproblems.

The optimal solution of the RMP may be integer, but this is often not the case. To find an integer solution generally, branching or cutting must be applied. This is often difficult using the λ-variables, which lack much of the context of the original problem. Although for many problems it is possible to define a relationship between the x- and λ-variables, for example, using a binary indicator b_i^r for each $r \in \Omega$ and $i = 0, 1, \cdots, n$, equal to one if λ_r uses x_i, and this is used to translate branching constraints and cuts from the original problem to the

RMP.

Bibliography

Alba, E. (2005). *Parallel metaheuristics: a new class of algorithms*. Wiley, Chichester, UK.

Alba, E., Luque, G., and Nesmachnow, S. (2013). Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48.

Altner, D. S., Ahuja, R. K., Ergun, O., and Orlin, J. B. (2014). Very large-scale neighborhood search. In Burke, E. K. and Kendall, G., editors, *Search Methodologies*, pages 339–367. Springer, New York, USA.

Applegate, D. L., Bixby, R. E., Chvatal, V., and Cook, W. J. (2006). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, NJ, USA.

Arora, S. and Barak, B. (2009). *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, USA.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA.

Bellman, R. (1962). Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63.

Bellmore, M. and Malone, J. C. (1971). Pathology of traveling-salesman subtour-elimination algorithms. *Operations Research*, 19(2):278–307.

Blum, C. (2012). Hybrid metaheuristics in combinatorial optimization: A tutorial. In Dediu, A.-H., Martn-Vide, C., and Truthe, B., editors, *Theory and Practice of Natural Computing*, pages 1–10. Springer, Berlin, DE.

Blum, C., Puchinger, J., Raidl, G. R., and Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151.

Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308.

Cobham, A. (1965). The intrinsic computational difficulty of functions. In Bar-Hillel, Y., editor, *Logic, Methodology and Philosophy of Science, Proceedings of the 1964 International Conference*, pages 24–30. North Holland, Amsterdam, Holland.

Congram, R. K., Potts, C. N., and van de Velde, S. L. (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, New York, USA.

Cook, W., Cunningham, W., Pulleyblank, W., and Schrijver, A. (2011). *Combinatorial Optimization*. Wiley, Chichester, UK.

Crainic, T. G. and Toulouse, M. (2010). Parallel meta-heuristics. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 497–541. Springer, New York, USA.

Crowder, H. and Padberg, M. W. (1980). Solving large-scale symmetric travelling salesman problems to optimality. *Management Science*, 26(5):495–509.

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

Dantzig, G. B. (1998). *Linear programming and extensions*. Princeton university press, Princeton, NJ, USA.

Eastman, W. L. (1958). Linear programming with pattern constraints. PhD Thesis, Harvard University.

Eiben, A. E. and Smith, J. E. (2003). *Introduction to evolutionary computing*. Springer, Berlin, DE.

Gale, D., Kuhn, H. W., and Tucker, A. W. (1951). Linear programming and the theory of games. In Koopmans, T. C., editor, *Activity Analysis of Production and Allocation*, pages 24–30. Wiley, New York.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, USA.

Gendreau, M. and Potvin, J.-Y. (2005). Metaheuristics in combinatorial optimization. *Annals of Operations Research*, 140(1):189–213.

Gendreau, M. and Potvin, J.-Y. (2010). *Handbook of metaheuristics*. Springer, New York, USA.

Geoffrion, A. M. (1974). *Lagrangean relaxation for integer programming*. Springer, US.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–546.

Glover, F. and Hao, J.-K. (2011). The case for strategic oscillation. *Annals of Operations Research*, 183(1):163–173.

Glover, F. and Laguna, M. (1998). *Tabu Search*. Springer, New York, USA.

Glover, F., Laguna, M., and Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684.

Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278.

Hansen, P. (1986). The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. In *Proceedings of the Congress on Numerical Methods in Combinatorial Optimization*. Capri, Italy.

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

Hentenryck, P. V. and Bent, R. (2009). *Online Stochastic Combinatorial Optimization*. The MIT Press, Cambridge, MA.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control and artificial intelligence*. Michigan University Press, Ann Arbor, MI, USA.

Johnson, D. S. (2002). A theoreticians guide to the experimental analysis of algorithms. In Goldwasser, M. H., Johnson, D. S., and McGeoch, C. C., editors, *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges*, pages 215–250. AMS, Providence, RI, USA.

Johnson, D. S. and McGeoch, L. (1997). The traveling salesman problem: A case study in local optimisation. In Aarts, E. and Lenstra, J., editors, *Local Search in Combinatorial Optimization*, pages 215–310. Wiley, Chichester, UK.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 302–311. ACM, New York, NY, USA.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, USA.

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

Khachiyan, L. (1977). Convergence rate of the game processes for solving matrix games. *USSR Computational Mathematics and Mathematical Physics*, 17(6):78–88.

Kirkpatrick, S., Gelatt, Jr, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *ORSA Journal on Computing*, 220(4598):671–680.

Klee, V. and Minty, G. J. (2002). How good is the simplex algorithm? In Shisha, O., editor, *Inequalities-III*, pages 159–175. Academic Press, New York, USA.

Kleywegt, A. J., Shapiro, A., and Homem-de Mello, T. (2002). The Sample Average Approximation Method for Stochastic Discrete Optimization. *SIAM Journal on Optimization*, 12(2):479–502.

Korte, B. and Lovász, L. (1981). Mathematical structures underlying greedy algorithms. In *Fundamentals of Computation Theory*, volume 117 of *Lecture Notes in Computer Science*, pages 205–209. Springer Berlin Heidelberg.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics*, 2(1-2):83–97.

Land, A. H. and Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, pages 497–520.

Lin, S. (1965). Computer solutions of the travelling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269.

Lourenço, H. R., Martin, O. C., and St utzle, T. (2010). Iterated local search: Framework and applications. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 363–397. Springer, New York, USA.

Maniezzo, V., Stützle, T., and Voß, S. (2009). *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*. Springer, New York, USA.

Markowitz, H. M. and Manne, A. S. (1957). On the solution of discrete programming problems. *Econometrica*, pages 84–110.

Merz, P. (2012). Memetic algorithms and fitness landscapes in combinatorial optimization. In *Handbook of Memetic Algorithms*, pages 95–119. Springer, Berlin, DE.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):393–410.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*. Wiley, New York, USA.

Osman, I. H. and Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623.

Papadimitriou, C. (1994). *Computational Complexity*. Addison-Wesley, Reading, MA, USA.

Papadimitriou, C. H. and Steiglitz, K. (1977). On the complexity of local search for the traveling salesman problem. *SIAM Journal on Computing*, 6(1):76–83.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403 – 2435.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(0):1985 – 2002.

Raidl, G. R. (2006). A unified view on hybrid metaheuristics. In Almeida, F., Blesa Aguilera, M. J., Blum, C., Moreno Vega, J. M., Perez Perez, M., Roli, A., and Sampels, M., editors, *Hybrid Metaheuristics*, pages 1–12. Springer, Berlin, DE.

Raidl, G. R., Puchinger, J., and Blum, C. (2010). Metaheuristic hybrids. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 469–496. Springer, New York, USA.

Rechenberg, I. (1973). *Evolutionsstrategie Optimierung technischer systeme nach prinzipien der biologischen evolution*. Friedrich Frommann Verlag, Stuttgart, DE.

Reidys, C. M. and Stadler, P. F. (2002). Combinatorial landscapes. *SIAM review*, 44(1):3–54.

Repoussis, P. P., Tarantilis, C. D., and Ioannou, G. (2009). An evolutionary algorithm for the open vehicle routing problem with time windows. In Pereira, F. B. and Tavares, J., editors, *Bio-inspired Algorithms for Vehicle Routing*, pages 55–76. Springer, Berlin.

Resende, M., Ribeiro, C., Glover, F., and Mart, R. (2010). Scatter search and path-relinking: Fundamentals, advances, and applications. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 87–107. Springer, New York, USA.

Schrijver, A. (2003). *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, Berlin, DE.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In Maher, M. and Puget, J.-F., editors, *Principles and Practice of Constraint Programming CP98*, pages 417–431. Springer.

Shor, N. Z. (1970). Convergence rate of the gradient descent method with dilatation of the space. *Cybernetics*, 6(2):102–108.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66.

Sörensen, K. and Sevaux, M. (2006). Ma— pm: memetic algorithms with population management. *Computers & Operations Research*, 33(5):1214–1225.

Steiglitz, K. and Papadimitriou, C. H. (1982). *Combinatorial optimization: Algorithms and complexity*. Printice-Hall, New Jersey, USA.

Talbi, E.-G. (2002). A taxonomy of hybrid metaheuristics. *Journal of heuristics*, 8(5):541–564.

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*, volume 74. Wiley, Chichester, UK.

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.

Chapter 3

The vehicle routing problem with release and due dates

In this chapter, the vehicle routing problem with release and due dates (VRPRDD) is formally introduced. Results on the complexity of the problem are discussed, and the set of proposed benchmark instances are described.

3.1 Problem definition

The VRPRDD is defined on a complete graph $G = (V, A)$, where the vertex set is $V = \{0, 1, \ldots, n\}$ and arc set is $A = \{(i, j)\colon i, j \in V, i \neq j\}$. Vertex 0 corresponds to the depot, while $V' = V \backslash \{0\}$ represents the set of n customer vertices. For customer (vertex) i, the order to be delivered is characterized by an individual load $q_i \geq 0$, a due date $d_i \geq 0$, a release date $r_i \geq 0$, and a weight $w_i \geq 0$. A homogeneous fleet of m vehicles, each with capacity Q, is stationed at the depot (vertex) 0. Each arc $(i, j) \in A$ has an associated distance cost c_{ij} and travel time τ_{ij}. We assume each τ_{ij} includes the service time at vertex i (which is zero if i is the depot). If the triangle inequality holds for travel times, then it also holds for the modified τ_{ij} values that include equal service times.

A solution x to the VRPRDD comprises m routes, one for each vehicle, where each customer is assigned to exactly one route and empty routes are allowed. Each route r is an elementary circuit in G that contains the depot. More precisely, for any route $r = 1, \ldots, m$, let R^r be the set of customers visited in r and let $n^r = |R^r|$ be the number of customers in r. Then a route r can be represented by a permutation $(\sigma^r(1), \ldots, \sigma^r(n^r))$ of the elements of R^r, where $\sigma^r(i)$ is the i^{th} customer to be visited, for $i = 1, \ldots, n^r$. The route starts and ends at the depot, so we naturally set $\sigma^r(0) = \sigma^r(n^r + 1) = 0$. The solution is feasible if the capacity constraint $\sum_{i \in R^r} q_i \leq Q$ is satisfied for $r = 1, \ldots, m$. Further, the vehicle for route r leaves the depot at time $\max_{i \in R^r} r_i$, so that the vehicle's earliest departure time is equal to the maximum release date of the orders for the customers visited in r.

The objective function f to be minimized is a convex combination of the total distance cost and total weighted tardiness, and is defined by

$$f(x) = \alpha \sum_{r=1}^{m} \sum_{i=0}^{n^r} c_{\sigma^r(i), \sigma^r(i+1)} + (1 - \alpha) \sum_{r=1}^{m} \sum_{i \in R^r} w_i \max\{0, S_i - d_i\}, \tag{3.1}$$

where x is a solution, S_i is the time at which the service for customer i starts in solution x, $\max\{0, S_i - d_i\}$ is the tardiness for customer i and $0 \leq \alpha \leq 1$ defines the relative weight of the two objective function components. Since f is a non-decreasing function of the arrival times S_i, then vehicle idle time is never beneficial. Therefore, without loss of generality, we may assume that

$$S_{\sigma^r(i)} = \max_{j \in R^r}\{r_j\} + \sum_{j=1}^{i} \tau_{\sigma^r(j-1),\sigma^r(j)}, \forall\, i = 1, \ldots, n^r, r = 1, \ldots, m. \tag{3.2}$$

Figure 3.1 presents a small-sized instance I, together with two different solutions a and b. Instance I has $n = 5$, $m = 2$, $Q = 3$, and $q_i = w_i = 1$ for $i \in V'$. For each arc $(i, j) \in A$, we have $\tau_{ij} = d_{ij}$, and these values are shown on the edges. For each customer i, the values $[r_i, d_i]$ are given in the problem instance within Figure 3.1, and solutions a and b show the values $(S_i, w_i T_i)$ for the given routing. The respective total distance travelled and total weighted tardiness are reported below the solutions. If $\alpha = 0.7$, then a is optimal and $f(a) = 7.6$ and $f(b) = 8.4$, but if $\alpha = 0.3$, then b is optimal and $f(a) = 8.4$ and $f(b) = 7.6$. Finally, if $\alpha = 0.5$, then both solutions are optimal and $f(a) = f(b) = 8$.

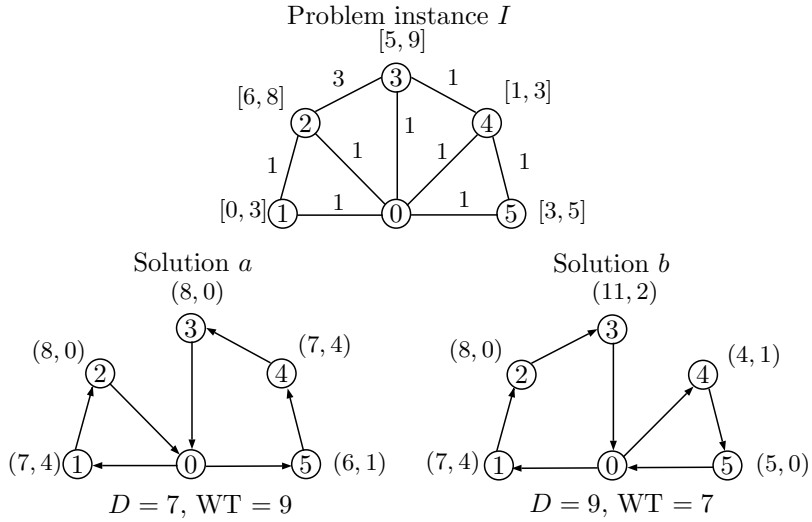

Figure 3.1: VRPRDD instance and optimal solutions for different values of α.

3.2 Complexity

We claim that the VRPRDD is unary NP-hard for all values of α satisfying $0 \leq \alpha \leq 1$, and this is validated as follows. For $\alpha = 1$, or for $0 < \alpha < 1$ and $d_i = \infty$ for $i \in V'$, then (3.1) reduces to the total distance travelled objective function. For this objective, the problem is equivalent to the CVRP, which is unary NP-hard by reduction from the travelling salesman problem. For $\alpha = 0$, $Q = \infty$, $m = 1$, and $r_i = 0$ and $w_i = 1$ for $i \in V'$, then the problem is the total tardiness problem with sequence dependent set-up times. The total tardiness objective is reducible from the makespan objective (Lenstra et al. 1977), and the makespan problem with sequence dependent set-up times is unary NP-hard by reduction from the Hamiltonian path problem. Notice also that the Hamiltonian path problem is reducible to the travelling salesman problem.

These complexity results hold for the cases of equal weights w_i, equal release dates r_i, unlimited vehicle capacity Q, a single vehicle, and equal or non-restrictive due dates d_i. Therefore, even after relaxing these aspects of the problem, the VRPRDD is still unary NP-hard, which reveals some of the inherent complexity of the problem.

3.3 Benchmark instances

We propose three problem types that have different values for α in (3.1), and a set of 96 benchmark instances for the VRPRDD. This enables us to perform computational experiments with the aim of assessing the performance of the proposed solution methods and obtaining insights into the problem.

Generating instances of scheduling problems is discussed by Hall and Posner (2001), who propose a number of principles and properties for generating adequately representative and unbiased data. Following these principles, we restrict the set of problem features varied to enable greater comparability and to facilitate analysis of the key features of the novel problem that we study. To reduce any possible bias in the instances, we encourage size- and scale-invariance in the problem features, and we also use the uniform distribution to generate many of the new features of the instances. We define $U_{\mathbb{Z}}[c, d]$ as the integer uniform distribution between integers c and d inclusive.

We extend four CVRP instances introduced by Christofides et al. (1979), where $n = 50$ and $m = 5$, $n = 100$ and $m = 10$, $n = 150$ and $m = 14$, and $n = 199$ and $m = 20$. The coordinates of customers are randomly distributed in the interval $[0, 100]$, so that they lie inside a square with sides of length 100, and the depot is relatively close to the centre. The instances are well documented and are available from the website `http://neo.lcc.uma.es`. The optimal CVRP solutions for the instances are found by Pecin (2014).

Let I be a CVRP instance with m_I vehicles, and $D_{\max}$ be the duration of the longest route in the best known solution of I. After generating the travel times, we introduce $E(\tau)$ as an estimate of the average travel time between vertices. Specifically

$$E(\tau) = \frac{\sum_{i \in V'} \sum_{j \in \mathcal{N}_i} \tau_{ij}}{|V'| \, |[n^{1/2}]|}, \tag{3.3}$$

where the operator $|[a]|$ is defined as previously and rounds a to the nearest integer, and $\mathcal{N}_i$ as the set of $|[n^{1/2}]|$ vertices that have smallest travel times to vertex i. We choose the $n^{1/2}$ vertices with shortest travel time because a number of studies, such as Beardwood et al. (1959) and Johnson et al. (1996), identify a relationship between the expected cycle costs of the TSP and $n^{1/2}$.

Various parameters are used to generate the instances: $e_m \geq 0$ defines the proportion of additional vehicles above m_I, so that the number of vehicles is $(1+e_m)m_I$, $b \geq 0$ is the spread of the release dates as a proportion of $D_{\max}$, $k \in \mathbb{Z}^+$ is the looseness of the due dates as a multiple of $E(\tau)$, and $0 \leq v \leq 1$ is half of the range that k can vary within as a proportion of k.

We adapt and extend I as follows:

1. Set $m = (1 + e_m)m_I$, $w_i = 1$, for all $i \in V'$, and $\tau_{ij} = c_{ij}$, for all $i, j \in V$.
2. Generate $r_i = X$, for all $i \in V'$, where $X \sim U_{\mathbb{Z}}\big[0, |[bD_{\max}]|\big]$.

3. Calculate $E(\tau)$, and generate $d_i = r_i + |[(k + Y - kv)E(\tau)]|$, for all $i \in V'$, where $Y \sim U[0, 2kv]$.

For simplicity, unit weights and unit vehicle speeds are used. Notice that if the underlying CVRP instance has a feasible solution, then the resulting VRPRDD instance also has a feasible solution because $e_m \geq 0$ and hence $m \geq m_I$.

In Table 3.1, the different values of the parameters used to generate the instances are presented. Introducing constraints and objectives related to the arrival times of customers to the CVRP affects the total distance travelled in optimal solutions. This is adjusted in the objective (3.1) by selecting different values of α, where we consider the objectives weighted equally and use weightings to increase the emphasis on each objective individually. In some instances, it may be beneficial to use more vehicles than in the optimal CVRP solution, and sometimes considerably more. Therefore, we consider two settings for e_m, resulting in instances with either an equal number of vehicles to the underlying CVRP instance, or 50% more vehicles. Preliminary experiments with 100% more vehicles suggest that more than 50% has little effect on the optimal solution for most of the instances. The release dates are a novel consideration in VRPs, and we use a range of spreads to enable a more detailed analysis. To decide on appropriate values for parameter k, we consider the range of the number of customers in routes in the best known solutions, and the average number of customers per route n/v, where v is the number of routes in the best known solution.

Table 3.1: Instance generation parameters

Instance parameter		Values
α	objective weight	$0.3, 0.5, 0.7$
e_m	proportion of extra vehicles	$0, 0.5$
b	spread of release dates	$0.25, 0.50, 0.75, 1.00$
k	looseness of due dates	$4, 6, 8$
v	range in looseness of due dates	0.25

Notice that if b and k are sufficiently low and high, respectively, then the distance travelled will tend to dominate the objective for any value of α. Alternatively, high b and low k result in the weighted tardiness dominating the objective for any value of α. If the weighted tardiness has high priority, then more vehicles will be used to enable a greater variety of vehicle departure times and earlier arrival times at customers; otherwise, additional vehicles often represent extra distance travelled. As a consequence, if there is a low number of vehicles available, then the priority of the weighted tardiness will increase. The considerations above demonstrate some of the complex interplay between the problem features, and emphasize the need to generate a set of varied instances to gain a better understanding of the impact on solutions of these key features.

Considering the complexity of the VRPRDD, we have also generated a number of smaller instances. To construct instances with $n = \{20, 30\}$, we randomly select n customers from the corresponding instance with $n = 50$ and reduce the number of vehicles by the same proportion as the customers, $n/50$. A small number of the the new instances did not have feasible solutions because all the demand of the customers does not fit in the available vehicles. To make these instances feasible, Q is increased by half the excess demand until the instance has feasible solutions. This approach was chosen to attempt to retain the distribution of the

demands relative to the vehicle capacity in the underlying instance, and avoid introducing any unintended bias to the instances. To test whether an instance has feasible solutions we apply the path relinking algorithm from Chapter 6.

Bibliography

Beardwood, J., Halton, J. H., and Hammersley, J. M. (1959). The shortest path through many points. *Mathematical Proceedings of the Cambridge Philosophical Society*, 55:299–327.

Christofides, N., Mingozzi, A., and Toth, P. (1979). The vehicle routing problem. In Christofides, N., Mingozzi, A., Toth, P., and Sandi, C., editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, UK.

Hall, N. G. and Posner, M. E. (2001). Generating experimental data for computational testing with machine scheduling applications. *Operations Research*, 49(6):854–865.

Johnson, D. S., McGeoch, L. A., and Rothberg, E. E. (1996). Asymptotic experimental analysis for the held-karp traveling salesman bound. In *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 341–350. Society for Industrial and Applied Mathematics.

Lenstra, J., Kan, A. R., and Brucker, P. (1977). Complexity of machine scheduling problems. In P.L. Hammer, E.L. Johnson, B. K. and Nemhauser, G., editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343 – 362. Elsevier.

Pecin, D. (2014). Exact algorithms for the capacitated vehicle routing problem. PhD Thesis, Pontifícia Universidade Católica do Rio de Janiero, Rio de Janiero, Brazil.

Chapter 4

Vehicle routing and scheduling problems

In this chapter, we describe two types of vehicle routing problem (VRP) that have similarities to the VRP with release and due dates (VRPRDD). A review of some milestones in the development of exact algorithms and heuristics for these VRPs is presented, where we seek to identify currently available techniques that are successful in providing tight lower bounds and optimal solutions, or efficiently finding high-quality solutions, respectively. Lastly, the main features of these exact algorithms and heuristics for one of the problems are compared and contrasted.

4.1 The capacitated vehicle routing problem

The capacitated VRP is the first VRP studied in the literature, and is suggested by Dantzig and Ramser (1959) in their seminal paper. We will formally define the problem as follows.

Definition 4.1 (*capacitated vehicle routing problem* (CVRP))**.** Given a complete graph $G = (V, A)$, where the vertex set is $V = \{0, 1, \ldots, n\}$ and arc set is $A = \{(i, j) : i, j \in V, i \neq j\}$. Vertex 0 corresponds to the depot, while $V' = V \backslash \{0\}$ represent the set of n customer vertices. Each arc $(i, j) \in A$ has an associated distance $d_{ij} \in \mathbb{Z}^+$, and each customer $i \in V \backslash \{0\}$ has an individual demand $q_i \in \mathbb{Z}^+$. A route $r = \{0, i_1, \cdots, i_k, 0\}$ is defined as an elementary cycle in G which includes the depot.

Find a set of m routes with minimum total distance, such that each customers receives their entire demand in a single visit, and for each route the demand of the assigned customers does not exceed Q.

If the distance matrix is symmetric, or $d_{ij} = d_{ji}$ for $i, j \in V$, then the graph can be defined as $G = (V, E)$, where the edge set is $E = \{(i, j) : i, j \in V, i < j\}$. The CVRP is clearly NP-hard, because the TSP is a special case that arises when $m = 1$ and $Q = \infty$.

4.1.1 Exact algorithms

In this section, we present a number of mathematical formulations of the CVRP and describe some exact algorithms to address these formulations. We focus on two important MIP models of the CVRP, and give a review of some important contributions and developments. For a

more detailed discussion, we refer the reader to the chapters on classical and new exact algorithms by Semet et al. (2014) and Poggi and Uchoa (2014), respectively.

Arc-based formulations

The *two-index* formulation of the CVRP is suggested by Laporte et al. (1984), and is an extension of the MIP formulation of the TSP proposed by Dantzig et al. (1954). For simplicity, we present the case for a symmetric distance matrix, but it can easily be adapted for an asymmetric distance matrix.

Let $\delta(i) \subset E$ be the set of edges connected to $i \in V$, $\delta(i) = \{(i,j),(j,i) \in E\}$ and $r(S)$ be a lower bound on the number of vehicles needed to feasibly visit $S \subseteq V$. A trivial value for the latter is the *bin packing lower bound* $\lceil \sum_{i \in S} q_i / Q \rceil$, but a tighter definition can tighten the formulation. We define $O(n^2)$ integer variables x_e representing the number of times $e \in E$ is used in the solution, if e is connected to the depot then $x_e \in \{0,1,2\}$ and otherwise $x_e \in \{0,1\}$.

$$(F) \quad z(F) \min \sum_{e \in E} c_e x_e, \tag{4.1}$$

$$\text{s.t.} \sum_{e \in \delta(i)} x_e = 2, \qquad \forall\, i \in V', \tag{4.2}$$

$$\sum_{e \in \delta(0)} x_e = 2m, \tag{4.3}$$

$$\sum_{e \in \delta(S)} x_e \geq 2r(S), \qquad S \subseteq V', \tag{4.4}$$

$$x_e \in \{0,1\}, \qquad e \notin \delta(0), \tag{4.5}$$

$$x_e \in \{0,1,2\}, \qquad e \in \delta(0). \tag{4.6}$$

Constraints (4.2) and (4.3) ensure that each customer is entered and exited and the depot is entered and exited m times, respectively. Constraints (4.4) are similar to the subtour elmination constraints from the IP formulation of the TSP, and require that the number of edges used that are incident to vertices in $S \subseteq V$ is at least twice $r(S)$. This prevents subtours and ensures that the route capacities are not violated. If $r(S)$ is defined as the bin packing lower bound, then these constraints are usually described as the *rounded capacity* constraints. Unfortunately, there are an exponential number of constraints (4.4), and although it is possible to replace these constraints with an adaptation of the Miller et al. (1960) subtour elimination constraints, the linear relaxation generally provides a considerably weaker lower bound.

Similarly to the TSP, a cutting-planes algorithm is usually adopted to solve F, where constraints (4.4) are removed and separated as necessary. A variety of other families of valid inequalities are proposed for F, and all the valid inequalities of the TSP are still valid. These inequalities and the related separation procedures are explained by Naddef and Rinaldi (2002). These authors also describe cutting-planes and BC algorithms, which are shown to solve benchmark instances with up to 45 and 135 customers, respectively.

A *one-commodity flow* (OC) formulation is presented by Gouveia (1995), which restricts subtours but also has polynomial size. This is achieved by defining a commodity flow on the

arcs that cumulates the demand of each route. Extending this idea, Baldacci et al. (2004) introduce a *two-commodity flow* formulation, but Letchford and Salazar-González (2006) show by projection that this achieves the same lower bound as the OC formulation. More recently, Letchford and Salazar-González (2015) have presented a number of stronger multi-commodity flow formulations that define n-commodities, one for each customer.

Path-based formulations

The *set-partitioning* (SP) formulation of the CVRP is a path-based formulation rather than arc-based, and is first suggested by Balinski and Quandt (1964), and . Let Ω be the set of all feasible routes, and for $r \in \Omega$, c_r is the cost of r and a_i^r is a binary indicator equal to 1 if $i \in V$ is visited by r. We define $O(|\Omega|)$ binary variables λ_r, which are equal to 1 if $r \in \Omega$ is selected in the solution. Then the SP formulation can be defined as follows.

$$\text{(SP)} \quad z(\text{SP}) = \min \sum_{r \in \Omega} c_r \lambda_r, \tag{4.7}$$

$$\text{s.t.} \sum_{r \in \Omega} a_i^r \lambda_r = 1, \qquad \forall\, i \in V', \tag{4.8}$$

$$\sum_{r \in \Omega} \lambda_r \leq m, \tag{4.9}$$

$$\lambda_r \in \{0, 1\}, \qquad \forall\, r \in \Omega. \tag{4.10}$$

Constraints (4.8) require that each customer is visited exactly once, and (4.9) requires that up to m routes are used. Ω has exponential size, and computing each c_r value requires solving the TSP for the associated set of vertices. Although the formulation is usually solved using a (delayed) column generation (CG) algorithm, where the pricing problem (PP) is a capacitated elementary shortest path problem (CESPP).

Note that the definition of the set Ω and costs c_r are very general, and any constraints or costs defined for individual routes or customers are implicit in the SP formulation. Further constraints will reduce the cardinality of Ω, although computing the costs or solving the PP becomes harder.

The relationship between the λ_r variables in the SP formulation and the x_{ij} variables in the F formulation is explored by Baldacci et al. (2004), who observe that many inequalities and branching strategies from F can be applied to the SP formulation. The authors also prove that the linear relaxation of the SP formulation is a tighter lower bound than the linear relaxation of the F formulation if the bin packing lower bound is used as the vehicle bound in (4.4), by demonstrating a variety of valid inequalities that are implied by the linear relaxation of the SP formulation but not the linear relaxation of the F formulation. Further valid inequalities are shown to be implied by Letchford and Salazar-González (2006).

CG algorithms seem a natural choice for solving the linear relaxation of the SP formulation, but experience suggests that the lower bound is not always tight enough to solve popular instances in acceptable computational without valid inequalities. This is demonstrated by Fukasawa et al. (2006), who propose a branch, price and cut (BCP) algorithm that builds on the branch and cut (BC) algorithm of Lysgaard et al. (2004).

To reduce the complexity of the PP, Fukasawa et al. (2006) solve a capacitated shortest path problem without 2-cycles using a label-setting algorithm, and also propose a number

of heuristics based on this algorithm. In this relaxation, Ω is defined as the set of not necessarily elementary *q-routes* without 2-cycles. It is notable that the formulation now has both an exponential number of variables and constraints.

The cuts separated by Fukasawa et al. (2006) are described as *robust* in the terminology of Poggi de Aragao and Uchoa (2003), because the dual variable associated to each cut can be incorporated in the PP without increasing the complexity or modifying the structure. Otherwise, cuts are *non-robust* and introducing a cut to the RMP will impact the complexity of the PP.

The current most successful exact algorithms for the CVRP are BPC algorithm proposed by Pecin (2014) and Contardo and Martinelli (2014), and these incorporate a variety of developments in CG. Many of these developments are shared with CG for the VRPTW and will be discussed in the following section. For further details, the interested reader is referred to Poggi and Uchoa (2014). .

4.1.2 Heuristics

The complexity of the CVRP and the limited size of instances that could be solved to optimality historically, leads to a multitude of heuristics proposed in the literature. Extensive surveys can be found in Cordeau et al. (2002), Laporte (2009), and more recently Laporte et al. (2014).

Constructive heuristics

Many constructive heuristics for the CVRP are proposed in the first three decades of study. These heuristics have low time complexity, but the objective values achieved are relatively far from optimal.

The *savings heuristic* is proposed by Clark and Wright (1964) as an extention of the same heuristics for the travelling salesman problem, and is the first heuristic for the CVRP. The heuristic starts with a route for each individual customer, and then computes the improvement in objective value or saving from connecting any two routes to form a single route. This can be calculated for two routes involving customers i and j, respectively, as $s_{ij} = d_{i0} + d_{0j} - d_{ij}$.

During the algorithm, routes are only combined at the extremities, and the savings are therefore valid throughout. The savings are ordered by non-decreasing value, and the associated routes are considered to be combined in order. The method has been noted by Laporte and Semet (2002) to find good routes toward the start of the construction, but this deteriorates towards the end.

Some of the deficiencies of the approach are improved by Gaskell (1967), who propose calculating the savings as $s'_{ij} = d_{i0} + d_{0j} - \lambda d_{ij}$, where the parameter $\lambda \geq 0$ controls the relative importance of the distance from the depot compared to the distance between the route extremities. Notice that different solutions are produced by the savings heuristic for different λ values.

Other constructive heuristics are popular for the CVRP. The *sweep heuristic* is proposed by Gillett and Miller (1974) for euclidean problems. This rotates a half-line rooted at the depot, and includes customers in an open route in order of intersection with the half-line. When a customer will exceed the capacity constraint then the current route is closed and a new route is opened.

Another classical example is the family of sequential *insertion heuristics* (see, e.g. Mole and Jameson 1976, Christofides et al. 1979), where the general idea is to initially order the customers, then in this order insert the customers in the least costly and feasible position in the existing partial solution. A key variation in the approaches described is sequential or parallel route construction.

A number of hierarchical decomposition heuristics are proposed for the CVRP. Most commonly the problem is decomposed by separating the partitioning of customers into routes and the sequencing of customers. These heuristics can generally be categorised as *route first and cluster second*, which is first proposed by Newton and Thomas (1974), and *cluster first and route second*, which is first proposed by Fisher and Jaikumar (1981). As the name suggests, the first category constructs a tour through all customers and the depot, then partitions this into routes involving the depot. The first phase requires solving a TSP to produce a tour of all the customers, and Prins (2004) shows this can be partitioned optimally by solving a shortest path problem. Note that this is not the optimal solution to the original problem but the partitioning sub-problem.

In the second approach, we partition the customers and then solve the TSP for each partition. The partitioning phase is modelled as a generalised assignment problem by Fisher and Jaikumar (1981), where m seed locations are chosen in areas of high customer density. The customers must each be assigned to a location such that the total demand of customers assigned to any location is less than Q, and the cost of assigning a set of customers to a location is a linear approximation of the distance of a tour containing those customers. For each resulting partition, the optimal sequence is found by solving the TSP.

Neighbourhood search

Neighbourhoods for the CVRP can be categorised as *intra-route*, if they operate on a single route, or *inter-route*, if they operating on more than one route. Inter-route neighbourhoods are most commonly defined to operate on two routes, and sometimes the union of the inter- and intra-route neighbourhoods are used.

Neighbourhoods can be described by changing k edges used in a solution described as k-Opt, or through moving the positions of customers. Although these definitions are equivalent, the distinction can be helpful when considering the evaluation of feasibility and the improvement in objective value associated to a move. Additionally, this distinction can also enable a better appreciation of how much the objective function may change from a move. For the CVRP, the contribution of a customer to the objective value is only dependent on the edges incident to it, and therefore, considering neighbourhoods as changing edges reveals the effect of a move on the objective value. If the contribution to the objective function or feasibility of a move is dependent on more than the edges incident to the customers, for example on the arrival times at the customers, then defining neighbourhoods as changing the positions of customers represents the effect of the move more clearly.

Intra-route neighbourhoods are often borrowed directly from the TSP, and these can usually be described as subsets of the neighbourhoods 2-Opt, 3-Opt and 4-Opt. 2-Opt is often used as an intra-route neighbourhood, and can be defined as either changing two edges or reversing the orientation of a subsequence of consecutive customers. Notice that if the distance matrix is asymmetric then a 2-Opt move changes all the edges within the subsequence

of customers between the two edges involved in the move.

Some specially defined subset of k-opt neighbourhoods are popular in the literature, for example:

- *Or-Opt* - Remove a subsequence of consecutive customers from its position in a route and reinsert it in another position of the same route.
- k-insert - Remove a subsequence of between 1 and k consecutive customers from its position and reinserting it into another position on the same route or into a different route.
- k-swap - Exchange the positions of two distinct subsequences of between 1 and k consecutive customers from the same route or different routes.
- 2-Opt* - Exchange the ends of two routes, where one can be empty.
- CROSS-exchange - Exchange two subsequences of consecutive customers from two different routes, where one subsequence can be empty.

Or-Opt is proposed by Or (1976) for the TSP, and is a subset of 3-Opt. The *k-insert* neighbourhood is also a subset of 3-Opt and also a superset of Or-Opt. The *k-insert* and *k-swap* neighbourhoods are subsets of 3-Opt and 4-Opt, respectively. The *2-Opt** neighbourhood is proposed by Potvin and Rousseau (1993), and is a subset of inter-route 2-Opt. *CROSS-exchange* is suggested by Taillard et al. (1997), and is the union of subsets of 3-Opt and 4-Opt, and is a superset of 2-Opt*.

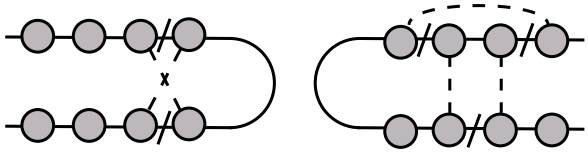

Figure 4.1: From left to right, intra-route 2-Opt and Or-Opt.

A larger neighbourhood is described by Osman (1993), and is defined by exchanging up to k customers from one route with up to k customer from another route. This definition includes any inter-route neighbourhood, and has size exponential in both n and k which can be prohibitive for even small values of n and $k > 1$.

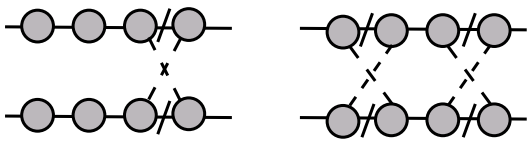

Figure 4.2: From left to right, 2-Opt* and CROSS-exchange.

Variable depth neighbourhood search is proposed by Lin and Kernighan (1973) for the TSP, and is developed by Glover (1992), under the term *ejection-chain neighbourhoods*. Rego and Roucairol (1996) design an ejection-chain neighbourhood for the CVRP, which searches for compound moves which are a cycle of inter-route insert moves, such that one customer is removed and introduced to each route.

A similar approach is proposed by Thompson and Psaraftis (1993), who search a very large neighbourhood defined by cyclic insertions of k customers between b routes. The search

for an improving move is transformed into finding a negative cost cycle in a graph defined by the moves. This graph can be efficiently constructed, but finding the best neighbour is shown to be NP-hard. Ergun and Orlin (2006) develop this idea to construct compound moves from independent or so-called weakly-independent moves in a variety of neighbourhoods. Although the neighbourhood proposed has exponential size, it is shown to be searchable in polynomial time using dynamic programming.

Ruin-and-recreate heuristics are proposed for the CVRP by Shaw (1998). These originally use constraint programming, but have been extended to using constructive heuristics and Prescott-Gagnon et al. (2009) describe a BB algorithm. A considerable number of constructive heuristics are proposed, and many can be found in Pisinger and Ropke (2007).

Single solution based heuristics

A variety of single solution based heuristics are proposed for the CVRP, and generally differ in the techniques used to introduce diversity. For example, *simulated annealing* (SA) algorithms are proposed by Osman (1993) and Chen et al. (2007). Although, the *deterministic annealing* algorithms suggested Li et al. (2005), and more recently Gröer et al. (2011), are shown to perform better empirically.

A considerable interest in *tabu search* (TS) heuristics is evidenced by the algorithms of Gendreau et al. (1994), Cordeau et al. (1997) and Cordeau et al. (2001b). Although more recently, a more random approach to diversification is used. Kilby et al. (1999) propose applying *guided local search* (GLS) to the CVRP, and a competitive implementation is more recently proposed by Zachariadis and Kiranoudis (2010). GLS is also an important component in the *variable neighbourhood search* of Kytöjoki et al. (2007), which is the best performing heuristic for very large-scale instances.

Population based heuristics

Many of the most successful metaheuristics for the classical instances of the CVRP are adaptations or enhancements of *genetic algorithms* (GAs) and the *evolutionary algorithms* metaphor. A general review of these approaches can be found in Potvin (2009).

A considerable variety of *Hybrid GAs* (HGAs) are proposed, for example, by Prins (2004), Mester et al. (2007), Nagata and Bräysy (2008) and Vidal et al. (2012). The importance of preserving and introducing diversity amongst solutions in the population when NS is applied to offspring solutions is emphasised by Prins (2004), Sörensen and Sevaux (2006) and Vidal et al. (2012). To recombine solutions Nagata and Bräysy (2008) extend the *edge-assembly crossover* proposed by one of the authors for the TSP. Both Prins (2004) and Vidal et al. (2012) use the *big tour* solution representation, and a feasible solution is retrieved using the optimal *split* algorithm proposed by the former. This solution representation allows solutions to be recombined using basic crossovers for permutations.

Sörensen and Schittekat (2013) propose a hybrid PRA, but find that is not statistically better with the PRA for the Augerat et al. (1995) instances. The authors consider a number of implementation choices, however, a single solution is always returned from each relinking, and the relinking procedure minimizes the trajectory length. Better success is reported by Ho and Gendreau (2006), who improved solution quality and speed using a hybrid PRA and

TS in comparison to the TS alone. These authors address the Christofides et al. (1979) and Golden et al. (1998) instances.

Some limited interest is shown in *swarm optimisation* (SO) methods for the CVRP. For example, Doerner et al. (2004) and Reimann et al. (2004) propose *ant colony optimisation* algorithms, Marinakis and Marinaki (2010a) describe a *bee mating algorithm*, and Marinakis and Marinaki (2010b) propose a *particle swarm optimisation* and GA hybrid.

A comparison of the best performing approaches for the Golden et al. (1998) instances is presented by Vidal et al. (2013a). The comparison is made on the average percentage difference of objective value from the *best known solutions* in the literature. The authors show their own HGA, the parallel R-to-R matheuristic of Gröer et al. (2011), and the HGA of Nagata and Bräysy (2008) to be the three best performing heuristics. The best achieving single-solution or trajectory-based metaheuristic is the GLS and TS hybrid algorithm of Zachariadis and Kiranoudis (2010).

4.2 Vehicle routing problems with time windows

Early mention of VRPs with time windows can be found in the papers of Schrage (1981) and Bodin and Golden (1981), but research attention is truly directed to these problems after the paper by Solomon (1987). In contrast to the VRPRDD, the customer orders are assumed to be available for delivery immediately and (hard) time window constraints are defined by lower and upper limits on the start time of service at each customer. Additionally, the number of vehicles is usually unlimited and the objective is hierarchical, where minimizing the number of vehicles required is the primary objective and minimizing the total distance travelled is the secondary objective, and otherwise the number of vehicles is omitted from the objective. We will formally define the problem as follows.

Definition 4.2 (*vehicle routing problem with time windows* (VRPTW))**.** Given a complete graph $G = (V, A)$, where the vertex set is $V = \{0, 1, \ldots, n\}$ and the arc set is $A = \{(i, j) : i, j \in V, i \neq j\}$. Vertex 0 corresponds to the depot, while $V' = V \backslash \{0\}$ represent the set of n customer vertices. Each arc $(i, j) \in A$ has an associated distance d_{ij} and travel time $\tau_{ij} > 0$, where τij includes any service time at customer i. For each customer $i \in V \backslash \{0\}$, we are given an individual demand q_i, and a time window $[e_i, l_i]$, where e_i and l_i are the earliest and latest times that a vertex can be serviced respectively. A route $r = \{0, i_1, \cdots, i_k, 0\}$ is defined as an elementary cycle in G which includes the depot.

Find a set of up to m routes with minimum total distance, such that each customer is visited once and receives their entire demand, and for each route:

- The demand of the assigned customers does not exceed Q,
- the departure time from and arrival time back to the depot is no less than e_0 and no greater than l_0 respectively,
- service at a customer i must complete in the interval $[e_i, l_i]$.

Often both the τ_{ij} and d_{ij} values are assumed to be symmetric and satisfy the triangle inequality. It is also possible to reduce the arc set as described in Desrochers et al. (1992), for example $(i, j) \in A$ can be removed if $e_i + \tau_{ij} > l_j$. As mentioned, the objective is to

either minimises the total distance travelled, or minimise the number of routes before the total distance travelled.

The VRPTW is a restriction of the CVRP, which arises when $e_i = 0$ and $l_i = \infty$ for $i \in V$, and is therefore NP-hard. Furthermore, Savelsbergh (1985) proves that even if we assume the distance matrix is symmetric and satisfies the triangle inequality, then finding a feasible solution for a fixed number of vehicles is unary NP-complete.

The vehicle routing problem with soft time windows (VRPSTW) is a generalization of the VRPTW, allowing violations of the time windows with penalties for earliness and tardiness. Again, if the number of vehicles is unlimited, then the objective is again hierarchical, where minimizing the number of vehicles is the primary objective and minimizing the total distance plus the weighted earliness and tardiness is the secondary objective.

Fu et al. (2008) provide some motivations for the VRPSTW over the VRPTW, and a classification of six types of soft time windows. Popular examples are type 1 that relaxes the upper limit on the time window, type 4 that is the same as type 1 but also introduce (hard) time deadlines for each customer, and type 2 that relaxes both the upper and lower limits on the time windows. Note that the studies in the literature on the VRPSTW predominately use equal earliness and tardiness weights in the objective.

4.2.1 Exact algorithms

The significant practical value and popularity of the VRPTW motivates a sustained research effort to develop efficient exact algorithms, and many different approaches are proposed. In this section, we will give a brief survey of the developments in these algorithms. For the interested reader, further information on exact algorithms for the VRPTW can be found in the review of Desaulniers et al. (2010), and more recently Desaulniers et al. (2014).

The literature is generally sparser and more fragmented for the VRPSTW, and to the best of our knowledge no algorithms that separate cuts are proposed for any variant. The most successful exact approaches for these problems currently rely heavily on implicit route enumeration, and therefore instances that have a greater number of feasible routes are generally the most difficult.

Branch-and-cut algorithms

The *three-index* formulation of the VRPTW is one of the earliest MIP models of the problem, and presentations are given by Kohl et al. (1999) and Cordeau et al. (2001a). The former authors attempt to tighten the formulation by introducing the *k-path cuts*, which extended the rounded capacity constraints from the CVRP. The latter authors describe a Lagrangian relaxation approach, and note the general difficulty of finding tight lower bound using these types of formulation for the VRPTW. A tighter formulation is proposed by Kallehauge et al. (2007), including the proposed path inequalities.

In the most successful BC algorithms for these formulations, a large number and variety of cuts are required to tighten the lower bound at each node. It must also be noted that the most successful of these, suggested by Kallehauge et al. (2007), is not able to solve most popular instances with 50 or more customers in one hour of computation time. Another family of valid inequalities is proposed for the VRPTW by Lysgaard (2006), called the reachability inequalities, and are further extended by Avella et al. (2013). Although tt must be noted

that few valid inequalities are described specifically for the VRPTW, and usually general routing facets are considered in the literature. This is attributed to the difficulty of finding effective valid inequalities and the large numbers of cuts required to find acceptable lower bounds.

Branch-price-and-cut algorithms

The current most successful exact algorithms for the VRPTW and VRPSTW are branch and price (BP) or branch, price and cut (BPC) algorithms. These algorithms work with set-partitioning (SP) or set-covering (SC) formulations, which are identical to the same formulations of the CVRP. For the VRPTW, the PP is an elementary shortest path problem with resource constraints (ESPPRC), and is proven to be NP-hard by Dror (1994). Whereas for the VRPSTW, the PP is an ESPPRC with soft time windows, and as a generalisation of the ESPPRC it clearly has at least equal complexity.

Branching strategies. In BP and BPC algorithms for VRPTWs, branching constraints are mostly formulated using the integrality of the number of routes, and the original arc-variables. As discussed in reference to the CVRP, the values for the arc-variables can easily be calculated using the solution to the RMP and binary indicators that are equal to one if a route uses an arc.

Most uniquely, Baldacci et al. (2008) propose a two-stage approach for the CVRP, that is extended to the VRPTW by Baldacci et al. (2011). In the first stage, a column and cut generation (CCG) algorithm is used to find a concise set of routes and cuts, and an attempt is made to solve this with a traditional BC algorithm.

Pricing problems and algorithms. In a CG approach, the PP must be solved repeatedly and represents the majority of the computational time. Therefore, a considerable research effort is made to explore techniques to do this more efficiently without compromising the lower bound. The first BP algorithm for the VRPTW is presented in the seminal paper by Desrochers et al. (1992). The authors relax the elementary route constraints, and the PP is then a shortest path problem with resource constraints (SPPRC) that can be solved in pseudo-polynomial time. This is marginally tightened by forbidding cycles involving two customers using the 2-cycle elimination approach presented by Christofides et al. (1981).

To explore the efficiency of solving the elementary pricing problem, Feillet et al. (2004) implement the DP formulation described by Beasley and Christofides (1989). This extends the DP formulation of the travelling salesman problem proposed by Held and Karp (1962), introducing resource constraints and relaxing the requirement that every customer must be visited. One interesting observation of Feillet et al. (2004) is that for resources that are consumed monotonically, if a customer cannot be visited feasibly at the current level of consumption then it can be added to the set of customers that cannot be visited. This approach more actively establishes the feasibility of an extention, and increases the potential to dominate more states in the DP algorithm.

Righini and Salani (2006) propose a general bounded bi-directional DP approach, where the set of feasible routes is decomposed into sets of either bounded forward or backward paths. These paths are then combined to produce a set of feasible routes. The size of the

sets of bounded forward or backward paths is smaller than the original set of feasible routes. Although it is not clear whether this reduces the overall computation time for either PP, because the combination of the paths can be time consuming.

Developing on the work of Feillet et al. (2004), decremental state-space relaxation (DSSR) is proposed independently by Boland et al. (2006) and Righini and Salani (2008). DSSR for DP is generally analogous to cutting-planes for integer programming. A state-space relaxation (SSR) is applied to the DP formulation of the PP that allows non-elementary routes. If the optimal solution to this relaxation is non-elementary and therefore has cycles, the formulation is modified to restrict at least one of these cycles. In the original proposals, some of the repeated customers are made critical, and in the DP formulation critical customers are constrained to be visited once. This process continues until the optimal solution is elementary and it is therefore optimal for the ESPPRC. We point out that in the original approach, the size of the DP formulation at each iteration is exponential in the number of critical customers. To mitigate this, Boland et al. (2006) propose a variety of approaches that try to efficiently select the critical customers to add in each iteration.

Opting to tighten the lower bounds from the non-elementary PP, Irnich and Villeneuve (2006) introduce *k-cycle* elimination as a generalisation of 2-cycle elimination. A detailed DP formulation is presented, but the size increases by a factor of $O(k!)$ compared to the SPPRC, and it is intractable for even relatively moderate values of k. An alternative approach is suggested by Baldacci et al. (2011), described as the ng-route relaxation. This tightens the SPPRC by preventing customers being visited again while they feature in the neighbourhood of all customer subsequently visited. The neighbourhoods for each customer $i \in V'$ are the customers that are mostly likely to be visited after i. Baldacci et al. (2011) suggest that good lower bounds are achieved if the neighbourhood for each customer comprises the Δ customers with smallest reduced arc cost. A DSSR algorithm for the ng-route relaxation is suggested by Martinelli et al. (2014). This allows larger neighbourhoods, because the customers are activated in each neighbourhood as the restrictions in each iteration.

To find columns of negative reduced cost, it is not always necessary to solve the PP to optimality. Importantly, Kohl et al. (1999) propose using heuristics to address the PP, and only solve it to optimality when these fail. Pricing heuristics are generally simple, and these are either based on constructive and neighbourhood search heuristics, or heuristic DP algorithms. Notably, a multi-start tabu search (MTS) for the ESPPRC is presented by Desaulniers et al. (2008). The MTS algorithm is controlled by two parameters, the number of times to restart the search with another basic column and the number of iterations to search neighbourhoods that either insert or remove a customer. The authors claim that the performance of this heuristic is less influenced by unstable dual variables and a large number of negative cost cycles in solutions.

Ioachim et al. (1998) study a similar problem to the PPs of the VRPTWs, which is described as the SPPRC and linear arrival time costs. There are time windows for each customers, and the sum of the weighted customer arrival times is added to the objective, where the weight can be positive or negative. Two DP formulations are compared, based on partial-discretisation of the time windows, or representing the arrival time costs of a path as a piecewise linear function of the feasible arrival times at the last customer in the path. Tagmouti et al. (2007) present a node-routing transformation of the arc routing problem

with time-dependent service costs, which could also be described as the arc routing problem with general time windows and no waiting. The PP presented is an ESPPRC and general time windows, and the authors present a DP formulation inspired by Ioachim et al. (1998) and Feillet et al. (2004). The elementary shortest path problem with resource constraints and type 4 soft time windows is described by Qureshi et al. (2009) as the PP for the type 4 VRPSTW. The authors also present a direct extension of the DP formulation presented by Feillet et al. (2004).

A more complex PP is described by Liberatore et al. (2010) for the type 2 VRPSTW. This is the ESPPRC with type 2 soft time windows, which is identical to the PP described by Tagmouti et al. (2007) except that waiting is permitted. For any given route in this problem, the number of states associated to scheduling the start of customer service at different times is exponential in the number of possible start times. The DP formulation presented has many similarities to the second formulation of Ioachim et al. (1998), and represents the weighted earliness and tardiness of a path as a function of the arrival time at the last customer. This is extended to form a bounded bi-directional DP formulation, and a number of pricing heuristics and a DSSR algorithm are also proposed.

In the three-stage Addressing the efficiency of finding tight lower bounds in a very different manner, Baldacci et al. (2008) describe a column and cut generation (CCG) algorithm. bi-direction DP algorithms are used to implicitly enumerate the set of feasible routes with negative reduced cost, using an approach analogous to the increasing lower bound interpretation of Dijkstra's Algorithm.

and a new SSR for the PP that is described as the *ng-route relaxation*. This relaxation provides a tight lower bound on the optimal solution to the PP by preventing customers being revisited if they feature in the neighbourhood of the subsequent customers, but does not guarantee an elementary route. Similarly to Baldacci et al. (2008), each time a set of routes is generated in the progress of the three dual heuristics then the DP is tightened by improving the estimation of the dual variables and separating cuts. The final DP formulation is used to generate the set of elementary routes of negative reduced cost similarly to Baldacci et al. (2008), but fathoms a greater number of routes using the information from the previous heuristics.

Column and cut generation. Column generation (CG), and column and cut generation (CCG) algorithms are used to solve the RMP and any branching subproblems, and manage algorithms and heuristics to solve the PP and the separation problems. Similarly to the branching constraints, valid inequalities for the SP or SC formulations are also defined on the original arc-variables. As discussed, many of these are usually valid inequalities for the CVRP.

Two approach are most notable, and both are proposed for the VRPTW. The first is the CCG algorithm proposed by Desaulniers et al. (2008). Notably, the authors propose a multi-phase pricing algorithm that uses a variety of heuristics to solve the PP more efficiently. The algorithm separates the generalised k-path inequalities from the arc-based formulation of the VRPTW, but also introduces constraints to the PP on repeated customers in the optimal solution to the RMP. This idea is adapted by Contardo et al. (2015), who suggest introducing the constraints into the PP by updating the neighbourhoods from the ng-route relaxation, or

introducing the strong degree cuts (SDCs), proposed in Contardo and Martinelli (2014), to the RMP formulation.

The second notable CCG algorithm for the VRPTW is proposed by Baldacci et al. (2011). Extending the three-stage CCG algorithm of Baldacci et al. (2004) for the CVRP, an additive lower bound is calculated by applying three heuristics in turn to the dual of the linear relaxation of the SP formulation. These heuristics use a combination of subgradient optimisation and CCG to tighten the dual variables. Each heuristic works with a partial set of columns and cuts from the previous heuristic, and the heuristics are applied in increasing complexity. This complexity is managed by using the increasingly accurate dual variables obtained from the previous heuristic, thereby reducing the number of columns with negative reduced cost and the number of cuts. The algorithm highlights the power of non-robust valid inequalities, and shows that the effect on the complexity of the PP can be handled by introducing the inequalities in a carefully controlled manner.

Comparison of algorithms and components

For the VRPTW, solving popular instances with up to 100 customers is usually possible, but many instances with 200 customers or more cause significant difficulties. For the VRPSTW, the situation is less clear but the intractability of the problem is demonstrated by Liberatore et al. (2010). These authors cannot solve most instances of the type 2 VRPSTW with 75 customers or more in one hour of computation time and for some instances find no valid lower bound.

Certain algorithmic strategies and components are more frequently used in the algorithms considered, and there are combinations of these which seem increasingly popular. To observe this, a comparison of the main features of the algorithms is presented in Table 4.1. The columns from left to right represent, whether or not the algorithm can handle soft time windows (as indicated by a "Y"); the master formulation; the branching strategies; the cuts separated; relaxations of the PP; heuristics for the PP; relaxation of the PP in DSSR; and restrictions in DSSR. If an algorithm is only proposed for the PP then the first three columns from the left have a line, and if it is only proposed for the master problem or its linear relaxation then the third columns from the left has a line.

The use of the covering or partitioning formulations of the RMP seems to be balanced, and a clear trade-off between the stability of the dual variables and the tightness of the lower bound is not apparent. The only exception is the partially-elementary set-partitioning formulation proposed by Desaulniers et al. (2008). A number of branching strategies are considered, with branching on the number of vehicles and arcs being the most common. Although much more of the research effort appears to be directed at the efficiency and effectiveness of the column and cut generation.

Historically, the complexity of the elementary PP appears to have encouraged authors to relax the PP, but the papers show an increased awareness of the importance of the tighter lower bound associated to the elementary PP. Further, the more recent variety of techniques to control the complexity of calculating this or similar lower bounds, suggests that the tightness and efficiency of the lower bounding procedure is critical.

Table 4.1: Features of heuristics for the VRPTW and VRPSTW

		RMP			PP			
							DSSR	
	STW	Form.	Branch	Cuts	Relax	Heur.	Relax	Restrict
D92		C,NE	V, D, A		2-cycle			
I98	Y	-	-	-	t-route			
K99		C,NE	RF	2-path	2-cycle	DP		
F04		-	-	-				
I06		-	-	-	3-cycle			
T07		P,E	A					
J08		C,E	Str, V	SR		DP		
R08		-	-	-			2-cycle	C+A
D08		P,PE	A	2-path, SR		DP, TS	2-cycle	C+A
Q09	Y	P,E	V, A					
L10	Y	C,E	V, A			C, DP, NS	t-route	C+MV
B11		P,NE+E	R	SC, SR, WSR	t-route, ng-route			
C15		P,E	-			DP, TS	2-cycle	C+MV
“ ”		P ,NE+E	-		2-cycle	“ ”	2-cycle	C+RMP
“ ”		P,NE+E	-		2-cycle	“ ”	2-cycle	C+RMP,F
“ ”		P,NE+E	-	SD	2-cycle	“ ”		
“ ”		P,NE+E	-	SD	ng-route	“ ”		

Note: Algorithms. D92: Desrochers et al. (1992); I98: Ioachim et al. (1998); K99: Kohl et al. (1999); F04: Feillet et al. (2004); I06: Irnich and Villeneuve (2006); T07: Tagmouti et al. (2007); J08: Jepsen et al. (2008); R08: Righini and Salani (2008); D08: Desaulniers et al. (2008); Q09: Qureshi et al. (2009); L10: Liberatore et al. (2010); B11: Baldacci et al. (2011); C15: Contardo et al. (2015).
Formulations. C: Set covering; P: Set partioning; PE: Partially elementary.
Branching strategies. V: Vehicles; D: Distance; A: Arcs; RF: Ryan and Foster; Str: Strong branching; R: Routes in a subsequent phase.
Cuts. SR: Subset row; SC: Strengthened capacity; WSR: Weak SR; SD: Strong degree.
DSSR restrictions. Crit: Critical customers; ng: ng-route neighbourhoods; A: All cycles of the optimal solution; MV: One of the most visited customer in the optimal solution; F: First cycle in the optimal solution; RMP: Cycles from the solution to the RMP.
Heuristics. C: Constructive; DP: Heuristic dynamic programming; NS: Neighbourhood search; TS: Tabu search.

Using efficient heuristics to find columns during the initial iterations of a CG algorithm seems to have become standard, and the popularity hints at the importance to the efficiency of the CG algorithms. DSSR is used in the majority of the methods proposed recently, and appears to be an important technique to efficiently calculate the elementary lower bound.

It is notable that no valid inequalities are used in the two algorithms that consider STW and solve the problem to optimality, but this is reflected in the lack of BC algorithms for the VRPSTW generally. This may also suggest that valid inequalities are less useful for VRPs with less constraints and more complex objectives.

It is interesting to note the breaks in tradition in the work of Righini and Salani (2008), Baldacci et al. (2011), and Contardo et al. (2015). The first authors reveal the efficiency of DSSR algorithms for solving the elementary PP, and show the potential of the elementary lower bound. The second authors highlight the power of non-robust cuts and techniques to control the associated increase in complexity of the PP, and give strong evidence towards the focus on CCG algorithms and tightening the duality gap before branching. The last authors review a comprehensive variety of techniques to enforce elementary routes in the solution to the RMP, and the most novel technique is to restrict these routes in the RMP using non-robust cuts.

4.2.2 Heuristics

As a result of the complexity of the VRPTWs, a significant variety of heuristics are proposed in the literature. The earliest of these are constructive and neighbourhood search (NS) heuristics. For further details, Bräysy and Gendreau (2005a) provide a comprehensive survey. Metaheuristic frameworks have since received significant attention, as evidenced by the surveys of Bräysy and Gendreau (2005b), Gendreau and Tarantilis (2010), and Desaulniers et al. (2014), for hard time windows and for both hard and soft time windows in the extensive survey of Vidal et al. (2013a). The evolutionary algorithms paradigm receives particular attention, and dedicated surveys are given by Bräysy et al. (2004) and more recently by Potvin (2009). There is also a notable trend in these surveys towards hybridization of strategies and components of different metaheuristics, and this aims to balance the individual advantages and disadvantages.

Although historically less heuristics are proposed for the VRPSTW directly, the problem is often used in heuristics for the VRPTW by defining weights α and β appropriately. This enables feasible solutions to be found more easily and the search space is less restrictive.

Constructive heuristics

In this section we will describe some classical constructive heuristics for the VRPTW and VRPSTW. For the interested reader, a detailed survey is given by Bräysy and Gendreau (2005a).

Solomon (1987) experiments with a number of constructive heuristics extended to the VRPTW from the CVRP, including the savings, sweep and insertion heuristics. He finds that insertion heuristics appear to perform best on a set of benchmark instances introduced by the author. This may be partially attributed to the complexity of finding a feasible solution using constructive heuristics, and the greater flexibility of insertion heuristics.

Bramel and Simchi-Levi (1996) extend the cluster-first and route-second approach, proposed by Fisher and Jaikumar (1981) for the CVRP, to the VRPTW. An initial customer for each route is found by solving a *capacitated location routing problem with time windows*, and the routes are constructed by inserting the remaining customers in a greedy fashion. The authors report improvements to around 50% of the solutions obtained by Solomon (1987).

One of the earliest heuristics for a VRPSTW is proposed by Koskosidis et al. (1992). This is for type 2 time windows, and extends the cluster-first and route-second approach proposed in Fisher and Jaikumar (1981) for the CVRP. The main aim of this approach is to solve the VRPTW and it improves a number of best known solutions, but violations of the time window are also accepted if the overall objective is minimised.

Balakrishnan (1993) propose three sequential route-building insertion heuristics for the VRPSTW with type 3 time windows. These are based on the nearest neighbour and savings heuristics, and vary the criteria used to initialise routes and select customers for insertion.

A number of authors describe a *route-first and schedule-second* approach, where the schedule of customer arrival times is determined for a given routing. For further details on these scheduling subproblems, the reader is referred to the recent surveys presented by Hashimoto and Yagiura (2008) and Vidal et al. (2015a).

Single solution based heuristics

Many of the earlier implementations of metaheuristics for the VRPTW and VRPSTW rely on single solution based and neighbourhood-centric frameworks. These are often extensions of NS and differ in the techniques used to achieve diversification. Although historically the initial solution had a significant impact on the objective quality of the final solution in many of these heuristics, it is now common to use random or simple constructive procedures and the initial solution has little influence on the overall solution quality. We now discuss some important implementations of single solution based metaheuristics.

Tabu search (TS) is historically popular, with notable TS heuristics for the VRPTW and VRPSTW being proposed by Cordeau et al. (2001b) and Fu et al. (2008), respectively. These heuristics both require a definition of solution attributes to construct the tabu list, and the TS and guided local search (GLS) hybrid algorithm of Cordeau et al. (2001b) also relies on these for defining the guiding penalties. In contrast, the TS algorithm of Fu et al. (2008) uses a strategy of exploring a random set of neighbours from the union of several neighbourhoods in each iteration. Solutions are also represented as a single giant tour with the depot between routes, and although this slightly increases the number of solutions it also results in greater connectivity of the search space. Cordeau et al. (2001b) also increase the connectivity of the search space by relaxing a number of constraints, such as those defined by the vehicle capacities and time windows. The amounts of infeasibility are individually weighted and penalized in the objective function, and are periodically adapted to facilitate controlled exploration of the search space and to adjust the numbers of infeasible solutions. More details on different relaxations for the time windows can be found in the review of Vidal et al. (2015a).

Another elementary framework with some popularity is iterated local search (ILS), and ILS heuristics are proposed by Cordeau and Maischberger (2012) for the VRPTW and by Ibaraki et al. (2008) for the VRPSTW with convex penalties on time-window violations. The

ILS algorithm of Ibaraki et al. (2008) uses an internal variable neighbourhood search (VNS) algorithm that explores a variety of neighbourhoods of increasing size, and Cordeau and Maischberger (2012) use a slightly improved version of the TS and GLS hybrid algorithm of Cordeau et al. (2001b). The differences between these internal heuristics are notable, and contrast the efficient search of a range of neighbourhoods with fewer neighbourhoods and diversification techniques. These differences also require different types of perturbation or kick. Accordingly, Ibaraki et al. (2008) choose a random neighbour from a larger neighbourhood than any used in the VNS algorithm, and Cordeau and Maischberger (2012) use a large neighbourhood (LN) or ruin-and-recreate heuristic. Ibaraki et al. (2008) also describe an efficient route-first and schedule-second evaluation technique in the NS, which decomposes the problem into finding a set of routes and then scheduling the start times of customer services for each route.

Pisinger and Ropke (2007) present an adaptive large neighbourhood search applied to the VRPTW, this iteratively applies separate ruin (removal) and recreate (insertion) operators to an incumbent solution. The generality and flexibility of the framework is notable, the operators to apply at each iteration of the method are chosen via roulette wheel, adaptively biased by the previous performance in memory, and a simulated annealing type acceptance criterion is used. A hybridization of exact and heuristic methods is proposed by Prescott-Gagnon et al. (2009), suggesting a large neighbourhood search with an exact branch-and-price algorithm for the recreation operator.

Notably, Mouthuy et al. (2011) propose a variable neighbourhood descent search for the types 1, 2 and 3 VRPSTW. The neighbourhoods extend the exponential cyclic transfers neighbourhood of Thompson and Orlin (1989), and are applied in a three-phases, first, minimising the number of vehicles, then the total time window violations, and finally the total distance travelled. Finding the best neighbour when the objective is time window violation is shown to be NP-hard by Thompson and Orlin (1989), and in case a heuristic is used, but otherwise these neighbourhoods are search exactly.

Hybrid evolutionary algorithms

Recently, much of the research focus shifts to population-based metaheuristics, which have advantages and drawbacks. Specifically, these methods usually offer greater adaptive control over the balance of diversity and intensity, although they are often observed to converge to good-quality solutions slowly. Hybridization of both single-solution and population-based metaheuristics is often proposed to balance such strengths and weaknesses. Hybrid evolutionary algorithms (HEAs) are particularly popular, and use single solution-based or NS heuristics to improve solutions generated as combinations of solutions stored in a population. These HEAs are the subject of this section and some important contributions are discussed.

In the majority of the HEAs described below, the number of vehicles is fixed and if the objective is hierarchical it is not handled explicitly. If this is the case, then either a route minimization heuristic is applied initially to fix the number of routes as suggested by Nagata et al. (2010), or an iterative approach is used to increase or decrease the number of vehicles depending on whether a feasible solution is found as suggested by Hashimoto and Yagiura (2008). Notice that this increases the similarity between the objective function of the problem solved by these HEAs and the VRPRDD.

A very early HEA is proposed by Taillard et al. (1997) for the type 1 VRPSTW with a fixed number of vehicles, and is described as a hybrid algorithm based on adaptive memory programming (AM), TS and GLS. The heuristic maintains a steady-state and elitist population of solutions, and in each iteration an offspring solution is constructed probabilistically from routes in the population. Each offspring is then improved by the TS and GLS hybrid algorithm before replacing the worst solution in the population if it has better objective value. The authors also suggest decomposing the instance, where the decomposition is based on a current solution, and they describe an iterative decomposition and recomposition (ID) framework using their guided TS to improve the initial population of solutions.

The heuristic of Taillard et al. (1997) can be characterized as a hybrid genetic algorithm (HGA) with a multi-parent crossover, but other studies typically give greater importance to the management and recombination of solutions in the population. For example, the HGA of Berger et al. (2003) for the VRPTW evolves two subpopulations independently, which individually minimize the total distance traveled, and the weighted infeasibility and number of vehicles required. Individual fitness functions are used for parent selection, and the subpopulations interact when a feasible solution with a lower number of vehicles is identified. A similar approach is described by Vidal et al. (2013b) in their HGA for the VRPTW and VRPSTW (see Vidal et al. 2014). Subpopulations for feasible and infeasible solutions are managed independently in their HGA, but the interaction is through selecting solutions for recombination from the union of these subpopulations.

Given the use of heuristics to improve the offspring solutions, the diversity of a population may fall quickly with the solutions becoming more similar. A simple approach which reduces the chances of this is suggested by Nagata et al. (2010) in their HGA for the VRPTW, and only considers replacing parent solutions with their offspring. If a distance between solutions is defined, then the diversity of a solution can be measured by its distance to solutions in the population. In the guided TS $(\mu + \mu)$-evolutionary strategy (ES) suggested promoted by randomly selecting solutions for the next iteration weighted by the distance to other solutions in the population and the objective value. Vidal et al. (2013b) also use a distance between solutions and introduce a measure of diversity into the fitness function, which is used for parent selection and population management. This permits the diversity of the population to be managed adaptively, and if the diversity falls, then more diverse offspring solutions will survive population management and be selected for recombination.

A variety of methods for recombining solutions are proposed, and often these technique vary quite significantly. For example, Hashimoto and Yagiura (2008) propose a path-relinking algorithm for the VRPTW. In the path-relinking (PR) procedure, the solution attributes are defined as the edges and the distance between two solutions as the number of different edges. By iteratively reducing the distance between an initial and a guiding solution using inter- and intra-route neighbourhoods, then a trajectory of offspring solutions is constructed. Interesting, Repoussis et al. (2009) use the entire population to generate a set of offspring using a deterministic approach. Another type of specialized crossover operator is proposed by Nagata et al. (2010) that builds on the powerful edge assembly-based crossover (EAX) proposed for the CVRP and TSP in previous work by the first author. Similar to PR, the EAX has significantly less randomization and more structural consideration than most traditional crossover operators, and produces a set of offspring solutions. In contrast, a traditional

ordered crossover (OX) is used by Vidal et al. (2013b), where solutions are represented as a single giant tour without the depot. The optimal partitioning into routes is then found using an adaptation of the dynamic programming split algorithm presented by Prins (2004), which runs in polynomial time. Uniquely, Mester et al. (2007) propose a $(1 + 1)$-ES that produces solutions from a single parent and can be viewed as an ES guiding an adaptive large neighbourhood search.

Comparison of heuristics

Certain algorithmic strategies and components are frequently used in heuristics for the VRPs considered, and there are combinations of these which seem popular. To observe this, a comparison of the main features of the heuristics is presented in Table 4.2. The columns from left to right are: The algorithm; whether the heuristic can be applied to a version of the VRPSTW (as indicated by a "Y"); the main metaheuristic frameworks; how insert, swap, 2-Opt, 2-Opt* and large neighbourhoods are used; details of NS as defined by the exploration strategy ,whether infeasible solutions are allowed (as indicated by a "Y") and the types of neighbourhood pruning; the type of solution recombination and whether infeasible offspring solutions are produced and how they are treated; and the population management strategy.

From Table 4.2, we observe that all the heuristics feature NS, and except for F08 (Fu et al. 2008) they are hybrids. This suggests that NS is vital in achieving competitive performance, and that a variety of strategies in combination performs best for these complex problems. In the NS components, the exploration strategies are generally balanced between accepting the best-improving or first-improving neighbour. Interestingly, for V13 (Vidal et al. 2013b), this HGA achieves more diversity by exploring neighbours from the union of a number of neighbourhoods in a random order and accepting the first-improving neighbour. Also, for I08 (Ibaraki et al. 2008), their ILS and VNS hybrid algorithm uses an aspiration plus (A+) strategy, in which a neighbourhood is explored until it is complete or the computation time taken approximately exceeds that used for preprocessing. It would appear that the benefits of efficiently searching neighbourhood in each iteration outweigh potential improvements in objective value that may occur by finding the best-improving neighbour.

All of the more recent heuristics use controlled exploration of infeasible solutions, and except N10 (Nagata et al. 2010) they all explore infeasible solutions in the NS component. Alongside the adaptive control of the penalties on infeasibility, considering some infeasible solutions appears to be critical for instances with tight constraints and complex objectives. Note that all of these heuristics relax the time window constraints, and this increases the similarity between the VRPTW and VRPSTW, and therefore the VRPRDD.

A variety of solution recombination approaches are suggested in the HEAs considered, and all except V13 (Vidal et al. 2013b) explicitly consider the structure of solutions more than traditional crossovers. This would suggest that diversity is important in creating offspring solutions, but often the loss of the structural properties of the solution from traditional crossovers obscures the translation of information from the parents to the offspring. It is also notable that both H08 (Hashimoto and Yagiura 2008) and N10 (Nagata et al. 2010) produce a set of offspring from each recombination, and V13 implicitly enumerates the set of offspring resulting from different partitions of the giant tour created by their crossover.

Table 4.2: Features of heuristics for the VRPTW and VRPSTW

	STW	Frameworks	Neighbourhood search								Population		
			Neighbourhoods					Str.	Infeas.	Prun.	Recombination		Management
			Ins.	Swap	2-O	2-O*	LNs				Type	Infeas.	
T97	Y	AM, ID, GLS, TS	+	+			−	B		A	Insertion		Steady-state, elit.
B03		HGA	+				x	F	Y	D	Insertion	Repair	Periodic, elit., tailored fit., subpops
F08	Y	TS	x	x	x	+		B	Y	R	—	—	—
H08		PRA	x	+		+		F	Y	S, D	PR	Accept	Steady-state, elit., unique
I08	Y	ILS, VNS	x′	x′	−	+		A+	Y	S, D	—	—	—
N10		HGA	x′	x′		+		F		D	EAX	Repair	Steady-state, elit., repl. parent
C12		ILS, GLS, TS	+				x	B	Y		—	—	—
V13	Y	ID, HGA	x′	x′	−	+		RF	Y	S, D	OX + split	Copy + rep.	Periodic, diversity in fit., subpops

Note: Algorithms. T97: Taillard et al. (1997); B03: Berger et al. (2003); F08: Fu et al. (2008); H08: Hashimoto and Yagiura (2008); I08: Ibaraki et al. (2008); N10: Nagata et al. (2010);
C12: Cordeau and Maischberger (2012); V13: Vidal et al. (2013b).
Frameworks. AM: Adaptive memory programming; GLS: Guided local search; HGA: Hybrid genetic algorithm; ID: Iterative decomposition-recomposition; ILS: Iterated local search; PRA: Path-relinking algorithm; TS: Tabu search.
Neighbourhoods. Ins: Insert neighbourhood; Swap: Swap neighbourhood; 2-O: 2-Opt; 2-O*: 2-Opt*; LNs: Large neighbourhoods.
Neighbourhood features. −: Intra-route; +: Inter-route; x: Intra- and inter-route; ′: original and reverse subsequences both considered.
Neighbourhood exploration. A+: Aspiration plus; B: Best improvement; F: First improvement; RF: Random first improvement.
Neighbourhood pruning. A: Approximate; D: Dynamic; R: Random; S: Static.
Recombination. EAX: Edge assembly crossover; OX: Order-based crossover; PR: Path-relinking.

The management of the population in earlier HEAs is purely elitist, but different techniques are increasingly introduced to maintain and more carefully introduce diversity to the population. This more recent development appears to be important for achieving current performance levels and particularly for improving the robustness of the results. The adaptive control of population diversity as used in V13 (Vidal et al. 2013b) is a particularly important development, which can enable the population to escape the equivalent of local optima when the diversity of the population falls.

If infeasible offspring solutions are produced by the HEAs, then often a repairing NS is applied that attempts to find a feasible solution whilst retaining objective quality. For V13 (Vidal et al. 2013b), the HGA does not repair solutions automatically but a with a certain probability. In both V13 and B03 (Berger et al. 2003), the HGAs use infeasible solutions more explicitly and a separate subpopulation of infeasible solutions is evolved. These strategies and techniques suggest that infeasible solutions are still important in both the population and NS, although finding feasible solutions quickly can be helpful for problems with tight constraints such as the VRPTW.

Neighbourhoods The neighbourhoods used for the VRPTW are similar to those used for the CVRP. A review and visual representations of the neighbourhoods proposed historically for the VRPTW and VRPSTW are given by Bräysy and Gendreau (2005a). For any NS, the time complexity of evaluating the objective value of a neighbour is critical to the efficiency. For the VRPTW, the difference in objective function for neighbourhoods based on exchanging a bounded number of arcs can be evaluated in $O(1)$ time. Importantly, Kindervater and Savelsbergh (1997) prove that the use of preprocessing requiring $O(n)$ time allows the feasibility of the time windows (and route loads and durations) to be evaluated in $O(1)$ time. Furthermore, Vidal et al. (2013b) show that the use of preprocessing requiring $O(n)$ time permits a popular measure of time window infeasibility termed time-warp to be also evaluated in $O(1)$.

For the types of VRPSTW in which both earliness and tardiness are permitted, the most efficient NS approaches use the route-first and schedule-second technique. This is because even for a given route the schedule of customer service start times is non-trivial. Dynamic programming is used to construct piecewise-linear cost functions in a preprocessing phase, and structures such as heaps or balanced binary trees are used to store and query these efficiently. If only earliness or tardiness is permitted, then the schedule is trivial, but the cost functions are still constructed. For all the types of soft time windows described and neighbourhoods defined by a bounded number of arc exchanges, Ibaraki et al. (2008) show that the use of preprocessing requiring $O(n^2)$ time allows an evaluation that requires $O(\log n)$ time. Further, for any convex piecewise-linear penalty function of earliness and tardiness, these complexities increase to $O(n \sum_i h_i)$ and $O(\log \sum_i h_i)$ respectively, where h_i is the number of segments of the penalty function of customer $i \in V'$. Note that these general techniques are first described independently by Ibaraki et al. (2005), Ergun and Orlin (2006) and Hendel and Sourd (2006) for the VRP with general time windows, and the single machine weighted tardiness and earliness-tardiness scheduling problems respectively. More details on the implementations and other functions on the completion times of activities can be found in the comprehensive review of Vidal et al. (2015b).

A direct comparison of the neighbourhoods used in the heuristics described is complicated by the use of different exploration and pruning strategies. Instead a high-level view is adopted and five underlying neighbourhoods are identified: *insert*, changing the position of a subsequence (of consecutive customers); *swap*, exchanging the positions of two distinct subsequences; *2-Opt*, removing two arcs and reconnecting the associated route segments differently; *2-Opt**, exchanging subsequences at the ends of two routes; and large neighbourhoods, which usually have exponential size and include ruin-and-recreate heuristics. Most of the traditional neighbourhoods proposed are subsets or compositions of subsets of the first four of these neighbourhoods. Although 2-Opt* is the least general and a subset of inter-route 2-Opt, it is included separately due to its apparent importance and frequency of use.

We notice in Table 4.2 that a considerable number of neighbourhoods are frequently used, and this applies even in the HEAs. This again hints at the underlying complexity of the problems, and the difficulty of designing an efficient and effective NS. A greater variety of neighbourhoods are generally used for the VRPSTW, and this may be attributable to the increased complexity and size of the search space. Subsets of the insert and swap neighbourhoods are the most popular, and the use of both inter- and intra-route neighbourhoods is also increasing. The intra-route subset of 2-Opt is used in three of the four heuristics for the VRPSTW, and one of these heuristics is also the most recently proposed of those considered for the VRPTW. Exploring neighbours resulting from considering the subsequences involved in the move in reverse order is also gaining popularity for both insert and swap. Possible reasons for this include instances with wider time windows that allow such a reversal to reduce the objective value, and the additional diversification, especially when used in VNS and first-improvement methods.

LNs are used in three of the heuristics, and a general trend is observed toward larger neighbourhoods. For this reason, considerable efforts are made to achieve efficiency in NS and this is evidenced by the range of pruning techniques that either statically or dynamically restrict the neighbours considered, and the exploration and evaluation strategies that are used. Uniquely, Taillard et al. (1997) propose selecting a set of neighbours in each iteration using an approximate evaluation that requires $O(1)$ time, and Fu et al. (2008) randomly restrict the neighbourhoods. These general issues again reveal the important balance between the greater potential for improving solutions and the efficiency of exploring the neighbourhood.

Bibliography

Augerat, P., Belenguer, J. M., Benavent, E., Corberan, A., Naddef, D., and Rinaldi, G. (1995). Computational results with a branch and cut code for the capacitated vehicle routing problem. Research Report 949-M, Universite Joseph Fourier, Grenoble, France.

Avella, P., Boccia, M., and Vasilyev, I. (2013). Lifted and local reachability cuts for the vehicle routing problem with time windows. *Computers & Operations Research*, 40(8):2004–2010.

Balakrishnan, N. (1993). Simple heuristics for the vehicle routeing problem with soft time windows. *Journal of the Operational Research Society*, pages 279–287.

Baldacci, R., Christofides, N., and Mingozzi, A. (2008). An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming Series A*, 115(2):351–385.

Baldacci, R., Hadjiconstantinou, E., and Mingozzi, A. (2004). An exact algorithm for the capaci-

tated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, 52(5):723–738.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283.

Balinski, M. L. and Quandt, R. E. (1964). On an integer program for a delivery problem. *Operations Research*, 12(2):300–304.

Beasley, J. and Christofides, N. (1989). An algorithm for the resource constrained shortest path problem. *Networks*, 19(4):379–394.

Berger, J., Barkaoui, M., and Bräysy, O. (2003). A route-directed hybrid genetic approach for the vehicle routing problem with time windows. *Proc. INFOR*, 41:179–194.

Bodin, L. and Golden, B. (1981). Classification in vehicle routing and scheduling. *Networks*, 11(2):97–108.

Boland, N., Dethridge, J., and Dumitrescu, I. (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, 34(1):58–68.

Bramel, J. and Simchi-Levi, D. (1996). Probabilistic analyses and practical algorithms for the vehicle routing problem with time windows. *Operations Research*, 44(3):501–509.

Bräysy, O., Dullaert, W., and Gendreau, M. (2004). Evolutionary algorithms for the vehicle routing problem with time windows. *Journal of Heuristics*, pages 587–611.

Bräysy, O. and Gendreau, M. (2005a). Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118.

Bräysy, O. and Gendreau, M. (2005b). Vehicle routing problem with time windows, part II: Metaheuristics. *Transportation Science*, 39(1):119–139.

Chen, S., Golden, B., and Wasil, E. (2007). The split delivery vehicle routing problem: Application, algorithms, test problems, and computational results. *Networks*, 49(0):318–329.

Christofides, N., Mingozzi, A., and Toth, P. (1979). The vehicle routing problem. In Christofides, N., Mingozzi, A., Toth, P., and Sandi, C., editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, UK.

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

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

Contardo, C., Desaulniers, G., and Lessard, F. (2015). Reaching the elementary lower bound in the vehicle routing problem with time windows. *Networks*, 65(1):88–99.

Contardo, C. and Martinelli, R. (2014). A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, 12:129–146.

Cordeau, J.-F., Desaulniers, G., Desrosiers, J., Solomon, M. M., and Soumis, F. (2001a). Vrp with time windows. In Toth, P. and Vigo, D., editors, *The Vehicle Routing Problem*, pages 157–193. SIAM, Philadelphia, USA.

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., Gendreau, M., Laporte, G., Potvin, J.-Y., and Semet, F. (2002). A guide to vehicle routing heuristics. *Journal of the Operational Research society*, pages 512–522.

Cordeau, J.-f., Laporte, G., and Mercier, A. (2001b). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research Society*, 52(8):928–936.

Cordeau, J.-F. and Maischberger, M. (2012). A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*, 39(9):2033–2050.

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

Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, 6(1):80–91.

Desaulniers, G., Desrosiers, J., and Spoorendonk, S. (2010). The vehicle routing problem with time windows: State-of-the-art exact solution methods. In Cochran, J., editor, *Wiley Encyclopedia of Operations Research and Management Science*, pages 5742–5749. Wiley, New York.

Desaulniers, G., Lessard, F., and Hadjar, A. (2008). Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, 42(3):387–404.

Desaulniers, G., Madsen, O., and Ropke, S. (2014). The vehicle routing problem with time windows. In Toth, P. and Vigo, D., editors, *Vehicle Routing: Problems, Methods and Applications*, pages 5742–5749. SIAM, Philadelphia, US.

Desrochers, M., Desrosiers, J., and Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, 40(2):342–354.

Doerner, K., Hartl, R., Kiechle, G., Lucka, M., and Reimann, M. (2004). Parallel ant systems for the capacitated vehicle routing problem. In Gottlieb, J. and Raidl, G., editors, *Evolutionary Computation in Combinatorial Optimization*, pages 72–83. Springer, New York, USA.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in vrptw. *Operations Research*, 42(5):977–978.

Ergun, Ö. and Orlin, J. B. (2006). Fast neighborhood search for the single machine total weighted tardiness problem. *Operations Research Letters*, 34(1):41–45.

Feillet, D., Dejax, P., Gendreau, M., and Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229.

Fisher, M. L. and Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124.

Fu, Z., Eglese, R., and Li, L. Y. O. (2008). A unified tabu search algorithm for vehicle routing problems with soft time windows. *Journal of the Operational Research Society*, 59(5):663–673.

Fukasawa, R., Longo, H., Lysgaard, J., Poggi de Aragao, M., Reis, M., Uchoa, E., and Werneck, R. F. (2006). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming. Series A.*, 106:491–511.

Gaskell, T. (1967). Bases for vehicle fleet scheduling. *Operational Research Quarterly*, pages 281–295.

Gendreau, M., Iori, M., Laporte, G., and Martello, S. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1291–1304.

Gendreau, M. and Tarantilis, C. (2010). Solving large-scale vehicle routing problems with time windows: The state-of-the-art. Tech Rep. 04, CIRRELT.

Gillett, B. E. and Miller, L. R. (1974). A heuristic algorithm for the vehicle-dispatch problem. *Operations research*, 22(2):340–349.

Glover, F. (1992). New ejection chain and laternating path methods for traveling salesman problems. In Balci, O., Sharda, R., and Zenios, S., editors, *Computer Science and Operations Research: New Developments in their Interfaces*, pages 449–509. Pergamon Press, Oxford, UK.

Golden, B. L., Wasil, E. A., Kelly, J. P., and Chao, I. M. (1998). Metaheuristics in vehicle routing. In Crainic, T. G. and Laporte, G., editors, *Fleet Management and Logistics*, pages 33–56. Springer, New York, USA.

Gouveia, L. (1995). A result on projection for the vehicle routing ptoblem. *European Journal of Operational Research*, 85(3):610–624.

Gröer, C., Golden, B., and Wasil, E. (2011). A parallel algorithm for the vehicle routing problem. *INFORMS Journal on Computing*, 23(2):315–330.

Hashimoto, H. and Yagiura, M. (2008). A path relinking approach with an adaptive mechanism to control parameters for the vehicle routing problem with time windows. In Hemert, J. and Cotta, C., editors, *Evolutionary Computation in Combinatorial Optimization*, pages 254–265. Springer, New York, USA.

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

Hendel, Y. and Sourd, F. (2006). Efficient neighborhood search for the one-machine earlinesstardiness scheduling problem. *European Journal of Operational Research*, 173(1):108–119.

Ho, S. and Gendreau, M. (2006). Path relinking for the vehicle routing problem. *Journal of Heuristics*, 12(1-2):55–72.

Ibaraki, T., Imahori, S., Kubo, M., Masuda, T., Uno, T., and Yagiura, M. (2005). Effective local search algorithms for routing and scheduling problems with general time-window constraints. *Transportation Science*, 39(2):206–232.

Ibaraki, T., Imahori, S., Nonobe, K., Sobue, K., Uno, T., and Yagiura, M. (2008). An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. *Discrete Applied Mathematics*, 156(11):2050 – 2069.

Ioachim, I., Gelinas, S., Soumis, F., and Desrosiers, J. (1998). A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, 31(3):193–204.

Irnich, S. and Villeneuve, D. (2006). The shortest-path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing*, 18(3):391–406.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511.

Kallehauge, B., Boland, N., and Madsen, O. G. B. (2007). Path inequalities for the vehicle roting problem with time windows. *Networks*, 49(4):273–293.

Kilby, P., Prosser, P., and Shaw, P. (1999). Guided local search for the vehicle routing problem with time windows. In Vo, S., Martello, S., Osman, I., and Roucairol, C., editors, *Meta-Heuristics*, pages 473–486. Springer, New York, USA.

Kindervater, G. and Savelsbergh, M. (1997). Vehicle routing: Handling edge exchanges. In Aarts, E. and Lenstra, J. K., editors, *Local Search in Combinatorial Optimisation*, pages 337–360. Wiley.

Kohl, N., Desrosiers, J., Madsen, O. B., Solomon, M. M., and Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116.

Koskosidis, Y. A., Powell, W. B., and Solomon, M. M. (1992). An optimization-based heuristic for vehicle routing and scheduling with soft time window constraints. *Transportation science*, 26(2):69–85.

Kytöjoki, J., Nuortio, T., Bräysy, O., and Gendreau, M. (2007). An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research*, 34(9):2743 – 2757.

Laporte, G. (2009). Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416.

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

Laporte, G., Ropke, S., and Vidal, T. (2014). Heuristics for the vehicle routing problem. In Toth, P. and Vigo, D., editors, *Vehicle Routing: Problems, Methods and Applications*, pages 87–116. SIAM, Philadelphia, US.

Laporte, G. and Semet, F. (2002). Classical heuristics for the capacitated VRP. In Toth, P. and Vigo, D., editors, *The vehicle routing problem*, pages 101–117. SIAM, Philadelphia, USA.

Letchford, A. and Salazar-González, J.-J. (2015). Stronger multi-commodity flow formulations of the capcitated vehicle routing problem. *European Journal of Operational Research*, 244(3):730–738.

Letchford, A. N. and Salazar-González, J.-J. (2006). Projection results for vehicle routing. *Mathematical Programming Series B*, 105(2-3):251–274.

Li, F., Golden, B., and Wasil, E. (2005). Very large-scale vehicle routing: New test problems, algorithms, and results. *Computers & Operations Research*, 32(0):1165–1179.

Liberatore, F., Righini, G., and Salani, M. (2010). A column generation algorithm for the vehicle routing problem with soft time windows. *4OR*, 9(1):49–82.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516.

Lysgaard, J. (2006). Reachability cuts for the vehicle routing problem with time windows. *European Journal of Operational Research*, 175(1):210–223.

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.

Marinakis, Y. and Marinaki, M. (2010a). Bumble bees mating optimization algorithm for the vehicle routing problem. In Panigrahi, B. K., Shi, Y., and Lim, M.-H., editors, *Handbook of Swarm Intelligence*, pages 347–369. Springer, New York, USA.

Marinakis, Y. and Marinaki, M. (2010b). A hybrid genetic - particle swarm optimisation algorithm for the vehicle routing problem. *Expert Systems with Applications*, 37(0):1446 – 1455.

Martinelli, R., Pecin, D., and Poggi, M. (2014). Efficient elementary and restricted non-elementary route pricing. *European Journal of Operational Research*, 239(1):102–111.

Mester, D., Bräysy, O., and Dullaert, W. (2007). A multi-parametric evolution strategies algorithm for vehicle routing problems. *Exepert Systems with Applications*, 32(0):508 – 517.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):393–410.

Mole, R. H. and Jameson, S. R. (1976). A sequential route-building algorithm employing a generalised savings criterion. *Operational Research Quarterly*, 27(2):503–511.

Mouthuy, S., Massen, F., Deville, Y., and Van Hentenryck, P. (2011). A multi-stage very large-scale neighborhood search for the vehicle routing problem with soft time-windows. In *Proceedings of the 9th Metaheuristics International Conference (MIC2011)*.

Naddef, D. and Rinaldi, G. (2002). Branch-and-cut algorithms for the capacitated VRP. In Toth, P. and Vigo, D., editors, *The vehicle routing problem*, pages 53–81. SIAM, Philadelphia, USA.

Nagata, Y. and Bräysy, O. (2008). Efficient local search limitation strategies for vehicle routing problems. *Evolutionary Computation in Combinatorial Optimisation*, 4972:48–60.

Nagata, Y., Bräysy, O., and Dullaert, W. (2010). A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, 37(4):724–737.

Newton, R. M. and Thomas, W. H. (1974). Bus routing in a multi-school system. *Computers & Operations Research*, 1(2):213–222.

Or, I. (1976). *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking*. PhD thesis, Northwestern University,, Illinois, USA.

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, 41(4):421–451.

Pecin, D. (2014). Exact algorithms for the capacitated vehicle routing problem. PhD Thesis, Pontifícia Universidade Católica do Rio de Janiero, Rio de Janiero, Brazil.

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 Toth, P. and Vigo, D., editors, *Vehicle Routing: Problems, Methods and Applications*, pages 59–86. SIAM, Philadelphia, US.

Poggi de Aragao, M. and Uchoa, E. (2003). Integer programming reformulation for robust branch-and-cut-and-price. In Wolsey, L., editor, *Annals of Mathematical Programming in Rio: A conference in honour of Nelson Maculan*, pages 56–61.

Potvin, J.-Y. (2009). State-of-the art review: Evolutionary algorithms for vehicle routing. *INFORMS Journal on Computing*, 21(4):518–548.

Potvin, J.-Y. and Rousseau, J.-M. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340.

Prescott-Gagnon, E., Desaulniers, G., and Rousseau, L. (2009). A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, 54(4):190–204.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(0):1985 – 2002.

Qureshi, A., Taniguchi, E., and Yamada, T. (2009). An exact solution approach for vehicle routing and scheduling problems with soft time windows. *Transportation Research Part E*, 45(6):960–977.

Rego, C. and Roucairol, C. (1996). A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In *Meta-Heuristics*, pages 661–675. Springer US.

Reimann, M., Doerner, K., and Hartl, R. F. (2004). D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, 31(4):563–591.

Repoussis, P. P., Tarantilis, C. D., and Ioannou, G. (2009). Arc-guided evolutionary algorithm for the vehicle routing problem with time windows. *IEEE Transactions on Evolutionary Computation*, 13(3):624–647.

Righini, G. and Salani, M. (2006). Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273.

Righini, G. and Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51(3):155–170.

Savelsbergh, M. W. (1985). Local search in routing problems with time windows. *Annals of Operations Research*, 4(1):285–305.

Schrage, L. (1981). Formulation and structure of more complex/realistic routing and scheduling problems. *Networks*, 11(2):229–232.

Semet, F., Toth, P., and Vigo, D. (2014). Classical exact algorithms for the capacitated vehicle routing problem. In Toth, P. and Vigo, D., editors, *Vehicle Routing: Problems, Methods and Applications*, pages 37–58. SIAM, Philadelphia, US.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In Maher, M. and Puget, J.-F., editors, *Principles and Practice of Constraint Programming CP98*, pages 417–431. Springer, New York, USA.

Solomon, M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265.

Sörensen, K. and Schittekat, P. (2013). Statistical analysis of distance-based path relinking for the capacitated vehicle routing problem. *Computers & Operations Research*, 40(12):3197–3205.

Sörensen, K. and Sevaux, M. (2006). Ma— pm: memetic algorithms with population management. *Computers & Operations Research*, 33(5):1214–1225.

Tagmouti, M., Gendreau, M., and Potvin, J.-Y. (2007). Arc routing problems with time-dependent service costs. *European Journal of Operational Research*, 181(1):30–39.

Taillard, E., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J. Y. (1997). A new neighbourhood structure for the vehicle routing problems with time windows. Tech Rep. CRT-95-66, Centre du researche sur les transportes, Université Montréal.

Thompson, P. M. and Orlin, J. B. (1989). The theory of cyclic transfers. Operations Research Centre Working Paper, MIT.

Thompson, P. M. and Psaraftis, H. N. (1993). Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations research*, 41(5):935–946.

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. (2013a). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231(1):1 – 21.

Vidal, T., Crainic, T.-G., Gendreau, M., and Prins, C. (2013b). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475 – 489.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2014). A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658–673.

Vidal, T., Crainic, T.-G., Gendreau, M., and Prins, C. (2015a). Time-window relaxations in vehicle routing heuristics. *Journal of Heuristics*, 21(3):329–358.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2015b). Timing problems and algorithms: Time decisions for sequences of activities. *Networks*, 65(2):102–128.

Zachariadis, E. E. and Kiranoudis, C. T. (2010). A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem. *Computers & Operations Research*, 37(12):2089–2105.

Chapter 5

Mixed integer programming formulations

5.1 Introduction

Solving vehicle routing problems (VRPs) to optimality has significant interest and this is evidenced by the significant number of papers published on the subject. For examples, we refer the reader to the chapters by Semet et al. (2014) and Desaulniers et al. (2014). The inherent complexity of these problems causes considerable difficulties, and therefore a variety of MIP formulation have been proposed. Often the trade-off in these formulations and the difference between their performance is associated to the complexity of the formulation (or time taken to solve the linear relaxation), and the strength of the lower bound provided.

The most successful formulations for the capacitated VRP (CVRP) are currently based on set-partitioning (e.g., Pecin 2014, Baldacci et al. 2011) or two-index formulations (e.g., Lysgaard et al. 2004). For the VRP with time windows (VRPTW), only set-partitioning formulations have received considerable attention (e.g., Baldacci et al. 2011, Contardo et al. 2015). Unfortunately, these formulations have an exponential number of variables or constraints, respectively, and cut or column generation must be used to solve the linear relaxation efficiently.

We propose extensions of two classical polynomial-size formulations of the CVRP to the VRP with release and due dates (VRPRDD). The first formulation will be described as the m-commodity formulation, and is suggested by Letchford and Salazar-González (2006). The second is suggested by Gouveia (1995) and is usually described as a one-commodity flow formulation. We formulate the variables and constraint necessary to model the departure time of a route as a commodity flow on the arcs, and tighten some constraints for the m-commodity flow formulations. The formulations are generally related by aggregation or disaggregation of the variables, although we show that some different constraints for the route departure times can be formulated if variables for each vehicle are used.

To compare the formulations, the performance of a commercial branch and cut algorithm is evaluated on the proposed benchmark instances with $n = 20$ or 30. The performance is compared in terms of the best lower bound, and number of optimal solutions found and proved in the time limit. The computational results confirm the complexity of the problem, and only small instances with loose due dates and similar release dates can be solved in the

time limit. The results also suggest that the one-commodity flow formulation has superior performance, but this is less evident when the scheduling aspects of the problem have greater influence.

The structure of the chapter is organised as follows. The formulations of the VRPRDD are presented and discussed in § 5.2. The requirement of a bound on the arrivals times and some techniques to calculate this are presented in § 5.3. The results of the computational experiments using a commercial branch and cut algorithm are given in § 5.4. Lastly, § 5.5 concludes our findings.

5.2 Formulations

In this section, we extend the m- and one-commodity flow formulations of the CVRP to include release and due dates. The one-commodity flow variables are the aggregation of the variables of the m-commodity flow formulation, but route indices for the variables enable an additional set of constraints to be formulated. Both formulations proposed require an upper bound on the latest arrival time back to the depot, T, or more specifically upper bounds on the arrival time at $j \in V$ after traversing (i, j), T_{ij}. For simplicity, let $r_{ij} = \max\{r_i, r_j\}$ where $r_0 = 0$, $r^+ = \max_{i \in V'}\{r_i\}$, $V_i = V \backslash \{i\}$, and $A' = A \backslash \{(0, i), (i, 0) \in A : i \in V'\}$.

5.2.1 m-commodity flow formulation

Modelling the *m-commodity flow formulation* of the CVRP requires $O(n^2m)$ binary variables $x_{ij}^k \in \{0, 1\}$ that are equal to one if route k traverses $(i, j) \in A$, and $O(n^2m)$ variables $f_{ij}^k \in \mathbb{R}^+$ that represent the cumulative demand of the customers in route $k \in K$ that are visited before (i, j) is traversed, and if $x_{ij}^k = 0$ then $f_{ij}^k = 0$. To model the VRPRDD, we also define $O(n^2m)$ variables $u_{ij}^k \in \mathbb{R}^+$ and $v_{ij}^k \in \mathbb{R}^+$, representing the time that route k finishes traversing $(i, j) \in A$ and the departure time of route k if it traverses $(i, j) \in A$, respectively, and if $x_{ij}^k = 0$ then $u_{ij}^k = v_{ij}^k = 0$. Lastly, we define $O(n)$ variables $z_i \in \mathbb{R}^+$ for tardiness at customer $i \in V'$. Let $\tau_{00} = 0$, $q_0 = -\sum_{i \in V'} q_i$, and $k(V')$ be a lower bound on the number of vehicles, such as $\lceil q(V')/Q \rceil$. We will denote this formulation as MC, and it is defined as

follows.

$$z(\text{MC}) = \min \left\{ \alpha \sum_{(i,j)\in A} c_{ij} \sum_{k\in K} x_{ij}^k + (1-\alpha) \sum_{i\in V'} w_i z_i \right\} \tag{5.1}$$

s.t.

$$\sum_{j\in V_i} \sum_{k\in K} x_{ij}^k = 1, \qquad \forall\, i \in V', \tag{5.2}$$

$$\tag{5.3}$$

$$\sum_{j\in V'} x_{0j}^0 = 1, \tag{5.4}$$

$$\sum_{j\in V'} x_{0j}^k \leq 1, \qquad \forall\, k \in K, k > 0 \tag{5.5}$$

$$\sum_{j\in V'} \sum_{k\in K} x_{0j}^k \geq k(V'), \tag{5.6}$$

$$\sum_{j\in V_i} x_{ij}^k - x_{ji}^k = 0, \qquad \forall\, i \in V, k \in K, \tag{5.7}$$

$$\sum_{j\in V\setminus\{i\}} (f_{ij}^k - f_{ji}^k - q_i x_{ij}^k) = 0, \qquad \forall\, i \in V, k \in K, \tag{5.8}$$

$$q_i^+ x_{ij}^k \leq f_{ij}^k \leq (Q - q_i^+) x_{ij}^k, \qquad \forall (i,j) \in A, k \in K, \tag{5.9}$$

$$\sum_{j\in V\setminus\{i\}} (v_{ij}^k - v_{ji}^k) = 0, \qquad \forall\, i \in V, k \in K, k > 0, \tag{5.10}$$

$$r_{ij} x_{ij}^k \leq v_{ij}^k \leq r_{\max} x_{ij}^k, \qquad \forall (i,j) \in A, k \in K, k > 0, \tag{5.11}$$

$$\sum_{j\in V\setminus\{i\}} (u_{ij}^k - u_{ji}^k - \tau_{ij} x_{ij}^k) = 0, \qquad \forall\, i \in V, k \in K, \tag{5.12}$$

$$(r_{\max} + \tau_{0i} + \tau_{ij}) x_{ij}^0 \leq u_{ij}^0 \leq T_{ij} x_{ij}^0, \qquad \forall (i,j) \in A, \tag{5.13}$$

$$v_{ij}^k + (\tau_{0i} + \tau_{ij}) x_{ij}^k \leq u_{ij}^k \leq T_{ij} x_{ij}^k, \qquad \forall (i,j) \in A, k \in K, k > 0, \tag{5.14}$$

$$\sum_{j\in V\setminus\{i\}} \sum_{k\in K} u_{ji}^k - z_i \leq d_i, \qquad \forall\, i \in V', \tag{5.15}$$

$$z_i \geq 0, \qquad \forall\, i \in V', \tag{5.16}$$

$$x_{ij}^k \in \{0,1\}, \qquad \forall (i,j) \in A, k \in K. \tag{5.17}$$

Clearly the formulation has both $O(n^2 m)$ columns and rows, and (3.1) calculates the objective function. Constraints (5.2)-(5.8) and (5.17) define the m-commodity flow formulation of the CVRP. Constraints (5.10) ensure that the departure time from the depot of every route $k \in K : k > 0$ is consistent, and constraints (5.12) ensure that the arrival times at customers are consistent. Constraints (5.13) and (5.14) bound the u-variables and relate them to the v-variables. Constraints (5.4) and (5.13) require route 0 to be used and depart from the depot at $r_{\max}$, this is valid because at least one route must be used and one route must depart at this time. Constraints (5.15) ensure that the tardiness for each customer is consistent, and constraints (5.16) lower bound the z-variables. Lastly, (5.17) give the binary requirement on the x-variables.

5.2.2 One-commodity flow formulation

A tight *one-commodity flow* formulation of the CVRP is suggested by Gouveia (1995). An extension of this formulation for the VRPRDD is achieved if we aggregate each set of variables in MC except the z-variables. This leads to $O(n^2)$ of each type of variable, for example, we now have $x_{ij} \in \{0,1\}$ for $(i,j) \in A$, equal to one if $(i,j) \in A$ is traversed by any route, and $f_{ij} \in \mathbb{R}^+$ for $(i,j) \in A$, representing the cumulative demand of customers that are visited in the route that traverses $(i,j) \in A$ before the arc is traversed. The formulation, which we call OC, can then be stated as follows.

$$z(\text{OC}) = \min \left\{ \alpha \sum_{(i,j)\in A} c_{ij}x_{ij} + (1-\alpha) \sum_{i\in V'} w_i z_i \right\} \tag{5.18}$$

s.t.

$$\sum_{j\in V\setminus\{i\}} x_{ij} = 1, \qquad \forall\, i \in V', \tag{5.19}$$

$$\sum_{j\in V\setminus\{i\}} x_{ji} = 1, \qquad \forall\, i \in V', \tag{5.20}$$

$$k(V') \le \sum_{j\in V'} x_{0j} = \sum_{j\in V'} x_{j0} \le m, \tag{5.21}$$

$$\sum_{j\in V\setminus\{i\}} (f_{ij} - f_{ji}) = q_i, \qquad \forall\, i \in V, \tag{5.22}$$

$$q_i^+ x_{ij} \le f_{ij} \le (Q - q_i^+)x_{ij}, \qquad \forall (i,j) \in A, \tag{5.23}$$

$$\sum_{j\in V\setminus\{i\}} (v_{ij} - v_{ji}) = 0, \qquad \forall\, i \in V, \tag{5.24}$$

$$r_{ij}x_{ij} \le v_{ij} \le r_{\max}x_{ij}, \qquad \forall (i,j) \in A, \tag{5.25}$$

$$\sum_{j\in V\setminus\{i\}} (u_{ij} - u_{ji} - \tau_{ij}x_{ij}) = 0, \qquad \forall\, i \in V, \tag{5.26}$$

$$v_{ij} + (\tau_{0i} + \tau_{ij})x_{ij} \le u_{ij} \le T_{ij}x_{ij}, \qquad \forall (i,j) \in A, \tag{5.27}$$

$$\sum_{j\in V\setminus\{i\}} u_{ji} - z_i \le d_i, \qquad \forall\, i \in V', \tag{5.28}$$

$$z_i \ge 0, \qquad \forall\, i \in V', \tag{5.29}$$

$$x_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A. \tag{5.30}$$

The formulation has both $O(n^2)$ columns and rows, and the objective function is equivalent to that of MC. Constraints (5.19)-(5.22), (5.23) and (5.30) represent the one-commodity flow formulation of the CVRP. Constraints (5.24), (5.26), and (5.28) ensure that the departure time of a route, arrival times, and tardiness are consistent, respectively. Constraints (5.27) and (5.25) bound and link the u- and v-variables. Notice that constraints (5.13) from MC cannot be formulated without the k index.

A second commodity could be defined for the f-, u- and v-variables, suggested for the f-variables in the two-commodity flow formulation of the CVRP presented by Baldacci et al. (2004). Although Letchford and Salazar-González (2006) have shown by projection that the two-commodity flow formulation of the CVRP gives the same lower bound as the one-commodity flow formulation. These authors also point out that this result does not hold for

defining two commodities for the arrival time in commodity flow formulations of the VRPTW. The result would seem to hold for the commodity variables in the VRPRDD, because there is no idle time or other unknown resource consumptions across the arcs that would be tightened.

5.3 Bounding the arrival times

The tightness of the T_{ij} values used in bounding the u-variables, directly influences the lower bound and the integrality gap for the x-variables when solving the linear relaxation of these formulations. This is clear if we consider that for any $x_{ij} > 0$ as T_ij increases then the upper bound on u_{ij} increases. Note that these bounds are also important for reducing the number of variables in *time-indexed* formulations.

It is easy to verify that $T_{0i} \leq r_{\max} + \tau_{0i}$ for $i \in V'$, but finding tight bounds for the T_{ij} values or even T, appears to be a much more difficult problem. Notice that calculating this bound can be formulated as an optimisation problem, which we will call the capacitated elementary latest path problem with release dates (CELPPRD). This problem requires finding one of the latest completing routes in G such that the capacity is not exceeded. The CELPPRD is unary NP-hard, and this is proven by a trivial reduction from the longest path problem.

The ECLPPRD can be represented by an MIP formulation involving $O(n^2)$ variables $x_{ij} \in \{0, 1\}$, equal to one if $(i, j) \in A$ is traversed by the route, and $O(n)$ variables $y_i \in \{0, 1\}$, equal to one if $i \in V'$ has the latest release date amongst customers visited. We also require the f-variables with the same definition. The formulation can then be stated as follows, where we set $q_0 = -\sum_{i \in V'} q_i$.

$$z(\text{LF}) = \max \sum_{i \in V'} r_i y_i + \sum_{(i,j) \in A} \tau_{ij} x_{ij} \tag{5.31}$$

s.t.

$$\sum_{j \in V_i} x_{ij} \leq 1, \qquad \forall\, i \in V', \tag{5.32}$$

$$\sum_{j \in V'} x_{0j} = 1, \tag{5.33}$$

$$\sum_{j \in V_i} x_{ij} - x_{ji} = 0, \qquad \forall\, i \in V, \tag{5.34}$$

$$y_i \leq \sum_{j \in V_i} x_{ij}, \qquad \forall\, i \in V', \tag{5.35}$$

$$\sum_{i \in V'} y_i = 1, \tag{5.36}$$

$$\sum_{j \in V_i} f_{ij} - f_{ji} = q_i, \qquad \forall\, i \in V, \tag{5.37}$$

$$q_i^+ x_{ij} \leq f_{ij} \leq (Q - q_i^+) x_{ij}, \qquad \forall (i, j) \in A, \tag{5.38}$$

$$y_i \in \{0, 1\}, \qquad \forall\, i \in V', \tag{5.39}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall (i, j) \in A. \tag{5.40}$$

The objective function maximises the time that the path terminates at a. Constraints (5.32) ensure that each customer is visited at most once, and (5.33) that the path originates and

terminates at the depot. Constraints (5.34) ensure that each customer $i \in V_a'$ is entered and exited the same number of times. Constraints (5.35) link the x- and y-variables, and constraint (5.36) ensures that only one release date is chosen as the departure time. Constraints (5.37) and (5.38) ensure consistent values for and bound the f-variables respectively. Lastly, (5.39) and (5.40) give the domains of the x- and y-variables. Notice that if the x-variables are integer then integer values for the y-variables are implicit.

The problem represented by LF can be relaxed to a modified transportation problem by replacing constraints (5.37) and (5.38) with the following constraint.

$$\sum_{i \in V'} q_i \sum_{j \in V_i} x_{ij} \leq Q. \tag{5.41}$$

Th resulting maximisation transportation problem includes an additional profit that can be collected from one of the destinations satisfied, and has a resource constraint on the transportations. The solution to this relaxed problem may contain subtours, and comprises a set of disjoint cycles and a disjoint route.

Other more trivial approaches require upper bounds on the travel time to each customer, for example, $\tau_i^+ = \max_{(j,i) \in A}\{\tau_{ji}\}$ for $i \in V$. A bound can be found in linear time if we solve a linear knapsack problem, where we define an item for each customer $i \in V'$ and the profit and weight are τ_i^+ and q_i, respectively.

5.4 Computational Results

In this section, we present a comparison of the performance of the proposed formulations on the benchmark instances with 20 or 30 customers (§ 5.4.1), and a detailed analysis of results for the better performing formulation independently (§ 5.4.2). These experiments seek to evaluate the formulations, in terms of the strength of the lower bound and number times the optimal solutions can be found in the time limit.

The formulations are implemented and solved using CPLEX 12.6.1 (Concert C++ API). The deterministic parallel branch and cut algorithm of CPLEX with default settings is used to attempt to solve the formulations, and the computational time limit is set at four hours. The results are taken from a single run for each instance, and all the experiments are performed using four cores of an Intel Xeon E5–2670 2.6GHz processor with 16GB of RAM (running Linux Red Hat Enterprise Server release 6.3). This number of cores seems justifiable considering the increasing prevalence of desktop computers that can accommodate this many threads. Note that the computational time reported is the elapsed time rather than the cumulative time the cores are processing. The upper bounds for each instance are taken from a single run of the path-relinking algorithm of Chapter 6 with a three minute computational time limit.

5.4.1 Comparison of results for the formulations

The two formulation presented differ in the number of variables and constraints by approximately a factor of m, although the MC formulation also allows constraints (5.13) to be formulated. To gain a greater appreciation of the difference between the formulations, we compare the results on the benchmark instances for both formulations.

In Table 5.1, we present the average relative gap between the best lower bound found and the best upper bound, and the percentage of instances for which optimal solutions are identified. Note that for a number of instances the RAM became full within the computational time limit, and this required setting CPLEX to use disk space to store nodes from the BC algorithm. These instances have tighter scheduling aspects, such as a greater spread of release date or tighter due dates, and are fully detailed in Chapter A.

Table 5.1: Results for different formulations.

			Gap (%)		Opt (%)	
α	n	m	MC	OC	MC	OC
0.3	20	2	38.65	32.39	25.0	25.0
		3	27.25	15.27	25.0	66.7
	30	3	36.49	32.67	0.0	8.3
		4	27.48	20.08	0.0	8.3
0.5	20	2	47.47	44.62	0.0	8.3
		3	34.78	27.49	0.0	8.3
	30	3	16.87	11.52	16.7	41.7
		4	14.83	4.99	8.3	66.7
0.7	20	2	26.57	23.94	16.7	25.0
		3	20.55	12.93	8.3	58.3
	30	3	25.84	21.52	0.0	8.3
		4	20.54	13.63	0.0	8.3
Av.			28.11	21.75	8.3	27.8

From this comparison, it is clear that the OC formulation achieves a considerably tighter gap on average across all the instances. This formulations and the associated lack of symmetry appears particularly effective for higher values of α and m, representing greater emphasis on the total distance objective and a greater number of vehicles. Equivalently, the MC formulation is more effective when total weighted tardiness objective has more weight and there are less vehicles available. Some of the increased effectiveness of the MC formulation may be attributable to the addition of constraints (5.13), which tighten the departure time for one of the routes.

The results are consistent across the different values of n, but for $n = 20$ the gap is generally a little higher. The allocation of customers to routes is more difficult in these instances, because the capacity constraint is more restrictive. The tighter lower bound from the OC formulation also results in a greater number of instances being solved within the computational time limit and lower computational times. It is notable that the upper bounds from the path-relinking algorithm are optimal for all of the instances that are solved to optimality, and no integer solutions are found that improve the upper bounds.

5.4.2 Detailed results for the OC formulation

To assess the strength of these formulations across the different instances, considering the novel aspects of the problem in more detail, we present an in-depth analysis of the OC formulation results. In Table 5.2, we present the average relative gap between the best lower bound found and the best upper bound, and average computational time in seconds. The results are taken from the single run on each instance, and to analyse the effects for the more novel aspects of the problem, we have averaged the results for all values of n and m.

Table 5.2: Results for OC formulation

		Gap (%)			Time (s)		
b	k	$\alpha = 0.3$	0.5	0.7	0.3	0.5	0.7
0.25	4	31.45	36.18	25.38	12823.0	14400.1	14400.5
	6	4.67	8.11	4.90	7828.2	7632.8	7730.7
	8	0.00	0.00	0.00	291.0	332.7	342.2
0.50	4	48.26	42.22	39.17	14400.9	14400.5	14401.7
	6	14.67	13.40	7.43	10881.2	8792.1	10857.9
	8	3.37	4.17	2.20	7367.4	7315.2	7338.1
0.75	4	55.12	48.31	39.89	14404.1	14402.6	14400.3
	6	25.10	20.82	19.87	11441.6	11850.6	12216.3
	8	12.23	5.42	8.78	10890.4	10465.7	10816.4
1.00	4	57.16	48.78	42.47	14404.8	14401.6	14402.8
	6	28.64	23.50	14.47	14400.2	12150.3	14400.3
	8	20.55	14.92	11.52	10855.2	11332.4	11129.5
Av.		25.10	22.15	18.01	10832.3	10623.0	11036.4

The results show varied performance, and this supports the suggestion of a varied set of instances from the previous chapter. Notably, all the instances with $b = 0.25$ and $k = 8$ are solved. However, it is clear that the gap is considerably larger for instances with higher values of b and lower values of k, representing a greater spread of release dates and tighter due dates, respectively. The difficulty of assigning customers to routes and scheduling the visits increases in these instances, and this appears to weaken the lower bound.

Tighter due dates, which are represented by lower values of k, can cause the weighted tardiness to become larger. This appears to have a considerable effect on the computational time and gap for these formulations, and even instances with low values for b still require considerable computational time. It is easy to imagine that these instances have a considerable number of fractional variables in the linear relaxations, to reduce the total weighted tardiness.

These insights suggest that the majority of instances of the VRPRDD are not easily handled using arc-based and polynomial-size MIP formulations. When the scheduling aspects of the problem are more important, particularly tighter due dates, then the lower bound becomes considerably weaker and the time taken to solve the problem or increase the lower bound grows drastically.

5.5 Conclusion

In this chapter, we propose the first two MIP formulations of the vehicle routing problem with release and due dates (VRPRDD). These formulations extend well-known commodity flow formulations of the capacitated vehicle routing problem (CVRP), defining either a single or m commodities for each resource, where m is the number of vehicles. This requires formulating variables and constraints to model the departure time of a vehicle as a commodity flow on the arcs. We also reveal an additional type of constraint for the vehicle departure time that can be modelled in the m-commodity formulation. A thorough computational evaluation and comparison of the formulations is performed, using the branch and cut algorithm of CPLEX.

Observing the results, it is clear that the VRPRDD represents a complex problem, and a

considerable number of the instances cannot be solved using relatively large computational time and power. This also supports the conclusions from the previous chapter, affirming the significant effects associated with introducing release dates and weighted tardiness into a vehicle routing problem (VRP). The formulation with one commodity for each resource has generally superior performance, and this is attributed to less symmetry in the formulation. It is notable that the performance is comparatively worse for instances where the scheduling aspects of the problem are tighter. It is also notable that no solutions are identified that improve the upper bounds, and therefore all instances with proven optimal solutions are solved to optimality by the path-relinking algorithm of Chapter 6.

The amount of computational time, and in some cases memory, required to solve these instances suggests that these formulations have limited applicability. As a future research direction, it may be interesting to extend the one-commodity flow formulation to an arc-time-indexed formulation. This is expected to considerably improve the duality gap by addressing many of the deficiencies of the proposed formulations in respect to the scheduling aspects of the problem, although it requires a pseudopolynomial number of binary variables and the memory requirements may be too restrictive to solve practical instances. To improve the lower bound from the m-commodity formulation, symmetry breaking constraints can be added to distinguish the binary variables associated to each vehicle, for example, requiring that the index of the first customer on each route is smaller than the next. Although it not clear how these constraints will impact the performance of the branch and cut algorithm.

The results also support the growing interest in efficient decomposition techniques, such as column generation, to solve more complex VRPs, and this is considered in the following chapter. The duality gap may also be reduced by increasing the number of constraints in the formulation. Although cuts are added by the CPLEX branch and cut algorithm, many of these are not specialised for the type of constraints and variables in the formulations. Notice also that many valid inequalities for the CVRP are still valid for the VRPRDD. In summary, the insights yielded from the analyses presented here should prove helpful in the future for designing formulations and exact algorithms for this and similar problems.

Bibliography

Baldacci, R., Hadjiconstantinou, E., and Mingozzi, A. (2004). An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations research*, 52(5):723–738.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations research*, 59(5):1269–1283.

Contardo, C., Desaulniers, G., and Lessard, F. (2015). Reaching the elementary lower bound in the vehicle routing problem with time windows. *Networks*, 65(1):88–99.

Desaulniers, G., Madsen, O., and Ropke, S. (2014). The vehicle routing problem with time windows. In Toth, P. and Vigo, D., editors, *Vehicle Routing: Problems, Methods and Applications*, pages 119–160. SIAM, Philadelphia, US.

Gouveia, L. (1995). A result on projection for the vehicle routing ptoblem. *European Journal of Operational Research*, 85(3):610–624.

Letchford, A. N. and Salazar-González, J.-J. (2006). Projection results for vehicle routing. *Mathematical Programming*, 105(2):251–274.

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.

Pecin, D. (2014). Exact algorithms for the capacitated vehicle routing problem. PhD Thesis, Pontifícia Universidade Católica do Rio de Janiero, Rio de Janiero, Brazil.

Semet, F., Toth, P., and Vigo, D. (2014). Classical exact algorithms for the capacitated vehicle routing problem. In Toth, P. and Vigo, D., editors, *Vehicle Routing: Problems, Methods and Applications*, pages 37–58. SIAM, Philadelphia, US.

Chapter 6

A path-relinking algorithm

6.1 Introduction

Finding good-quality solutions and upper bounds efficiently for vehicle routing problems (VRPs) allows instances of practical size to be addressed, and improves the efficiency of exact algorithms. We propose a novel path-relinking algorithm (PRA) to address the VRP with release and due dates (VRPRDD), and this is based on a hybrid evolutionary framework rooted in scatter search and concepts proposed by Glover (1989) to enhance tabu search. Our algorithm utilises the exploratory potential and adaptive memory of evolutionary algorithms, but relies on more intelligent solution recombination and population (diversity) management.

A new relinking procedure is described to enable efficient exploration of the search space between the solutions in the population, and which introduces some controlled randomization. To intensify the search and improve the convergence of the algorithm, an efficient and powerful neighbourhood search (NS) is applied to some solutions resulting from relinking. To provide a comparator algorithm to the PRA, we embed the neighbourhood search into a standard iterated local search algorithm (ILS).

The remainder of the chapter is organised as follows. In § 6.2, we describe the proposed PRA in detail. The comparator ILS algorithm is presented in § 6.3. Computational experience is reported in § 6.4. Lastly, § 6.5 concludes our findings.

6.2 Path-relinking algorithm

PRAs and the closely related scatter search are EAs incorporating concepts for strategic recombination of solutions and population diversity management. Formalised by Glover et al. (2000), these frameworks share many features of HGAs, such as a population of solutions, recombination, and NS to improve offspring solutions. In contrast, solutions are recombined in an intelligent manner by progressively adapting an initial solution to involve more features of a set of guiding solutions. This is termed path-relinking or simply relinking. At each step of relinking a solution is created, and the resulting solutions form a trajectory connecting the initial and guiding solutions. A more detailed discussion of the variations, design choices and some applications of PRAs are given by Resende et al. (2010).

We propose a PRA for the VRPRDD, and a general outline is given in Algorithm 6.1. Notable features of our method are the successful hybridization of concepts from different evolutionary frameworks, the efficient and effective NS, and the novel relinking procedure.

The algorithm relies on various parameters: β is an infeasibility penalty that is used in equation (6.1) (see § 6.2.1 for our method of computing β), μ is the reduced size of the population after updating, λ is the number of additional solutions in the population before the population is reduced in size, γ is the number of relinkings that are performed without improving the best-known solution until the population is refreshed, κ is the number of offspring that are generated between updates to the value of β, and $T_{\max}$ is the computation time limit that is used as a termination criterion.

Algorithm 6.1 Path-relinking algorithm

Set the parameter values μ, λ, γ, κ, $T_{\max}$
Generate an initial population P and compute the value of β
while *running time* $< T_{\max}$
    Select initial and guiding solutions x_I and x_G
    Construct trajectory from x_I to x_G and select a subset S of these solutions (relinking)
    Improve solutions in S by applying NS
    Set $P = P \cup S$
    if $|P| \geq \mu + \lambda$ **then**
        Reduce population size to μ (population management)
    if *Number of relinkings without improving both best and best feasible solution* $\geq \gamma$ **then**
        Refresh population
    if *Number of offspring since most recent penalty update or population refresh* $\geq \kappa$ **then**
        Update infeasibility penalty β
return Best feasible solution

A population comprising a set of feasible and infeasible solutions P is evolved through iterative relinking, application of NS and population management. Two parent solutions x_I and x_G are selected and relinked (§ 6.2.4), and a subset S of resulting offspring solutions are improved through NS (§ 6.2.2). These improved solutions then enter P if not already present. P is reduced to size μ when it reaches size $\mu + \lambda$, by iteratively removing the solution with worst fitness (§ 6.2.5). Evaluation of the fitness, used in population reduction and selection of solutions for relinking, follows the approach developed in Vidal et al. (2012). More precisely, fitness is measured as a combination of rank in the population for both the objective function value and diversity (§ 6.2.3). If the best feasible solution and best solution do not improve after γ relinkings, then the population is refreshed (§ 6.2.5). Infeasible solutions encountered during the search are penalised by the level of infeasibility, as defined in (6.1). The associated infeasibility penalty weight β is adaptively updated each time κ offspring are generated since β has been updated or the population has been refreshed (§ 6.2.1). The algorithm terminates once the running time exceeds $T_{\max}$.

6.2.1 Infeasibility penalty

Exploring infeasible regions of the search space in heuristic approaches is conjectured to facilitate transition between feasible regions, which may otherwise be distant or unreachable (e.g., see Glover and Hao 2011). In exact approaches, a number of constraint relaxation techniques have proven useful, such as Lagrangian, linear and state-space relaxation. We relax the route capacity constraints and penalise the level of infeasibility, as shown in (6.1). The penalty on the level of infeasibility is controlled by adapting β in (6.1) (see Chapter 3)

during the search. This yields a penalized objective function

$$f^p(x) = f(x) + \beta \sum_{r=1}^{m} \max \left\{ 0, \sum_{i \in R^r} q_i - Q \right\}, \tag{6.1}$$

where β is a parameter, and m, r and R^r are defined as in Chapter 3.

To initially prevent infeasibility, $\beta = \beta_0$, where β_0 is a large value. After three solutions have been generated in the initial population and NS applied, β is set to be the average of the cost per unit of demand for each route. To encourage a useful trade-off between infeasibility and objective value throughout the search, β is adaptively updated each time κ offspring solutions are produced since the most recent update of the penalty or refreshing of the population. Let p_f be the target proportion of feasible solutions, ϵ the range of this proportion, $\bar{p}_f$ the proportion of the κ solutions that are feasible after NS, and δ the update factor. The value of β is then updated accordingly using

$$\beta = \begin{cases} \beta\delta, & \text{if } \bar{p}_f < p_f - \epsilon, \\ \beta/\delta, & \text{if } \bar{p}_f > p_f + \epsilon, \\ \beta, & \text{otherwise.} \end{cases} \tag{6.2}$$

Although the size of the search space is increased by allowing infeasibility, our preliminary experiments with NS (using random first-improvement) exhibit a faster progression through the search space. This effect is attributed to a greater proportion of improving neighbours for some of the solutions, which therefore creates many additional paths in the search space.

6.2.2 Neighbourhood search

We apply a NS to improve and evaluate solutions before considering their introduction to the population. This is an aggressive improvement procedure which enables fast progression to solutions with low objective values. The neighbourhoods used are popular for both VRPs and machine scheduling problems. In hybrid approaches such as our PRA, NS is applied frequently and represents a large proportion of the computation time. In the later parts of the section, we describe an efficient move evaluation procedure and a memory structure to reduce this time requirement.

Let n^r and $(\sigma^r(1), \ldots, \sigma^r(n^r))$ be defined as in Chapter 3. We consider subsets of four well-known neighbourhood structures, where the sizes are restricted by the parameters s_1 and s_2.

- N_1 - Insert (or relocate): Remove $(\sigma^r(i), \ldots, \sigma^r(j))$ and insert immediately after $\sigma^{r'}(i') \notin (\sigma^r(i-1), \sigma^r(i), \ldots, \sigma^r(j))$, where $1 \leq i \leq j \leq n^r$, $0 \leq i' \leq n^{r'}$, $j - i < s_1$ and $1 \leq r, r' \leq m$. The reverse order of $(\sigma^r(i), \ldots, \sigma^r(j))$ is also considered.
- N_2 - Swap: Exchange the positions of $(\sigma^r(i), \ldots, \sigma^r(j))$ and $(\sigma^{r'}(i'), \ldots, \sigma^{r'}(j'))$, where $1 \leq i \leq j < n^r$, $1 \leq i' \leq j' < n^{r'}$, $j - i < s_1$, $j' - i' < s_1$, $1 \leq r \leq r' \leq m$, and the subsequences are disjoint. The reverse order of either or both subsequences are also considered.
- N_3 - Inter-route 2-Opt: Reverse the order of $(\sigma^r(i), \ldots, \sigma^r(i'))$, where $1 \leq i < i' \leq n^r$, $i' - i < s_2$ and $1 \leq r \leq m$.

- N_4 - 2-Opt*: Exchange $(\sigma^r(i),\ldots,\sigma^r(n^r))$ and $(\sigma^{r'}(i'),\ldots,\sigma^{r'}(n^{r'}))$, where $1 \leq i \leq n^r+1$, $1 \leq i' \leq n^{r'}+1$, $i > 1$ or $i' > 1$, $i \leq n^r$ or $i' \leq n^r$ and $1 \leq r < r' \leq m$. By not considering $i = i' = 1$ and $i = i' = n^r + 1$, the cases where all customers are exchanged from r and r' and no customers are exchanged from r and r', respectively, are eliminated.

The neighbourhoods above can be described in terms of either vertex relocation, or the λ-Opt neighbourhood proposed by Lin (1965). Neighbourhoods N_1 and N_2 operate on one or two routes, whereas N_3 operates on a single route and N_4 on two routes. The inherent symmetry in neighbourhoods N_2 and N_4 is removed by specifying that $r \leq r'$ in N_2 and $r < r'$ in N_4. Neighbourhoods N_1 and N_4 can create an additional route by considering one empty route, or remove a route by combining two routes. Initial experiments with a number of other traditional neighbourhoods suggest that a combination of those described perform best. Interestingly, this corroborates the suggestions from our analysis of commonly-used neighbourhoods in § 4.2.2.

If we fix s_1 and s_2, then neighbourhoods N_1, N_2 and N_4 all have size $O(n^2)$, while N_3 has size $O(n)$. We explore the composite neighbourhood $N = N_1 \cup N_2 \cup N_3 \cup N_4$ as follows. We define subsets of each neighbourhood by each pair of vertices for $\sigma^r(i)$ and $\sigma^{r'}(i')$ in the definitions above, where we set $\sigma^r(i) = \sigma^{r'}(i)$ for N_3, and term these *sub-neighbourhoods*. We search the composite neighbourhood by considering pairs (i, j) in a random order, where i and j are customers or a depot for each route. For each pair (i, j), we evaluate the associated composite sub-neighbourhood, and we accept the best neighbour of the first sub-neighbourhood that contains an improving neighbour. Inspired by Nagata and Bräysy (2008) and Vidal et al. (2012), our initial experiments with this approach show an effective balance between computational speed and improvement to the objective value is achieved.

A computational bottleneck is created in NS by the significant number of moves that must be evaluated. This is especially significant for the tardiness objective that often leads to linear evaluation, or constant-time approximate evaluation (see, for example, Taillard et al. 1997). To reduce the time complexity of move evaluation, we extend the evaluation by "concatenation" approach that is developed by Kindervater and Savelsbergh (1997), and more recently expounded by Vidal et al. (2015). As an example, given routes r and r', let us assume that an insert move is applied, such that $(\sigma^r(i),\ldots,\sigma^r(j))$ is inserted after customer $\sigma^{r'}(i')$. If we define $\oplus$ as the concatenation operator, then the two modified routes of this neighbouring solution can be found as $(\sigma^r(0),\ldots,\sigma^r(i-1)) \oplus (\sigma^r(j+1),\ldots,\sigma^r(n^r+1))$ and $(\sigma^{r'}(0),\ldots,\sigma^{r'}(i')) \oplus (\sigma^r(i),\ldots,\sigma^r(j)) \oplus (\sigma^{r'}(i'+1),\ldots,\sigma^{r'}(n^{r'}+1))$.

The following preprocessed data is used to evaluate much of the information about a route:

- $D(\sigma^r(i),\ldots,\sigma^r(j))$ is the total distance traveled while visiting $(\sigma^r(i),\ldots,\sigma^r(j))$ in order;
- $Q(\sigma^r(i),\ldots,\sigma^r(j))$ is the total load for $(\sigma^r(i),\ldots,\sigma^r(j))$;
- $L(\sigma^r(i),\ldots,\sigma^r(j))$ is the duration of the route that visits $(\sigma^r(i),\ldots,\sigma^r(j))$ in order;
- $R_{\max}(\sigma^r(i),\ldots,\sigma^r(j))$ the latest release date for customers in $(\sigma^r(i),\ldots,\sigma^r(j))$.

For a single customer $i \in V'$, we have $D(i) = 0$, $Q(i) = q_i$, $L(i) = 0$, $R_{\max}(i) = r_i$. Preprocess-

ing this data or evaluating a concatenation of subsequences is then achieved by successively applying the following operators. If $\pi = (\sigma^r(i), \ldots, \sigma^r(j)) \oplus (\sigma^{r'}(u), \ldots, \sigma^{r'}(v))$ for suitably defined i, j, u, v, r and r', then

$$D(\pi) = D(\sigma^r(i), \ldots, \sigma^r(j)) + c_{\sigma^r(j)\sigma^{r'}(u)} + D(\sigma^{r'}(u), \ldots, \sigma^{r'}(v)), \tag{6.3}$$

$$Q(\pi) = Q(\sigma^r(i), \ldots, \sigma^r(j)) + Q(\sigma^{r'}(u), \ldots, \sigma^{r'}(v)), \tag{6.4}$$

$$L(\pi) = L(\sigma^r(i), \ldots, \sigma^r(j)) + \tau_{\sigma^r(j)\sigma^{r'}(u)} + L(\sigma^{r'}(u), \ldots, \sigma^{r'}(v)), \tag{6.5}$$

$$R_{\max}(\pi) = \max\{R_{\max}(\sigma^r(i), \ldots, \sigma^r(j)), R_{\max}(\sigma^{r'}(u), \ldots, \sigma^{r'}(v))\}. \tag{6.6}$$

Proposition 6.1. *Preprocessing using the operators (6.3)-(6.6) requires $O(n^2)$ time. For any neighbourhood in which a neighbouring solution is obtained by moving a constant number of subsequences to different positions, checking feasibility and evaluating its total distance cost require $O(1)$ time.*

Proof. Proof. There are $O(n^2)$ subsequences of consecutive customers. Using (6.3),

$$D((\sigma^r(i), \ldots, \sigma^r(j)) = D((\sigma^r(i), \ldots, \sigma^r(j-1)) \oplus D(\sigma^r(j)). \tag{6.7}$$

Thus, the preprocessed distance data is computed for each subsequence in $O(1)$ time, and for all subsequences in $O(n^2)$ time. A similar argument applies for the preprocessed load, duration and latest release date data.

For a neighbouring solution involving the movement of a constant number of subsequences, equations (6.3) and (6.4) are applied to obtain the distance and load for each route in $O(1)$ time. Thus, the capacity constraint can be checked in $O(1)$ to ascertain whether the neighbouring solution is feasible, and its total distance cost can also be evaluated in $O(1)$ time. □

Note that for the additive preprocessing data, only $O(n)$ subsequences are needed to find the data on any subsequence. Considering distance, for example, $D(\sigma^r(1), \ldots, \sigma^r(j))$ for $j = 1, \ldots, n^r$ and $r = 1, \ldots, m$ can be computed in $O(n)$ time. From these values, we can evaluate $D(\sigma^r(i), \ldots, \sigma^r(j))$ for any i and j in $O(1)$ time using

$$D(\sigma^r(i), \ldots, \sigma^r(j)) = D(\sigma^r(1), \ldots, \sigma^r(j)) - D(\sigma^r(1), \ldots, \sigma^r(i)). \tag{6.8}$$

We now consider how to evaluate the weighted tardiness efficiently. First, let $E_{\sigma^r(i')}$ be the earliness of customer $\sigma^r(i')$ in a subsequence of route r. For example, if the subsequence of r begins with customer $\sigma^r(i)$, where $1 \leq i \leq i' \leq n^r$, then $E_{\sigma^r(i')} = \max\{0, d_{\sigma^r(i')} - L(\sigma^r(i), \ldots, \sigma^r(i'))\}$. We also define the function $g^r_{i,j}(t)$ for $t \geq 0$ to be the total weighted tardiness for customers in the sequence $(\sigma^r(i), \ldots, \sigma^r(j))$, where the service for customer $\sigma^r(i)$ starts at time t, for all i and j satisfying $1 \leq i \leq j \leq n^r$ and $r = 1, \ldots, m$. We first focus on the computation of the values $g^r_{i,n^r}(t)$.

The functions $g^r_{i,j}(t)$ are non-decreasing, continuous and piecewise-linear, and we adapt the method of Ergun and Orlin (2006) for constructing the functions. We can represent $g^r_{i,n^r}(t)$ by a sequence $(b_0, g^r_{i,n^r}(b_0)), (b_1, g^r_{i,n^r}(b_1)), \ldots, (b_k, g^r_{i,n^r}(b_k))$ of $(t, g^r_{i,n^r}(t))$ values at which the gradient of the function changes, where $k \leq n^r - i + 1$. Although the values of k and $b_1, \ldots, b_k$ depend on i and r, for notational conciseness we omit introducing subscripts i and superscripts r. Each pair $(b_h, g^r_{i,n^r}(b_h))$ for $h = 0, 1, \ldots, k$ is referred to as a *breakpoint*,

where $b_0 = 0$ is the first point in the domain of the function $g^r_{i,n^r}(t)$. Using this representation, the function can be expressed as

$$g^r_{i,n^r}(t) = \begin{cases} g^r_{i,n^r}(b_h) + \frac{g^r_{i,n^r}(b_h+1) - g^r_{i,n^r}(b_h)}{b_{h+1}-b_h}(t - b_h), & \text{if } b_h \le t < b_{h+1}, \\ & h = 0, \ldots, k-1, \\ g^r_{i,n^r}(b_k) + \sum_{h=i}^{n^r} w_{\sigma^r(h)}(t - b_k), & \text{if } t \ge b_k. \end{cases} \tag{6.9}$$

For a given t, knowledge of the breakpoints allows the value of $g^r_{i,n^r}(t)$ to be computed in $O(\log n)$ time using bisection search to find the segment corresponding to the value of t.

Our procedure for computing breakpoints for $g^r_{i,n^r}(t)$ for $1 \le i \le n^r$ and $1 \le r \le m$ is given in Algorithm 6.2. It relies on the observation that for a subsequence $(\sigma^r(i), \ldots, \sigma^r(n_r))$, the $b_1, b_2, \ldots, b_k$ correspond to the distinct values among $E_{\sigma^r(i)}, \ldots, E_{\sigma^r(n_r)}$, and that the order of these earliness values does not change if the time that service starts at $\sigma^r(i)$ is changed. Our algorithm sets $\nu = (\nu(1), \ldots, \nu(n_r - i + 1))$ to be a sequence of the customers of $E_{\sigma^r(i)}, \ldots, E_{\sigma^r(n^r)}$ ordered by non-decreasing earliness, and $\bar{\nu}$ stores the sequence ν used in the previous iteration. Note that the function $\text{Add}(\sigma, \nu)$ inserts the customer σ in the sequence of customers ν so that $E_{\nu(1)} \le \ldots \le E_{\nu(n^r - i + 1)}$.

Algorithm 6.2 Procedure to create functions $g^r_{i,n^r}(t)$

for all $1 \le r \le m$
  Set $i = n^r$, and $\nu = \emptyset$
  while $i \ge 1$
    Set $E_{\nu(h)} = \max\{0, E_{\nu(h)} - \tau_{\sigma^r(i)\sigma^r(i+1)}\}$ for $1 \le h < n^r - i$
    Set $E_{\sigma^r(i)} = \max\{0, d_{\sigma^r(i)} - L(\sigma^r(i), \ldots, \sigma^r(n^r))\}$
    $\nu =$Add$(\sigma^r(i), \nu)$
    Set $k = 0$ and $b_k = 0$
    Choose the largest index $j \in \{1, \ldots, n^r - i + 1\}$ such that $E_{\nu(j)} = E_{\nu(1)}$
    Compute $W = \sum_{h=1}^{j} w_{\nu(h)}$
    if $E_{\nu(j)} = 0$ **then**
      Compute $g^r_{i,n^r}(b_k) = \sum_{i'=i}^{n^r} w_{\sigma^r(i')} \max\{0, L(\sigma^r(i), \ldots, \sigma^r(i')) - d_{\sigma^r(i')}\}$
    else
      Set $g^r_{i,n^r}(b_k) = 0$
    while $j < n^r - i + 1$
      Set $k = k + 1$ and $b_k = E_{\nu(j+1)}$, and compute $g^r_{i,n^r}(b_k) = g^r_{i,n^r}(b_{k-1}) + W(b_k - b_{k-1})$
      Choose the largest index $j' \in \{j + 1, \ldots, n^r - i + 1\}$ such that $E_{\nu(j')} = E_{\nu(j+1)}$
      Set $W = W + \sum_{h=j+1}^{j'} w_{\nu(h)}$ and $j = j'$
    Set $i = i - 1$

Having used Algorithm 6.2 to compute the breakpoints, function $g^r_{i,n^r}(t)$ is specified by (6.9). The function $g^r_{i,j}(t)$ for $j = 1, \ldots, n^r$ is then defined by

$$g^r_{i,j}(t) = g^r_{i,n^r}(t) - g^r_{j+1,n^r}(t + L(\sigma^r(i), \ldots, \sigma^r(j)) + \tau_{\sigma^r(j),\sigma^r(j+1)}). \tag{6.10}$$

This allows us to define $Z(\sigma^r(i), \ldots, \sigma^r(j))$ as the total weighted tardiness for the customers $(\sigma^r(i), \ldots, \sigma^r(j))$ visited in order, starting from the depot. That is computed using

$$Z(\sigma^r(i), \ldots, \sigma^r(j)) = g^r_{i,j}(R_{\max}(\sigma^r(i), \ldots, \sigma^r(j)) + \tau_{0,\sigma^r(i)}), \tag{6.11}$$

where $R_{\max}(\sigma^r(i),\ldots,\sigma^r(j))$ is the time that the vehicle departs from the depot. Analogous to equations (6.3)-(6.6) in which $\oplus$ is defined as a concatenation operator for subsequences, we can also evaluate the total weighted tardiness for a concatenation of subsequences using

$$Z(\pi) = g^r_{i,j}(R_{\max}(\pi) + \tau_{0,\sigma^r(i)}) + g^{r'}_{u,v}(R_{\max}(\pi) + \tau_{0,\sigma^r(i)} + L(\sigma^r(i),\ldots,\sigma^r(j)) + \tau_{\sigma^r(j),\sigma^{r'}(u)}), \tag{6.12}$$

where $\pi = (\sigma^r(i),\ldots,\sigma^r(j)) \oplus (\sigma^{r'}(u),\ldots,\sigma^{r'}(v))$.

Recall that some neighbours under the insert, swap and inter-route 2-Opt neighbourhoods reverse a subsequence of jobs. To evaluate such neighbours efficiently, we also compute values of $\bar{g}^r_{i,j}(t)$ for $t \geq 0$, which is the total weighted tardiness for customers in the sequence $(\sigma^r(j),\ldots,\sigma^r(i))$, where the service for customer $\sigma^r(j)$ starts at time $t \geq 0$, in a similar way to our computation of $g^r_{i,j}(t)$.

Proposition 6.2. *Preprocessing using Algorithm 6.2 to determine the breakpoints $(b_h, g^r_{i,n^r}(b_h))$ for $h = 0,1,\ldots,k$, $i = 1,\ldots,n^r$ and $r = 1,\ldots,m$ requires $O(n^2)$ time. For any neighbourhood where a neighbouring solution is obtained by moving a constant number of subsequences to different positions, evaluating its total weighted tardiness after preprocessing requires $O(\log n)$ time.*

Proof. Proof. Steps 4-18 of Algorithm 6.2 require $O(n)$ for each of the n choices of r and i, which gives a time complexity for these steps of $O(n^2)$ overall. For Step 6, the sequence π is updated by finding the appropriate position into which $\sigma^r(i)$ is inserted by applying bisection search. Thus, Step 6 requires $O(\log n)$ time for each $\sigma^r(i)$, and $O(n \log n)$ time overall. This establishes that all breakpoints are computed by Algorithm 6.2 in $O(n^2)$ time.

Iterative application of equation (6.12) shows that evaluating a neighbouring solution created by moving a constant number of subsequences to different positions requires a constant number of evaluations of functions $g^r_{i,j}(t)$ for various values of i, j, r and t. Each $g^r_{i,j}(t)$ function is obtained from (6.10) by evaluating two $g^r_{i,n^r}(t)$ functions. Lastly, each $g^r_{i,n^r}(t)$ is computed from (6.9) in $O(\log n)$ time, which is the time for requirement for applying bisection search to find the correct piecewise linear segment corresponding to t in (6.9). Combining these statements, we obtain an overall time complexity of $O(\log n)$ for evaluating the total weighted tardiness after moving a constant number of subsequences. □

6.2.3 Fitness function

The diversity amongst solutions in the population is a key consideration in PRAs, and increasingly in EAs more widely. If the population diversity is too low, then the exploratory potential of the algorithm becomes weak. This is because less information is captured in the population, and the shorter relinking trajectories that are produced result in less of the search space being explored. If solutions in the population are more diverse, then we may expect more interesting and varied relinking trajectories, particularly if these solutions have good objective values. To address the dual objectives of improving solution quality but retaining a useful level of diversity, a variety of approaches are suggested and a number are discussed in § 2.3.3 and § 4.2.2. We have chosen to follow the biased fitness approach of Vidal et al. (2012), which combines the rank of a solution in the population both for the objective value and for a measure of diversity.

To measure the diversity in the population, it is useful to define a measure of the distance between two solutions x and x'. No single choice of distance measure appears ideally suited for the problem. However several features of a solution may convey meaningful information, and different attributes/distance measures can be defined. Considering computational effort, we propose the use of arcs $(i,j) \in A$ as the attributes, with the distance between two solutions being the number of different arcs used in those solutions. Let $I_{ij}(x, x' : \xi)$ be a binary indicator for any pair of solutions x and x', and arc $(i,j) \in A$ used in x, which is defined by

$$I_{ij}(x, x' : \xi) = \begin{cases} 1, & \text{if } (i,j) \text{ is used in solution } x \text{ and not in } x', \\ 1-\xi, & \text{if } i \neq 0,\ j \neq 0,\ (i,j) \text{ is used in solution } x \text{ and } (j,i) \text{ in } x', \\ 0, & \text{otherwise.} \end{cases} \tag{6.13}$$

where ξ is a parameter in the interval $0 \leq \xi \leq 1$. Parameter ξ, for $\xi > 0$, reduces the distance if arcs are used in opposite directions in the two solutions. This definition of $I_{ij}(x, x' : \xi)$ is used to create the effect of using a balance of edge- and arc-based distance. If $\xi = 1$ then the distance is edge-based, and if $\xi = 0$ then the distance is arc-based. Observe that $\xi = 1$ considers the assignment of customer vertices to the same routes more explicitly, but can underestimate the distance between solutions when the schedules are important or the distances between customers are asymmetric, and vice versa for $\xi = 0$. Observe that in the second expression (6.13) the depot arcs are not considered.

Notice that $\sum_{(i,j)\in x} I_{ij}(x, x' : \xi) = \sum_{(i,j)\in x} I_{ij}(x', x : \xi)$ if and only if x and x' contain the same number non-empty routes. To adjust for solutions with varying numbers of non-empty routes, we therefore define $\rho(x)$ as the number of non-empty routes in a solution x. The distance between x and x' is then defined as

$$\text{dist}(x, x' : \xi) = \sum_{(i,j)\in x} I_{ij}(x, x' : \xi) + \max\{0, \rho(x) - \rho(x')\}, \tag{6.14}$$

where $\text{dist}(x, x' : \xi) = \text{dist}(x', x : \xi)$ and $0 \leq \text{dist}(x, x' : \xi) \leq n + m$ for any choice of x and x'. This measure of distance is also related to the Hamming distance for solutions represented by $n \times n$ binary matrices with the entry in row i and column j set equal to one if $(i,j) \in A$ is used in the solution.

Let $N_c(x)$ be a set containing n_c solutions x' from the population with smallest values of $\text{dist}(x, x' : \xi)$. We define the *diversity* $\mathcal{D}(x)$ of x as the average distance to solutions of this set, which is given by

$$\mathcal{D}(x) = \frac{1}{n_c} \sum_{x' \in N_c(x)} \text{dist}(x, x' : \xi). \tag{6.15}$$

Let N_e be a set containing n_e solutions from the population with lowest objective value. We also define $p(x)$ and $\delta(x)$ as the rank of a solution x in the population, ordered by non-decreasing value of the penalised objective (6.1) and non-increasing value of the diversity (6.15), respectively. We can then define the *fitness* of a solution x in the population (which is to be minimised) as

$$F(x) = p(x) + (1 - n_e/|P|)\, \delta(x). \tag{6.16}$$

If $|P| > n_e$, the coefficient of $\delta(x)$ prevents the solutions N_e from being given the worst fitness

in the population. Even if one of these solution has the lowest diversity it will not be given the worst fitness (for a proof, see Vidal et al. (2012)).

The population can contain both feasible and infeasible solutions. However, if all of the solutions are infeasible, then it may be difficult to navigate to a feasible region of the search space. For this reason, and to retain a feasible solution if one is found, we modify the ranking mechanism by assigning the best feasible solution in the first position of the ranking defined by p. This will also encourage this solution to be selected more frequently in relinking, thereby ensuring further emphasis on exploring the feasibility boundary and achieving feasibility. This boundary is expected to be of particular interest for instances where the capacity constraints are tight.

6.2.4 Parent selection and relinking

Path-relinking is first suggested by Glover (1989) as a technique for exploring trajectories connecting solutions found during NS. The method is now considered more generally as a recombination approach, and compared to crossover in GAs relies less on randomization. First, two solutions are selected as the *initial solution* x_I, and the *guiding solution* x_G. Starting from x_I, a trajectory is constructed by iteratively applying neighbourhood moves to introduce attributes from x_G. The number of different attributes describes a distance measure between solutions. Also, the trajectory contains offspring solutions with decreasing distance to x_G and increasing distance from x_I. Note that x_G can be defined as a set of solutions, although we choose x_G to be a single solution in our implementation.

We use binary tournament to select two parent solutions for relinking, which encourages some variety in the selection but prioritises lower fitness. Once two different solutions are selected, these are randomly assigned to be either x_I or x_G. For the relinking procedure described and the problem considered, our initial experiments suggest taking x_I as the solution with highest fitness produces trajectories structurally different to the reverse, although neither dominates the other. Both feasible and infeasible solutions can be selected for x_I and x_G, and this encourages further exploration of the boundary of feasibility.

For any solution x we need to define its attributes, and for a pair of solutions x and x' we need to define the set of attributes that are in x' and not in x, and the cardinality of this set. As described in § 6.2.3, the arcs (i, j) of a solution x can be regarded as the attributes. Further, for solutions x and x', we define $\mathcal{A}(x, x') = \{(i, j) \in A\colon (i, j) \in x', (i, j) \notin x\}$ as the set of attributes in x' that are not in x, and $\text{dist}(x, x' : 0) = |\mathcal{A}(x, x')|$ as stated in (6.14), as the cardinality.

Let $\Delta = \text{dist}(x_I, x_G : 0)$, N_r be a chosen set of neighbourhoods which introduce attributes, S be the set of solutions returned from relinking, ϕ be a parameter representing the average number of offspring returned from a relinking, $\phi' = \Delta/(\phi+1)$, $\bar{\phi}$ be a parameter the indicates the next iteration at which the current solution is copied to S, and the operator $||[a]||$ rounds a to the nearest integer. A general outline of the relinking procedure is given in Algorithm 6.3.

The current solution x_c is initially set equal to x_I, and at each iteration of relinking we seek to progressively reduce $\text{dist}(x_c, x_G : 0)$ by at least one. While $\text{dist}(x_c, x_G : 0) > 2$, we randomly order the neighbourhoods N_r, and choose the move which improves the penalised objective (6.1) the most from the first neighbourhood featuring a move that introduces an

Algorithm 6.3 Relinking procedure

Set the parameter value ϕ', initialise $S = \emptyset$ and $x_c = x_I$
Randomly pick an integer $\bar{\phi}$ from $\left[|[0.75\phi']|, |[1.25\phi']|\right]$
while $\text{dist}(x_c, x_G : 0) > 2$
    Randomly order neighbourhoods N_r
    Set N_c as the first neighbourhood with a move introducing an attribute in x_G to x_c
    Apply most improving move of N_c that introduces an attribute of $\mathcal{A}(x_c, x_G)$ to x_c
    if *Number of iterations since* $\bar{\phi}$ *updated* $= \bar{\phi}$ **then**
        Copy x_c into S and randomly pick an integer $\bar{\phi}$ from $\left[|[0.75\phi']|, |[1.25\phi']|\right]$
return S

attribute from $\mathcal{A}(x_c, x_G)$ to x_c. Every $\bar{\phi}$ iterations, we copy x_c into the set S, and each time $\bar{\phi}$ is chosen as a random integer between $|[0.75\phi']|$ and $|[1.25\phi']|$.

To design neighbourhoods for the set N_r, we consider introducing an arc $(i, j) \in \mathcal{A}(x_c, x_G)$. As suggested in Reghioui et al. (2007), we define a block as a subsequence of consecutive customers that are connected in both x_c and x_G, and the immediate predecessor and successor of the subsequence in x_c and x_G are different. Therefore, removing a block from its current position in x_c causes no increase in $\text{dist}(x_c, x_G : 0)$. Furthermore, inserting this block in a position that introduces the desired arc $(i, j) \in \mathcal{A}(x_c, x_G)$ into x_c decreases $\text{dist}(x_c, x_G : 0)$. If $i, j \in R^r$ for x_c, then there is a block-insert neighbour which introduces (i, j). This neighbour has the block $(\sigma^r(p), \cdots, \sigma^r(p+q))$, where $\sigma^r(p) = j$, relocated to immediately after vertex i. If $i \in R^r$ and $j \in R_{r'}$ for x_c, where $r \neq r'$, then there is a 2-Opt* neighbour which introduces the desired arc (i, j) into x_c and does not increase $\text{dist}(x_c, x_G : 0)$. This neighbour exchanges the subsequences $(\sigma^r(p), \cdots, \sigma^r(n^r))$ and $(\sigma^{r'}(q), \cdots, \sigma^{r'}(n^{r'}))$, where $\sigma^r(p-1) = i$ and $\sigma^{r'}(q) = j$. The 2-Opt* moves to introduce an arcs $(0, j)$ for $j \in V'$ in x_c, create a new route, and are considered at any iteration if $\rho_{x_c} < \rho_{x_G}$. For further clarity, an example relinking is illustrated in Chapter B.

The penalised objective (6.1) is used in relinking to select neighbourhood moves, which increases the number of offspring that can be explored through relinking. Notice that solutions that are infeasible with respect to the capacity are also possible in the trajectory for any value of β, due to the constraint on introducing at least one arc of $\mathcal{A}(x_c, x_G)$ to x_c. If no capacity feasible solutions can be reached under this criterion, then infeasible solutions will be visited. By definition, the trajectory is guaranteed to return to the level of feasibility of x_G by the end. A solution which is not neighbouring either x_I or x_G can only be found if $\Delta > 4$. Otherwise, only one or two offspring solutions would be produced from the relinking of x_I and x_G, and both offspring neighbour x_I or x_G. If x_I and x_G are locally optimal for the neighbourhoods N_r then this relinking is unnecessary, however this is not strictly true in our PRA because the value of β may since have changed in (6.1).

Relinking to different solutions in the population enables a comprehensive exploration of the locality of a solution in "promising" directions. Further, the use of the penalised objective (6.1), and varying between the inter- and intra-route neighbourhood in N_r, compounds the potential for local exploration and also encourages a more comprehensive exploration of the search space between solutions.

Proposition 6.3. *For given x_I and x_G, relinking can be performed in $O(\Delta^2 n)$ time.*

Proof. Proof. Finding Δ from (6.14) and identifying the set $\mathcal{A}(x_I, x_G)$ can be performed in $O(n)$ time, and $|\mathcal{A}(x_I, x_G)| = O(\Delta)$. The number of solutions in the trajectory and consequently the number of iterations of the relinking procedure is therefore $O(\Delta)$. At each iteration, the maximum number of possible moves is $O(\Delta)$. The 2-Opt* moves are identified in $O(1)$ time from $\mathcal{A}(x_c, x_G)$, but the block-insert moves are found in $O(n)$ time. Thus, the overall time complexity is $O(\Delta^2 n)$. □

Note that an alternative implementation in which data is preprocessed in $O(n^2)$ time at each iteration requires $O(\Delta n^2)$ time. Thus, the more straightforward implementation that leads to the result in 6.3 is preferable.

The neighbourhoods used in relinking are restricted and therefore solutions in the trajectory are not guaranteed to be locally optimal. We perform NS on each solution in S prior to considering introduction to the population. Defining S as the complete trajectory would require performing NS on each of these solutions, which represents a significant computational effort. Introducing a large number of solutions to the population from a single relinking may also encourage lower diversity in the population.

Solutions closer to each other in the trajectory have a greater probability of sharing basins of attraction, and therefore leading to the same local optima. This fact is also exaggerated by the use of similar neighbourhoods in the NS and relinking procedures. Various techniques have been proposed to select offspring solutions in the literature. For example, Hashimoto and Yagiura (2008) propose storing a fixed number of the best solutions that have better objective value than their immediate neighbours in the trajectory. A simpler policy is described by Reghioui et al. (2007), who store solutions a fixed number of iterations apart in the trajectory. In contrast, Sörensen and Schittekat (2013) return a single solution, which has the best objective value or is at the midpoint of the trajectory.

Preliminary experiments on the VRPRDD suggest the PRA described is robust to the approach used to select S. We therefore decide to retain ϕ approximately equidistant solutions from the trajectory. This reduces the computational effort and obtains a spread of solutions with some additional randomization. The current solution x_c is periodically copied into S, and the number of iterations to wait for the next copying is selected as a random integer from the interval $\big[|[0.75\phi']|, |[1.25\phi']|\big]$. The relinking procedure therefore produces $|S| \sim U\big[|[0.75\phi]|, |[1.25\phi]|\big]$ solutions. Each solution is improved by NS and enters the population if not already present. Relinking can be performed between the same initial and guiding solutions during the algorithm, and the randomization ensures some variation in S.

6.2.5 Population management

The population is initialised with 1.5μ randomly generated solutions improved by NS. This is motivated by beginning the search with diverse solutions which have at least moderately low objective values. When the population size reaches $\mu + \lambda$, it is reduced to μ considering all solutions in the population. The population is reduced by iteratively removing the solution with the highest fitness and updating the fitness of the remaining solutions. As discussed in § 6.2.4, the set of solutions S returned from each relinking are improved by NS and are introduced to the population if not already present. Notice that solutions joining the population

after relinking are immediately available to be selected as parent solutions.

Many population-based approaches refresh the population or remove a portion of the population and re-initialise with newly generated solutions. This is usually performed if the progress of the search, or the diversity of the population, appears to have stagnated. In the PRA, each time γ relinkings are performed and neither the best quality solution nor the best quality feasible solution are improved, then the population is refreshed. The top $\mu/3$ solutions in the population are retained and μ new solutions are generated identically to the population initialization. This attempts to utilise progress made during the search, whilst introducing a significant amount of diversity. Clearly, if $n_e \leq \mu/3$, then the feasible solution with best value and $\max\{0, n_e - 1\}$ other solutions with lowest value for the penalised objective are retained.

6.3 Iterated local search

ILS is an elementary stochastic extension of NS, applying a kick to escape whenever a local optimum is reached. The simplicity and generality of the framework are key features, and it performs competitively when applied to numerous problems in combinatorial optimization. For an overview of the design, implementation and some applications of ILS, we refer to Lourenço et al. (2010).

In Algorithm 6.4, we describe the basic implementation of our comparator ILS algorithm for the VRPRDD. This is primarily a comparison for the proposed PRA, and the NS of § 6.2.2 is used as the improvement procedure. In the initialization, the current solution is generated randomly and is then improved by NS. An alternate solution is produced by kicking the current solution, NS is then applied and the resulting solution replaces the current solution if it has lower penalised objective value (6.1).

Algorithm 6.4 Iterated local search

1: Initialise current solution x
2: Improve x by applying NS
3: **while** *running time* $< T_{\max}$
4: Apply η random moves to x producing x' (apply a kick)
5: Improve x' by applying NS
6: **if** $f(x') < f(x)$ **then** set x as x'
7: **return** x

For the kick, we apply η random moves in the swap neighbourhood with $s_1 = 1$ (see § 6.2.2). The kick can produce infeasible solutions with respect to vehicle capacity, and the level of infeasibility is restrictively penalised using $\beta = \beta_0$ in (6.1), as defined in § 6.2.1. As a result, when NS is applied to infeasible solutions, the objective will be dominated by initially achieving feasibility. The only parameters of this method are the kick size η, and a limit on computation time $T_{\max}$, which is used as a termination criterion.

6.4 Computational experiments

In this section, we present the results from the calibration of the algorithms (§ 6.4.1), a comparison of the performance of the proposed heuristics on the benchmark instances with between 50 and 199 customers (§ 6.4.2), and an analysis of the PRA results independently (§ 6.4.3). Finally, a sensitivity analysis of the PRA parameters is performed (§ 6.4.4). A validation of the PRA on instances of the more familiar asymmetric VRP are presented in Chapter C.

The algorithms are implemented in C++ and complied using GCC 4.8.1. All experiments are performed using a single core of an Intel Xeon E5–2670 2.6GHz processor (running Linux Red Hat Enterprise Server release 6.3).

6.4.1 Algorithm calibration

Common to metaheuristics and particularly EAs, the PRA presented relies on a variety of correlated parameters. The objective of parameter calibration is find a set of parameters to optimise some performance measures of the algorithm. Mercer and Sampson (1978) suggest using metaheuristics to calibrate metaheuristics, termed metacalibration, but this early work reports insufficient computational resources.

To calibrate the PRA presented, we apply the covariance matrix adaptation evolutionary strategy (CMA-ES) proposed by Hansen and Ostermeier (2001). The CMA-ES has performed well in a variety of settings, and in particular EAs have been calibrated to achieve competitive performance (Smit and Eiben 2009, Vidal et al. 2012). To evaluate a set of parameters, the PRA is executed on a training-set of instances that are selected to represent the problem range. The calibration objective is to minimise the average percentage gap in the objective value across the training-set compared to the best found in all previous experiments.

Our preliminary experiments suggested a number of parameters are more robust, and these are fixed at values that appear to perform well. For the parameter limiting the size of the neighbourhoods N_1 and N_2, $s_1 = 2$ is found to provide a good compromise between quality and efficiency. We set the parameter limiting the size of N_3 to $s_2 = 10$ because reversals of large sections of routes appear to increasingly disturb the schedules. To initially prevent infeasibility, the initial value of the penalty β is set to $\beta_0 = 1000$. For the parameters controlling the infeasibility penalty, the update rate is $\delta = 1.2$, the range of the target proportion of feasible offspring is $\epsilon = 0.05$, and the number of offspring between updates is $\kappa = 50$. Finally, the number of iterations considered in the condition on population refresh is set to $\gamma = 80$.

Similarly to Vidal et al. (2012), we tune the remaining parameters separately for the different problem types. This may reveal any dependency in the parameters on the value of α, and the results can also be used to infer good parameter settings. In Table 6.1, the best solutions of the parameter calibration are given for the three values of α, and the parameter values used in the main experiments that follow are given in the "Final values" column. Multiple settings for the final parameter values are given in order of increasing α. The boundaries on the parameter used in the metacalibration are also given in the "Range" column. Some of these are simply proportions, μ and λ are set through observations from the literature, and ϕ and p_c are estimated.

Table 6.1: Results from meta-EA calibration of PRA parameters.

Parameter		Range	$\alpha = 0.3$	0.5	0.7	Final values
μ	population size	$[10, 40]$	10	14	13	**12**
ϕ	number of solutions from relinking	$[1, 30]$	6	5	5	**5**
λ	number of solutions in generation	$[\phi, 100]$	8	14	28	**10/15/30**
p_f	target proportion of feasible solutions	$[0, 1]$	0.38	0.5	0.53	**0.5**
p_e	proportion of elite solutions	$[1/\mu, 1]$	0.12	0.14	0.20	**0.1/0.2/0.2**
p_c	proportion of solutions in diversity	$[0, 0.33]$	0.12	0.27	0.21	**0.1/0.25/0.25**
ξ	edge reduction in dist$'$	$[0, 1]$	0.43	0.28	0.82	**0.5**

The results suggest that the best parameters for the values of α are generally similar, and the most diverse are λ, p_e and p_c. We found that averaging and rounding the other parameters across the problem types caused negligible degradation of the final objective values achieved. For $\alpha = 0.5$ and 0.7, averaging p_e and p_c also caused negligible degradation of the final objective values.

The problem type dependent parameters suggest that the objectives ideally require different exploration strategies in the PRA. When the weighted tardiness has a greater emphasis, then it is preferable to more regularly manage the population. Additionally, only the closest solution in the population is considered in the diversity of solutions, and this leads to a maximally diverse population. Relinking and NS are therefore performed more frequently among solutions with low fitness, and the diversity is given almost equal weight in the fitness.

When the total distance traveled has a greater emphasis, then the population is managed less regularly. A further contrast to the case of weighted tardiness dominating is that the three closest solutions are considered in the diversity, and this allows clusters of solutions to be formed by a smaller penalization of solutions with lower distances to the two closest solutions. These differences encourage greater and more localised exploration in each generation before the population is reduced.

For the ILS, we used the same set of training instances and consider $\eta = 1, 2, 3, 4$. The results suggested that $\eta = 2$ achieves the best objective values in acceptable computation time for the values of α considered. Larger values of η are not considered due to the results, the greater disturbance to the solutions and the resulting infeasibility that often occurs.

6.4.2 Comparison of algorithm performance

To evaluate the relative performance of the PRA and ILS methods for the VRPRDD, results for the benchmark set are compared in Table 6.2. For comparability, the total running time is set as $T_{\max} = 10$min for both PRA and ILS, and the results are based on 10 independent runs. The computational time limit has been chosen based on the running times for similar instance sizes of heuristics for VRPTWs, and considering that the practical setting is planning at the start of a period. The values PIA and PIB represent the average percentage improvement in the objective value achieved by the PRA compared to the ILS, for the average and best of the 10 runs respectively. Further, TBS_P and TBS_I represent the average time in minutes that the solution with best objective value of a run is first identified, for the PRA and ILS respectively.

Considering the results from these experiments. the PRA appears superior, and achieves better objective values in shorter computation time than ILS. Notably, both PIA and PIB

Table 6.2: Comparison of the PRA and ILS results.

n	$\alpha = 0.3$				$\alpha = 0.5$				$\alpha = 0.7$			
	PIA	PIB	TBS_P	TBS_I	PIA	PIB	TBS_P	TBS_I	PIA	PIB	TBS_P	TBS_I
50	2.64	0.39	1.11	3.90	1.84	0.10	0.84	3.54	1.13	0.17	0.78	2.99
100	2.09	0.48	2.96	5.26	1.36	0.36	2.33	4.73	0.83	0.14	1.84	4.78
150	3.77	1.72	4.38	6.86	2.27	0.87	3.80	6.52	1.61	0.55	3.87	6.87
199	3.79	2.11	5.85	7.67	2.75	1.58	5.40	7.44	2.13	1.34	5.84	7.35
Avg.	3.07	1.18	3.57	5.92	2.06	0.73	3.09	5.56	1.42	0.55	3.08	5.50

are largest when the weighted tardiness has a higher objective weighting. This suggests that the contrasting features of the PRA, such as a more directed exploration of the search space and the information contained in a diverse population of solutions, are useful for small α. Equivalently, this may be attributable to the ILS finding a larger number of local optima, which therefore have a greater chance of being distant from the optimal solutions, and the reliance on only accepting solutions with improving objective value.

The PRA achieves good quality objective values more robustly than the ILS, as evidenced by the large PIA values. The ratio between the PIA and PIB values is similar for the different values of α, and this suggests that the relative robustness is consistent. Notably, both PIA and PIB increase with n, showing that the PRA achieves greater improvement in objective value over the ILS for the larger instances. These instances clearly have a larger search space and are expected to contain a greater number of local optima. These results support the previous conjectures relating to the contrasting features, and may also suggest that the effect of the kick in ILS is not large or varied enough for larger instances. The PIA is notably lower for $n = 100$, which suggests that the robustness of the methods are more similar. This characteristic may be attributable to these instances having the most sparsely distributed set of customers.

Considering TBS_P and TBS_I, we observe that PRA finds the solution with best objective value faster on average than ILS. Again, this suggests that the contrasting features of the PRA are more effective and also efficient. When α is lower, then both TBS_P and TBS_I are larger, which suggests that a higher objective weighting for the weighted tardiness increases the time taken for the algorithms to converge and find the best solutions. Finally, note that the growth of TBS_P with respect to n is larger than for TBS_I, but this is expected for a population-based method. This may be a consideration when solving larger instances, although the PIA and PIB are also increasing with n and we would expect longer computation times to achieve better objective values.

6.4.3 PRA results on the instances

To assess and validate the benchmark instances, we present a detailed analysis of the PRA results. We consider the effects on the objective values of the different parameters that define the problem features and the time taken to find the best solution.

In Table 6.3, we present the average results for the PRA across the 10 runs with $T_{\max}$ set at 10 minutes, where Av represents the average objective value, and TBS is the average time taken to find the best solution of the run. To analyse the effects of the problem attributes introduced, we average the results for all the underlying CVRP instances or equivalently for

all values of n.

Table 6.3: Average results of PRA on benchmark instances.

b	k	$\alpha = 0.3$				$\alpha = 0.5$				$\alpha = 0.7$			
		$e_m = 0$		$e_m = 0.5$		$e_m = 0$		$e_m = 0.5$		$e_m = 0$		$e_m = 0.5$	
		Av	TBS	Av	TBS	Av	TBS	Av	TBS	Av	TBS	Av	TBS
0.25	4	1052.6	3.7	735.9	1.5	1088.2	4.0	946.8	1.9	1102.9	2.9	1077.3	2.9
	6	458.0	3.9	407.6	3.1	650.0	3.2	632.8	1.9	817.0	2.7	815.4	2.1
	8	304.4	4.3	304.4	4.8	503.2	3.7	503.2	4.1	694.4	3.9	694.4	3.3
0.50	4	1477.9	3.6	953.9	2.6	1410.6	3.1	1130.5	1.3	1319.2	2.3	1253.3	1.6
	6	623.9	3.4	478.5	3.5	795.0	3.7	741.1	2.6	943.1	3.7	937.3	2.9
	8	364.5	3.9	360.3	4.4	585.4	4.0	584.3	4.4	778.8	4.5	778.5	3.7
0.75	4	1834.1	3.9	1175.7	2.1	1688.9	3.6	1308.0	2.3	1520.9	3.4	1396.4	2.9
	6	855.7	4.2	550.2	3.1	975.3	2.4	829.7	1.8	1074.0	2.9	1047.1	2.9
	8	429.9	4.5	396.8	3.6	661.4	3.0	646.6	3.2	865.6	3.3	862.0	3.8
1.00	4	2119.5	4.3	1335.9	2.5	1914.4	4.2	1428.8	1.8	1675.6	3.9	1494.5	2.0
	6	1064.7	3.9	627.6	2.0	1147.1	3.9	913.8	2.3	1202.0	3.5	1144.4	2.4
	8	523.7	5.5	438.4	3.5	748.6	3.8	709.3	3.7	943.4	3.5	938.5	2.9
Avg.		925.7	4.1	647.1	3.1	1014.0	3.6	864.6	2.6	1078.1	3.4	1036.6	2.8

The experiments show contrasting performance, and this suggests a varied set of instances. The instance parameters appear to have regular effects across the set, and respect the principles discussed in the instance generation (Chapter 3). Different results are produced using the values of α proposed, and this reveals the effect of different weightings in the objective.

A greater spread of release dates, which is represented by larger values for b, can cause the weighted tardiness to become larger. This increases the overall objective values found, particularly if the due dates are tight as indicated by low values of k. The value of TBS also increases, and this is presumably because more computation time is necessary to find good partitions and schedules of the customers.

The value of TBS is lower when the due dates are tighter, suggesting the instances can be solved more quickly when a greater proportion of solutions have high tardiness. Notice that this will cause a large number of feasible solutions (and infeasible solutions with respect to the vehicle capacity) to have high objective values, resulting in a steeper landscape. The value of TBS also decreases as the value α becomes higher. This may be attributable to the greater diversity of objective values amongst solutions when the total distance traveled becomes dominant, which enables the algorithm to navigate the search space more easily.

Increasing the number of vehicles by increasing e_m permits more flexibility in vehicle departure times and therefore customer arrival times. The greater flexibility can decrease the weighted tardiness for certain instances and values of α. If the change in total distance traveled due to an additional route is less than the reduction in weighted tardiness, then another vehicle clearly improves the objective value. Also, if greater numbers of routes are used in a solution, then the scheduling and capacity constraints will be less restrictive. This is most prominent when the scheduling is tight, i.e., when b is large or k is small.

6.4.4 Sensitivity analysis of PRA

We now present our results from experiments performed to analyze the individual importance of some of the main components in the PRA. We disable each component independently, and the results for each alternative are again taken from 10 independent runs with $T_{\max}$ set at 10 minutes. The variants of the PRA with a component disabled are: "No NS" that does not apply NS to offspring solutions; "No Infe." that does not adjust the infeasibility penalty β; and "No Div." that does not include diversity in the fitness function. The results from these variants are summarised in Table 6.4, where the PRA column refers to the version with no disabling. The value Gap represents the worsening of the average objective value compared to the full PRA as a percentage of the latter, and again TBS is the average time taken to find the best solution in a run.

Table 6.4: Sensitivity analysis of the PRA.

α		No NS	No Infe.	No Div.	PRA
0.3	Gap	4.10	0.46	0.28	–
	TBS	5.80	4.98	3.62	3.57
0.5	Gap	2.72	0.22	0.17	–
	TBS	5.60	4.42	3.37	3.09
0.7	Gap	1.80	0.19	0.09	–
	TBS	5.55	4.44	3.36	3.08

The experiments confirm the importance of each component for solving the VRPRDD with the proposed PRA. Considering the results apart from those where NS is disabled, we note that all variants of the method still have better average performance than the ILS. The most important component tested appears to be the NS, but allowing solutions that are infeasible with respect to capacity and incorporating a diversity measure also play an important role in the performance. Removing the different components appears to have a similar effect for the different values of α, but the results suggest these components are most critical to performance for lower values of α. Note that the effect of disabling the components may be exaggerated because the algorithm parameters are not changed accordingly.

Considering infeasible solutions and an adaptive penalty weight both improves the objective value and significantly decreases the TBS values. This supports the results from preliminary experiments, and the conjecture on the utility of infeasible solutions for heuristics, as described in § 6.2.1. The diversity measure used in the fitness function appears to contribute the least of the components tested, although when α is lower, the gap is almost equal for either disabling the infeasibility or diversity. This suggests both components are important for minimizing the weighted tardiness, but the diversity less so when the total distance becomes more dominant.

To promote comparability between the PRA and ILS, we have used a computational time limit as the stopping criterion. To analyse the progress of the PRA independently of the computer used, and to understand the effectiveness of the stopping criterion, we have performed experiments using different stopping criteria. In Table 6.5, the results for 600, 800 and 1000 iterations of the PRA are displayed in order. The results for each instance are taken from 10 independent runs of the PRA. In the table, Gap is the average percentage gap between the best objective value achieved and that of the PRA with the standard stopping criterion, and T is the total running time in minutes.

Table 6.5: Sensitivity analysis of the stopping criterion

α		600it	800it	1000it
0.3	Gap	0.05	0.02	0.00
	T (min)	3.12	4.17	5.21
0.5	Gap	0.03	0.02	0.02
	T (min)	2.81	3.73	4.61
0.7	Gap	0.02	0.01	0.00
	T (min)	2.95	3.90	4.85
Av.	Gap	0.03	0.02	0.01
	T (min)	2.96	3.93	4.98

The results show that the performance of the PRA increases with the number of iterations performed, and that similar results can be achieved in less time if an iteration limit is used as the stopping criterion. When α is smaller, a greater number of iterations are required to achieve similar results, and each iteration requires more computational time.

The parameters for the PRA have been independently calibrated for each value of α. We have performed some experiments to analyse the effect of using the parameters for $\alpha = 0.5$ for all values of α, and how the performance compares. In Table 6.6, we have summarised these results for each value of α. The results are taken from 10 independent runs for each instance. In the colum titles, Gap represents the average percentage difference between the

objective values obtained and those from the PRA with the calibrated parameters, and TBS is the average time at which the best solution was found during the run. In these experiments we have reduced the floating point tolerance from $1e-9$ to $1e-6$, and this reduces the total running time of the PRA proportionately.

Table 6.6: Sensitivity analysis of algorithm parameters

	PRA	PRA fixed param.	
α	TBS	TBS	Gap (%)
0.3	2.19	2.13	0.01
0.5	1.76	-	-
0.7	2.06	1.96	0.00
Av.	2.00	2.05	0.01

These experiments show that the performance is almost identical using the same set of parameters for all values of α. This also suggests a robustness in the algorithm to the choice of parameters, in terms of the objective values obtained and the computational time required. As we have found in many of the other experiments the negative effect of using a single set of parameters is most pronounced when α is smaller and the scheduling objective is more heavily weighted.

6.5 Conclusion

In this chapter, we propose a path-relinking algorithm (PRA) to address the vehicle routing problem with release and due dates (VRPRDD), and an iterated local search algorithm (ILS) as a comparator. The PRA builds on features of recently proposed heuristics for similar problems, although notably it is conceptually more simple than many. Some contributions include the efficient neighbourhood search and evaluation procedures, the path-relinking procedure, and the balanced edge- and arc-based solution distance used in measuring population diversity.

Observing the results from these heuristics, there are significant effects associated with introducing release dates and weighted tardiness into a vehicle routing problem (VRP). Also, the performance of the PRA is shown to dominate the ILS for the benchmark set introduced, considering either the average time to find the best solution or the average or best objective values. Moreover, the results support the growing interest in more sophisticated and population-based heuristics for VRPs. This initial and comprehensive investigation of heuristics for the VRPRDD should prove valuable in initiating further research of the problem. Moreover, the insights yielded from the analyses presented should prove helpful in the future for designing exact algorithms and heuristics for this and similar problems.

The relative computational time of the heuristics seems acceptable in a practical setting, and is similar to heuristics for related VRPs. Considering the practical nature of the VRPRDD, an interesting extension of this work would be to use the heuristics in a case-study of real instances. Population-based heuristics are popular for multi-objective optimisation, the PRA already considers multiple objectives during population management, and this suggest

an interesting extention to address the multi-objective version of the problem. Furthermore, the PRA is shown to have competitive performance for the asymmetric capacitated VRP by simply changing the parameters, and this warrants further investigation into the performance of the PRA for other VRPs.

Bibliography

Ergun, Ö. and Orlin, J. B. (2006). Fast neighborhood search for the single machine total weighted tardiness problem. *Operations Research Letters*, 34(1):41–45.

Glover, F. (1989). Tabu search - part 1. *ORSA Journal on Computing*, 1(2):190–206.

Glover, F. and Hao, J.-K. (2011). The case for strategic oscillation. *Annals of Operations Research*, 183(1):163–173.

Glover, F., Laguna, M., and Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–95.

Hashimoto, H. and Yagiura, M. (2008). A path relinking approach with an adaptive mechanism to control parameters for the vehicle routing problem with time windows. In Hemert, J. and Cotta, C., editors, *Evolutionary Computation in Combinatorial Optimization*, pages 254–265. Springer, Berlin, Heidelberg.

Kindervater, G. and Savelsbergh, M. (1997). Vehicle routing: Handling edge exchanges. In Aarts, E. and Lenstra, J. K., editors, *Local Search in Combinatorial Optimisation*, pages 337–360. Wiley.

Lin, S. (1965). Computer solutions of the travelling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269.

Lourenço, H. R., Martin, O. C., and Sttzle, T. (2010). Iterated local search: Framework and applications. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, volume 146, pages 363–397. Springer, New York.

Mercer, R. and Sampson, J. (1978). Adaptive Search Using a Reproductive Meta-Plan. *Kybernetes*, 7(3):215–228.

Nagata, Y. and Bräysy, O. (2008). Efficient local search limitation strategies for vehicle routing problems. *Evolutionary Computation in Combinatorial Optimisation*, 4972:48–60.

Reghioui, M., Prins, C., and Labadi, N. (2007). Grasp with path relinking for the capacitated arc routing problem with time windows. In Giacobini, M., editor, *Applications of Evolutionary Computing*, pages 722–731. Springer, Berlin, Heidelberg.

Resende, M., Ribeiro, C., Glover, F., and Mart, R. (2010). Scatter search and path-relinking: Fundamentals, advances, and applications. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 87–107. Springer, New York.

Smit, S. and Eiben, A. (2009). Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the 11th Congress on Evolutionary Computation*, pages 399–406, Picastaway, NJ. IEEE Press.

Sörensen, K. and Schittekat, P. (2013). Statistical analysis of distance-based path relinking for the capacitated vehicle routing problem. *Computers & Operations Research*, 40(12):3197–3205.

Taillard, E. P., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186.

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. (2015). Timing problems and algorithms: Time decisions for sequences of activities. *Networks*, 65(2):102–128.

Chapter 7

A column generation algorithm

7.1 Introduction

As discussed in Chapter 5, solving vehicle routing problems (VRPs) to optimality has considerable interest and a variety of formulations are proposed in the literature. The lower bound provided by polynomial-size formulations is often too weak to solve practical instances using a branch and bound algorithm, and this is also true of the VRP with release and due dates (VRPRDD) as shown in Chapter 5.

To achieve a tighter lower bound, a Dantzig-Wolfe reformulation is proposed, where the pricing problem (PP) is an elementary shortest path problem with resource constraints (ESPPRC) and release and due dates. To solve the formulation efficiently, a column generation (CG) algorithm and two dynamic programming (DP) formulations of the pricing problem (PP) are proposed. The formulations of the PP model the dependency between the time that a vehicle departs from the depot and the weighted tardiness of the assigned customers as a pointwise function of the release dates, or by decomposing the states over the release dates.

The efficiency of the CG algorithm relies on a multi-phase pricing algorithm that utilises dynamic programming (DP) heuristics to find columns quickly and a decremental state-space relaxation (DSSR) algorithm that exploits an efficient label-setting algorithm to solve an ng-route relaxation at each iteration. A comprehensive computational evaluation and comparison is performed using the proposed benchmark instances with $n = 20$ or 30.

The remainder of the chapter is organized as follows. The Dantzig-Wolfe reformulation and the definition of the pricing problem are given in § 7.2. § 7.3, describes the CG algorithm in detail, and computational experience is reported in § 7.4. Lastly, § 7.5 concludes our findings.

7.2 Set covering formulation

In the OC formulation of Chapter 5, the constraints (5.19) and (5.20) and link the variables associated to each route, and the remaining constraint decompose into identical subsets, one for each route. This suggests a Dantzig-Wolfe reformulation with constraints (5.19) and (5.20) defining the *master problem*, and the remaining constraints defining the PP.

To simplify the presentation, we define $a^+ = \max\{0, a\}$, and scale the c_{ij}- and w_i-values by α and $(1 - \alpha)$, respectively. Let Ω be the set of feasible routes, and for $r \in \Omega$, let c^r be

the objective value of r, and a_i^r be a binary indicator equal to one if $i \in V'$ is visited by r and $a_0^r \equiv 0$. The *master problem* can then be formulated as follows.

$$z(\text{LP}) = \min \sum_{r \in \Omega} c^r \lambda^r, \tag{7.1}$$

$$\text{s.t.} \sum_{r \in \Omega} a_i^r \lambda^r = 1, \qquad \forall\, i \in V', \tag{7.2}$$

$$k(V') \leq \sum_{r \in \Omega} \lambda^r \leq m, \tag{7.3}$$

$$\lambda^r \geq 0, \qquad \forall\, r \in \Omega. \tag{7.4}$$

The objective function sums the objective value of each route in the solution, and as a side-effect linearises the weighted tardiness. Covering constraints (7.2) require that each customer is visited once, and convexity constraints (7.3) ensure a feasible number of routes are selected. Notice that constraint (5.21) is also included in the master problem. The domain of the λ-variables is non-negative and $\lambda^r \leq 1$ is implied. For further details on the process, we refer the reader to the relevant section in Chapter 2.

LP is the linear relaxation of a set-partitioning formulation of the VRPRDD, which we will denote by SP. Constraints (7.2) can be relaxed to obtain the linear relaxation of a set-covering formulation. This allows more than one visit to each customer, and the constraint are reformulated as follows.

$$\sum_{r \in \Omega} a_i^r \lambda^r \geq 1, \forall\, i \in V', \tag{7.5}$$

We denote the set-covering formulation as SC and its linear relaxation as LC. If we assume that the c_{ij} and τ_{ij} values satisfy the triangle inequality and G is complete, the optimal solutions of SC and SP are equal. Continuing with the previous assumptions, the optimal solution of LC is a lower bound for LP, which is in turn a lower bound for the VRPRDD. Mathematically this can be expressed as

$$z(\text{LC}) \leq z(\text{LP}) \leq z(\text{SC}) = z(\text{SP}). \tag{7.6}$$

If the previous assumptions are not satisfied, then SC is a lower bound on SP, but (7.6) is now stated as follows and LC is a weaker bound on LP and therefore the VRPRDD.

$$z(\text{LC}) \leq z(\text{LP}), z(\text{SC}) \leq z(\text{SP}). \tag{7.7}$$

Let γ_0 be a free dual-variable, and γ_i for $i \in V'$ be non-negative dual-variables, which are associated to constraints (7.3) and (7.5) respectively. Observe that the dual of LC can be stated as follows.

$$z(\text{DC}) = \max \sum_{i \in V'} \gamma_i + m\gamma_0, \tag{7.8}$$

$$\text{s.t.} \sum_{i \in V} a_i^r \gamma_i \leq c^r, \qquad \forall r \in \Omega, \tag{7.9}$$

$$\gamma_i \geq 0, \qquad \forall i \in V'. \tag{7.10}$$

Note that the dual of the LP formulation is a relaxation of DC and is obtained by removing constraints (7.10). Although LP provides a tighter lower bound on the VRPRDD, the dual variables are unrestricted in sign and therefore more unstable. For these reasons, we have chosen to use the LC formulation.

Property 7.1. *If G is complete and the τ_{ij} and c_{ij} values satisfy the triangle inequality, in any optimal solution to LC each visit to a customer must have a negative contribution to the reduced cost.*

Proof. Let us assume to the contrary that x is an optimal solution of an instance of LC, and features a visit to a customer which increases the reduced cost. This customer can be removed from the solution without worsening the feasibility or objective value, a contradiction. □

G can no longer be assumed to be complete if a branching constraint for an arc has been added to LC. Additionally, it may be possible to remove arcs between customers which will not be present in a feasible or optimal solution, such as $\{(i,j) \in A \colon q_i + q_j > Q\}$.

In the simplex method, all of the variables in a formulation and the associated columns are considered explicitly, and usually the reduced cost of each variable is calculated in every iteration. This approach is intractable for LC because of the number of variables, instead, we propose a CG algorithm that considers a restricted set of columns and generates others as necessary. This process iterates between solving the master problem with a restricted set of columns, or the restricted master problem (RMP), and generating columns with negative reduced cost that are added to the restricted set of columns. If no such column exists, then the complimentary-slackness conditions of linear programming ensure that the solution is optimal. Therefore, a pricing problem (PP) must be defined to find a column of negative reduced cost or thus prove that one does not exist.

7.2.1 Pricing problem

The PP for LC or LP is an ESPPRC with release and due dates. To simplify the presentation, let the reduced arc costs be defined as follows, where $\gamma_0 = 0$ because it is constant in the PP..

$$\bar{c}_{ij} = c_{ij} - 1/2(\gamma_i + \gamma_j), \qquad \forall (i,j) \in A, \tag{7.11}$$

The PP can then be formulated as follows.

$$z(\text{PP}) = \min \sum_{(i,j)\in A} \bar{c}_{ij} x_{ij} + \sum_{i \in V'} w_i z_i \tag{7.12}$$

s.t.

$$\sum_{j \in V'} x_{ij} \leq 1, \qquad \forall\, i \in V' \tag{7.13}$$

$$\sum_{j \in V'} x_{0j} = \sum_{j \in V'} x_{j0} = 1, \tag{7.14}$$

(5.22)-(5.30).

The objective function defines the reduced cost of a column, and other than the constraints from the OC formulation, the PP also includes constraints (7.13) that are implied in the OC formulation, and constraint (7.14) that is decomposed from constraint (5.21). This problem is unfortunately still unary NP-hard, and similarly to the proof of complexity for the VRPRDD in Chapter 3, it can be proven by reduction to the TSP for $0 < \alpha \leq 1$, and the Hamiltonian path problem for $\alpha = 0$.

7.3 Column generation algorithm

Algorithm 7.1 presents the skeleton of our CG algorithm. We define P as the current column pool of the RMP; z_{RMP} as the optimal objective value of the RMP, which is an upper bound on the master problem; c as a column, and $\text{rc}(c)$ as its reduced cost; C as a set of columns; z_P as the optimal objective value of the PP, which represents the most negative reduced cost; and UB as an upper bound on the original problem. The algorithm relies on two parameters that trigger column management, if the size of P exceeds ρ or the time taken to solve the RMP exceeds t_{LP} seconds.

Algorithm 7.1 Column generation algorithm

```
1: Set the parameter values t_LP, ρ, and UB
2: Set LB= 0, and P to the set of initial columns
3: repeat
4:     Solve RMP with columns P (Linear programming)
5:     if solving RMP takes more than t_LP time or |P| > ρ then
6:         Remove columns c ∈ P: rc(c) >UB−LB
7:     Solve PP with dual variables to find columns C (Pricing algorithm)
8:     Set P = P ∪ C
9:     if optimal column found then
10:        if z_RMP + m z_P > LB then
11:            LB = z_M + m z_P
12: until z_P ≥ 0
13: return RMP current solution, LB
```

First, P is initialised with a set of $n + 1$ artificial columns with high cost, one for each constraint, to ensure that the RMP always has a feasible solution. The algorithm is also hot-started with the routes from the best solution found by the PRA presented in Chapter 6.

The RMP is then solved to produce the initial dual-variables, and using these the PP is solved to find columns of negative reduced cost. If the PP is solved to optimality, then the Lagrangian lower bound can be considered to improve LB(Lines 10-11). The RMP is solved with the new P, and this process continues until $z_P \geq 0$ and no columns of negative reduced cost exist. If solving the RMP in any iteration requires at least t_{LP} seconds or $|P| \geq \rho$, the columns in P with reduced cost greater than the duality gap are removed.

7.3.1 Pricing algorithm

A CG algorithm will require the PP to be solved a large number of times, and the computation time required will often dominate the global computational time. Heuristics for the PP are often used to generate columns of negative reduced cost more efficiently, although these heuristics become progressively less effective as the CG algorithm continues and the dual variables stabilise. We propose a two-phase variable neighbourhood descent, applying more complex pricers until a column with negative reduce cost is found. Algorithm 7.2 presents a skeleton of our pricing algorithm.

Algorithm 7.2 Pricing algorithm

1: Set the parameter value phase and $C = \emptyset$
2: **if** phase $= 0$ **then**
3: Use heuristic pricer one and put columns in C
4: **if** $C = \emptyset$ **then**
5: Use heuristic pricer two and put columns in C
6: **if** $C = \emptyset$ **then**
7: Set phase $= 1$
8: Use exact pricer and put columns in C
9: **return** C and phase

When the heuristics fail to find any columns with negative reduced cost, an exact pricer is applied to find a column or prove that one does not exist. Considering the decaying effectiveness of the heuristic pricers, the first heuristic is no longer applied once the second heuristic fails for the first time.

7.3.2 Formulations of the pricing problem

Following conventions for similar problems, we pose the PP as an elementary shortest path problem with resource constraints (ESPPRC) between sink and source nodes. The source is defined as the depot 0 with the ingoing arcs are removed, and the sink is defined as a dummy depot $n + 1$ with the outgoing arcs removed. Solutions are then capacity-feasible paths between the source and sink $(0, i_1, \ldots, i_k, n + 1)$, which visit a permutation of $k \leq n$ customers. DP is a popular approach to solve similar ESPPRCs, and implicitly enumerates the feasible routes by progressively extending incomplete paths originating at the source.

There is no restriction on the cost of arcs in the PP, and negative cost cycles will often exist because the dual variables are unstable in the earlier iterations of a CG approach. As a consequence, elementary paths are not guaranteed by cost minimisation, although finiteness of the paths is implied by both the capacity constraint, and the combination of positive travel

times and weights, and the due dates. If a path that departs from the depot at time t is extended to a customer $i \in V'$ such that $r_i > t$, the departure time of the resulting path is increased to r_i, and an increase in departure time from the depot may result in more tardiness for customers visited in the path. Considering the DP principle of optimality, any increase in the weighted tardiness for the customers previously visited in the path must be modelled by the state variables.

The weighted tardiness for customers in a path is a non-decreasing piecewise linear function of the departure time from the depot. These functions have breakpoints at which the gradient changes, and these are the times that departing from the depot incurs additional weighted tardiness for customers in the path. For example, the function for a path that only visits customer $i \in V'$, has a single breakpoint at $d_i - (r_i + \tau_{0i})$ and the gradient of the following interval is w_i. Idle time prior to departing from the depot is not profitable, and therefore a path must depart at one of the customer release dates $R = \{r_i \colon i \in V'\}$. It is therefore sufficient to represent the weighted tardiness of a path as a non-decreasing pointwise function of the release dates. For further clarity, a depiction of the functions for a path that visits customers i and k can be seen in Figure 7.1, where the line represents a piecewise linear weighted tardiness function, and the points represent the corresponding pointwise weighted tardiness function.

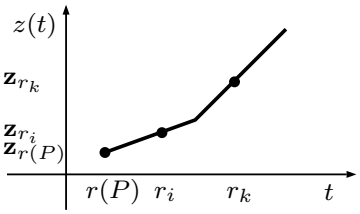

Figure 7.1: Illustration of the weighted tardiness of a path as a function of the departure time from the depot.

We propose two DP formulations for the PP; the first defines of states to include a vector of length $|R|$, representing the increase in the weighted tardiness of a state if the departure time from the depot is increased to a later release date, and the other relies on the observation that the weighted tardiness function decomposes over R. Backward DP is not possible because the optimal state is unknown for either definition of states and tight bounds are not available. Therefore, a forward-reaching DP algorithm is used, following the general approach proposed by Desrochers and Soumis (1988) for the SPPRC. More details on the algorithm used to solve the DP formulations are given in § 7.3.3.

Tardiness function formulation

For the tardiness function (TF) formulation, let a state σ^k be defined by the set of state variables $(\mathbf{z}^k, L^k, s^k, i^k)$, representing a path $P = (0, L^k \backslash \{0, i^k\}, i^k)$ that originates from the depot and visits any permutation of $L^k \subseteq V_+$ such that 0 is the origin and $i \in L^k$ is visited last, and the path has duration s. The $\mathbf{z}^k$-variable is a vector of length $|R|$, where each element z_t^k for $t \in R$ such that $t \leq r(L^k)$ is equal to zero, and for $t \in R$ such that $t > r(L^k)$ is equal to the increase in the weighted tardiness of the path if the departure time is increased to t. The superscript k will be omitted if it is unnecessary in the context. For brevity, let $q(A) = \sum_{i \in A} q_i$ and $r(A) = \max_{i \in A}\{r_i\}$

The gradients of the piecewise weighted tardiness functions change at the breakpoints. When a state $(\mathbf{z}, L, s, i)$ is extended to $j \in V'$, a breakpoint is created at time $b = d_j - (s + \tau_{ij})$, if one does not exist, and the gradients of the following intervals are increased by w_j. For the pointwise weighted tardiness function captured in $\mathbf{z}$, introducing a customer j creates a new vector $\mathbf{z}'$, which is related as follows.

$$z'_t = \begin{cases} z_t - z_{r(L\cup\{j\})}, & r(L \cup \{j\}) \leq t \leq b, \\ z_t - z_{r(L\cup\{j\})} + w_j(t-b), & t > b, \\ 0, & \text{otherwise.} \end{cases} \tag{7.15}$$

The associated state-space graph $H(\mathcal{V}_1, \mathcal{A}_1)$, is defined by the following sets of nodes and edges.

$$\mathcal{V}_1 = \{(\mathbf{z}, L, s, i) \colon \mathbf{z} \text{ defined as above}, L \subseteq V_+, q(L) \leq Q, \tau_{0i} \leq s \leq T, \forall\, i \in L\}, \tag{7.16}$$

$$\mathcal{A}_1 = \{((\mathbf{z}', L\backslash\{i\}, s - \tau_{ji}, j), (\mathbf{z}, L, s, i)) \colon \mathbf{z} \text{ related to } \mathbf{z}' \text{ as defined in (7.15)}, \\ \forall\, j \in L\backslash\{i\}, \forall (\mathbf{z}, L, s, i) \in \mathcal{V}_1\}. \tag{7.17}$$

For a state $S \in \mathcal{V}_1$, we define $\Phi(S)$ as the set of predecessor states from $\mathcal{A}_1$. The recursion is then stated as follows.

$$f(\mathbf{z}, L, s, j) = \min_{\forall\, \Phi(\mathbf{z},L,s,i)} \{f(\mathbf{z}', L\backslash\{i\}, s - \tau_{ji}, j) + \mathbf{z}'_{r(L)} + \bar{c}_{ji} + \\ w_i(r(L) + s - d_i)^+\}, \forall\, (\mathbf{z}, L, s, i) \in \mathcal{V}, \tag{7.18}$$

This is initialised with $f(\mathbf{0}, \{0\}, 0, 0) = 0$ and $f(\mathbf{0}, \{0\}, s, 0) = \infty$, $0 < s \leq T$, and the optimal solutions are found from $f(\mathbf{z}, L, s, n+1)$.

Bounding the magnitude of the vectors $\mathbf{z}$ for a path is difficult, because a closed form expression is difficult to find without loosening the bound. It also requires an upper bound on the weighted tardiness, such as $C = T \sum_{i \in V'} w_i$. We know from Property 7.1 that the reduced cost of a route must be negative, and therefore a lower bound on the reduced arc costs of a route can also be used. The number of possible vectors $\mathbf{z}$ for any state can be bounded as follows.

$$O\left(\sum_{i=0}^{|R|} \binom{C}{|R|}\right) = O\left(\sum_{i=0}^{n} \binom{C}{n}\right) = O(2^C). \tag{7.19}$$

The TF formulation therefore has $O(2^{C+n}nT)$ states, but this does not reveal much about the expected number of states. The number of outgoing arcs in the state-space graph for any node other than those associated to the depot is at most $n-1$, and in total $O(|A|)$, and the forward extension between two nodes takes $O(n)$ time. The time complexity of a DP algorithm that in the worst-case treats every state once is therefore $O(n|A|T2^{C+n})$.

From (7.17), it is clear that extending a state $(\mathbf{z}, L, s, i)$ by appending an arc $\{(i, j) \in A$ such that $j \in V_+\}$ produces the state $(\mathbf{z}', L \cup \{j\}, s + \tau_{ij}, j)$, where $\mathbf{z}'$ can be constructed using

(7.15). Note also that an extension is only feasible if the following conditions are satisfied.

$$j \notin L, \tag{7.20}$$

$$q(L) + q_j \leq Q. \tag{7.21}$$

The efficiency of any algorithm to solve a DP formulation depends on the number of generated states, and it is therefore essential to avoid unnecessarily extending states. If two states are equal, then it is sufficient to retain one copy of the state if one optimal solution is sufficient. Furthermore, given two states σ^1 and σ^2, then σ^1 dominates σ^2 or $\sigma^1 \prec \sigma^2$ if it is proven that extending both states is unnecessary.

Proposition 7.1. *Let $\sigma^1, \sigma^2 \in \mathcal{V}_1$, be two distinct states such that $i^1 = i^2$. If the following conditions are satisfied, then σ^1 dominates σ^2, or $\sigma^1 \prec \sigma^2$.*

$$s^1 \leq s^2, \tag{7.22}$$

$$L^1 \subseteq L^2, \tag{7.23}$$

$$f(\sigma^1) + z_t^1 \leq f(\sigma^2) + z_t^2, \qquad \forall\, t \in R\colon t \geq r(L^2). \tag{7.24}$$

Proof. It is easy to verify that any feasible extension of the path encoded by σ^2 is also feasible for σ^1. The objective value of σ^2 resulting from a feasible extension is at least equal to that of σ^1; this is guaranteed for the cost of the customer j by (7.22) and (7.23), and for the existing path and any increase to the departure time from the depot by (7.24). □

Conditions (7.23) and (7.24) require $O(n)$ and $O(|R|) = O(n)$ time to verify, respectively, and therefore checking the dominance relation for two states has linear complexity. Conditions (7.23) cannot be true unless $r(L^1) \leq r(L^2)$, $q(L^1) \leq q(L^2)$, and $|L^1| \leq |L^2|$, and these additional conditions can be used to reduce the number of times that (7.23) is verified.

As pointed out by Feillet et al. (2004), if a constrained variable does not improve by extension, then extensions to customers that are infeasible from a given state due to this variable will never be feasible for states descended from the given state. For a given state σ, let $U = \{j \in V'\colon j \notin L, q(L) + q_j > Q\}$. A new definition of states is now the set of variables $(\mathbf{z}, L, U, s, i)$, where $L \cup U$ is used rather than L to test for equality of states and feasibility of an extension, and the set of these states $\mathcal{V}_1'$. It is interesting to note that the worst case bound on the number of states is unchanged by introducing the U-variable, but weaker dominance conditions can be described as follows.

Proposition 7.2. *Let $\sigma^1, \sigma^2 \in \mathcal{V}_1'$, be two distinct states such that $i^1 = i^2$. If the following conditions are satisfied then $\sigma^1 \prec \sigma^2$.*

$$s^1 \leq \begin{cases} s^2 + r(L^2) - r(L^1), & r(L^1) > r(L^2), \\ s^2, & r(L^1) \leq r(L^2), \end{cases} \tag{7.25}$$

$$q(L^1) \leq q(L^2), \tag{7.26}$$

$$L^1 \cup U^1 \subseteq L^2 \cup U^2, \tag{7.27}$$

$$f(\sigma^1) + z_t^1 \leq f(\sigma^2) + z_t^2, \qquad \forall\, t \in R\colon t \geq r(L^2). \tag{7.28}$$

It is easy to verify the correctness of this dominance relation in a similar fashion to Proposition 7.1, although the conditions on the load and departure time of the states are no longer implied, because (7.23) is replaced by (7.27). Therefore, (7.26) is required and (7.22) is modified to produce (7.25). The latter condition ensures that the time that σ^2 arrives at a customer from a feasible extension is at least equal to the time that σ^1 arrives at the same customer.

Release date decomposition formulation

The following DP formulation decomposes the PP into $O(n)$ sub-problems, one for each release date $r \in R$, and will be called the *release date decomposition* (RD) formulation. Let a state be defined by the set of variables (L, r, s, i), representing a path $(0, L\backslash\{i\}, i)$ originating at the source and departing at time $r \in R$, that visits any permutation of $L \subseteq V'_+$ such that the customer or depot $i \in L$ is visited last and the path has duration s.

The departure time from the depot is fixed for each state, and a feasible route can only include customers from the set $\{i \in V' \colon r_j \leq r\}$, but must include one from the subset $\{i \in V' \colon r_i = r\}$. We define $u(L, r)$ as a binary indicator equal to one if $r(L) = r$, and a state σ such that $u(L, r) = 1$ will be described as *strongly feasible*, and otherwise as *weakly feasible*. Again, let $q(L) = \sum_{j \in L} q_j$, $r(L) = \max_{j \in L}\{r_j\}$ and $a^+ = \max\{0, a\}$.

The state-space graph $H_2(\mathcal{V}_2, \mathcal{A}_2)$ associated to the RD formulation is defined by the following sets of nodes and edges.

$$\mathcal{V}_2 = \{(L, r, s, i) \colon L \subseteq V'_+, q(L) \leq Q, r \in R, r(L) \leq r, \tau_{0i} \leq s \leq T, i \in L \cup \{0\}\}, \tag{7.29}$$

$$\mathcal{A}_2 = \{((L\backslash\{i\}, r, s - \tau_{ji}, j), (L, r, s, i)) \colon j \in L\backslash\{i\}, (L, r, s, i) \in \mathcal{V}_1\}. \tag{7.30}$$

The recursion is then stated as follows.

$$f(L, r, s, i) = \min_{j \in L\backslash\{i\}} \{f(L\backslash\{i\}, r, s - \tau_{ji}, j) + \bar{c}_{ji} + w_i(r + s - d_i)^+\}, \quad \forall (L, r, s, i) \in \mathcal{V}_2. \tag{7.31}$$

This is initialised by $f(\{0\}, r, 0, 0) = 0$ and $f(\{0\}, r, s, 0) = \infty$ for $r \in R$ and $0 < s \leq T$. The optimal solutions are the strongly feasible routes with lowest objective value and are given by the following expression.

$$\min_{u(L,r)=1} \{f(L, r, s, n + 1)\}. \tag{7.32}$$

There are $O(n^2 T 2^n)$ states or nodes in $\mathcal{V}_1$. Again, the number of outgoing arcs of any node other than those associated to the depot is $O(n)$, in total $O(|A|)$, and computing the forward extension between two nodes takes $O(n)$ time. Therefore, the time complexity of a DP algorithm that treats at most each state once is $O(|A| n^2 T 2^n)$.

A state is again extended by appending an arc $(i, j) \in A$, and similarly produces the state $(L \cup \{j\}, r, s + \tau_{ij}, j)$. An extension is feasible if and only if constraints (7.20), (7.21), and the following constraint are satisfied.

$$r_j \leq r. \tag{7.33}$$

Proposition 7.3. *Let $\sigma^1, \sigma^2 \in \mathcal{V}_2$, be two distinct states such that $i^1 = i^2$, $r^1 = r^2$, and $u(L^1, r^1) = u(L^2, r^2)$. If the following conditions are also satisfied, then $\sigma^1 \prec \sigma^2$.*

$$s^1 \leq s^2, \tag{7.34}$$

$$L^1 \subseteq L^2, \tag{7.35}$$

$$f(\sigma^1) \leq f(\sigma^2). \tag{7.36}$$

Proof. It is easy to verify that any feasible extension of σ^2 is also feasible for σ^1. Both $r^1 = r^2$ and $u(L^1, r^1) = u(L^2, r^2)$, and this is due to the following observations.

1. To ensure constraint (7.33) is satisfied when extending σ^1 to any feasible extension of σ^2, then $r^1 \geq r^2$.
2. Condition (7.35) implies $r(L^1) \leq r(L^2)$, and if $r^1 \geq r^2$ then additionally $u(L^1, r^1) \leq u(L^2, r^2)$.
3. If $u(L^1, r^1) < u(L^2, r^2)$, then any extension of σ^1 may be weakly feasible, but σ^2 and therefore any descendants are strongly feasible.

Lastly, the reduced cost of σ^2 is at least equal to that of σ^1 from condition (7.36). Referring to (7.31), the difference in the change of reduced cost resulting from any extension of σ^1 compared to the same extension of σ^2 is proportionate to $s^1 + r^1 - s^2 - r^2$. We know $r^1 = r^2$ and $s^1 - s^2 \leq 0$ from (7.34), and this completes the proof. □

Verifying (7.23) requires $O(n)$ time, and therefore the time complexity of evaluating the dominance relation is $O(n)$. Notice that again the last condition cannot be true unless $q(L^1) \leq q(L^2)$, $r(L^1) \leq r(L^2)$ and $|L^1| \leq |L^2|$, and these additional conditions can be used to reduce the number of times that (7.23) is verified.

Proposition 7.4. *Let $\sigma^1, \sigma^2 \in \mathcal{V}_2'$, be two distinct states such that $i^1 = i^2$ and $u(L^1, r^1) = 1$. If the following conditions are satisfied then $\sigma^1 \prec \sigma^2$.*

$$s^1 + r^1 \leq s^2 + r^2, \tag{7.37}$$

$$q(L^1) \leq q(L^2), \tag{7.38}$$

$$L^1 \cup U^1 \subseteq L^2 \cup U^2, \tag{7.39}$$

$$f(\sigma^1) \leq f(\sigma^2). \tag{7.40}$$

If $u(L^1, r^1) = 0$, then $\sigma^1 \prec \sigma^2$ if and only if $u(L^2, r^2) = 0$ and $r^1 = r^2$.

The proof of correctness for Proposition 7.4 is similar to the proof of Proposition 7.3, if L^1 and L^2 are replaced by $L^1 \cup U^1$ and $L^2 \cup U^2$, respectively. However, (7.39) does not imply (7.38), and the latter is therefore required. Further, strongly feasible states can dominate both strongly and weakly feasible states with any departure time, but weakly feasible states can only dominate weakly feasible states with the same departure time. This is because for any feasible extension of σ^2 through arc (i, j), then $r_j \leq r^1$ is implied for σ^1 by the set U^1.

Notice that (7.39) can only be true if $|L^1 \cup U^1| \leq |L^2 \cup U^2|$, and this can be similarly used to improve the average computation time of the dominance procedure. For states $\sigma^1, \sigma^2 \in \mathcal{V}_1'$, the relation $\prec^l$ is defined by comparing, in order, the values of the s-variable, $q(L^1)$ and

$q(L^2)$, the vectors representing the L-variables, and the reduced costs. Again the dominance conditions can be weakened by considering earlier departure times from the depot for weakly feasible states, but the lack of transitivity is even more pronounced because strongly feasible states can dominate weakly feasible states.

7.3.3 Labelling algorithm

It is easy to verify that all of the dominance relations defined above give partial orders on the respective sets of states, but some of the states are incomparable. A label-setting algorithm requires a total order that preserves the dominance through the existence of a state variable that cannot be improved by extension. This is described as the qualifying condition of a label-setting algorithm for an SPPRC by Desrochers and Soumis (1988), who show that if states are extended in such an order then the states that are not dominated when a stage is treated, will never be dominated.

For example, a total order that preserves dominance for the states $\mathcal{V}_1$ can be defined using the following relation. For $\sigma^1, \sigma^2 \in \mathcal{V}_1$ then $\sigma^1 \prec^s \sigma^2$ if $s^1 < s^2$. Notice that the value of the s-variable increases when a state is extended, because the τ_{ij} values are strictly positive. Therefore, the $\prec^s$ order is dominance preserving, or mathematically, $\sigma_1 \prec^s \sigma^2 \rightarrow \sigma^2 \not\prec \sigma^1$. If the stages of the DP formulation are taken to be the increasing values of the s-variable, then it can be solved by algorithms requiring time complexity that is polynomial in the number of states and n.

More generally, $\sigma^1 \prec^l \sigma^2$ if σ^1 is lexicographically less than σ^2, comparing the values of the variables involved in domination that satisfy the qualifying condition, followed by the remaining variables involved in domination. This relation is a generalisation and extension of the lexicographic relations proposed by Desrochers and Soumis (1988) and Boland et al. (2006), respectively. We note that for the purposes of computational efficiency, the sets L or equivalently θ are not considered in the lexicographic comparison of states.

To improve the efficiency of the algorithm, the number of states that each state is compared to for dominance is limited by two parameters p_T and p_Q. When a state σ^1 is created it is compared to states σ^2 such that $s^2 \geq s^1 - T/p_T, q^2 \geq q^1 - Q/p_Q$ to test if it is dominated, and states σ^2 such that $s^2 \leq s^1 + T/p_T, q^2 \leq q^1 + Q/p_Q$ to test if it dominates.

For a given DP formulation, let $l(s)$ be a lower bound on the reduced cost of final states reachable from a state s, U be an upper bound on the optimal objective value, K be the set of stages in order, where $K(s)$ be the stage of state s, and let us decompose the set of states into subsets S_i^k for $k \in K$ and $i \in V_+$. Algorithm 7.3 presents a general label-setting algorithm (GLA) in the style of Desrochers and Soumis (1988), which can be applied to all of the DP formulations presented. The functions feasible(σ, i) and extend(σ, i) test the feasibility of extending the state σ to $i \in V'_+$ and create the resulting state, respectively. The function dominates(σ_1, σ_2) tests if $\sigma_1 = \sigma_2$ or $\sigma_1 \prec \sigma_2$, and solutions(S) returns the set of routes constructed by backtracking from the states in S. For compatibility of the lower bound and dominance tests, we require that for two states σ_1 and σ_2, if $\sigma_1 \prec \sigma_2$ then $l(\sigma_1) \leq l(\sigma_2)$.

The GLA begins by creating the initial states, and then adds each of these to the corresponding set of states. The algorithm then proceeds to treat the states of each stage in increasing order, testing the lower bound for each state before extending the state to the depot $n+1$ and all feasible customers. For each state σ_2, created by an extension, the lower

Algorithm 7.3 PLA: A general label-setting algorithm for the ESPPRC

1: Set $U = 0$, S_0^k to initial states and $S_i^k = \emptyset, \forall\, k \in K, i \in V'_+$.
2: **for all** $k \in K, i \in V$
3: **for all** $\sigma_1 \in S_i^k$
4: **if** $l(\sigma_1) \leq U$ **then**
5: **for all** $j \in V'_+$
6: **if** feasible(σ_1, j) **then**
7: Compute $\sigma_2 = \text{extend}(\sigma_1, j)$.
8: **if** $l(\sigma_2) \leq U$ **then**
9: Set $I = \textit{false}$.
10: **if** $j = n + 1$ **then**
11: Set $U = \min\{U, f(\sigma_2)\}$
12: **for all** $\sigma_3 \in S_j^p, p \in K : p \leq K(\sigma_2)$
13: $I = \text{dominates}(\sigma_3, \sigma_2)$
14: **if** $I = \textit{false}$ **then**
15: Add σ_2 to $S_j^{B(\sigma_2)}$
16: **for all** $\sigma_3 \in S_j^p, p \in K : p \geq K(\sigma_2)$
17: **if** dominates(σ_2, σ_3) **then**
18: Remove σ_3 from S_j^p.
19: **else**
20: Remove σ_1 from S_i^k.
21: **return** solutions$\left(\bigcup_{k \in K} S_{n+1}^k\right)$

bound is also tested. Following this, σ_2 is tested for being dominated by states ending at the same vertex in stages that are at most equal to the stage of σ_2. If σ_2 is dominated then it can be ignored, otherwise it is added to the respective set of states and may dominate any states that end at the same vertex in stages that are at least equal to the stage of σ_2.

Let us consider the time complexity of the GLA, and define $O(v)$ as the maximum number of states ending at any vertex and therefore $O(nv)$ is the maximum number of states. The first two loops (Steps 2 and 3) require at most $O(nv)$ iterations, treating each state once. For each iteration, the state to treat is found in constant time and is extended $O(n)$ times. Across all iterations there are $O(|A|v)$ extensions. Each extension requires computing a lower bound (Step 4), and comparing the state for dominance with at most $O(v)$ other states (Steps 12-18). The time complexity of removing any dominated states from the sets S_i^k and adding the extension state is $O(v)$ if the sets are implemented as linked lists and these changes are performed whilst checking the dominance. If $g(n)$ and $h(n)$ are the time complexities of the dominance and lower bound, respectively, then the time complexity of each extension is $O(g(n) + vh(n))$. For all the extensions, and therefore the GLA, the time complexity is $O(|A|v(g(n) + vh(n)))$, and if $g(n) \leq O(vh(n))$ it is $O(|A|v^2h(n))$.

The bound on the time complexity of the GLA is not very informative, but shows that the worst case time complexity of the algorithm is proportional to number of arcs, the square of the number of dominant states and the complexity of the dominance test. Notice that the improvements in running time from the reduction in the number of states due to dominance has no direct effect on the time complexity. Therefore, strategies such as reducing the number of states compared for dominance will improve the running time of the GLA most when the

dominance is not very effective and the bound on the number of states is tighter.

7.3.4 Decremental state-space relaxation

The number of states for either of the formulations presented above is exponential and this results in exponential time complexity. The PP can be relaxed to reduce the number of states, but this may result in infeasible solutions, and introducing these to the RMP may weaken the lower bound produced. The state variables representing the sets of customers which cannot be visited have the most significant impact on the number of states, but relaxing the requirement that routes are elementary (or simple), and the implied constraints, can produce a considerably weaker lower bound. To tighten this relaxation, Christofides et al. (1981) propose a technique described as state-space relaxation (SSR). This maps the set of customers that cannot be visited to a set of additional state variables representing the implied constraints or other valid constraints, and this increases the elementarity and therefore lower bound.

If the optimal solution to an SSR has at least one customer that is visited more than once, then the solution is infeasible for the original problem. To further tighten the lower bound by reducing the number of times that customers are visited, Righini and Salani (2008) and Boland et al. (2006) independently propose decremental state-space relaxation (DSSR). This approach has many parallels with cutting planes in linear and integer programming, and can be described as follows. After solving the non-elementary relaxation, a state variable is introduced or its domain is enlarged to cause at least one of the customers that is visited more than once to be visited less, without restricting any feasible solutions to the original problem. Solving the resulting DP produces a solution that is at least as feasible and provides as tight a lower bound on the optimal objective value for the original problem. Continuing in this fashion results in the optimal solution to the original problem in a finite number of iterations.

Relaxations

Following the notation of Christofides et al. (1981), we first define the (q, r)-route relaxation of the TF formulation. Let the L-variable be mapped to $q = q(L)$ and $r = r(L)$, the state-space graph is then defined by the following sets of nodes and edges.

$$\mathcal{V}_1^r = \{(\mathbf{z}, q, r, s, i) \colon \mathbf{z} \text{ as defined, } q_i \leq q \leq Q, r \geq r_i, r + \tau_{0i} \leq s \leq T, \forall\, i \in V'_+\}, \tag{7.41}$$

$$\mathcal{A}_1^r = \{((\mathbf{z}', q - q_i, r', s - \tau_{ji}, j), (\mathbf{z}, q, r, s, i)) \colon \mathbf{z} \text{ and } \mathbf{z}' \text{ related by (7.15)}, r' \leq r, \\ \forall\, j \in V_+ \backslash \{i\}, \forall (q, u, r, s, i) \in \mathcal{V}_1^r\}. \tag{7.42}$$

The recursion is now stated as follows.

$$f(\mathbf{z}, q, r, s, i) = \min_{r' \leq r, \forall\, j \in V \backslash \{i\}} \{f(\mathbf{z}', q - q_i, r', s - \tau_{ji}, j) + \mathbf{z}_r + \bar{c}_{ji} + \\ w_i(r + s - d_i)^+\}, \forall (\mathbf{z}, q, r, s, i) \in \mathcal{V}_1^r. \tag{7.43}$$

The time complexity of the resulting DP is $O(n^2|A|QT2^C)$. The dominance conditions are now similar to those in Proposition 7.2 if r^k and q^k are substituted for $r(L^k)$ and $q(L^k)$,

except that (7.27) is no longer required.

For the RD formulation, we define the (q, u)-route relaxation as follows. Let the L-variable be mapped to $q = q(L)$ and $u = u(L, r)$, the state-space graph is then defined by the following sets of nodes and edges.

$$\mathcal{V}_2^r = \{(q, u, s, r, i) \colon q_i \leq q \leq Q, u \in \{0, 1\}, r_i \leq r, r + \tau_{0i} \leq s \leq \bar{t}_i, \forall\, r \in R, \forall\, i \in V\}, \quad (7.44)$$

$$\mathcal{A}_2^r = \{((q - q_i, u', r, s - \tau_{ji}, j), (q, u, r, s, i)) \colon u' \leq u, \forall\, j \in V \backslash \{i\}, \\ \forall\, (q, u, r, s, i) \in \mathcal{V}_2^r\}. \quad (7.45)$$

The recursion is now stated as follows.

$$f(q, u, s, r, i) = \min_{u' \leq u, \forall\, j \in V' \backslash \{i\}} \{f(q - q_i, u', r, s - \tau_{ji}, j) + \bar{c}_{ji} + w_i(s - d_i)^+\}, \\ \forall\, (q, u, r, s, i) \in \mathcal{V}_2^r. \quad (7.46)$$

The time complexity of the relaxed formulation is $O(n|A|QT)$, which is pseudo-polynomial.

The dominance conditions are similar to those in Proposition 7.4, but are now stated as follows.

Proposition 7.5. *Let $\sigma^1, \sigma^2 \in \mathcal{V}_2^r$, be two distinct states such that $i^1 = i^2$ and $u^1 = 1$. If the following conditions are satisfied then $\sigma^1 \prec \sigma^2$.*

$$r^1 \geq r^2, \quad (7.47)$$

$$s^1 + r^1 \leq s^2 + r^2, \quad (7.48)$$

$$q^1 \leq q^2, \quad (7.49)$$

$$f(\sigma^1) \leq f(\sigma^2). \quad (7.50)$$

If $u^1 = 0$ then $\sigma^1 \prec \sigma^2$ if and only if $u^2 = 0$ and $r^1 = r^2$, and therefore (7.47) are unnecessary.

Again, the proof relies on the fact that a state must be more feasible, strongly feasible and optimal to dominate another.

These relaxations can be tightened by introducing the variable $k = |L|$, and are then named the (k, q, r)- and (k, q, u)-route relaxations, respectively. The state-space graph is easily adapted and the number of states is increased by a factor of n. The dominance conditions are the same as those from the (q, r)- and (q, u)-route relaxations, except that the following condition is also required.

$$k^1 \leq k^2. \quad (7.51)$$

These dominance conditions are stricter and give a tighter lower bound on the original problem, because (7.23) can only be true if (7.51) is true.

Another relaxation technique is suggested by both Righini and Salani (2008) and Boland et al. (2006), and defines a set of critical customers N. A state variable θ is introduced to the (q, u)-relaxation and similarly to L it represents customers that are forbidden from being visited, but not necessarily all that have been visited. Only customers $i \in V' \cap N$ that are extended to are added to θ. In comparison to the (q, r)- and (q, u)-route relaxations the

number of states is a factor of $O(2^{|N|})$ larger. Again the dominance condition are the same as for the (q, r)- and (q, u)-route relaxations, except that the following condition is also required.

$$\theta \subseteq \theta'. \tag{7.52}$$

More recently, Baldacci et al. (2011) propose the NG-route relaxation, which prevents customers from being visited again depending on which customer are subsequently visited. Let $N_i \subseteq V'$ be neighbourhoods of customers considered most likely to be visited immediately after $i \in V'$ (and including i). At any state, the set θ comprises the customers visited which feature in the sets N_i of all subsequently visited customers. Let $\Delta = \max_{i \in V'} |N_i|$, then compared to the (q, r)- and (q, u)-route relaxations the number of states is a factor of $O(2^{\Delta})$ larger. If $|N_i| = n$ for $i \in V'$, then θ will be equivalent to the set L, and if each customer in N is in each N_i it is equivalent to the critical node relaxation. The dominance conditions are the same as for the critical node relaxation.

Restrictions

Let us consider an optimal route $(0, \ldots, i_1, i_2, \ldots, i_k, i_1, \ldots, 0)$ for one of the relaxations described. The most simple approach to ensure that a customer such as i_1 is visited less is to use the critical node relaxation. If i_1 is added to the set N, then when the DP is solved i_1 will only be visited once. Another approach is to update the neighbourhoods from the NG-route relaxation. For neighbourhoods N_i, then i_1 must be added to N_{i_j} for $j \in \{2, \ldots, k\}$. When the DP is solved then any cycles at customer i_1 visiting a subset of $\{i_j : j \in \{2, \ldots, k\}\}$ are not possible. These restrictions are independent of the DP formulation used. The number of DSSR iterations is clearly bounded by the number of possible restrictions, and is therefore $O(n)$ using the set N, and $O(n^2)$ using neighbourhoods N_i. Comparing the two, in each iteration the time complexity grows equally in the worst case and the lower bound improves equally in the best case.

If a customer is visited three times or more, or more than one customer is visited more than once, then a decision must also be made about which cycles to restrict. Often a simple approach is used such as restricting the first customer visited twice or the customer visited the most times when updating N, and the first cycle or all the cycles for the NG-route neighbourhoods. If more than one optimal solution is found, then a decision must also be made about which solutions are considered for restrictions on cycles. Again a simple approach is usually taken, such as considering cycles from the first optimal solution identified. For more detail about these considerations using the set N for DSSR see the paper by Boland et al. (2006).

7.3.5 Heuristic dominance

Heuristic dominance conditions are more aggressive and may lead to the optimal solution or even all solutions with negative reduced cost being dominated. These conditions allow more states to be dominated and this makes the time taken to solve the DP formulation shorter. The approach considered is proposed by Desaulniers et al. (2008) and focuses on the customers whose γ-variable has increased the most since the last iteration of column generation.

Let $0 \leq n_{\text{crit}} < n$ be an integer parameter and N_{crit} be a list of the n_{crit} customers whose γ-variable has increased most since the last time column generation was needed (except in the first iteration). Only these customers are considered in the dominance conditions, and the dominance condition concerning the set of customers visited in the path is reformulated as follows.

$$\{L \cap N_{\text{crit}}\} \subseteq \{L' \cap N_{\text{crit}}\}. \tag{7.53}$$

Note that the full set of customers visited is considered for feasibility when states are extended and therefore only elementary paths are found.

7.4 Computational experiments

In this section, we present results from some computational experiments. These experiments seek to examine the effectiveness and efficiency of using the LC formulation to generate lower bounds for the VRPRDD, and to compare the efficiency and effectiveness of the alternate formulations of the PP. We consider the effectiveness because although both formulations return the same optimal solution, it is not necessarily true that the same set of columns with negative reduced cost are obtained in the same time span, and therefore if the algorithm is prematurely terminated the lower bounds obtained may be different. To give a better appreciation of the differences in the results across the instance types, we will consider the smaller instances used the previous chapter, which have $n = 20$ or 30.

Our preliminary experiments suggested a number of parameters of the CG algorithm are more robust, and these are fixed at values that appear to perform well. For the parameters limiting the number of states compared for dominance, $p_Q = p_T = 1/3$ was found to provide a good balance between quality and efficiency. We set the parameters that trigger managing the column pool to $t_{\text{LP}} = 2$ and $\rho = 10,000$. To calculate the upper bound on the arrival times at customers, T, we used the same procedure described in the relevant section of Chapter 5. Although this value may be too small when the elementary requirement on routes is relaxed, with the proposed dominance conditions and procedure this limit is not exceeded.

As discussed, the RMP is hot-started with the routes from the best solution identified by the PRA in Chapter 6, and the objective value is taken as the initial upper bound. In the first two stages of the pricing algorithm, we use the RD formulation with heuristic dominance, and set $n_{\text{crit}} = 0$ and $n_{\text{crit}} = \lfloor n/m \rfloor$, respectively. Notice that the first DP heuristic ignores the set L in the dominance conditions (but not when extending a state), and therefore has a pseudopolynomial number of states and time complexity. The values for n_{crit} are not necessarily optimal, but have been found to produce acceptable results in limited preliminary testing. The exact pricer uses DSSR with the ng-route relaxation, but we consider two strategies for initialising and updating the NG-route neighbourhoods.

The algorithms are implemented in C++ and complied using GCC 6.4.1. Gurobi 6.0.0 (C API) is used to model and solve the linear programs, applying the dual simplex algorithm with default settings. All experiments are performed using a single core of an Intel Xeon E5–2670 2.6GHz processor (running Linux Red Hat Enterprise Server release 6.3).

7.4.1 Comparison of formulations and update strategies

In this section, we compare the results of the CG algorithm using the two DP formulation porposed for the PP, and the two strategies for initialising and updating the NG-route neighbourhoods. This results in four configurations of the DSSR algorithm and therefore four versions of the CG algorithm that will be denoted by the abbreviation of their DP formulation and a subscript for the strategy.

The first strategy, denoted by a subscript of one, is inspired by Baldacci et al. (2011) and Boland et al. (2006), and initialises the neighbourhoods for each customer $i \in V'$ with the $\lfloor n/m \rfloor$ customers $j \in V'$ with smallest reduced arc costs $\bar{c}_{ji}$. At each iteration of the DSSR algorithm, if the optimal solution contains a cycle, the first cycle to finish from the start of the route is restricted by updating the neighbourhoods appropriately. For the second strategy, denoted by a subscript of two, we use the same initialisation except we reduce the initial sizes of the neighbourhoods to $\lfloor 0.8n/m \rfloor$. For the update, if the optimal solution contains a cycle, then all the cycles of the five solutions with least reduced cost are restricted by updating the neighbourhoods appropriately. This update approach is inspired by Martinelli et al. (2014).

In Table 7.1, we present the results from these experiments. The CG algorithm is deterministic and therefore results are taken from a single run on the LC formulation of each instance. The stopping criteria are solving LC or exceeding one hour of computational time.

Table 7.1: Comparison of results solving LC

			Gap (%)				Time (s)			
α	n	m	TF_1	RD_1	TF_2	RD_2	TF_1	RD_1	TF_2	RD_2
0.3	20	2	0.22	0.22	0.24	0.24	609.65	769.99	44.43	54.84
		3	1.11	1.11	1.11	1.11	4.51	5.60	3.38	3.99
	30	3	2.69	3.28	2.69	2.69	1006.91	1274.52	201.19	223.29
		4	1.88	1.88	1.88	1.88	41.67	62.13	26.52	30.39
0.5	20	2	0.26	0.26	8.57	8.57	469.07	444.13	393.86	609.39
		3	0.79	0.79	0.79	0.79	3.11	3.59	2.84	3.14
	30	3	10.33	10.33	2.37	2.37	681.33	1013.63	193.72	210.91
		4	1.44	1.44	1.44	1.44	30.76	40.29	24.80	27.50
0.7	20	2	9.19	8.42	0.09	0.09	648.34	654.54	264.70	502.44
		3	0.89	0.89	0.89	0.89	3.49	4.43	3.38	3.80
	30	3	1.78	1.78	1.78	1.78	151.14	195.40	109.32	118.77
		4	0.97	0.97	0.97	0.97	23.63	28.51	20.82	22.73
Av.			2.63	2.61	1.90	1.90	306.13	374.73	107.41	150.93

From these results, it is clear that strategy two has considerably lower computational times on these instances for both formulations, and mostly this strategy achieves equal or tighter lower bounds. The improvement in running time is particularly evident for instances with lower values of α, representing more weight on the total weighted tardiness. These instances also have larger and more unstable dual variables, leading to more cycles in solutions to the relaxed pricing problem.

It is interesting to note that the LC formulation can be solved faster for instances with more vehicles, and therefore a smaller value for the ratio n/m. This is expected because this ratio is closely related to the number of customers in solutions to the PP, and solving the PP represents almost all of the computational time. The gap is larger for these instances in general, and this may be attributed to a greater number of fractional columns that are permitted by the greater number of vehicles available.

The gap is larger for strategy two on average for instances with $\alpha = 0.3$, $n = 20$, $m = 2$, and this is because no valid lower bound is found for one of these instances. Using strategy one, the average gap for the RD formulation is slightly smaller, although using strategy two this becomes equal. Generally, the results in the table reveal the significant impact of the update strategy on the computational time and ability to solve LC for the instance.

The TF formulation appears to be faster on average, and this may be due to the better performance on the instances where the scheduling aspects of the problem are less influential. In Table 7.2, we present a more detailed breakdown of the computational time in seconds using strategy two and each of the formulations. As discussed, the results are taken from a single run of the CG algorithm on each instance and the maximum computational time is one hour.

Table 7.2: More detailed comparison of computational times using the different formulations of the PP.

		$\alpha = 0.3$		0.5		0.7	
b	k	TF_2	RD_2	TF_2	RD_2	TF_2	RD_2
0.25	4	24.35	39.65	210.66	853.96	272.35	990.70
	6	28.08	44.80	20.48	24.23	16.93	20.75
	8	17.02	19.61	17.63	18.84	19.54	21.42
0.50	4	65.00	76.05	32.96	39.30	28.86	31.46
	6	22.29	26.23	19.70	23.25	20.51	22.18
	8	45.82	54.79	60.87	65.77	77.36	80.57
0.75	4	139.79	157.67	110.87	112.43	55.87	57.60
	6	79.41	85.45	86.93	95.60	65.19	69.57
	8	36.77	39.55	31.82	38.02	28.97	31.84
1.00	4	1010.73	1003.07	1054.66	1063.43	502.96	495.09
	6	146.05	166.87	148.51	170.44	83.14	97.41
	8	100.18	110.08	50.57	47.55	22.97	24.61
Av.		142.96	151.98	153.80	212.74	99.55	161.93

The gaps are generally equal between these formulations using strategy two, but the computational times give us a greater appreciation of the differences in performance between the different formulations. It is still clear that for the majority of the instances, the TF formulation is more efficient. This is particularly so for larger values of α, which reduce the importance of the weighted tardiness, and therefore the release dates, and the decomposition of the states over the release dates introduces unecessary complexity in the RD formulation. Although for the instances with the greatest value of b, representing the greatest spread of release dates, the RD formulation is sometimes marginally faster.

It is also notable that there seems to be a correlation between the efficiency of the RD formulation and the influence or tightness of the scheduling aspects. This is somewhat expected, because these factors directly influence the number of different cost vectors and therefore states in the TF formulation, although in the RD formulation this has little effect. In general, the computational time seems to mostly dependent on the value of b, representing the spread of release dates, although the value of k, representing the tightness of the due dates, can also have an impact and compounds the effect of a large spread of release dates.

7.4.2 Strength of lower bounds from column generation

Considering the most efficient CG algorithm from the previous sections, it is also important to analyse the tightness of the lower bound provided and efficiency of the LC formulation in comparison to other formulations. We compare the results of the LC formulation, using the proposed CG algorithm with the TF_2, and the linear relaxation of the OC formulation (LOC) from Chapter 5, using the CPLEX cutting planes algorithm. The latter algorithm is the same as the branch and cut algorithm described in Chapter 5, except that a sequential approach is used by reducing the number of threads to one and the search is stopped if the root node is solved or after one hour of computational time.

In Table 7.3, we present the average relative gap between the best lower bounds found and the provided upper bounds, and the average time in seconds, for the two algorithm and formulation pairs. The results are summarised for each spread of release dates and tightness of due dates, because this was found to reveal the most differences between the formulations. It is immediately evident from these results that the lower bound from the LC formulation is far superior to that from the LOC formulation, but the time required is also greater. Further, the difference in the gap appears to be negatively correlated to the difference in computational time.

Table 7.3: Comparing results for solving the linear relaxation of different formulations.

		Gap (%)		Time (s)	
b	k	LC	LOC	LC	LOC
0.25	4	0.85	44.03	169.12	3.32
	6	1.77	21.28	21.83	2.27
	8	0.82	7.39	18.06	1.34
0.50	4	1.85	52.96	42.27	3.52
	6	0.89	26.45	20.84	2.18
	8	1.37	14.52	61.35	1.96
0.75	4	1.33	55.96	102.17	3.62
	6	1.34	34.45	77.18	2.46
	8	1.27	18.98	32.52	2.01
1.00	4	18.53	57.56	856.11	4.19
	6	0.36	36.58	125.90	2.70
	8	0.72	27.44	57.91	2.02
Av.		2.60	33.13	132.11	2.63

As we have found for many analyses, the performance of algorithm and formulation pairs is most different when the scheduling aspects of the problem are more influential. Clearly, the increased complexity of the scheduling aspects causes the lower bound from the LOC formulation to be considerably weaker, whereas the LC formulation appears to give a much tighter lower bound. This difference in performance may be in-part attributed to the increase in the total weighted tardiness when a convex combination of routes are used, because each fractional route incurs a proportion of the weighted tardiness of that route, rather than calculating the weighted tardiness of each customer from the sum of arrival times in routes to which it is partial assigned.

There is a noticeably large gap for the LC formulation when $b = 1$ and $k = 4$, and this is because for two of these instances a valid lower bound is not identified in the computational time limit. The considerable difference in the relative gap for the LC formulation in com-

parison to the increase in computational time seems to be justifiable for the majority of the instances considered. This may also be improved by adjusting the initialisation and update strategy for the neighbourhoods in the DSSR algorithm to adapt more to the structure of the instance.

7.5 Conclusion

In this chapter, we propose a set covering formulation of the vehicle routing problem with release and due dates (VRPRDD), and a column generation (CG) algorithm to solve the linear relaxation. The CG algorithm builds on features of recently proposed algorithms for similar problems, although it applies these to the new problem. Contributions include two formulations of the pricing problem (PP), using a pointwise linear function of the departure time of a route to model the weighted tardiness or decomposing the states over the release dates to remove the dependency on the departure time from the depot, and a variable neighbourhood descent pricing algorithm. This pricing algorithm applies heuristics of increasing complexity before resorting to an exact decremental state-space relaxation (DSSR) algorithm using a general label-setting algorithm to solve the ng-route relaxations. We also trial two strategies for the DSSR algorithm that have greater emphasis on either initialising or updating the ng-route neighbourhoods.

Observing the results of the CG algorithm, the performance using the TF formulation and the strategy with greater emphasis on updating the ng-route neighbourhoods is shown to dominate all others. Using this strategy, either formulation achieves the same lower bound for the instances in the computational time, but the TF formulation is shown to be more efficient. The results from the better performing version of the CG algorithm are also compared against the linear relaxation of the OC formulation of Chapter 5 solved using the cutting planes algorithm of CPLEX. These results suggest that the LC formulation has a superior lower bound, particularly when the scheduling aspect of the problem are influential, and if an efficient CG algorithm is used to solve the formulation then the relative increase in computational time is acceptable.

In our experiments, the importance of exploring and improving the techniques to restrict non-elementary routes in the DSSR algorithm is highlighted, and this would be an interesting area for future research. Furthermore, the efficiency of the CG algorithm may be improved if the restrictions for the DSSR algorithm, or for another approach to restrict non-elementary routes, are taken from the optimal solutions to a relaxed restricted master problem (RMP), as proposed by Contardo et al. (2015). This may considerably reduce the number of restriction required to solve the original RMP, although the powerful strong-degree constraints considered by these authors do not hold for set-covering formulations. Using the linear relaxation of the set-covering (LC) formulation in a branch-and-price algorithm for the VRPRDD seems to be an appropriate development. Especially considering that the commercial branch and cut algorithm in Chapter 5 required more computing power, memory and time, and still has a worse lower bound for some instances.

Bibliography

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations research*, 59(5):1269–1283.

Boland, N., Dethridge, J., and Dumitrescu, I. (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, 34(1):58–68.

Christofides, N., Mingozzi, A., and Toth, P. (1981). State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2):145–164.

Contardo, C., Desaulniers, G., and Lessard, F. (2015). Reaching the elementary lower bound in the vehicle routing problem with time windows. *Networks*, 65(1):88–99.

Desaulniers, G., Lessard, F., and Hadjar, A. (2008). Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, 42(3):387–404.

Desrochers, M. and Soumis, F. c. (1988). A generalised permanent labelling algorithm for the shortest path problem with time windows. *INFOR*, 26:193–214.

Feillet, D., Dejax, P., Gendreau, M., and Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229.

Martinelli, R., Pecin, D., and Poggi, M. (2014). Efficient elementary and restricted non-elementary route pricing. *European Journal of Operational Research*, 239(1):102–111.

Righini, G. and Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51(3):155–170.

Chapter 8

Conclusion

In this thesis, we present a new vehicle routing problem (VRP) that considers the time that customer orders become available for delivery at the depot, and is described as the VRP with release and due dates (VRPRDD). The objective is to minimise a convex combination of operational cost and customer service objectives, represented by the total distance travelled and total weighted tardiness, respectively. If a relationship between the operational costs and the perceived cost of tardy deliveries to customers can be established, then the problem can be applied at an operational level to find delivery routes for a single period. Another option is to evaluate a small number of values for the weight that balances the objective components, and potentially the individual customer tardiness weights, to find a solution which reaches a suitable compromise for the decision maker. This also suggests an alternative more strategic application of the problem to evaluate different possible scenarios and decisions. As one example, given a set of customers with representative release and due dates and weights, the effect of different numbers of vehicles and their capacity on the other features of the solution can be investigated.

8.1 Contributions and insights

We propose commodity flow formulations of the VRPRDD to attempt to find lower bounds and optimal solutions, a path-relinking algorithm (PRA) to find efficient upper bounds and an iterated local search algorithm (ILS) as a comparator, and a set-covering (SC) formulation and column generation (CG) algorithm to find tighter lower bounds. Some of the main contributions from these algorithms and heuristics include efficient neighbourhood search and move evaluation procedures; a path-relinking procedure; a balanced edge- and arc-based solution distance used in measuring population diversity; constraints and variables to formulate the departure time from the depot of a route as a commodity flow on the arcs; two dynamic programming formulations of the pricing problem (PP) and respective ng-route relaxations; an efficient dominance procedure for a general label-setting algorithm; two strategies for initialising and updating ng-route neighbourhoods in a decremental state space relaxation (DSSR) algorithm; and extensive computational experiments for all of the above.

Considering the results of this extensive computational evaluation of the VRPRDD, and heuristics and algorithms applied to the problem, it is clear that there are significant effects associated with introducing release dates and weighted tardiness into a VRP. The commodity flow formulations have considerable duality gaps, and therefore cannot be used to solve many

of the instances with 20 or 30 customers in four hours using the parallel branch-and-cut algorithm of CPLEX with four processors. The heuristics proposed perform acceptably on the instances considered, and the PRA finds optimal solutions for all the instances with 20 or 30 customers that now have known optimal solutions. Even for larger instances with up to 199 customers, the computational time of the PRA is acceptable and the performance dominates that of the ILS in terms of quality and efficiency. The linear relaxation of the set-covering formulation has a greatly superior lower bound across the variety of instances, but solving it requires considerably more computational time and this is particularly true when the scheduling aspects of the problem have greater influence. The computation time is controlled using the carefully designed CG algorithm, and this is particularly dependent on computational time of the DSSR algorithm for the pricing problem.

This introduction and the initial investigations of the VRPRDD should prove valuable in initiating and promoting research of the problem, and the integration of further aspects of machine scheduling into VRPs. A problem of particular interest motivated the VRPRDD, and requires scheduling the processing of customer orders and deliveries. This investigation is particularly important considering the novelty of these types of problems, and the immediate practicality for managing the operations of supply chains. Another particularly interesting variant of the problem considers both objectives simultaneously and will further explore the trade-off.

For practitioners, the results and discussion should present a clear analysis of some of the effects associated with explicitly considering release dates for customer orders, and introducing the customer service level objective. In addition, we reveal relationships and dependencies between the problem features that hint at the complex and varied interplay between these in different instances of the problem. Furthermore, the insights yielded from the analyses presented should prove helpful in the future for designing exact algorithms and heuristics for this and similar problems.

8.2 Further directions

A variety of avenues for future research are initiated by this thesis, and some have already been suggested. These range from improvements to algorithms and techniques presented to extensions of the problem and the novel features, and we mention some of the more interesting examples.

To find the tight lower bounds provided by the set-covering formulation more efficiently, avenues to consider include developing heuristics for the PP with less dependency on negative cost cycles; further analysis of the techniques to initialise and update the ng-route neighbourhoods in the DSSR algorithm; and restricting non-elementary routes in the restricted master problem rather than the PP. These improvements will also contribute to extending the CG algorithm to develop an efficient branch, price and cut algorithm to solve the problem to optimality. This naturally leads to another area for research, exploring valid inequalities and branching strategies for the arc- and path-based formulations of the VRPRDD.

From a more practical viewpoint, the definition of the problem and any further variants may be informed by a case-study on release dates and the representation of customer service level objectives in industrial applications. It may be interesting to analyse the features of

real-world problems that affect the mathematical structure of the problem, and as a result the proposed formulations, heuristics and lower bounds.

The results of the PRA and its competitive performance for the asymmetric capacitated VRP, motivate further investigation into applying the PRA to other VRPs, and even extention to address the bi-objective VRPRDD. Further, the performances of the both the PRA and the CG algorithm suggest that with a few modifications these approaches may perform well for machine scheduling problems that are closely related to the VRPRDD.

Appendix A

Memory intensive instances

The following instances required enabling CPLEX to use disk space to store nodes from the branch and cut algorithm.

Table A.1: Table of instances that required more memory.

α	n	m	b	k	MC	OC
0.3	20	2	0.25	4	x	x
			0.5	6	x	
			0.75	4	x	
				6	x	
			1	4	x	x
				8	x	
		3	0.5	4		x
			0.75	4		x
			1	4		x
	30	4	0.75	4		x
			1	4		x
0.5	20	2	0.25	4	x	
			0.5	4	x	
			0.75	4	x	
				6	x	
			1	4	x	x
				8	x	
		3	1	8	x	
	30	3	0.75	4		x
			1	4		x
				8		x
		4	0.25	4		x
			0.75	4		x
			1	4		x
0.7	20	2	0.5	4	x	
			0.75	4	x	x
				6	x	x
			1	4	x	
		3	0.5	8	x	
	30	3	0.5	4		x
		4	1	4		x
Total					18	18

Appendix B

Illustration of an example path-relinking

In Figure B.1, an example of a relinking is given, where the vertical lines separating customers in x_I represent arcs not included in x_G. We assume the first neighbourhood chosen is 2-Opt* and the neighbour selected introduces $(3, 4)$, which reduces $\text{dist}(x_c, x_G : 0)$ by one. The following neighbourhood is also 2-Opt* and $(1, 3)$ is introduced, this also connects two routes and removes $(1, 0)$ which is not present in x_G, and $\text{dist}(x_c, x_G : 0)$ is therefore reduced by two. The third neighbourhood is block-insert, introducing $(9, 8)$ and as a side-effect $(0, 1)$ and $(8, 0)$, therefore reducing $\text{dist}(x_c, x_G : 0)$ by three. This continues until the sixth move produces x_G.

Figure B.1: An example of a relinking trajectory.

$\text{dist}(x_I, x_G) = 11$	1. 2-Opt*$(8, 4)$ $\text{dist}(x_c, x_G) = 10$	2. 2-Opt*$(3, [3])$ $\text{dist}(x_c, x_G) = 8$	3. bl-ins$(8, 8, [2])$ $\text{dist}(x_c, x_G) = 5$
x_I 0 | 7 2 |10 6 | 0 0 | 3 | 8 | 1 | 0 0 | 4 | 5 | 9 | 0	0 | 7 2 |10 6 | 0 0 | 3 4 | 5 | 9 | 0 0 | 8 | 1 | 0	0 | 7 2 |10 6 | 0 0 | 8 | 1 3 4 | 5 | 9 | 0	0 | 7 2 |10 6 | 0 0 1 3 4 | 5 | 9 8 0
x_G 0 10 6 5 7 2 0 0 1 3 4 9 8 0	4. bl-ins$(7, 2, [1])$ $\text{dist}(x_c, x_G) = 3$	5. 2-Opt*$(7, 9)$ $\text{dist}(x_c, x_G) = 2$	6. 2-Opt*$(9, 5)$ $\text{dist}(x_c, x_G) = 0$
	0 10 6 | 7 2 0 0 1 3 4 | 5 | 9 8 0	0 10 6 | 9 8 0 0 1 3 4 | 5 7 2 0	0 10 6 5 7 2 0 0 1 3 4 9 8 0

Note: Vertical lines represent arcs not in x_G.

Appendix C

PRA results on the asymmetric capacitated vehicle routing problem

To validate the performance of the PRA on a more familiar problem, and to compare the performance with independent methods, we have also applied the algorithm to the asymmetric capacitated vehicle routing problem (AVRP). This problem is chosen due to the relationship to the VRPRDD and the emphasis on directed routes, the latter has similarities to the effect of the scheduling aspects of the VRPRDD. The set of instances considered is proposed by Fischetti et al. (1994) and extended by Pessoa et al. (2008). There are 24 instances with between 33 and 70 customers, and for each n there are three different values for m. The best known solutions (BKS) are taken from Pessoa et al. (2008) and Subramanian et al. (2013), and are proven to be optimal for all the instances except $n = 71$ and $m = \{5, 10\}$.

In Table C.1, we compare the performance of the PRA with an ILS algorithm and a HGA, proposed by Subramanian et al. (2013) and Vidal et al. (2014), respectively. The parameter settings for $\alpha = 0.7$ are used, except that $\eta = 1$ and therefore the distance used in calculating the fitness of solutions in population management (§ 6.2) is arc-based. The stopping criteria are now 250 relinkings without improving the best or best feasible solutions, and a 30 minutes limit on the total computational time. The results are taken from 10 independent runs for each instance, and note that all of the algorithms achieved the best known solution for each of the instances in at least one run. From the results, it is clear that with a small increase in one parameter and some stopping criteria, then the PRA is competitive with the two best performing methods for the AVRP.

Bibliography

Fischetti, M., Toth, P., and Vigo, D. (1994). A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. *Operations Research*, 42(5):846–859.

Pessoa, A., De Aragão, M. P., and Uchoa, E. (2008). Robust branch-cut-and-price algorithms for vehicle routing problems. In *The vehicle routing problem: Latest advances and new challenges*, pages 297–325.

Subramanian, A., Uchoa, E., and Ochi, L. S. (2013). A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10):2519–2531.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2014). A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658–673.

Table C.1: Comparison of performance on AVRP instances

			S13		V12		PRA	
n	m	BKS	Avg	Time (s)	Avg	Time (s)	Avg	Time (s)
33	2	1406	1406.0	0.57	1406.0	11.40	1406.0	0.89
33	4	1773	1773.0	0.47	1773.0	9.00	1773.0	0.88
33	8	2672	2672.0	0.47	2672.0	6.60	2672.0	1.01
35	3	1644	1644.0	0.64	1644.0	9.00	1644.0	1.12
35	5	2110	2110.0	0.61	2110.0	9.60	2110.0	1.56
35	10	3338	3338.0	5.67	3338.0	7.20	3339.2	2.15
38	3	1654	1654.0	0.69	1654.0	10.20	1654.0	0.98
38	6	2289	2289.0	0.6	2289.0	9.00	2289.0	1.16
38	12	3705	3705.0	0.77	3705.0	7.80	3705.0	1.37
44	3	1740	1740.0	1.06	1740.0	12.60	1740.0	1.25
44	6	2303	2303.0	0.88	2303.0	13.80	2303.0	2.31
44	11	3544	3544.0	2.46	3544.0	14.40	3559.1	2.55
47	3	1891	1891.0	1.26	1891.0	16.20	1891.0	1.42
47	5	2283	2283.0	1.79	2283.0	18.00	2283.0	2.27
47	10	3325	3325.0	1.29	3325.0	15.60	3326.2	2.63
55	3	1739	1739.0	2.09	1739.0	25.20	1739.0	3.62
55	5	2165	2165.0	3.53	2165.0	20.40	2165.0	2.54
55	10	3263	3263.0	2.26	3263.0	19.20	3268.3	3.23
64	3	1974	1974.0	3.08	1974.0	25.20	1974.0	2.57
64	6	2567	2571.7	3.3	2567.4	42.00	2569.2	6.92
64	12	3902	3904.9	3.41	3902.0	28.20	3902.0	5.34
70	3	2054	2054.0	4.46	2054.0	28.80	2054.0	6.37
70	5	2457	2457.9	6.72	2457.0	34.20	2457.0	8.81
70	10	3486	3492.9	5.61	3488.4	38.40	3498.30	7.36
Av Time (s)			2.24		18.00		2.93	
Av Gap (%)			0.02		0.00		0.04	
CPU			Xeon 2.93G		Opt 2.2G		Xeon 2.6G	