OPTIMIZATION APPROACHES FOR VEHICLE ROUTING PROBLEMS WITH BLACK BOX FEASIBILITY

FLORENCE MASSEN

Thesis submitted in partial fulfillment of the requirements for the Degree of Doctor in Applied Sciences

August 2013

Institute of Information and Communication Technologies,
Electronics and Applied Mathematics
Louvain School of Engineering
Louvain-la-Neuve
Belgium

Thesis Committee:

Yves Deville (director)	Université catholique de Louvain, Belgium
Karl Dörner	Johannes Kepler Universität, Austria
Charles Pecheur	Université catholique de Louvain, Belgium
Pascal Van Hentenryck	University of Melbourne, Australia
Peter Van Roy (president)	Université catholique de Louvain, Belgium

ABSTRACT

Vehicle Routing Problems are concerned with determining how a set of vehicles can visit a set of customers while minimizing some objective. Rich Vehicle Routing Problems combine different real-world constraints in a same problem. In a subclass of these rich problems, a simple Vehicle Routing Problem is rendered more difficult by adding a complicated side-problem that must be solved in order to verify the feasibility of a route. Examples of such problems are Vehicle Routing Problems with loading constraints, where for each route a loading problem must be solved. Another example are Vehicle Routing Problems with Time Windows and scheduling constraints, where for each route a scheduling problem must be solved.

Existing optimization approaches for such problems make use of very specific knowledge of the side-problem that is to be solved for each route. The goal of this thesis is to devise optimization methods for this type of Vehicle Routing Problems, which are as independent as possible from the side-problem being solved per route. This is achieved, by introducing an abstract problem, the Vehicle Routing Problem with Black Box Feasibility. In this problem, each route must satisfy a set of hidden constraints. A black box function is provided, to test whether a given route respects the hidden constraints. No insight into this function is possible.

This thesis proposes three different optimization approaches for the Vehicle Routing Problem with Black Box Feasibility. They are all based

on the principle of reformulating the problem as a Set Partitioning Problem. This means that a set of feasible routes must be generated. A final feasible solution to the Vehicle Routing Problem with Black Box Feasibility is obtained by selecting a subset of these routes. This must be done such that the selected routes can be combined into a feasible solution to the Vehicle Routing Problem with Black Box Feasibility and such that the resulting solution minimizes the given objective.
The first approach, Pheromone-based Heuristic Column Generation, combines concepts from Ant Colony Optimization and Column Generation. The idea is to use the information from intermediary non-integral solutions of the Set Partitioning Problem to learn which arcs are more likely to appear in high-quality, feasible routes. This information is then employed in the generation of new feasible routes. A definitive solution to the Vehicle Routing Problem with Black Box Feasibility is obtained using a standard solver.
The second approach, Dive & Generate, combines the ideas of Backtracking, Heuristic Pruning and Restarts with a known primal heuristic. Here a set of solutions to the Vehicle Routing Problem with Black Box Feasibility is obtained by traversing a tree. At each level of this tree a specific route is added to the current partial solution. At selected nodes, the dual information from intermediary non-integral solutions to the Set Partitioning Problem is used to generate feasible routes that have the potential to improve this same solution.
The third approach, Heuristic Branch & Price also uses dual information in the process of generating feasible routes. Again, a set of solutions is obtained in the traversal of a binary tree. At each level a specific route feature is selected and will be forced to appear in the current partial solution on one node, and forbidden to do so on the sibling node.
Finally the use of these three methods in a decomposition-based approach is proposed.

Iterated F-Race, an automatic algorithm configuration procedure, has been used on the Pheromone-based Heuristic Column Generation approach, allowing to identify high-performing configurations and to do a basic impact analysis of the different components and parameters of this method.

The proposed approaches have been implemented into a coherent system which is available for download under the LGPL license at

http://becool.info.ucl.ac.be/resources/VRPBB. They are evaluated on two concrete instantiations of the Vehicle Routing Problem with Black Box Feasibility from the literature, the 3L-CVRP and the MP-VRP. The state-of-the-art approaches for these problems all make use of some kind of knowledge about the side-problem being solved for each route. The black box functions used for each of the problems correspond to algorithms used in the relevant state-of-the-art.
The results show that the Pheromone-based Heuristic Column Generation method is highly competitive on both the 3L-CVRP and MP-VRP. It is able to improve the best known solutions on a majority of benchmark instances, this while being completely agnostic w.r.t. the side-problem being solved for each route. Also, the Pheromone-based Heuristic Column Generation method allows to improve the average solution costs from state-of-the-art approaches on a great number of instances. The Dive & Generate and Heuristic Branch & Price approaches perform less well. The performance gap with the Pheromone-based Heuristic Column Generation procedure is more distinct on the MP-VRP. This can be explained by the problem size of the benchmark instances in this latter application.

Finally the Pheromone-based Heuristic Column Generation is also applied to variations of the 3L-CVRP and the MP-VRP where a different objective is considered. These variations haven't been considered in the literature so far, and first results are published here.

ACKNOWLEDGMENTS

First of all I'd like to thank my supervisor Yves for his patience, his steady support and the guidance he provided over the last years. His ability to quickly grasp ideas, to discern vagueness and to always ask the relevant questions has been extremely useful for my work and keeps astonishing me. Many thanks also to Pascal for his interest in my work and the fruitful discussions. I am also very grateful he invited me to visit his research group in Melbourne.

A big thank you to my office mates Vianney and Jean-Baptiste. I enjoyed coming to work these past years, knowing there'd be interesting conversations, whether they would be of scientific, philosophical or general nature. The numerous jokes and sometimes random discussions (where do paradigms go when they die?) were also much appreciated. Thanks guys! Thanks also to the other former and current beCool members, Pierre, Jean-Noël, Dung, Julien, Sébastien, Viet, Trung, Cyrille, Sascha and Renaud. It has been fun sharing time and beers with you on conferences, getting deeper insights into your work and your lifes. It has always been a pleasure to engage in funny and interesting conversations with Tania, Benjamin, Stéphane and Vanessa during our little breaks, thank you for that! Thank you also to everybody at INGI! Thanks to the NICTA researchers in Melbourne, especially to Carleton and Victor for welcoming me in their office. Many thanks also to Alessandra for her helpfulness and kindness. I am very grateful to the INGI department for giving me the opportunity to visit the NICTA lab in Melbourne.

Thanks to the jury for reading this thesis, asking intelligent questions and providing helpful comments.
Thanks to Fabien Tricoire for having provided us with his implementation of the feasibility check for the MP-VRP.
Thanks also to Manuel and Thomas for our collaboration and for Manuel's help with the R scripts which have come in handy for this thesis.

Finally many thanks go to my family and friends for their support during these last years. I am especially grateful to my parents who always encouraged me, supported me and offered kind words when things looked grim. They both invested a great deal of time and patience to help me through my first years of studies, for which I am infinitely thankful.

Last but not least, my deep gratitude goes to my husband Ronny who has supported me, and bore with me, during these last years. He listened to my explanations, to my presentations and reread my thesis with patience. Thanks for offering comfort when necessary, motivating me the rest of the time and for being just plain awesome!

I gratefully acknowledge the support from the Fonds National de la Recherche, Luxembourg.

CONTENTS

Part I

BACKGROUND

1 INTRODUCTION

The original Vehicle Routing Problem has been stated in 1959. Even after 60 years of research, large-scale instances or complicated variants of the problem still constitute a challenge for the scientific community. Research over the last decades has focused on so-called Rich Vehicle Routing Problems, where different complicating real-world constraints must be handled at the same time. This thesis is concerned with optimizing Vehicle Routing Problems where complicated side-constraints need to be satisfied by every route and no insight except for a feasibility check into these constraints is possible. A new generic type of Vehicle Routing Problem is introduced and different optimization approaches for this problem are designed, implemented and evaluated on concrete examples.

1.1 VEHICLE ROUTING PROBLEMS

Vehicle Routing Problems are basically concerned with finding a way to visit a given set of customer locations using a given set of vehicles in such a way that a cost function, often the total distance, is minimized. In the most basic version of this problem, each customer must be visited by exactly one vehicle and each vehicle performs one trip, starting and ending at a depot location. The question is thus to decide for each vehicle which set of customer locations it visits, and in which order it visits them. Typically a set of operational constraints must be fulfilled. These can limit the number or set of possible locations a given vehicle can visit, the exact time at which a given location is visited, the order in which a set of locations must be visited and so on. In Rich Vehicle Routing Problems many complicating constraints must be handled in one same problem. Among such Rich Vehicle Routing Problems are also problems which correspond to a "basic" problem where a set of complex side-constraints must be fulfilled by each individual route. Verifying the feasibility of a route in these problems necessitates the resolution of a complex side-problem. This is the type of problems considered in this thesis.

1.2 PROBLEM STATEMENT

Recent literature on Vehicle Routing Problems is concerned with Rich problems where for each route, in order to verify the feasibility, a complex side-problem needs to be solved. Examples are problems combining routing and loading, as well as routing and scheduling. Often the methods proposed to tackle this type of problems are very problem-specific and make wide use of knowledge of the side-problem being solved for each route.

The aim of this thesis is to propose methods able to tackle such problems using no further knowledge on the side-problem than a provided feasibility function to check the feasibility of a route. To do this a generalized reformulation for this type of problem is proposed. Different optimization procedures, all based on the idea of column generation, are designed and implemented in a coherent system. This system is available for download under the LGPL license at `http://becool.info.ucl.ac.be/resources/VRPBB`. The procedures are tested on two concrete example problems and compared to existing problem-specific approaches.

1.3 CONTRIBUTIONS

Several contributions are proposed in this thesis:

1. The formulation of a new generic problem, the Vehicle Routing with Black Box Feasibility is proposed. This problem embraces

any Vehicle Routing Problems where each route must verify a set of complicated side constraints, the satisfaction of which can only be checked in non-linear time complexity in the length of the route.

2. The combination of Ant Colony Optimization and Mixed Integer Programming in a novel way. Ants are used to generate routes, while information from the continuous relaxation of the Set Partitioning Problem formulated over all routes is used to update the pheromone matrix.

3. The design of a method using Ant Colony Optimization in a Column Generation context, of a Branch & Generate method as well as of a Heuristic Branch & Price method (the latter two based on the use of a randomized savings heuristic to do the pricing) for the Vehicle Routing Problem with Black Box Feasibility.

4. The implementation of the proposed approaches in a coherent system.

5. The evaluation of the proposed approaches on concrete examples of Vehicle Routing Problems with Black Box Feasibility. The concrete examples considered are the Three-dimensional Loading Capacitated Vehicle Routing Problem (3L-CVRP) and the Multi-Pile Vehicle Routing Problem (MP-VRP), both combining Vehicle Routing with Loading. The methods are executed on benchmark instances from the literature and compared to the problem-specific methods from the state-of-the-art.

6. The use of the Iterated F-Race procedure for the automatic parameter setting of one of the proposed methods. The resulting optimized configuration is used as a starting point to carry out a systematic analysis of the impact and effect of each of the considered parameters.

7. A variant of the concrete examples of Vehicle Routing Problems with Black Box Feasibility is considered. Instead of the classical objective a MinMax objective is optimized. Both problems haven't been studied under this aspect in the literature. First results on these variants are published in this thesis.

1.4 OUTLINE

The remainder of this thesis is organized as follows. Chapter 2 introduces the necessary background. The Vehicle Routing Problem is explained and a basic variant, the Capacitated Vehicle Routing Problem is formally introduced. Different optimization approaches are presented in an abstract way, and for each approach it is explained how it can be adapted to the Capacitated Vehicle Routing Problem.
The notion of Rich Vehicle Routing Problems is introduced in chapter 3. Then examples of Vehicle Routing Problems with complicated side-problems, such as these considered in this thesis, are given. An overview of existing libraries and frameworks designed for Rich Vehicle Routing Problems follows. A brief explanation of how the reviewed frameworks and libraries can be used to solve the problems considered in this thesis is provided.

Chapter 4 introduces the generic Vehicle Routing Problem with Black Box Feasibility. The advantages of using this generic problem formulation are explained. Each of the optimization methods presented in chapter 2 is reconsidered for application to this new problem.
Chapter 6 presents the Pheromone-based Heuristic Column Generation approach specifically designed for the Vehicle Routing Problem with Black Box Feasibility. A Branch & Generate and Heuristic Branch & Price approach for this same problem are proposed in chapters 7 and 8. Finally chapter 9 explains how all these methods can be used in cooperation with spatial decomposition.
The implementation of the proposed methods in a coherent system is described in chapter 10.

In chapter 11 the use of the Iterated F-Race approach to find an optimal parameter setting for the Pheromone-based Heuristic Column Generation method is detailed. The results obtained with the improved parameter setting are presented and a systematic analysis of the impact of each of the considered parameters is performed.
Chapters 12 and 13 introduce the reader to the Three-dimensional Loading Capacitated Vehicle Routing Problem and the Multi-Pile Vehicle Routing Problem, both instantiations of the Vehicle Routing Problem with Black Box Feasibility. In each chapter a review of existing work on the respective problem is presented; furthermore explanations on how the problem can be considered as a Vehicle Routing Problem

with Black Box Feasibility and on how existing approaches make use of problem-specific knowledge, are provided. Each of the approaches considered in chapters 6 to 8 are applied to these problems. Their performance is evaluated on benchmark instances from the literature and compared to the state-of-the-art for the concrete problem.
Chapter 14 considers the problems from chapters 12 and 13 under a MinMax objective. It is explained how the Pheromone-based Heuristic Column Generation method is adapted to this problem and results are presented.

Finally a general conclusion of this thesis along with perspectives for future work is given in 15.

1.5 PUBLICATIONS

The Vehicle Routing Problem with Black Box Feasibility and the Pheromone-based Heuristic Column Generation method have been presented at CPAIOR'12 [MDVH12]. The use of Iterated F-Race for parameter setting and the parametrical analysis of Pheromone-based Heuristic Column Generation has been presented at the Hybrid Metaheuristics conference (HM2013) [MLISD13].

The major publications are thus the following:

[MDVH12] F. Massen, Y. Deville, and P. Van Hentenryck, *Pheromone-based heuristic column generation for vehicle routing problems with black box feasibility*, Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (2012), 260–274.

[MLISD13] F. Massen, M. López-Ibáñez, T. Stützle and Y. Deville, *Experimental analysis of pheromone-based heuristic column generation using irace*, Hybrid Metaheuristics (2013), 92–106.

Extended abstracts and preliminary versions of this work have been presented at:

- Local Search for Constraint Satisfaction (LSCS) Workshop, Perugia, Italy, 2011

- Annual Meeting of the EURO Working Group on Vehicle Routing and Logistics Optimization (Verolog), Bologna, Italy, 2012
- Annual Meeting of the Belgian Operational Research Society (ORBEL), Brussels, Belgium, 2012
- Journées francophones de la programmation par contraintes (JFPC), Toulouse, France, 2012

2 VEHICLE ROUTING PROBLEMS

Vehicle Routing Problems (VRPs) constitute a class of combinatorial optimization problems concerned with the delivery of goods to customers. The original problem has been proposed in [DR59] where the authors present a real-world problem in which gasoline needs to be delivered to service stations. Given a certain number of trucks with a given tank size the authors want to find the routes to be served by these trucks such that every service station is visited once, such that the amount of gasoline to deliver on a route does not exceed the truck's tank size and such that the total distance traveled by all the trucks is minimized. This problem constitutes the simplest problem of the VRP class. It is formally introduced in Section 2.2. A brief overview of different well-known variants is also given in this section. Different methods for producing solutions to VRPs are presented in sections 2.4 to 2.10. First high-level notions relevant to VRPs are explained in 2.1.

2.1 PROBLEM NOTIONS

Vehicle Routing Problems are concerned with the distribution or collection of goods or services to customers situated in different geographical locations. This collection or distribution is ensured by visiting the customers at their location. VRPs ask how to execute these visits such as to minimize and/or maximize some quantity while respecting a set of constraints. This section introduces a high-level description of the different elements encountered in VRPs.

Customers	Customers represent the geographical locations that must be visited in order to perform some action. Such an action typically corresponds to a service or the delivery or pick-up of some goods.
Vehicles	Vehicles are the entities traveling to the customers in order to perform the needed action. It is common to have further properties attached to vehicles, delimiting the number or the set of customers a specific vehicle may visit.
Depot	Depots are geographical locations that differ in some way from customers. Often no action needs to be performed at the depot. In many basic VRPs, depots represent the "home" of the vehicles. That is the vehicles start from the depot in order to visit customers and return to the depot after having finished this task.
Road network	Vehicles use the road network in order to travel between locations. This network is usually represented by a graph, where customers and depots are represented by vertexes and the different road segments by edges linking the vertexes. Typically a cost is associated with traveling from one location to another in the road network, and thus also with traveling from one vertex to another in the corresponding graph.
Routes	A route corresponds to a sequence of locations visited by one vehicle. In many VRP variants routes start and end at a depot, visiting only customers in between.
Fleet	The fleet corresponds to the set of vehicles available to perform the visits to the customers. This fleet can

be homogeneous (all vehicles in the fleet share the same properties) or heterogeneous (different types of vehicles are available).

Objective The objective of VRPs is typically the minimization and/or maximization of some quantity. The most basic objective is the minimization of the total distance traveled by all vehicles. Another possible objective is the maximization of a profit incurred by visiting only a subset of the customers.

Solution In the more basic VRP variants a solution corresponds to a set of routes with a mapping from each route to a vehicle in the fleet. In more complex variants additional information may be necessary to characterize a solution, as for example a time schedule for each vehicle. Each solution allows to evaluate the objective and thus to assign a cost or gain to the solution.

Constraints Constraints are conditions that need to be met in a solution to the problem. Such constraints can apply to the shape and the number of the routes, but often additional properties associated with customers and vehicles give rise to further constraints, limiting for example the possible combinations of customers a vehicle can visit.

2.2 THE CAPACITATED VEHICLE ROUTING PROBLEM

The most basic Vehicle Routing Problem is the Capacitated Vehicle Routing Problem (CVRP). The CVRP is also the problem underlying to all VRP variants considered in the contributions of this thesis. This section describes the problem and gives the relevant notions.

2.2.1 *Problem Description*

The Capacitated Vehicle Routing Problem is occupied with visiting a set of customers with associated demands using a given number of vehicles of limited capacity. Vehicles are kept at a depot and start and end their routes at this depot. The goal is to find a route for each vehicle such that all customers are visited, the capacity of the vehicles is not exceeded and such that the total distance traveled by all the vehicles is minimized.

The problem is defined on a simple, complete and weighted graph $G = (V, A)$ representing the road network. The set of vertexes $V = \{0, \ldots, n\}$ represents the different locations and the set of arcs A represents the different roads in the network. The set of outgoing (incoming) arcs from vertex $i \in V$ is denoted by δ_i^- (δ_i^+). Vertexes $i = 1, \ldots, n$ are called customers, while vertex 0 is called depot. The weight c_{ij} of an arc $(i, j) \in A$ equals the distance on the road network between the locations matching vertexes i and j and thus represents the cost of including this arc in a solution. The distances are considered to be symmetric ($c_{ij} = c_{ji} \quad \forall i, j \in V$) and to respect the triangle inequality ($c_{ij} \leq c_{ip} + c_{pj} \quad \forall i, j, p \in V$). With each customer $i \in V \backslash \{0\}$ is associated a demand q_i. A set of homogeneous vehicles K, each of limited capacity Q, is available to perform the visits to the customers. Each vehicle may execute at most one route. Each route starts from the depot to visit a number of customers and then returns back to the depot. Since by definition the first and last vertexes in a route are the depot vertex, a route r can be seen as a tuple (S, σ) where $r.\sigma = \langle e_1, \ldots, e_m \rangle$ ($\bigcap_{i=1}^{m} \{e_i\} = \emptyset$) is a sequence of customers ($\{e_1, \ldots, e_m\} \subseteq V \backslash \{0\}$) and $r.S = \{e_1, \ldots, e_m\}$. The customer visited in the s^{th} ($1 \leq s \leq |r.S|$) position of route r is given by $r[s]$. For any route r, $r[0]$ and $r[|r.S| + 1]$ represent the depot. Note that if for any route r we have $r[s] = i$ and $r[s+1] = j$ ($0 \leq s \leq |r.S|$), this means that arc (i, j) is used in this route.

Finally the goal is to devise a solution $Sol = \{r_1, \ldots, r_m\}$ amounting to a set of routes such that:

1. $|Sol| \leq |K|$
   the solution does not use more vehicles than available

2. $\bigcap_{r_i \in Sol} r_i.S = \emptyset$ and $\bigcup_{r_i \in Sol} r_i.S = V \backslash \{0\}$
   each customer is visited exactly once

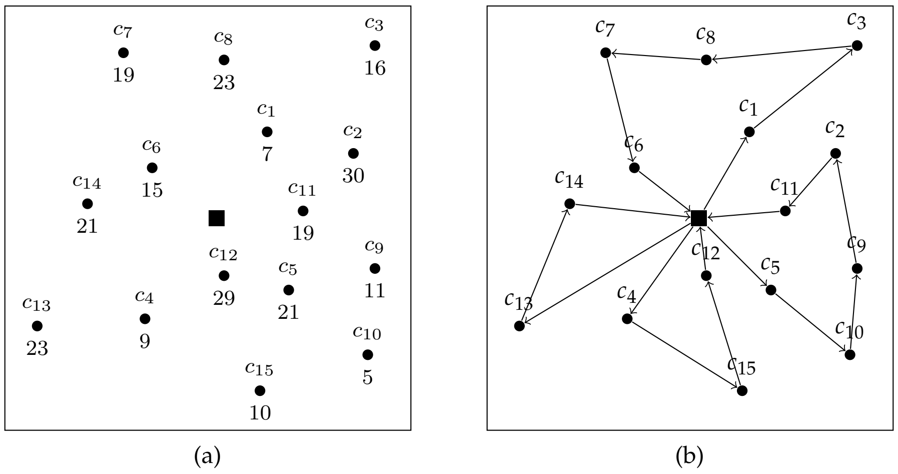


Figure 2.1: A CVRP instance with $|K| = 4$ and $Q = 90$ (a) and one of its feasible solutions (b). The customer demands are indicated below each customer vertex in the instance. In order to improve visibility the customer demands are not indicated in the solution.

3. $\sum_{j \in r.S} q_j \leq Q \qquad \forall r \in Sol,$
   the sum of the customer demands on a route does not exceed the vehicle capacity Q

4. min $Dist(Sol) = \sum_{r \in Sol} (\sum_{s=0}^{|r.S|} c_{r[s]r[s+1]})$
   the total distance is minimized

A solution respecting constraints 1-3 is called *feasible*. A solution respecting only part of, or none of these constraints is called *infeasible*. The quality of a solution Sol is evaluated using $Dist(Sol)$. Constraint 4 means that a solution Sol' is considered of higher quality than a solution Sol if $Dist(Sol') < Dist(Sol)$.
A solution Sol^* is called optimal if it is feasible and no other feasible solution Sol'' s.t. $Dist(Sol'') < Dist(Sol^*)$ exists.
Deciding whether a feasible solution for a given CVRP instance exists is NP-complete, while the problem of finding the optimal solution is NP-hard ([TV02]). Note that the Traveling Salesman Problem ([Coo11]) corresponds to a CVRP with $|K| = 1$ and $Q = \infty$. An example of a CVRP instance and of a feasible solution for this instance are given in Fig. 2.1.

2.2.2 *Variations of the CVRP*

A number of variants of the CVRP have been covered in the scientific literature see e.g. [GRW08]. One can consider three types of variations on the CVRP resulting in a new problem:

1. modifications to the structure of the routes
2. modifications to the objective function
3. additional constraints to be respected by the routes

In the following for each modification three examples of problems resulting from the considered variation are given.

Modifications to the structure of the routes

- **Multi-depot Vehicle Routing Problem** Instead of a single depot multiple depots are available. Routes start from any of the depots but must end at the same depot. See e.g. [CGL97].
- **Capacitated Vehicle Routing Problem With Split Delivery** Customers can be visited more than once by different vehicles. Their demand is split between different routes. It has to be determined which vehicle delivers which quantity of product to customers visited in several routes. See e.g. [ASH06].
- **Capacitated Vehicle Routing Problem With Multiple Trips** Vehicles may execute more than one route. Each route starts and ends at the depot. See e.g. [TLG96].

Modifications to the objective function

- **Vehicle Routing Problem with Profits** A profit is associated with visiting each customer. Not all customers must be visited in this problem. The goal is to find a solution that maximizes the total profit, defined as difference between the profit collected at the visited customers and the total distance covered by the fleet. See e.g. [ASV13].
- **MinMax Vehicle Routing Problem** Instead of minimizing the total distance the goal is to minimize the distance of the route with the highest distance (i.e. of the 'longest' route). See e.g. [GLT97].

- **Vehicle Routing Problem with Minimization of the vehicle fleet** Often the minimization of the number of vehicles necessary to execute all routes is stated as a primary objective in a lexicographic objective function. The secondary objective is the minimization of the total distance. See e.g. [BVH04]

Additional constraints to be respected by the routes

- **Capacitated Vehicle Routing Problem with Time Windows** Time Windows are associated with the depot and the customers. The service at each customer must start inside the customer's time window. The vehicle may however arrive early at a customer and wait for the start of the customer's time window. The service at each customer uses up a predefined amount of time. Vehicles must return to the depot by the end of the depot's time window. The time needed to travel between locations is relative to the distance between these locations. See e.g. [Sol87].

- **Capacitated Vehicle Routing with Pick-up and Delivery** With each customer is associated a pick-up and a delivery location. Some commodity is gathered at the pick-up location and must then be dropped off at the corresponding delivery location in the same route. This introduces precedence constraints between pick-up and delivery locations. Furthermore the currently used vehicle loading space will not steadily decrease or increase during the execution of the route. See e.g. [PDH08].

- **Capacitated Vehicle Routing Problem with Loading Constraints** The customers' demands are expressed as a set of multi-dimensional items. The loading space of the vehicle takes the same number of dimensions and is limited in each of those dimensions. For each route, a feasible loading of the items belonging to the customers visited on this route must exist in the loading space of the vehicle. See e.g. [GILM08].

This thesis focuses on VRPs that result from adding additional non-structural constraints to the vehicle routes that concern these routes individually. Further examples of such problems and the methods used to solve them are given in chapter 3.

2.2.3 *Optimization of the CVRP*

The goal of optimization methods is to find a good quality solution for a given optimization problem. Good quality solution means a solution that optimizes the value of the problem's objective function. Such approaches can either be *complete* or *incomplete*. *Complete* approaches are guaranteed to find an optimal solution and prove that this solution is an optimum. *Incomplete* approaches can find optimal solutions but are not guaranteed to do so and are not able to prove the optimality of a solution. The approaches presented here can also be split into *constructive* and *perturbative* approaches. *Constructive* approaches construct solutions from scratch while *perturbative* approaches modify a current solution in order to obtain a new solution.

Before entering the topic of optimization methods recurring notations used in the rest of this thesis are presented in the next section 2.3. The following sections 2.4 to 2.7, 2.9 and 2.10 will review the most common optimization methods. For each approach its basic principles will be outlined first. Then additional elements on how to apply the given method to the CVRP are given. In the following it is assumed that the considered optimization problems are minimization problems.

2.3 NOTATIONS AND OPERATIONS ON ROUTES

As mentioned earlier a route r can be seen as a tuple (S, σ) where $r.S$ represents the set of customers visited by r and $r.\sigma$ the sequence in which they are visited.

Positions of customers in a route
For each route r, $r.\sigma = \langle e_1, \ldots, e_m \rangle$ represents the sequence in which customers in $\{e_1, \ldots, e_m\} (= r.S)$ are visited. The vertex visited in the s^{th} position (i.e. e_s, $1 \leq s \leq |r.S|$) can be obtained by $r[s]$, i.e. $r[s] = e_s$. Furthermore $r[0]$ and $r[|r.S| + 1]$ default to 0, the depot vertex. The other way around, the position of a customer v in r s.t. $v \in r.S$ (the position corresponds to index s s.t. $e_s = v$) is obtained via $pos(v, r)$. Thus $r[pos(v,r)] = v \wedge e_{pos(v,r)} = v \quad \forall v \in r.S$. Finally the first and last customers visited in r are retrieved using $first(r)$ and $last(r)$. If $|r.S| \geq 1$, $first(r) = r[1]$ and $last(r) = r[|r.S|]$, else if $|r.S| = 0$ the

route is "empty" and $first(r) = last(r) = 0$.

Example: Let route r_{ex} where $r_{ex}.S = \{v_a, v_b, v_c\}$ and $r_{ex}.\sigma = \langle v_a, v_b, v_c \rangle$. The second customer in the route is v_b, $r_{ex}[2] = v_b$ and $pos(v_b, r_{ex}) = 2$. Furthermore $r_{ex}[0] = 0$ and $r_{ex}[3+1] = 0$. Finally $first(r) = v_a$ and $last(r) = v_c$.

Accumulated demand on a route
Given route r the accumulated demand on r is given by
$demand(r) = \sum_{i \in r.S} q_i$.

Distance of a route
Given route r the distance of r is given by $distance(r) = \sum_{i=0}^{|r.S|} c_{r[i]r[i+1]}$.

Operations on routes
Many solution and optimization approaches modify routes in some way or another. The most basic operations are defined here.

Inserting a customer in a route

Consider a route r with $r.\sigma = \langle e_1, \dots, e_{|r.S|} \rangle$. The insertion of customer i ($i \notin r.S \wedge i \neq 0$) into route r at position p
($1 \leq p \leq |r.S| + 1$) is denoted by $insert(i, r, p)$. Let r be the original route and $r' = insert(i, r, p)$. Then $r'.S = r.S \cup \{i\}$ and $r'[t] = r[t]$ $\forall\, 1 \leq t < p$ and $r'[p] = i$ and $r'[h] = r[h-1]$ $\forall\, p+1 \leq h \leq |r'.S|$. Analogously $r'.\sigma = \langle r.\sigma_1^{p-1}, i, r.\sigma_p^{|r.S|} \rangle$ where $r.\sigma_i^t$ denotes the possibly empty subsequence $\langle e_i, \dots, e_t \rangle$ of $r.\sigma$.

Example: Let route r_{ex} with $r_{ex}.S = \{v_a, v_b, v_c\}$ and $r_{ex}.\sigma = \langle v_a, v_b, v_c \rangle$, the insertion of i in the 2nd position in r_{ex} results in $r'_{ex} = insert(i, r_{ex}, 2)$ where $r'_{ex}.S = \{v_a, v_b, v_c, i\}$ and $r'_{ex}.\sigma = \langle v_a, i, v_b, v_c \rangle$. Similarly the insertion of i in the 4th position in r_{ex} results in $r''_{ex} = insert(i, r_{ex}, 4)$ where $r''_{ex}.S = \{v_a, v_b, v_c, i\}$ and $r''_{ex}.\sigma = \langle v_a, v_b, v_c, i \rangle$.

Removing a customer from a route

The removal of a customer i ($i \in r.S$) from route r is denoted by $remove(i, r)$. Let r be the original route and $r' = remove(i, r)$. Then $r'.S = r.S \backslash \{i\}$ and $r'[t] = r[t] \quad \forall\, 1 \leq t < pos(i, r)$, and $r'[h] = r[h+1]$

$\forall\, pos(i,r) \leq h \leq |r'.S|$. Analogously $r.\sigma' = \langle r.\sigma_1^{pos(i,r)-1}, r.\sigma_{pos(i,r)+1}^{|r.S|} \rangle$.

Example: The removal of v_b from route r_{ex} results in
$r'_{ex} = remove(v_b, r_{ex})$ where $r'_{ex}.S = \{v_a, v_c\}$ and $r'_{ex}.\sigma = \langle v_a, v_c \rangle$.

Merging two routes

Consider two routes r_1 and r_2 such that $r_1.S \cap r_2.S = \emptyset$ and where $r_1.\sigma = \langle e_{11}, \ldots, e_{1|r_1.S|} \rangle$ and $r_2.\sigma = \langle e_{21}, \ldots, e_{2|r_2.S|} \rangle$. Merging r_1 with r_2 is denoted by $r = r_1 \times r_2$ where $r.S = r_1.S \cup r_2.S$ and
$r.\sigma = \langle e_{11}, \ldots, e_{1,|r_1.S|}, e_{21}, \ldots, e_{2|r_2.S|} \rangle$.
Analogously $r[i] = r_1[i]\ \forall\, i \in 1, \ldots, |r_1.S|$ and $r[j + r_1.|S|] = r_2[j] \quad \forall\, j \in 1, \ldots, |r_2.S|$. Note that the merge operator $\times$ is not commutative, i.e. $r_1 \times r_2 \neq r_2 \times r_1$.

Example: Given two routes route r_x and r_y where $r_x.S = \{v_a, v_b, v_c\}$, $r_x.\sigma = \langle v_a, v_b, v_c \rangle$ and $r_y.S = \{v_g, v_h, v_i,\}$, $r_y.\sigma = \langle v_g, v_h, v_i \rangle$ the merge of r_x and r_y results in $r = r_x \times r_y$ where $r.S = \{v_a, v_b, v_c, v_g, v_h, v_i\}$ and $r.\sigma = \langle v_a, v_b, v_c, v_g, v_h, v_i \rangle$. The merge of r_y and r_x results in route $r' = r_y \times r_x$ where $r'.S = \{v_a, v_b, v_c, v_g, v_h, v_i\}$ and
$r'.\sigma = \langle v_g, v_h, v_i, v_a, v_b, v_c \rangle$.

2.4 CONSTRUCTION HEURISTICS

Construction heuristics are *incomplete* and *constructive* methods. The goal of a construction heuristic is to build a solution to a problem. This is done by iteratively adding features to an initially empty solution until a complete solution is obtained. Heuristics aim at obtaining good quality solutions.
Construction heuristics for VRPs construct a set of routes. During construction they try to keep the total distance of the solution as small as possible. In order to achieve this, the best possible way to extend the current route or solution is chosen (this is called *greedy* behavior) at each step. Such decisions are myopic as they consider only the current situation, taking decisions that might be good now but bad in the bigger picture. Note that while Construction heuristics for the VRP are guaranteed to provide a set of feasible routes, it is possible that these routes cannot be combined into a feasible solution if the number of routes obtained is higher than the number of available vehicles. In the

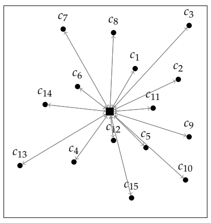
following, three commonly encountered construction heuristics for the CVRP are presented.

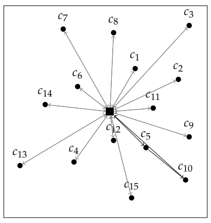
2.4.1 *Savings Heuristic*

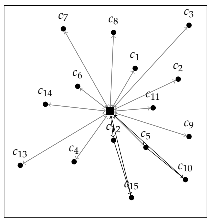
Clarke and Wright's Savings Heuristic [CW64] is one of the most well-known and oldest construction heuristics. In the original paper a parallel and a sequential version are proposed. Here the parallel version is presented. The heuristic starts by computing for every ordered pair (i, j) $(i, j \in V \backslash \{0\},\ i \neq j)$ a savings value as $s_{ij} = c_{i0} + c_{0j} - c_{ij}$. This value corresponds to the distance that can be saved by merging two routes r_1 and r_2 s.t. $last(r_1) = i$ and $first(r_2) = j$. The heuristic then puts each individual customer in a route of its own, such that we have $Sol = \{r_1, ..., r_n\}$ $(n = |V \backslash \{0\}|)$ and $r_i.S = \{i\}$ $(1 \leq i \leq n)$. The heuristic then starts merging the routes in Sol.
At each iteration the ordered pair (i,j) maximizing s_{ij} and such that $\exists\, r_a, r_b \in Sol$ with $last(r_a) = i$ and $first(r_b) = j$ and with $demand(r_a) + demand(r_b) \leq Q$ is determined. Route r_a is then merged with route r_b. That is a new route $r' = r_a \times r_b$ is created and used to replace r_a and r_b in Sol. The new set of routes Sol thus becomes $Sol = (Sol \backslash \{r_a, r_b\}) \cup \{r'\}$.
The heuristic continues merging routes until no further merge is possible given the current set of routes in Sol. It is this set of routes obtained at the end of the heuristic that is returned as solution. As mentioned before, if this set contains more routes than available vehicles in the problem instance (i.e. $|Sol| > |K|$), the solution is not feasible. See Figure 2.2 for a visualization of the savings heuristic.

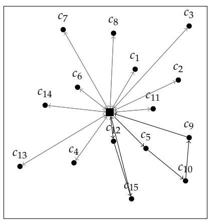
2.4.2 *Insertion Heuristics*

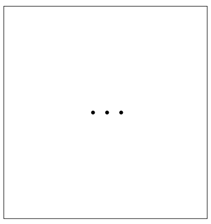
Several insertion heuristics have been proposed in the literature. The idea of these heuristics is to build a solution by inserting customers one after another into open routes. A route is said to be "open" if it is to be considered for insertion of new customers, "closed" if it isn't. Insertion heuristics can be sequential, that is there is always only one open route, or parallel, where several routes are open at the same time. Initially no customer is routed. At each iteration the heuristic needs to decide which customer to insert at which position in which route. In the simplest case, there is only one open route and among the cus-


c_7 c_8 c_3 c_1 c_2 c_6 c_{14} c_{11} c_{12} c_5 c_9 c_{13} c_4 c_{10} c_{15}
(a)


c_7 c_8 c_3 c_1 c_2 c_6 c_{14} c_{11} c_{12} c_5 c_9 c_{13} c_4 c_{10} c_{15}
(b)


c_7 c_8 c_3 c_1 c_2 c_6 c_{14} c_{11} c_{12} c_5 c_9 c_{13} c_4 c_{10} c_{15}
(c)


c_7 c_8 c_3 c_1 c_2 c_6 c_{14} c_{11} c_{12} c_5 c_9 c_{13} c_4 c_{10} c_{15}
(d)



(e)

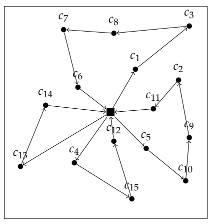

c_7 c_8 c_3 c_1 c_2 c_6 c_{14} c_{11} c_{12} c_5 c_9 c_{13} c_4 c_{10} c_{15}
(f)

Figure 2.2: Execution of the savings heuristic on a CVRP.

tomers that can be feasibly inserted in the open route the customer closest to the currently last visited customer in the open route is inserted at the end of this route (Nearest Neighbor Heuristic). More sophisticated versions consider inserting customers in different spots in the open routes.
Here the sequential Insertion Heuristic proposed in [MJ76] is presented in order to show the spirit of these heuristics. The following explanation is based on the one given in [TV02].

The Mole and Jameson heuristic starts with an empty route r_{cur} ($r_{cur}.S = \emptyset$) as the current open route and an initially empty set of routes $Sol = \emptyset$. At each iteration the profit of inserting in r_{cur} each non-routed customer i s.t. $q_i + demand(r_{cur}) \leq Q$ is computed. This is done in two steps.
First the minimal detour α_{ip} of visiting the customer in the current open route is determined. To do this the cost of inserting i in every position p ($1 \leq p \leq |r_{cur}.S| + 1$) in r_{cur} is computed as $\alpha_{ip} = c_{r_{cur}[p-1]i} + c_{ir_{cur}[p]} - \lambda c_{r_{cur}[p-1]r_{cur}[p]}$ where λ is a parameter. This α_{ip} value is minimal for some p, and this minimized value corresponds to the smallest possible detour. Next the distance that is saved by visiting i in r_{cur} rather than in a route of its own is evaluated. This is done along the formula $\beta_i = \mu c_{0i} + \alpha_{ip}$ where μ is a parameter. Finally the customer i

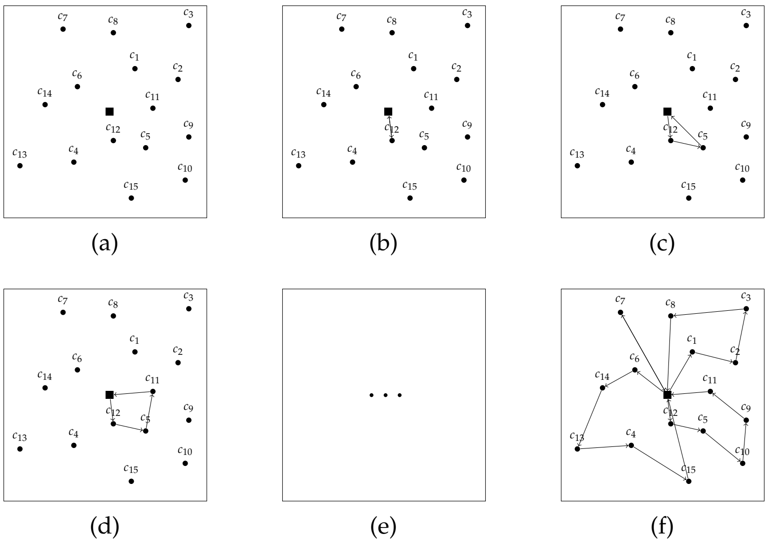


Figure 2.3: Execution of the Mole and Jameson Heuristic on a CVRP.

maximizing β_i is selected and inserted in the position p of r_{cur} causing the minimal detour ($r_{cur} = insert(i, r, p)$). The resulting route is then post-optimized using a 3-exchange (3-opt) optimization (see 2.5). Once no more customers can be feasibly added to the open route r_{cur}, r_{cur} is closed and added to Sol ($Sol = Sol \cup \{r_{cur}\}$) and a new empty route r_{cur} is opened. This procedure continues until all customers have been included in a route and the routes have been added to Sol. As for the Savings Heuristic, Sol can, at the end of the procedure, contain more routes than available vehicles. The execution of the Mole and Jameson Heuristic is visualized in Fig. 2.3.

2.4.3 *Sweep Heuristic*

The Sweep Heuristic is an instance of a so-called *Cluster first, route second* heuristic. These heuristics work in two phases. In a first phase the customers are partitioned into sets (clusters). In a second phase one route per cluster, visiting all the customers in the cluster is computed. This latter step is typically implemented by solving a Traveling Salesman Problem per cluster.

The Sweep Heuristic supposes the vertexes in V are distributed on a plane. With each customer $i \in V \backslash \{0\}$ are associated its polar coordi-

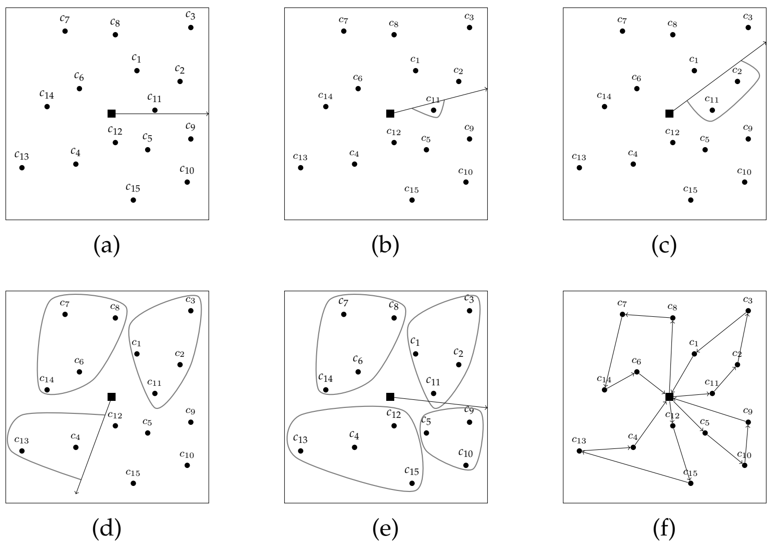


Figure 2.4: Intermediate steps in the execution of the Sweep Heuristic on a CVRP. Note that the Sweep heuristic is designed for instances where the customers are distributed in clusters.

nates w.r.t. the depot (θ_i, ρ_i), with for some customer $j \in V \backslash \{0\}$ $\theta_j = 0$. The heuristic starts with a current empty route r_{cur} ($r_{cur}.S = \emptyset$). A ray centered at the depot performs a full rotation, sweeping over the customer vertexes, in such a way that customer $j \in V \backslash \{0\}$ s.t. $\theta_j = 0$ is encountered first by the ray. The moment customer v is swept over by the ray, if $demand(r_{cur}) + q_v \leq Q$ it is added to the set of customers visited in the currently open route r_{cur} ($r_{cur}.S = r_{cur}.S \cup \{v\}$). If the customer v cannot be added to $r_{cur}.S$ then r_{cur} is closed and added to Sol ($Sol = Sol \cup \{r_{cur}\}$), a new empty route r_{cur} is opened and v is added to this new route. Once a full rotation has been performed the current route r_{cur} is added to Sol and all customers are visited in Sol. Then for every route r in Sol a Traveling Salesman Problem is solved to decide on the optimal sequence in which to visit the vertexes in the set $r.S \cup \{0\}$. Once this sequence has been determined $r.\sigma$ is adapted accordingly.

As for the previous heuristics, Sol may use more vehicles than available at the end. An example of the execution of the Sweep Heuristic on a CVRP instance is given in Fig. 2.4.

2.5 LOCAL SEARCH

Local Search is an *incomplete* and *perturbative* method. In the following only main concepts are presented, for a more exhaustive overview please refer to [HS04].
The basic idea of local search is fairly simple. The search considers solutions one by one and records the best quality feasible solution ever encountered. The space containing all possible solutions (feasible and infeasible) to a problem is called the *solution space*. Local Search is based on the observation that by modifying a given (feasible or infeasible) solution a new, different (feasible or infeasible) solution is obtained. Local Search thus moves through the solution space, each step corresponding to a perturbation of the current solution. Local Search will analyze each solution it encounters to check its feasibility (if needed) and records the best quality feasible solution encountered so far (called the *incumbent* solution). The process is stopped once a given stopping criterion is reached (e.g. execution time or iterations without improvement to the incumbent).
Given a current solution Sol the set of solutions that can be obtained by performing an operation op on Sol is called the neighborhood $N_{op}(Sol)$ of Sol. The operation op is performed by applying a neighborhood operator $\eta_{op}(\ldots, Sol)$ to the current solution. Such an operator typically takes several parameters indicating which parts of the current solution will intervene in the perturbation. The size of the neighborhood $N_{op}(Sol)$ of Sol depends on the different parameters considered for $\eta_{op}(\ldots, Sol)$. At each iteration in Local Search the neighborhood of the current solution is constructed and evaluated. One of the neighboring solutions is then selected as the new current solution. Note that often more than one neighborhood operator is used in Local Search. A visualization of the different concepts is given in Fig. 2.5.

The selection of the neighboring solution to move to is typically done using either a *First Improvement* or a *Best improvement* strategy. In *First Improvement* the search evaluates the neighboring solutions during the construction of the neighborhood and as soon as a neighboring solution improving the quality of the current solution is found, the search moves to it. In *Best improvement* the complete neighborhood is constructed and evaluated. The solution improving the solution quality the most is selected as new current solution. In a randomized *Best improvement* strategy one out of the ω best neighboring solutions is se-

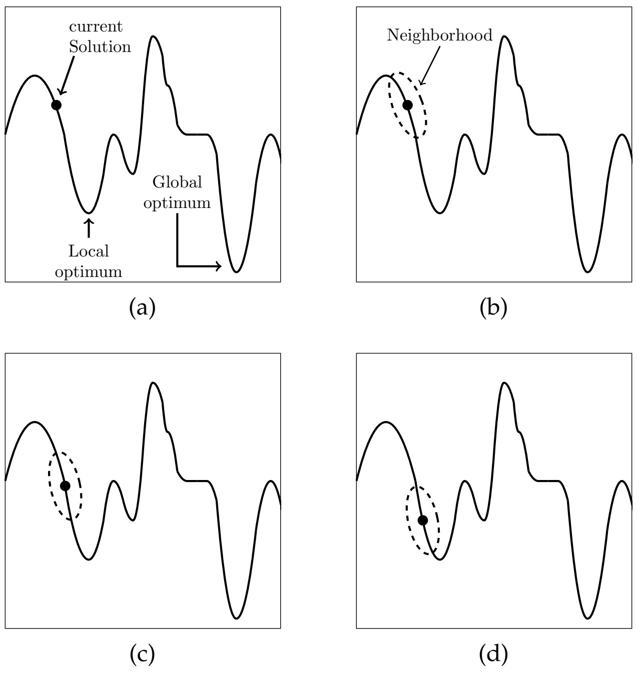


Figure 2.5: Local Search moving through the solution space towards a local optimum.

lected at random (ω being a parameter).

Two important concepts in Local Search are *Intensification* and *Diversification*. The solution space may contain optimal solutions and locally optimal solutions. Optimal solutions are called globally optimal to differentiate from locally optimal solutions. A local optimum is a solution that is optimal only in its neighborhood (i.e. none of the neighboring solutions improves it).
Intensification means that the search is concentrated in a specific area of the solution space as this area seems most promising (typically with the goal of ending up at a local optimum). *Diversification* means that the search is forced to explore different parts of the solution space in order to make sure different areas are covered and the search does not keep returning to the same local optimum. *Intensification* and *Diversification* measures are commonly implemented in the evaluation and selection of neighboring solutions.

Finally random decisions play a big role in the efficiency of Local Search approaches. Random decisions can be included in the choice of the neighborhood operator to use in the current iteration, the parameters used to construct this neighborhood, the selection of the neighboring solution or the restart from a randomly generated solution.

2.5.1 *Application to the CVRP*

When implementing a Local Search algorithm for a specific problem the following points need to be considered:

A) How is the initial solution constructed?

B) How is the quality of solutions evaluated?

C) How is the neighborhood of a solution computed?

Local Search has been extensively applied to Vehicle Routing Problems and has been proven to be efficient especially on large-scale instances where exact methods are intractable. In the remainder of this section examples of how the main local search steps are handled for CVRP are given.

(A) Construction of the initial solution
The initial solution is commonly built using construction heuristics such as for example the ones presented in 2.4. Sometimes random decisions are included in these heuristics, in order to be able to generate number of different initial solutions. An important decision that has to be taken is whether the initial solution should be forced to be feasible or not. An infeasible solution could for example use more vehicles than available in the fleet, or not respect the capacity constraints.

(B) Solution evaluation
In the most common cases the evaluation of a solution corresponds to the evaluation of the objective function on this solution. However when the search is allowed to visit infeasible solutions often a modified objective function is used for the evaluation. Infeasibility for the CVRP could correspond to using more vehicles than available or exceeding the capacity of the vehicles. The modified objective function includes then a measure of infeasibility, as for example the sum of the excess

demands per vehicle. The idea is to guide the search towards feasible solutions by penalizing infeasible solutions (the higher the infeasibility the higher the resulting objective value). Note that the violation of structural constraints such as the number of visits to a customer or routes starting and ending at the depot is usually not allowed.

(C) Construction of the neighborhood
There are typically three major decisions that need to be taken when constructing the neighborhood:

- are infeasible solutions considered?
- is the full neighborhood or only a portion computed?
- which neighborhood operator to use?

If the initial solution is allowed to be infeasible it makes sense to allow the search to move to infeasible neighboring solutions as it might be impossible to find a feasible solution in the neighborhood of the initial infeasible solution. Sometimes the search is not allowed to visit infeasible solutions anymore once a feasible solution has been encountered. In such cases the first focus of the search (and thus also of the neighborhood operators) is to find a feasible solution and once a feasible solution has been found, the focus shifts on the solution quality. The decision on whether or not a neighborhood should contain infeasible solutions is enforced through the choice of parameters for the corresponding neighborhood operator.

Computing the full neighborhood means considering all possible parts of the current solution for perturbation by a neighborhood operator. To construct a reduced neighborhood only a subset (possibly selected at random) of such parts is considered. Of course constructing the full neighborhood comes with a higher computation cost compared to constructing a reduced neighborhood, but potentially contains more improving solutions.

Neighborhood operators constitute the core of Local Search procedures. Typically one out of several neighborhood operators is selected at random to compute the neighborhood of the current solution. In the following, several well-known neighborhood operators for the CVRP are presented.

Relocate operator The relocate operator $\eta_{reloc}(i, r_1, p, r_2, Sol)$ ($r_1, r_2 \in Sol$) takes a customer i currently visited in r_1 ($i \in r_1.S$), removes it from r_1 and reinserts it at position p in route r_2 ($1 \leq p \leq |r_2.S| + 1$). The size of $N_{reloc}(Sol)$ is determined by the different routes, customers and positions considered for η_{reloc}.

Example: Let $Sol = \{r_1, r_2, r_3, r_4\}$ with $r_1.\sigma = \langle v_a, v_b, v_c\rangle$ and $r_2.\sigma = \langle v_d, v_e, v_f\rangle$. Then $\eta_{reloc}(v_b, r_1, 2, r_2, Sol)$ results in the modified solution $Sol' = \{r'_1, r'_2, r_3, r_4\}$ with modified routes $r'_1.\sigma = \langle v_a, v_c\rangle$ and $r'_2.\sigma = \langle v_d, v_b, v_e, v_f\rangle$.

Swap operator The swap operator $\eta_{swap}(i, r_1, j, r_2, Sol)$ ($r_1, r_2 \in Sol$) takes two customers i and j visited in different routes r_1 and r_2 ($r_1 \neq r_2, i \in r_1.S, j \in r_2.S$) and exchanges them. The size of $N_{swap}(Sol)$ is determined by the different routes and customers considered for η_{swap}.

Example: Let $Sol = \{r_1, r_2, r_3, r_4\}$ with $r_1.\sigma = \langle v_a, v_b, v_c\rangle$ and $r_2.\sigma = \langle v_d, v_e, v_f\rangle$. Then $\eta_{swap}(v_b, r_1, v_d, r_2, Sol)$ results in the modified solution $Sol' = \{r'_1, r'_2, r_3, r_4\}$ with modified routes $r'_1.\sigma = \langle v_a, v_d, v_c\rangle$ and $r'_2.\sigma = \langle v_b, v_e, v_f\rangle$.

k-exchange operators The k-exchange operator $\eta_{kex}(r_1, A_1, A_2, Sol)$ ($r_1 \in Sol$ and $A_1 \cap A_2 = \emptyset$) partitions a route r_1 into $k+1$ segments by dropping all arcs in A_1 from r_1 ($A_1 \subseteq \bigcup_{j=0}^{|r.S|}\{(r[j], r[j+1])\}$) and reconnecting the resulting segments using the arcs in A_2 ($A_2 \subseteq \bigcup_{j=0}^{|r.S+1|}(\delta^-_{r[j]} \cap \bigcup_{l=0}^{|r.S+1|}\delta^+_{r[l]})\}$). Note that some of the route segments may be reversed in the resulting route. The size of $N_{swap}(Sol)$ is determined by the different routes and arc sets considered for η_{swap}.

Example: Let $r_1.\sigma = \langle v_a, v_b, v_c, v_d, v_e\rangle$, $A_1 = \{(v_a, v_b), (v_d, v_e)\}$ and $A_2 = \{(v_a, v_d), (v_b, v_e)\}$. Then $\eta_{2ex}(r_1, A_1, A_2, Sol)$ results in the modified solution $Sol' = \{r'_1, r_2, r_3, r_4\}$ with modified route $r'_1.\sigma = \langle v_a, v_d, v_c, v_b, v_e\rangle$.

Cross operator The cross operator $\eta_{cross}(i, r_1, j, r_2, Sol)$ ($r_1, r_2 \in Sol$) exchanges the segments starting with i and j and ending at the depot, in routes r_1 and r_2 ($r_1 \neq r_2, i \in r_1.S, j \in r_2.S$). The size of $N_{cross}(Sol)$ is determined by the different routes and customers considered for η_{cross}.

Example: Let $r_1.\sigma = \langle v_a, v_b, v_c \rangle$ and $r_2.\sigma = \langle v_d, v_e, v_f \rangle$ then $\eta_{cross}(v_b, r_1, v_d, r_2, Sol)$ results in the modified solution Sol' with modified routes $r'_1.\sigma = \langle v_a, v_d, v_e, v_f \rangle$ and $r'_2.\sigma = \langle v_b, v_c \rangle$.

2.5.2 *Metaheuristics*

Metaheuristics extend basic Local Search with additional decision criteria, as for example in the decision on which neighbor solution to select, the evaluation of neighboring solutions and the neighborhoods to use in a given situation. The idea is to improve the performance of Local Search by taking more intelligent decisions. Metaheuristics have widely been applied to VRPs. In this section two common metaheuristics *Tabu Search* and *Variable Neighborhood Search* will be sketched out.

Tabu Search [GL98]
Tabu Search has been designed in order to help Local Search to escape from local optima. An additional structure called a *Tabu List* is added. In this list, aspects of solutions (complete solutions being intractable) that have already been encountered by the search are stored over a given number of iterations (corresponding to the length of the list). Every time the search wants to move to a neighboring solution, this solution is first scanned for forbidden aspects that are in the tabu list. If the neighboring solution is tabu (i.e. it contains aspects in the tabu list) then it is evaluated to verify whether it does improve the incumbent solution. If it does, the tabu criterion is overridden and the search may move to the neighboring solution even though it is tabu (this is called the *aspiration criterion*). If the neighboring solution is tabu and does not improve the best known solution then the search may not move to it and must select a different neighboring solution. Often the length of the tabu list is adapted dynamically in order to enhance *intensification* and *diversification*. If the search moves to a solution improving over the previous one, the length of the list is increased in order to intensify the search in the current area of the solution space. On the other hand if the solution quality deteriorates with respect to the previous solution the length of the list is reduced in order to allow the search to move on quickly, or even allow it to go back to the better solution after some iterations.

There are numerous examples of Tabu Search applied to CVRP and other variants of the VRP, see e.g. [CLM01, GHL94, Ba09]. A typical aspect that can be set tabu is which customer is visited in which route. If the search moves to a new solution by moving customer i from r_1 to r_2, moving i back to r_1 is set tabu.

Variable Neighborhood Search [MH97]
Instead of moving linearly as in classical Local Search, Variable Neighborhood Search (VNS) jumps through the solution space. This is done by using a set of k different Neighborhoods. Often these neighborhoods are ordered from 1 to k by increasing size. At each iteration of the search one neighborhood $N_i(Sol)$ $(1 \leq i \leq k)$ of the current solution Sol is computed. A neighboring solution Sol' in this neighborhood is selected at random. Sol' then figures as the initial solution for a basic Local Search procedure, which moves the search to a local optimum Sol''. If Sol'' improves the best known solution so far, the search moves there and it serves as the initial solution for a new VNS iteration using neighborhood N_1. If Sol'' doesn't improve the best known solution, Sol is kept as the current solution and the next neighborhood (if available) is selected (by doing $i = i + 1$) to be used in the next iteration.

Variable Neighborhood Search has been used on the CVRP but also other complex VRP variants, see e.g. [KNBG07, PDH10]. The neighborhoods used are typically similar to these seen in this section. Often the neighborhood operator used over several neighborhoods stays the same, but it is the size of the neighborhood that increases.

2.6 ANT COLONY OPTIMIZATION

Ant Colony Optimization (ACO) is a *incomplete* and *constructive* method. For a more exhaustive overview of ACO please refer to [DBS06]. The idea of ACO is to mimic the foraging behavior of ants. Each individual ant deposits pheromones on its way from the nest to the food source and back. When choosing between several possible ways the ants tend to favor the ways with higher pheromone deposits. If the ants are confronted with a long and a short path between nest and food source, the pheromone quantity on the shorter path will increase more quickly, as the time needed to traverse it is shorter than for the longer path. This will cause the ants to favor the shorter path and to deposit even more pheromones, leading the majority of ants to con-

verge towards traveling on the shorter path.
In Ant Colony Optimization, agents, called ants, construct solutions to the problem being optimized. An ant starts from an empty solution and extends it in several steps such as to reach a full solution. At each step the ant needs to decide how to further extend the current partial or empty solution into a new (possibly) partial solution. This means that at each step the ant needs to choose one among several ways to extend the current partial (or empty) solution by choosing one out of several alternative paths. This decision will be influenced by the pheromone quantity already deposed on the different paths the ants can choose from. Each ant in the colony will continue execution until it has built a full solution or fails to extend the current solution in a feasible way. Often these solutions are then optimized using a Local Search. In each iteration of ACO, an entire colony containing ℓ ants is executed. At the end of such an iteration, the incumbent solution is possibly updated and pheromones are deposited on the paths the ℓ ants took to build their solutions. This is done in a way that is proportional to the quality of the given solutions.
Graph-based problems such as the VRP are particularly well-suited to Ant Colony Optimization. This is because solutions to these problems correspond to sets of paths in the problem graph. An ant will select at each step an arc in the problem graph to add to the current solution Sol_{par}. Pheromones will then finally be deposited on the arcs appearing in the solutions constructed by the ants.

2.6.1 *Adaptation to the CVRP*

When adapting an Ant Colony Optimization approach to a particular problem the following choices need to be adapted:

A) How do the ants construct a solution?

B) How do the ants choose the next step to take?

C) How is the solution post-optimized?

D) How is the pheromone deposit updated?

There are numerous examples of CVRP and variants being handled using Ant Colony Optimization algorithms, see e.g. [RSD02, GTA99, FDHI10]. Note also that there are several variants of Ant Colony Optimization (AS, MinMax, etc.) differing most importantly on when and how pheromones are updated. The examples given in the rest of this section are based on a basic Ant System where the ants execute a simple Insertion heuristic.

(A) Construction of a solution
Each ant executes a construction heuristic. Again the heuristics seen in section 2.4 are appropriate choices. An ant executing a simple Insertion Heuristic will start at the depot and then at each step select the next vertex to move to. This means the ant builds one route at a time. At each step it will either choose to move to a customer vertex that hasn't been visited so far, or choose to move back to the depot, thereby closing the current route and opening a new current route with its next move. Using this heuristic, the ant can end up constructing a solution using more vehicles than available. There are two options in this case, either it is considered that the ant has failed to build a feasible solution, or the constructed solution is post-optimized by a Local Search with the objective of rendering the solution feasible.

(B) Selection of the next step
At each step the ant will choose which vertex to visit next, which corresponds to choosing an arc to add to its path. The ant's choice is random but biased towards arcs with a higher pheromone deposit and locally more attractive (based on some heuristic information). Let Sol_{cur} be the current set of closed routes, r_{cur} the current open route and v_{cur} the vertex the ant last added to its path. $Ex(Sol_{cur} \cup r_{cur})$ is the set of vertexes that can be feasibly used to extend r_{cur}. $Ex(Sol_{cur} \cup r_{cur})$ is then defined as $\{j \in V \backslash Vis(Sol_{cur} \cup r_{cur}) | demand(r_{cur} + q_j \leq Q)\}$, where $Vis(R) = \bigcup_{r \in R}\{r.S\}$ is the set of vertexes already visited in the routes in R.
The probability associated with visiting vertex j next is then given by

$$p_{v_{cur}j} = \begin{cases} \frac{\tau^{\alpha}_{v_{cur}j} \cdot \eta^{\beta}_{v_{cur}j}}{\sum_{s \in Ex(Sol_{cur} \cup r_{cur})} \tau^{\alpha}_{v_{cur}s} \cdot \eta^{\beta}_{v_{cur}s}} & \text{if } j \in Ex(Sol_{cur} \cup r_{cur}) \\ 0 & \text{otherwise} \end{cases}$$

where $\tau_{v_{cur}j}$ corresponds to the pheromone deposit on arc (v_{cur}, j) and $\eta_{v_{cur}j}$ to heuristic information. This information can be for example the inverse of the distance associated with arc (v_{cur}, j), i.e. $\eta_{v_{cur}j} = \frac{1}{c_{v_{cur}j}}$.

(C) Post-optimization of a solution The solution built by an ant is commonly post-optimized using a Local Search approach. It thus has to be decided how to compute neighborhoods and select neighbors etc. as seen in section 2.5. Common operator choices correspond to these seen in said section.

(D) Update of the pheromone deposit
At the end of a complete iteration, the pheromone deposit on the problem graph will be updated. The pheromone quantity on every arc in the problem graph will be evaporated in order to avoid the system to converge too rapidly to a solution. Then the solutions produced by all the ants during this iteration are used to update the pheromone deposit on the arcs of the problem graph. The pheromone deposit on arc (i,j) is updated according the following formula:

$$\tau_{ij} = \rho \cdot \tau_{ij} + \sum_{k=1}^{\ell} \sigma_{ij}^{k}$$

where ρ $(0 \leq \rho \leq 1)$ is called the trail persistence and corresponds to the fraction of the current pheromone quantity τ_{ij} that remains on (i, j); where σ_{ij}^{k} corresponds to the quantity of pheromones deposed by ant k. This latter quantity is 0 if arc (i, j) doesn't appear in the solution constructed by ant k and else depends on to the total cost of the solution.

2.7 BRANCHING SEARCH

Methods based on Branching Search [Hoo11] are *constructive* and can be *complete* or *incomplete*. In the following, Branching Search is introduced as an abstract method. Concrete examples of optimization techniques employing Branching Search are given at the end of this section. Two of these methods are presented in a more detailed fashion in the following sections 2.8 to 2.10.
Branching search is used to solve problems of the form $P(X, C, \dots)$ (denoted P in short-hand notation), where $X = \{x_1, \dots, x_n\}$ is a set of variables and $C = \{c_1, \dots, c_m\}$ is a set of constraints on these variables. The original problem $P(X, C, \dots)$ is divided into a set of smaller subproblems $\{P_1(X, C_1, \dots), \dots, P_s(X, C_s, \dots)\}$ which are then solved

individually. A subproblem $P_i(X, C_i, \dots)$ is created from problem P by adding μ_i new constraints to problem $P(X, C, \dots)$. Thus a subproblem P_i is created from the original problem P by imposing further constraints. Note that this means that any feasible solution for the subproblem P_i is feasible for the original problem P as well. The idea is to choose the new constraints in such a way that the set of feasible solutions to P_i, $S(P_i)$, is smaller than the set of feasible solutions to P, $S(P)$, i.e. $|S(P_i)| < |S(P)|$. At the same time the constraints must ensure that the set of feasible solutions to P_i, denoted by $S(P_i)$ is completely included in the set of feasible solutions of the original problem $S(P)$, i.e. $S(P_i) \subset S(P)$. Finally the different subproblems $P_1, \dots, P_s$ partition $S(P)$ in such a way that $\bigcap_{i=1}^{s} S(P_i) = \emptyset$ and that $\bigcup_{i=1}^{s} S(P_i) = S(P)$.

A search tree (*Branching search tree*) is constructed by applying the division into subproblems recursively to each new subproblem P_i derived from some problem P, see Fig. 2.6 for an example of such a tree. At the root node (level 0) of the Branching search tree the original problem P can be found. P is then divided into s_1 subproblems, therefore creating s_1 child nodes at level 1. This action is called *Branching*. The branching action is repeated at every level and for each of the newly created subproblems until resulting in subproblems of the form $P_u(X, C_u, \dots)$ where constraints C_u allow only one feasible value for each $x \in X$. Such subproblems correspond to leaf nodes in the Branching search tree. They furthermore correspond to a feasible solution to the original problem P. In the following P_i will be used to designate problem $P_i(X, C_i, \dots)$ as well as the node in the Branching search tree associated with this problem.

Methods that build and explore the entire Branching Search tree actually enumerate all assignments of values to the variables in X that are feasible for P. Even though the set of possible values for the variables may be limited, such an enumeration is not tractable for most problems. Therefore Branching Search must be sped up by *pruning*, that is cutting off, parts of the Branching search tree without exploring them.

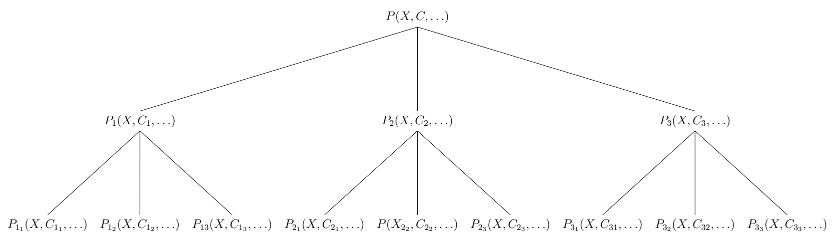

$P(X, C, \ldots)$
$P_1(X, C_1, \ldots)$
$P_2(X, C_2, \ldots)$
$P_3(X, C_3, \ldots)$
$P_{1_1}(X, C_{1_1}, \ldots)$
$P_{1_2}(X, C_{1_2}, \ldots)$
$P_{13}(X, C_{1_3}, \ldots)$
$P_{2_1}(X, C_{2_1}, \ldots)$
$P(X_{2_2}, C_{2_2}, \ldots)$
$P_{2_3}(X, C_{2_3}, \ldots)$
$P_{3_1}(X, C_{31}, \ldots)$
$P_{3_2}(X, C_{32}, \ldots)$
$P_{3_3}(X, C_{3_3}, \ldots)$

Figure 2.6: Branching search tree with associated Problem indicated at each node. In this tree each of the considered "non-leaf" problems is divided into 3 subproblems.

2.7.1 *Search strategies*

Complete methods, while pruning some parts, explore the remainder of the Branching search tree in its entirety. Different strategies for the tree traversal exist. At each point in the tree exploration there is a (possibly empty) set of open nodes (or subproblems) called the *frontier*. Nodes in the frontier are scheduled for exploration, that is, a node P_i is in the frontier if no attempts to derive further subproblems from problem P_i have been made so far.
The most basic search strategy is *Depth First Search* (DFS). In DFS the next node to be explored is the leftmost unexplored node (note that the decision as to which node is the leftmost one depends upon the branching heuristic, see below). Contrary to DFS, *Best First Search* associates a value $h(P_i)$ with each node P_i in the frontier. The node minimizing this value is greedily selected as next node to explore. Finally *Discrepancy Search* corresponds to a DFS with "wrong turns". With each leaf node is associated a value called *discrepancy*. This value corresponds to the number of times the DFS strategy must be "disobeyed" in order to travel from the root node to the considered leaf node. The tree is then explored in such a way that the leaf nodes are visited by non-decreasing discrepancy value.

2.7.2 *Branching Search in Optimization*

Integer Programming and Constraint Programming adapt Branching Search to solve optimization problems resulting in *Branch & Bound* and *Branch & Propagate*. The methods differ in several points:

- How is a problem represented? (Problem representation)

- Which constraints are added to create subproblems? (Subproblem creation)
- How is the Branching search tree pruned? (Pruning)
- How are nodes in the Branching search tree ordered? (Branching heuristic)
- How is the Branching search tree explored? (Search strategy)

Both *Branch & Bound* and *Branch & Propagate* are explained in further detail in the following sections. Finally note that Construction Heuristics can be seen as special cases of Branching Search where only one branch of the Branching search tree is explored. In Ant Colony Optimization each ant explores a branch of a Branching search tree, the tree varies however from one ant to the other.

2.8 BRANCH & BOUND

Branch & Bound is a *complete* and *constructive* method for solving, among others, integer linear programming problems based on Branching Search. Integer linear programming problems are modeled as follows:

$$\begin{array}{lr} \text{Minimize} & o^T x \\ \text{subject to} & Ax = b \\ & x \in \mathbb{Z}^n \end{array}$$

where x is a vector of variables of size n, c is a vector of costs associated with these variables and A and b are matrices of sizes $m \times n$ and $m \times 1$. The problem is thus to find a value for each variable in x (a solution) s.t. $o^T x$ is minimized and the constraints given by $Ax = b$ are respected (feasible solution). For simplicity only binary integer linear programming problems where $x \in \{0, 1\}$ are considered here.
The underlying idea of Branch & Bound is to enumerate all feasible solutions to the problem, while keeping track of the best feasible solution (the incumbent). Once all feasible solutions have been enumerated the incumbent corresponds to an optimal solution for the problem.

The following notations are used in this section:

- a solution corresponds to an assignment of values to each variable in x
- the set of feasible solutions to a problem P is denoted $Sol(P)$
- the set of optimal feasible solutions to problem P is denoted $Sol^*(P)$
- the *cost* or *value* of $sol \in Sol(P)$ corresponds to the evaluation of $o^T x$ and is denoted by $Obj(sol)$
- $d \in D$, where D is a $m \times n$ matrix means $d \in \{D_{11}, \ldots, D_{1n}, D_{21} \ldots, D_{2n}, \ldots, D_{m1}, \ldots, D_{mn}\}$

While different implementations and uses of Branch & Bound may exist, the given descriptions focus on *Integer Linear Programming*. The following explains how Branching Search can be adapted to result in Branch & Bound.

Problem Representation An integer linear programming problem is defined by its variables, the vector x, the vector of cost coefficients associated with these variables o and the constraints of the problem given by the equalities $Ax = b$. Note that the constraints on the bounds of the x variables are needed as well to define a problem. Since only binary problems are considered here, these bounds are not included expressively in the problem notation for readability reasons. The original problem is thus $P(x, Ax = b, o)$ and its decomposition into s subproblems results in $P_1(x, A^1x = b^1, o), \ldots, P_s(x, A^sx = b^s, o)$. As previously seen subproblems in Branching search are created by adding new constraints to the original problem. In the case of a linear program adding new constraints corresponds to adding new rows to the A and b matrices. Matrix A^i is thus of size $(m + \mu_i) \times n$ and $A^i_{u,v} = A_{u,v} \quad u \in 1, \ldots, m$ and $v \in 1, \ldots, n$ and $A^i_{u,v}$ with $u \in m+1, \ldots, m + \mu_i$ and $v \in 1, \ldots, n$ correspond to the additional constraints added. The same holds analogously for b^i.

Subproblem Creation Branching search can be easily applied to binary integer programming problems when creating subproblems by fixing a variable to a value. That is, given the original problem
$P(x, Ax = b, o)$ some variable $x_k \in x$ is selected. Then two subproblems $P_{k0}(x, A^{k0}x = b^{k0}, o)$ and $P_{k1}(x, A^{k1}x = b^{k1}, o)$ are created. In these subproblems $A^{k0}x = b^{k0}$ ($A^{k1}x = b^{k1}$) corresponds to the original constraints $Ax = b$ with the additional constraint $x_k = 0$ ($x_k = 1$). For each

new branching decision at problem P_i only variables that are not yet fixed to a given value by a constraint are considered for the subproblem creation.

Pruning Obviously, in order to maintain completeness, only subtrees not containing the optimal solution may be cut off from the tree, and therefore a method allowing to predict with certainty that a certain part of the tree does not contain an optimal solution to the original problem P is needed. In *Branch & Bound* this is done using upper and lower bounds. At each node (i.e. subproblem) P_i in the branching tree the global upper bound (UB) gives a cost that an optimal solution sol^* to P ($sol^* \in Sol^*(P)$) cannot exceed ($Obj(sol^*) \leq UB$). The lower bound (LB_i) gives at each node P_i a minimal cost that a globally optimal solution ($sol_i^* \in Sol^*(P_i)$) to the node's associated subproblem P_i cannot undermatch ($Obj(sol_i^*) \geq LB_i$). If problem P_i has no feasible solution, then $LB_i = \infty$. If at a certain node P_i, $LB_i \geq UB$ or $LB_i = \infty$, then it can be deduced that no solution improving the incumbent (and therefore no new optimum) can be found in the subtree of the current node P_i. The subtree can therefore be pruned without losing the completeness of the approach.
Thus when designing the pruning mechanisms for a Branch & Bound method for solving integer linear programming problems the following decisions need to be taken:

A) How is the upper bound computed?

B) How is the lower bound computed?

(A) Upper bound The upper bound corresponds to the cost of the incumbent solution. The incumbent solution can be initialized using a feasible solution obtained using appropriate construction heuristics. Alternatively, UB is initialized to ∞ and the first feasible solution encountered in the Branch & Bound tree can be used to initialize the incumbent and update UB. Each time the incumbent solution is updated, so is UB.

(B) Lower bound The lower bound of a problem is obtained by relaxing some constraints in this problem. Often the linear programming relaxation is used, where the constraints stating that the x variables must take integer values (integrality constraints) are relaxed. That is in a solution to this relaxation the x variables may take any value in $\mathbb{R}_+^n$.

The linear programming relaxation of the integer linear programming problem P is denoted $Relax(P)$. On the other hand a relaxed problem P' on which integrality constraints are imposed is denoted $Int(P')$. Thus $Int(Relax(P)) = P$. The cost $Obj(sol^*_{rel})$ of an optimal feasible solution sol^*_{rel} of $Relax(P)$ ($sol^*_{rel} \in Sol^*(Relax(P))$), is a lower bound on the cost $Obj(sol^*)$ of the optimal feasible solutions $sol^* \in Sol^*(P)$. Therefore for each node, the linear programming relaxation of the corresponding subproblem is optimally solved in order to provide a lower bound on the optimal solution value of the subproblem. If $Relax(P_i)$ is infeasible, then so is P_i and $Sol^*(Relax(P_i)) = Sol^*(P_i) = \{\perp\}$ and $Obj(\perp) = \infty$. For more information on how to solve $Relax(P_i)$ please refer to [Dan98].

Branching heuristic Assume that a linear programming relaxation is used to compute LB_i at each node P_i. The branching heuristic decides at each node P_i on which variable $x_k \in x$ to branch next. A common heuristic is to consider the set of variables Y $(\subseteq x)$ that take a fractional value in $Sol^*(Relax(P_i))$ and to select the variable $x_k \in Y$ that takes the value closest to 0.5 [Ach05].

Search strategy Two common heuristics are Depth-First Search (DFS) and Best First Search (BFS) ([Wol98]). The advantage of DFS is that is plunges down a branch in the tree allowing it to quickly find potentially feasible solutions. Since UB is used to prune parts of the search tree, it is important to initialize it with a feasible solution value as early as possible in the search. On the other hand, DFS has the tendency to get stuck in parts of the search tree and possibly lose time and effort exploring a subtree containing no improving or even no feasible solutions.
Best Bound Search, an implementation of BFS, uses the lower bound LB_i associated with each node P_i in the frontier as a heuristic means to select the next node to be explored. The node P_i minimizing LB_i is greedily selected as next node to explore. The reasoning is that this branch may contain feasible solutions of cost lower than any of the other nodes in the frontier. By discovering such solutions as early as possible UB can be tightened as early as possible and allow to prune a high number of subtrees later in the search.

2.9 COLUMN GENERATION FOR SET PARTITIONING PROBLEMS

Given a set of elements $\mathcal{U}$ called the universe and a set of sets called $\mathcal{R}$ such that $\bigcup_{\mathcal{I}\in\mathcal{R}} \mathcal{I} = \mathcal{U}$ and $o_{\mathcal{I}}$ the cost associated with set $\mathcal{I} \in \mathcal{R}$. The Set Partitioning Problem asks to identify the least cost combination of sets from $\mathcal{R}$ that contain all elements of $\mathcal{U}$ exactly once. The integer linear programming problem formulation is as follows:

$$\text{Minimize} \sum_{\mathcal{I}\in\mathcal{R}} o_{\mathcal{I}} x_{\mathcal{I}} \tag{2.1}$$

$$\text{subject to} \sum_{\mathcal{I}\in\mathcal{R}} a_{u\mathcal{I}} x_{\mathcal{I}} = 1 \quad \forall u \in \mathcal{U} \tag{2.2}$$

$$x_{\mathcal{I}} \in \{0,1\} \qquad \forall \mathcal{I} \in \mathcal{R} \tag{2.3}$$

$$a_{u\mathcal{I}} = \begin{cases} 1 & \text{if } u \in \mathcal{I} \\ 0 & \text{otherwise} \end{cases} \quad \forall u \in \mathcal{U}, \quad \forall \mathcal{I} \in \mathcal{R} \tag{2.4}$$

With each set $\mathcal{I} \in \mathcal{R}$ is associated a binary variable $x_{\mathcal{I}}$ indicating whether the set $\mathcal{I} \in \mathcal{R}$ is selected in the solution. To each set $\mathcal{I} \in \mathcal{R}$ is furthermore linked a column in matrix a. This problem formulation is referred to as $SPP(\mathcal{R})$ in the remainder of this section.

It often happens that in such set partitioning problems the set $\mathcal{R}$ is extensive, and adding one variable $x_{\mathcal{I}}$ per set $\mathcal{I} \in \mathcal{R}$ would result in a problem formulation containing too many variables to be efficiently solved using standard methods such as the Branch & Bound method using Linear Programming Relaxations described earlier. Therefore the problem is initially only formulated on a restricted set $\mathcal{R}_0^* \subset \mathcal{R}$, resulting in $SPP(\mathcal{R}_0^*)$. New sets, considered meaningful in the process of finding an optimal solution to the original problem, are iteratively added to the restricted set $\mathcal{R}_i^*$, resulting in the extended set $\mathcal{R}_{i+1}^*$. Adding a new set $\mathcal{I}$ to $\mathcal{R}_i^*$ corresponds to adding a new variable $x_{\mathcal{I}}$, its associated cost $o_{\mathcal{I}}$ as well as a new column to matrix a in the problem formulation $SPP(\mathcal{R}_i^*)$, resulting in the extended problem $SPP(R_{i+1}^*)$. This procedure in the large sense is called *Column Generation* and adding a new set $\mathcal{I}$ to $\mathcal{R}_i^*$ (i.e. new variable, associated cost and new column) is called *adding a column* to the problem.

The difficulty in Column Generation is to identify new sets $\mathcal{I}$ to add to $SPP(\mathcal{R}_i^*)$ that can help to find an optimal solution to the original problem $SPP(\mathcal{R})$. Only such sets should be added. The identification of such sets is not directly possible on an integer problem such as $SPP(\mathcal{R}_i^*)$. It is however possible to identify sets for $Relax(SPP(\mathcal{R}_i^*))$ that can help to find an optimal solution to
problem $Relax(SPP(\mathcal{R}))$. $Relax(SPP(\mathcal{R}))$ is also called the *Master Problem* (MP). The restriction to a strict subset $\mathcal{R}_i^*$ of $\mathcal{R}$ ($Relax(SPP(\mathcal{R}_i^*))$) is called the *Restricted Master Problem* (RMP).

When an optimal feasible solution $sol_{rel} \in Sol^*(Relax(SPP(\mathcal{R}_i^*))$ is available, the dual of this solution and the associated dual costs π^i are available as well. The dual costs associate via the constraints 2.2 a dual cost π_u^i with each element $u \in \mathcal{U}$. Using these dual costs the *reduced cost* $rc_\mathcal{I}$ of a set $\mathcal{I} \in \mathcal{R}$ can be computed as follows:

$$rc_\mathcal{I}(\pi^i) = o_\mathcal{I} - \sum_{u \in U} a_{u\mathcal{I}} \pi_u^i$$

For sets $\mathcal{I}$ already included in $\mathcal{R}_i^*$ ($\mathcal{I} \in \mathcal{R}_i^*$), the reduced cost will be greater or equal to 0 ($rc_\mathcal{I}(\pi^i) \geq 0$). For sets $\mathcal{I} \notin \mathcal{R}_i^*$ the reduced cost can take any value. If the reduced cost of a set $\mathcal{I}$ is negative ($rc_\mathcal{I}(\pi^i) < 0$) this indicates that adding $\mathcal{I}$ to $\mathcal{R}_i^*$ (resulting in $\mathcal{R}_{i+1}^*$) could lead to finding an optimal solution $sol'_{rel} \in Sol^*(Relax(SPP(\mathcal{R}_{i+1}^*))$ such that $Obj(sol'_{rel}) < Obj(sol_{rel})$. When no set $\mathcal{I} \in \mathcal{R}$ s.t. $rc_\mathcal{I}(\pi^i) < 0$ exists, then the current optimal solution $sol \in Sol^*(Relax(SPP(\mathcal{R}_i^*))$ is optimal for $Relax(SPP(\mathcal{R}))$ (i.e. $sol \in Sol^*(Relax(SPP(\mathcal{R}_i^*)))$ and $sol \in Sol^*(Relax(SPP(\mathcal{R})))$. Searching for new sets of negative reduced cost is called *pricing*.

2.9.1 *Branch & Price for Set Partitioning Problems*

Column Generation can be included in the Branch & Bound framework resulting in Branch & Price. In Branch & Price the original integer problem corresponds to $SPP(\mathcal{R})$. As before the problem is however formulated on a restricted set $\mathcal{R}_0^*$ resulting in $SPP(\mathcal{R}_0^*)$. This problem is then attacked using a Branch & Bound method combined with Column Generation. Each node SPP_j in the Branch & Price tree is associated with a current restricted set $\mathcal{R}_j^*$ (and with constraints introduced through branching). The initial problem at node SPP_j is thus problem $SPP_j(\mathcal{R}_j^*, \dots)$ (where $\dots$ designs additional branching constraints). The problem $Relax(SPP_j(\mathcal{R}_j^*, \dots))$ is solved and the resulting dual costs π^j are used to generate new sets $\mathcal{I}$ s.t. $rc_{\mathcal{I}}(\pi^j) < 0$. Note that care has to be taken that those new sets respect the constraints of problem $SPP_j(R_j^*, \dots)$. The new sets are then added to $\mathcal{R}_j^*$, resulting in a new problem. These steps of solving and pricing are repeated iteratively until at some point set $\mathcal{R}_j^{*\prime}$ ($\mathcal{R}_j^* \subseteq \mathcal{R}_j^{*\prime}$) is obtained and no further sets $\mathcal{I}$ s.t. $rc_{\mathcal{I}}(\pi'^j) < 0$ are proven to exist. Thus at the end of the procedure the problem associated with node SPP_j is problem $SPP_j(\mathcal{R}_j^{*\prime}, \dots)$ and a solution $sol \in Sol^*(Relax(SPP_j(\mathcal{R}_j^{*\prime}, \dots)))$ is optimal for the non-restricted problem ($sol \in Sol^*(Relax(SPP_j(\mathcal{R}, \dots)))$). Furthermore $Obj(Sol^*(Relax(SPP_j(\mathcal{R}_j^{*\prime}, \dots)))) \leq Obj(SPP_j(\mathcal{R}, \dots))$ (the optimal solution gives a lower bound on the optimal solution of $SPP_j(\mathcal{R}, \dots)$).

In the remainder of this thesis $\mathcal{R}_i^*$ will be used to denote the initial set of elements associated with node SPP_i and $\mathcal{R}_i^{*\prime}$ is used to denote the final set of elements associated with node SPP_i.

2.9.2 *Application to the CVRP*

The Capacitated Vehicle Routing Problem can be formulated as a Set Partitioning Problem (called SPP_{CVRP}) as follows:

$$\begin{aligned}
&\text{Minimize} \sum_{r \in \mathcal{R}} o_r x_r \\
&\text{subject to} \sum_{r \in \mathcal{R}} a_{ir} x_r = 1 \quad \forall i \in V \backslash \{0\} && (2.5) \\
&\sum_{r \in \mathcal{R}} x_r = K && (2.6) \\
&x_r \in \{0, 1\} \qquad \forall r \in \mathcal{R} && (2.7)
\end{aligned}$$

$$a_{ir} = \begin{cases} 1 & \text{if customer } i \text{ is visited in route } r \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in V \backslash 0, \quad \forall r \in \mathcal{R}$$

In this formulation the universe corresponds to the set of customers $\mathcal{U} = V \backslash \{0\}$ and set $\mathcal{R}$ is the set of all feasible routes, along with $K - 1$ empty routes to allow a solution using less routes than vehicles available. A binary variable x_r is thus associated with every route $r \in \mathcal{R}$. Furthermore a constraint 2.6 is added to restrict the number of routes used in the solution. The set $\mathcal{R}$ contains all possible routes such that for each route:

- no customer is visited more than once in the route
- the sum of the customers visited on the route does not exceed the vehicle capacity
- the route starts and ends at the depot

Since it is intractable to generate all feasible routes for most problem instances, column generation procedures are used to solve this problem. Thus only a restricted set of feasible routes $\mathcal{R}^*$ is used. Adding a new column to this problem means adding a new route r into $\mathcal{R}^*$. This of course also corresponds to adding to the problem formulation a new variable x_r, its associated cost o_r which is the total distance of route r and finally also a column in matrix a indicating which customers are visited in r.

The only point in the previously described Column Generation and Branch & Price methods that needs to be adapted to the problem at hand is how the Pricing subproblem is solved. Note that in practice the way subproblems are created in the Branch & Price tree is adapted as well, see e.g. [Fei10]. There are numerous examples of Column Generation or Branch & Price applied to Vehicle Routing see e.g. [Sal05, CRS09, BBMR10].

2.9.2.1 *Pricing for the CVRP*

Here, a brief overview of pricing for the CVRP is given. For simplicity, in the following it is assumed that no further constraints, introduced by branching, need to be respected by the generated routes.
Pricing for the CVRP corresponds to finding new feasible routes of negative reduced cost. With each customer $i \in V \backslash \{0\}$ will be associated a dual cost π_i and furthermore a dual cost π_0 is associated with constraint 2.6. The reduced cost of a route r is thus computed as:

$$rc_r(\pi) = o_r - \sum_{i \in V \backslash \{0\}} a_{ir} \pi_i - \pi_0$$

With $o_r = distance(r)$. This formula can be reformulated as:

$$rc_r(\pi) = \sum_{(i,j)\in A} b_{ijr}(c_{ij} - \pi_i)$$

where

$$b_{ijr} = \begin{cases} 1 & \text{if route } r \text{ uses arc } (i,j) \\ 0 & \text{otherwise} \end{cases} \quad \forall i,j \in V, \quad \forall r \in \mathcal{R}$$

The problem of finding feasible routes of negative reduced cost is the problem of finding elementary paths of negative cost respecting the capacity constraints from source node s to sink node t in an adapted graph $G'' = (V'', A'')$. Source and sink nodes correspond to replications of the depot, thus V'' corresponds to V where the original depot node has been replaced by two of its copies, $V'' = V \backslash (\{0\}) \cup \{s,t\}$. The set of arcs is adapted as $A'' = A \backslash (\delta_0^- \cup \delta_0^+) \cup (\delta_s^- \cup \delta_s^+) \cup (\delta_t^- \cup \delta_t^+)$ where δ_s^- is the set of arcs (i,j) s.t. $i = s \wedge j \in V\backslash\{0\}$ and δ_s^+ is the set of arcs (i,j) s.t. $j = s \wedge i \in V\backslash\{0\}$, and analogously for δ_t^- and δ_t^+. The weight of the arcs in A'' is adapted as $c''_{ij} = c_{ij} - \pi_i \quad \forall (i,j) \in A''$ where $\pi_s = \pi_t = \pi_0$.
The problem of finding the elementary path of lowest cost and respecting the capacity constraints in G'' corresponds to an Elementary Shortest Path Problem with Resource Constraints (ESPPRC). Note that as arc weights in G'' may be negative, the elementary constraint is necessary to avoid cycles. The problem is NP-hard, but when dropping the elementary constraint, pseudo-polynomial algorithms exist. [ID05].
These problems can be solved using labeling algorithms. The basic idea is to explore all feasible paths from the source s to the sink t by progressively extending partial paths from s to nodes i ($i \in V''$) until reaching t. However different paths from s to the same node i are constantly compared to each other and dominance between these paths is established. Only non-dominated partial paths will be further extended since dominated partial paths can not be part of the shortest path from s to t. With each path from node s to node i is associated a label indicating for each considered resource the amount consumed on the partial path from s to i. In the CVRP the resources considered correspond to the accumulated arc costs and demands. Thus with a given path p from s to i is associated a resource variable R_{pi}^{cost} and a resource variable R_{pi}^{demand}. Both resource variables are initialized to 0 at node s,

i.e. $R^{cost}_{p_{0s}} = 0$ and $R^{demand}_{p_{0s}} = 0$ where p_0 is considered the unique possible empty path from s to s. Each time a path p from s to i is extended to j, the values of the corresponding resource variables R^{cost}_{pj} and R^{demand}_{pj} are computed as $R^{cost}_{pj} = R^{cost}_{pi} + c''_{ij}$ and $R^{demand}_{pj} = R^{demand}_{pi} + q_j$. Of course paths p from s to some node j such that $R^{demand}_{pj} > Q$ are discarded.
The resources are used to establish dominance between two paths. A path p' from s to i dominates a path p'' from s to i if and only if $R^{cost}_{p'i} \leq R^{cost}_{p''i}$ and $R^{capacity}_{p'i} \leq R^{capacity}_{p''i}$ and one of these inequalities is strict.
For information on how this approach can be extended to guarantee elementary paths and further information on Resource-constrained Shortest Path Problems see [ID05].

It is sufficient to solve the ESPPRC heuristically to determine new feasible routes of negative reduced costs. It needs to be solved exactly only once per Column Generation to ensure no further feasible routes of negative reduced cost exist.

In heuristic Column Generation schemes, as the resulting method does not need to be complete, heuristics and metaheuristics such as those seen in previous sections can be used to generate feasible routes of negative reduced cost. This is done by either heuristically modifying routes currently in the optimal solution of the RMP [XCRA03], or (meta-) heuristically solving a Shortest Path Problem [PR09].

2.9.2.2 *A note on the Consecutive-Ones Property*

It can be shown that an optimal feasible solution to a bounded linear programming problem with integer right-hand side coefficients b, such as the Linear Programming relaxation of problem 2.1, is integral if the constraint matrix A is totally unimodular [GP10]. Note that the integrality or non-integrality of the optimal feasible solution is thus independent from the cost coefficients associated with the problem variables. If it is known that the matrix A for some integer problem is totally unimodular, then this means that the problem can be solved more efficiently. It is sufficient to solve the problem's Linear Programming relaxation, no Branch & Bound is necessary to obtain the optimal integral solution from the optimal fractional solution.
We call AP_{CVRP} the constraint matrix associated with the Set Parti-

tioning reformulation of the CVRP. It corresponds to the left-hand side coefficients of constraints Eqs. 2.5 and 2.6. Note also that for the CVRP the right-hand side coefficients are always integer. Thus if for some set of routes $\mathcal{R}^+$ the corresponding matrix $AP_{CVRP}(\mathcal{R}^+)$ is totally unimodular then an optimal feasible solution sol is integral ($sol \in Sol^*(Relax(SPP_{CVRP}(\mathcal{R}^+)))$) and thus $sol \in Sol^*(SPP_{CVRP}(\mathcal{R}^+))$.
In the case of a Set Covering Problem (SCP_{CVRP}, the equalities in constraints 2.5 and 2.6 are replaced by inequalities ($\geq$)), an optimal feasible solution to the LP relaxation of that problem is integral *if and only if* the corresponding constraint matrix AC_{CVRP} is totally unimodular [GP10].

One way to detect total unimodularity in binary matrices is via the *Consecutive-Ones Property* (C1P), as it is known that a matrix that has this property is totally unimodular (the inverse is not true). A matrix containing only entries in {0,1} (called $0/1$-matrix) is said to have C1P if its columns can be permuted such that in every row the $1s$ appear consecutively. Furthermore a $0/\pm1$ matrix is said to have C1P if (i) the entries in every row belong to either $\{0,1\}$ or $\{0,-1\}$ and (ii) the matrix columns can be permuted in such a way that in every row the $1s$ (or the $-1s$ respectively) appear consecutively [Dom09].
Let's first consider the case of problem $SPP_{CVRP}(\mathcal{R}^+)$. As each route in $\mathcal{R}^+$ needs exactly one vehicle, all coefficients in matrix $AP_{CVRP}(\mathcal{R}^+)$ in the row corresponding to constraint 2.6 are 1, and thus C1P is automatically respected for this row. Condition (i) is thus always respected. It depends on the nature of the routes (i.e. the combination of customers visited in these routes) in $\mathcal{R}^+$ whether matrix $AP_{CVRP}(\mathcal{R}^+)$ has C1P. With problem SCP_{CVRP} the row in matrix $AC_{CVRP}(\mathcal{R}^+)$ corresponding to constraint 2.6 (with inequality sign) will contain only -1 entries, thus again it will depend on the set of routes $\mathcal{R}^+$ whether matrix AC_{CVRP} has C1P.
Finally note that Set Covering Problems with C1P can be solved faster by using matrix A to represent a weighted graph and solving a minimum cost flow problem on this graph (see [Dom09]).

Example

Assume a CVRP with a central depot and four customers called c_1, c_2, c_3 and c_4. Assume furthermore 4 routes $r_a = \langle 1, 4, 2 \rangle$, $r_b = \langle 2, 3, 1 \rangle, r_c = \langle 3, 4 \rangle$ and $r_d = \langle 3 \rangle$. With each of the routes is associated an x variable. Matrix M depicts the constraint matrix corresponding to constraints 2.5 for this problem.

$$\mathrm{M} = \begin{array}{c|cccc} & x_a & x_b & x_c & x_d \\ c_1 & 1 & 1 & 0 & 0 \\ c_2 & 1 & 1 & 0 & 0 \\ c_3 & 0 & 1 & 1 & 0 \\ c_4 & 1 & 0 & 1 & 1 \end{array}$$

Matrix M does not have C1P as there is no way to arrange its columns in such a way that the 1s appear in blocks in each of its rows.
If we assume all distances equal to 1, the optimal solution to the LP relaxation of problem SCP_{CVRP} formulated over these routes will select all routes with $x_a = 0.5, x_b = 0.5, x_c = 0.5$ and have a cost 5.5. The optimal integral solution selects $x_a = 1$ and $x_d = 1$ with an optimal solution cost of 6.

If we now assume routes $r_a = \langle 1, 2 \rangle, r_b = \langle 2, 3, 1 \rangle, r_c = \langle 3, 4 \rangle$ and $r_d = \langle 3 \rangle$ the corresponding matrix M' corresponds to:

$$\mathrm{M}' = \begin{array}{c|cccc} & x_a & x_b & x_c & x_d \\ c_1 & 1 & 1 & 0 & 0 \\ c_2 & 1 & 1 & 0 & 0 \\ c_3 & 0 & 1 & 1 & 0 \\ c_4 & 0 & 0 & 1 & 1 \end{array}$$

Clearly this matrix has C1P and the optimal solution to the LP relaxation of problem SCP_{CVRP} formulated over these routes selects $x_a = 1, x_c = 1$. It is integral and has cost 6.

2.10 BRANCH & PROPAGATE

Branch & Propagate is a *complete* and *constructive* method to solve Constraint Satisfaction Problems of the form CSP(X,C,D) based on Branching Search. Such a problem is defined over a set of variables $X = \{x_1, \ldots, x_n\}$, the domains of these variables $D = \{D(x_1), \ldots, D(x_n)\}$ and a set of constraints $C = \{c_1, \ldots, c_m\}$ over the variables in X. The domain $D(x_i)$ of a variable x_i corresponds to the set of values $\{v_1, \ldots, v_d\}$ the variable can take. Each constraint $c \in C$ involves a set of variables $V(c)$ $(V(c) \subseteq X)$ and limits the allowed combinations of values that can be assigned to these variables. A complete assignment $\mathcal{A}$ maps to each variable x a value $\mathcal{A}(x_i) = v$ s.t. $v \in D(x_i)$. A feasible solution to a CSP is a complete assignment $\mathcal{A}$ such that every constraint $c \in C$ is respected in this assignment. A CSP is said to be unsatisfiable or infeasible if no feasible solution exists.
Note that CSPs are not optimization but satisfaction problems. Thus the exploration of the search tree stops once a feasible solution has been found, or the entire search tree has been explored. See section 2.10.2 for explanations of how optimization problems can be solved using CSPs and Branch & Propagate. While different implementations and uses of Branch & Propagate may exist the descriptions given here focus on *Constraint Programming*. The following explains how Branching Search can be adapted to result in Branch & Propagate.

Problem representation A Constraint Satisfaction Problem is defined by its variables, the set $X = \{x_1, \ldots, x_n\}$, the domains of these variables $D = \{D(x_1), \ldots, D(x_n)\}$ and the constraints on these variables $C = \{c_1, \ldots . c_m\}$. The original problem is thus $P(X, D, C)$ and its decomposition into s subproblems results in $P_1(X, C_1, D_1), \ldots, P_s(X, C_s, D_s)$. Subproblems are created by adding new constraints to the original problem. In the case of a CSP $P(X, C, D)$ a subproblem $P_i(X, C_i, D_i)$ is created by adding μ_i new constraints to problem $P(X, C, D)$ s.t. $C_i = \{c_1, \ldots, c_m, c_{m+1}, \ldots, c_{m+\mu_i}\}$ and by reflecting the impact of these new constraints on the domains, i.e. $D_i(x_j) \subseteq D(x_j) \quad \forall x_j \in X$. For further information on how the domains are adapted see *Pruning*.

Subproblem creation Different types of constraints can be added to P in order to create subproblems. Usually the same types of constraints are used throughout the construction of the Branching Search Tree. Three common ways ([RVBW06]) to create subproblems in Branch &

Propagate are given now. A common choice for creating subproblems for $P(X, C, D)$ is to select some variable $x_k \in X$ s.t. $|D(x_k)| > 1$ and create one subproblem for each of the d values in the domain of x_k by adding the constraint $x_k = v_i$ ($i = 1, \ldots, d$, $v_i \in D(x_k)$) to the original problem. Thus d subproblems $P_{kv_1}(X, C_{kv_1}, D_{kv_1}), \ldots, P_{kv_d}(X, C_{kv_d}, D_{kv_d})$ are created and $C_{kv_i} = C \cup \{x_k = v_i\}$. This strategy is known as *Enumeration*.
Another possibility is to split the domain of some variable $x_k \in X$ around some value v_i ($v_i \in D(x_k)$), thus creating two subproblems: one with the new constraint $x_k < v_i$ and one with the new constraint $x_k \geq v_i$ (*Domain Splitting*).
Finally in the *Binary Choice points* strategy two subproblems are created for some variable $x_k \in X$ and some value $v \in D(x_k)$, one where constraint $x_k = v$ is added and one where constraint $x_k \neq v$ is used.

Pruning Even though the search stops at the first feasible solution it is possible that no such solutions exists and a complete enumeration of all possible assignments $\mathcal{A}$ is intractable in most cases. Again the idea is to speed up the search by *pruning*. Obviously only parts not containing a feasible assignment may be cut from the search tree. Therefore a method allowing to predict with certainty that a certain subtree does not contain a feasible solution is needed. This is done using *constraint propagation*. At each node P_i, constraint propagation removes values from the domains of variables if these values cannot be part of a feasible solution to subproblem $P_i(X, C_i, D)$, resulting in $P_i(X, C_i, D_i)$. If for some variable x_k we have $D_i(x_k) = \emptyset$, then the subtree rooted at P_i is proven to contain no feasible solution and can be pruned.
In practice, propagation is done for each of the constraints $c_j \in C_i$. The propagation algorithm for c_j considers each of its variables $x_k \in V(c_j)$ in turn and detects which values in $D(x_k)$ may not participate in a feasible solution given the domains $D(y)$ of the other variables ($y \in V(c), y \neq x_k$). The detected values are then removed from $D(x_k)$. Removing a value from the domain of $D(x_k)$ may cause values from the domains of other variables y s.t. $\exists c \in C_i$ s.t. $\{x_k, y\} \subseteq V(c)$ to be removed as well. Thus all constraints in C and all their variables may be treated several times, and this until each constraint $c \in C_i$ reaches some level of consistency, such as *domain consistency*.
A constraint c is considered domain consistent if for each of its variables $x \in V(c)$ and for every value in $D(x)$ a value exists in the domain $D(y)$ of every variable $y \in V(c) \wedge y \neq x$ s.t. those values are part of

an assignment $\mathcal{A}$ respecting c. Other common consistencies are *Bound consistency* and *Forward Checking consistency*.
Choosing the level of desired consistency for each of the original and also the added constraints is a trade-off between reducing the number of explored nodes in the Branch & Propagate tree and the time spent at each node. The higher the consistency level, the higher the temporal complexity of the propagation algorithm, but also the higher the number of potentially pruned nodes. This decision is typically dependent on the constraint at hand.

Branching Heuristic The branching decision consists in deciding on which variable and possibly value(s) to impose the new constraint. A common heuristic is the *dom* heuristic [RVBW06]. It chooses to first impose constraints on the variable having the least values in its domain. The intuition behind this is that this allows to detect infeasibility as soon as possible, and thus allows to prune the biggest possible parts of the Branch & Propagate tree. Other heuristics (*deg*, *dom+deg*, *dom/deg*) furthermore use the number of constraints in which each variable appears in order to select the next variable on which to branch.
As seen in *Subproblem creation* the constraints imposed on a variable to create subproblems usually involve some value. Heuristics to choose these values can be problem-dependent or use some type of approximation of the pruning performed when posting the constraint implicating the given variable and value.

Search strategy Two common strategies used in Branch & Propagate are *Depth-First Search* (DFS) and *Discrepancy Search*. Since complete assignments are located in leaf nodes and the goal is to find a feasible assignment it makes sense to quickly descend in the search tree as is done with DFS. However if a bad branching decision has been taken at a low level in the Branch & Propagate tree, time and effort will be lost exploring a subtree that may not contain a feasible solution. Discrepancy Search remedies this. As the discrepancy w.r.t. the DFS strategy may be taken at any level in the search tree different distant parts of the search tree are explored in sequence.

2.10.1 *Branch & Propagate for Optimization*

A Constraint Optimization Problem $COP(X, C, D, Obj)$ is a CSP where instead of only finding a feasible solution, the goal is to find a fea-

sible solution minimizing some function $Obj(x_1, \dots, x_n)$. An optimal solution is a complete assignment $\mathcal{A}^*$ such that every constraint $c \in C$ is respected and such that $Obj(\mathcal{A}^*(x_1), \dots, \mathcal{A}^*(x_n))$ is minimized. This means that no other complete assignment $\mathcal{A}'$ such that every constraint $c \in C$ is respected and such that
$Obj(\mathcal{A}'(x_1), \dots, \mathcal{A}'(x_n)) < Obj(\mathcal{A}^*(x_1), \dots, \mathcal{A}^*(x_n))$ exists.

COPs can be solved by repetitively solving the underlying CSP. A variable o constrained to take the value of the objective function ($o = Obj(x_1, \dots, x_n)$) is added to the underlying CSP. Furthermore a constraint of the form $o < UB$ is added and UB is initialized to ∞. The resulting CSP is then called CSP_{oUB_0} and solved using Branch & Propagate. If no feasible assignment is found the COP is infeasible.
If a feasible assignment $\mathcal{A}$ is found then UB is updated to
$Obj(\mathcal{A}(x_1), \dots, \mathcal{A}(x_n))$ resulting in CSP_{oUB_1}. These steps are repeated until no further feasible assignment can be found for some CSP_{oUB_i}. In that case the current value of UB gives the optimal solution value and the last feasible assignment $\mathcal{A}$ corresponds to the optimal solution. Thus this method is complete. This is due to two facts: Branch & Propagate is guaranteed to find a feasible to CSP_{oUB_i} if one exists; UB is initialized to ∞ but then steadily decreases as it is updated with the cost of solutions feasible in the current CSP.

2.10.2 *Application to the CVRP*

Usually Constraint Programming is not applied as is to Vehicle Routing Problems. Hybrid, often incomplete, methods combining Constraint Programming with other optimization techniques such as Local Search ([Sha98]), Ant Colony Optimization ([Sol10]), Mixed Integer Programming ([Tho01]) and Column Generation ([RGP02, Cha06]) exist.

To adapt Branch & Propagate to the CVRP it is sufficient to model it as a Constraint Optimization Problem $COP(X, D, C, Obj)$. In order to simplify the modelization, the depot vertex is duplicated in G resulting in the problem graph $G' = (V', A')$. That is for each vehicle $k \in \{1, \dots, K\}$ a new start depot vertex SD_k and a new end depot vertex ED_k are created. The sets $SD = \{SD_1, \dots, SD_K\}$ and $ED = \{ED_1, \dots, ED_K\}$ are used to refer to the entirety of start depot and end depot vertexes in G'. The set of vertexes V' is adapted from V as follows: $V' = (V\backslash\{0\}) \cup SD \cup ED$ and the set of arcs becomes $A' = (A\backslash(\delta_0^- \cup \delta_0^+) \cup SD \times V\backslash\{0\} \cup ED \times V\backslash\{0\} \cup V\backslash\{0\} \times SD \cup V\backslash\{0\} \times ED$. In the following a modelization of the CVRP as COP is given. Variables, domains, constraints and objective function are defined followed by a description of the resulting model.

Variables X

$next_i$: vertex following vertex i in the route visiting i $\quad (\forall i \in V'\backslash\{ED\})$

$prec_i$: vertex preceding vertex i in the route visiting i $\quad (\forall i \in V'\backslash\{SD\})$

$vehicle_i$: vehicle visiting i $\quad (\forall i \in V')$

N_i : set of vertexes visited after vertex i by same vehicle $\quad (\forall i \in V')$

Domains D

$$D(next_i) = V'\backslash(SD \cup \{i\}) \quad \forall i \in V'\backslash ED \tag{2.8}$$
$$D(prec_i) = V'\backslash(ED \cup \{i\}) \quad \forall i \in V'\backslash SD \tag{2.9}$$
$$D(N_i) \subseteq V'\backslash(SD \cup \{i\}) \quad \forall i \in V' \tag{2.10}$$
$$D(vehicle_i) = \{1, \ldots, K\} \quad \forall i \in V' \tag{2.11}$$

Constraints C

$$next_i \neq next_j \quad \forall i, j \in V', i \neq j \tag{2.12}$$
$$i = prec_{next_i} \quad \forall i \in V'\backslash ED \tag{2.13}$$
$$vehicle_{SD_k} = k \quad \forall k \in K \tag{2.14}$$
$$vehicle_{ED_k} = k \quad \forall k \in K \tag{2.15}$$
$$vehicle_i = vehicle_{prec_i} \quad \forall i \in V'\backslash SD \tag{2.16}$$
$$N_i = \emptyset \quad \forall i \in ED \tag{2.17}$$
$$N_i = N_{next_i} \cup \{next_i\} \quad \forall i \in V'\backslash ED \tag{2.18}$$
$$\sum_{i \in V'\backslash(SD \cup ED)} (vehicle_i = k) * q_i \leq Q \quad \forall k \in \{1, \ldots, K\} \tag{2.19}$$

Objective Obj

$$Obj(next) = \sum_{i \in V'\backslash ED} d_{inext_i} \tag{2.20}$$

In this model four different types of variables are used. Only the $next_i$ variables are actually necessary to describe a solution, these are

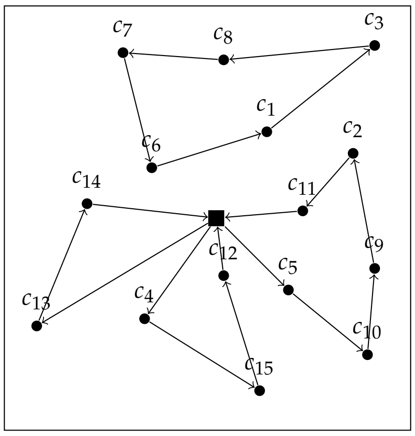


Figure 2.7: A subtour visiting customers c_1, c_3, c_8, c_7 and c_6
A subtour is a cycle in the problem graph not containing the depot vertex.

the so-called decision variables. The other variables allow to model and constrain the problem more easily. A solution to this COP does not directly represent a set of routes as seen in 2.2.1. However such a set can be easily extracted from the values of the *next* variables.

Constraints 2.12 ensure no vertex (and thus no customer) is visited more than once while constraints 2.13 are responsible for consistency between the $next_i$ and $prec_i$ variables. One vehicle is associated with each start and end depot vertex in constraints 2.14 and 2.15. With constraints 2.16 all vertexes visited on the path from start depot SD_k to end depot ED_k will be visited by the same vehicle. In combination with the N_i variables (see below) this allows to ensure that each vehicle may only be used once, and thus that the number of routes in a solution does not exceed the number of available vehicles. Constraints 2.18 link the N_i and $next_i$ variables. As, per definition of its domain the set N_i may not contain vertex i, subtours such as the one depicted in Fig. 2.7 are impossible. Thus only routes starting at a start depot and ending at an end depot are feasible in this model. Finally constraints 2.19 force the accumulated demands of customers visited by a same vehicle to respect the capacity limit of the vehicle.

2.10.2.1 *Adaptation of model using Global Constraints*

Global Constraints are constraints expressing complicated relations between a non-fixed number of variables ([RVBW06]). Such relations could also be expressed in a set of simpler constraints. However efficient propagation algorithms are associated with global constraints, which allows to achieve a better pruning of the implied variables' domains when compared to the pruning achieved with the conjunction of simpler constraints.

Pruning could be improved by using global constraints in the COP model for the CVRP. The following parts would be modified.

- Constraints 2.12 could be replaced by one *alldifferent* ([Lau78]) constraint over all the *next* variables.

  $$alldifferent(next_1, \dots, next_{n+K})$$

  where $next_1, \dots, next_n$ are the *next* variables associated with customer nodes $v_1, \dots, v_n$ and $next_{n+1}, \dots, next_{n+K}$ are the *next* variables associated with vertexes $SD_1, \dots, SD_K$.

- Constraints 2.19 could be expressed using a *bin packing* ([Sha04]) constraint.

  $$bin\_packing(q1, \dots, q_n, vehicle_1, \dots, vehicle_n, l_1, \dots, l_k)$$

  where $vehicle_i$ is the bin (vehicle) associated with customer $i \in V\backslash\{0\}$ and l_j is an additional variable representing the load of bin (vehicle) $j \in K$ with $D(l_j) = [0..Q] \quad \forall j \in K$.

- Constraints 2.17 and 2.18 could be replaced by a *directed acyclic graph* ([DDD05]) constraint.

  $$DAG(next_1, \dots, next_{n+K})$$

  where $next_1, \dots, next_n$ are the *next* variables associated with customer nodes $v_1, \dots, v_n$ and $next_{n+1}, \dots, next_{n+K}$ are the *next* variables associated with vertexes $SD_1, \dots, SD_K$. This would make the N_i variables superfluous.

3 VEHICLE ROUTING PROBLEMS WITH COMPLEX SIDE-CONSTRAINTS

In this chapter, existing high-level or generic approaches to tackle Rich Vehicle Routing Problems are reviewed. Then examples of Vehicle Routing Problems with complicated side-problems are given and the application of existing generic approaches (for Rich Vehicle Routing Problems) to these problems are considered.

3.1 RICH VEHICLE ROUTING PROBLEMS

The term rich is used in the literature to describe complicated Vehicle Routing Problems, combining number of real-world constraints into one problem. These additional constraints often impact the structure of routes and solutions (multiple depots, split deliveries, . . .) on one hand and on the other hand add additional (non-structural) constraints, such as the respect of time windows, to routes. Objective functions more complex than a basic distance minimization are common as well.

In the broad class of Rich Vehicle Routing Problem a specific subclass of problems can be identified. These are problems based on simple VRP variants where a set of additional constraints must be respected by each route. Verifying whether a route respects these constraints is a complicated problem by itself. The most notable examples of (more or less) basic VRPs with complicated side-problems in the literature are VRPs where for each route a loading problem must be solved, and VRPs with Time Windows where for each route a scheduling problem must be solved. They are briefly reviewed in the next section.

3.2 VRPS WITH COMPLICATED SIDE-PROBLEMS

There are several examples of problems combining routing and loading: in the 2L-CVRP ([ISGV07]) and the 3L-CVRP ([GILM06]) a two-, respectively three-dimensional, bin packing problem with complicating constraints must be solved for each route. In the case of the MP-VRP ([DFH+07]) a one-dimensional loading problem must be solved to verify the feasibility of a route. Recently ([ZTK12]) presented a problem where three-dimensional items must first be loaded unto pallets before loading the pallets into the truck. For each pallet a three-dimensional loading problem must be solved first. A further well-known problem combining routing and loading is the Auto carrier transportation problem [ABM98], where cars must be loaded onto specialized trucks and loading and unloading operations must be considered. For a recent overview of problems combining vehicle routing and loading the reader is referred to [IM10].

A second class of VRPs with complicated side-constraints are problems combining routing and scheduling. Here the feasibility of a route is verified by solving a scheduling problem (note that the one-dimensional loading problem of the MP-VRP can also be considered a scheduling problem). In these problems each customer must be visited in a given time window (thus we have a VRPTW) and breaks for the truck drivers must be scheduled in order to comply with a given set of rules (often corresponding to a specific legislation on working time of drivers). The exact problem depends on the rules considered. Recent work considers the European ([PGDDR10], [Goe09]) and the American legislation ([RCL13]). A comprehensive overview of work in this domain can be found in [MKKS11].

Most approaches for both types of problems integrate some kind of specific knowledge other than a basic feasibility check about the side-problem to be solved for each route (existing approaches for the

3L-CVRP and the MP-VRP will be reviewed in detail in the Applications part of this thesis). However the Large Neighborhood Search proposed in [Goe09] should be pointed out. The author proposes a Large Neighborhood Search based on the removal and reinsertion of customers. Customers may be removed from their route only if the resulting reduced route remains feasible. Reinsertion of customers is handled using an auction system, and is also only acceptable if the resulting route is feasible. In this method the only side-constraint specific information used is the feasibility check.

3.3 GENERIC/HIGH-LEVEL APPROACHES FOR RICH VEHICLE ROUTING PROBLEMS

The majority of approaches proposed in the literature for addressing Rich VRPs are metaheuristic or mathematical programming-based approaches (or combinations of both). These approaches are often very problem-specific and tailored to exactly fit the problem at hand. However efforts have been made to develop libraries and frameworks that can be used to model and solve broader classes of Rich Vehicle Routing Problems. The most recent approaches are reviewed here and considered for adaptation to VRPs with complicated side-problems.

The authors in [BBMR10] present an exact solution framework for Vehicle Routing Problems with additional constraints that can be reformulated as Set Partitioning Problems. The problem considered in this thesis falls into that category. The presented framework is based on dual ascent heuristics which are used to compute a near-optimal dual solution (to the Set Partitioning Problem) and on Column Generation. They show how their framework can be specialized to a range of problems (CVRP, VRP with Time windows (VRPTW), and Pick-up and Delivery Problem with Time Windows (VRPPDTW)) by providing problem-specific pricing algorithms. The fact that this framework is exact entails that the pricing problem needs to be solved exactly. Typically this is done by solving a Resource-constrained Shortest Path Problem (RCSPP) which then needs to be adapted to the side-problem to be solved. If the adaptation is to remain as generic as possible w.r.t. the side-problem, a feasibility check can be provided. This check must then be executed on each partial path considered in the RCSPP. Such an approach was tested in [Bon08] for the 2L-CVRP and shown to be intractable.

A unified modeling and solution framework for vehicle routing has been proposed in [Irn08]. The framework is based on the assumption that any constraints can be expressed as resources on paths or on segments. Efficient means of evaluating neighborhoods based on preprocessing are proposed. The authors apply their framework on the VRPTW, the multi-depot VRPTW and pick-up and delivery problems. They further on show how to model a set of recurring VRP variants in their framework. Expressing the constraints of the side-problem as resource is not possible without specific knowledge of the side-problem being solved per route.

In [GGW10] the authors propose a library of local search heuristics for the capacitated vehicle routing problem. Well-known construction heuristics and a large set of local search operators are implemented. A set of rules allows to easily adapt the core local search algorithm. It is also possible to interface the presented library with the SYMPHONY mixed-integer programming package. The authors show how the metaheuristics in the framework can be used to generate solutions, of which the individual routes are then added to a pool of routes on which a Set Partitioning Problem is then solved. As explained by the authors, it would be necessary to overwrite the *evaluate* method (used to evaluate a move in the neighborhood) of the local search part to adapt the libraries to a problem with side-problems. The approach being based on local search, it is not clear how it would perform on a problem where the feasible solution space is possibly not connected (and only a feasibility check for the side-problem provided).

In [DD08] it is shown how the (general-purpose) Greedy-Indirect Search Framework can be applied to VRPPDTW. In this framework the feasibility checker for a given solution is used as a black box, and can be interchanged easily with a different checker in order to adapt the framework to a different set of side-constraints. The framework is based on the concept of performing Local Search on a simplified representation (encoding) of a solution and then greedily decoding this simplified representation in order to construct a fully feasible solution for the original problem. In the case of the considered VRPPDTW an encoded solution corresponds to a sequence of vertexes. The greedy decoder executes a greedy insertion heuristic, where the vertexes are inserted in order of the given sequence to construct the corresponding solution. The black box feasibility checker is used to verify the feasibility of the partial solution resulting from each insertion. The approach being based on local search, it is not clear how it would perform on a problem where the feasible solution space is possibly not connected (and only a feasibility check for the side-problem provided).

Finally a metaheuristic framework for Rich Vehicle Routing Problems has been implemented in [Vog12]. The framework provides customizable implementations of standard construction heuristics, local search neighborhoods, and destruction and repair operators to be used in a Large Neighborhood Search context. Furthermore metaheuristic control mechanisms for Steepest Descent, Attribute-based Hill Climbing and Record-to-Record travel are provided. The framework can be adapted to specific problems by modifying the feasibility checks and different heuristic measures. The author shows for example how template classes can be customized to provide additional information on time windows (in a VRPTW context) to be used as guidance in the solution process. The framework is tested on the VRPTW, the VRP with split deliveries, the VRP with compartments, the Periodic VRP and a Truck and Trailer Problem. The author explains that the framework can be adapted to problems with complicated side-constraints (such as loading constraints) by providing external procedures to efficiently verify feasibility. The framework being heavily based on metaheuristics, the concerns w.r.t. its application to the problem considered in this thesis are the same as with local search. Does the framework allow to obtain good results on a problem where the feasible solution space is not connected?

Part II

CONTRIBUTIONS

4 THE VEHICLE ROUTING PROBLEM WITH BLACK BOX FEASIBILITY

In this chapter an abstraction for Vehicle Routing Problems with complex side-problems is presented. This abstraction is called the Vehicle Routing Problem with Black Box Feasibility (VRPBB). First, the problem formulation is presented. Different examples of how the abstraction can be instantiated to a concrete problem are given next. Then the features of a hypothetical optimization approach for the VRPBB are highlighted and the restriction of the VRPBB covered in this thesis is explained. Finally classical techniques for solving VRPs are briefly considered for adaptation to the VRPBB.

4.1 PROBLEM FORMULATION

The Vehicle Routing Problem with Black Box Feasibility is an abstraction of Vehicle Routing Problems with complex intra-route side-constraints. It is based on a basic Vehicle Routing Problem.

The VRPBB is defined on a simple complete weighted graph $G = (V, A)$. In the set of vertexes $V = \{0, \dots, n\}$ vertex 0 represents the depot, while vertexes $1 \dots, n$ represent the customers to be visited. Each arc $(i,j) \in A$, $(i,j \in V)$ has an associated weight c_{ij}. A set of homogeneous vehicles K is available to perform the visits to all customers.

In order to be feasible a route must respect two sets of non-structural intra-route constraints F_{wb} and F_{bb}. Let $feas(r,c) = true$ indicate that route r respects constraint c.

- Set F_{wb} contains so-called *white box* constraints. These constraints are known in detail and the evaluation of $feas(r,c)$ is done in $O(|r.S|)$ time complexity for every constraint $c \in F_{wb}$.
- Set F_{bb} contains *black box* constraints. These constraints are unknown and the evaluation of $\bigwedge_{c \in F_{bb}} feas(r,c)$ is done in $\Omega(|r.S|)$ time complexity. A function $feasbb(r)$, called *black box function*, to evaluate the feasibility of route r w.r.t F_{bb} is provided. However $feasbb(r)$ possibly only provides a non-exact verification of $\bigwedge_{c \in F_{bb}} feas(r,c)$, resulting in false negatives (i.e. route r is considered infeasible even though an exact verification of $\bigwedge_{c \in F_{bb}} feas(r,c)$ would prove its feasibility).

Finally the goal is to devise a solution (set of routes) $Sol = \{r_1, \dots, r_n\}$ such that:

1. $|Sol| \leq |K|$
   the solution does not use more vehicles than available
2. $\bigcap_{r_i \in Sol} r_i.S = \emptyset$ and $\bigcup_{r_i \in Sol} r_i.S = V \backslash \{0\}$
   each customer is visited exactly once
3. $\bigwedge_{c_{wb} \in F_{wb}} feas(r,c) \quad \forall r \in Sol$
   each route in the solution respects the white box constraints
4. $\bigwedge_{c_{bb} \in F_{bb}} feas(r,c) \quad \forall r \in Sol$
   each route in the solution respects the black box constraints
5. min $Obj(Sol)$
   the objective function is minimized

In the following a route is called *wb-feasible* if it respects constraints F_{wb} and *bb-feasible* if it respects constraints F_{bb}. A route is called *feasible* if it is both *wb*- and *bb*-feasible. A solution is called *VRP-feasible* if constraints 1 and 2 hold. It is called *wb*-feasible (*bb*-feasible) if all its routes are *wb*-feasible (*bb*-feasible). Finally a solution that is *VRP*-feasible and *wb*-feasible and *bb*-feasible is called *feasible*.

4.2 EXAMPLES OF VRPBB INSTANTIATIONS

The VRP with Black Box Feasibility is an abstraction, different parts (white box constraints, black box constraints, objective function) need to be instantiated in order to obtain a concrete routing problem. Examples of common instantiations of the abstract features of the VRPBB are given below:

White box constraints $\mathbf{F_{wb}}$ These constraints will typically correspond to the "classical" VRP constraints such as for example capacity constraints over the vehicles, constraints on the maximal length of a route or time window constraints for each customer.

Black box constraints $\mathbf{F_{bb}}$ These will typically correspond to some combinatorial problem. The black box function will need to solve the combinatorial problem in order to verify the feasibility of the route. Examples of such problems could be the loading of multi-dimensional items into the loading space of the vehicle or the scheduling of driver breaks.

Objective function Obj In the simplest case the objective function may correspond to the total distance of the solution or the number of vehicles. In the case of time windows it could, for example, correspond to accumulated waiting time or lateness.

4.3 FEATURES OF AN OPTIMIZATION APPROACH FOR THE VRPBB

Any optimization approach for the VRPBB will exhibit the following inter-dependent properties:

- Independence
- Flexibility
- Genericity

Independence Since only a black box function is provided to evaluate the feasibility w.r.t F_{bb} and no further insight is possible into F_{bb}, the optimization approach can not integrate or make use of any other information than the feasibility information provided by $feasbb(r)$ for each route r. This means that the optimization approach is as independent as possible from the black box constraints.

Flexibility The optimization approach is flexible in the sense that it can be applied to a fundamentally different problem by plugging a different black box function.

Genericity As the optimization approach is independent of the black box constraints it is also as generic as possible w.r.t. to the constraints in F_{bb}.

4.4 VRPBBS CONSIDERED IN THIS THESIS

In this thesis a restriction of the VRPBB is considered. This restriction is called CVRPBB henceforth.

- the objective function $Obj(Sol)$ corresponds to the minimization of the total distance
- F_{wb} correspond to capacity constraints. With each customer $i \in V \setminus \{0\}$ is associated a non-negative demand q_i. The vehicles all have a restricted capacity of Q. A route r is considered *wb*-feasible if and only if $\sum_{i \in r.S} q_i \leq Q$.

4.5 SOLVING THE CVRPBB

A method for solving CVRPBBs is needed. This method need not be *complete*, the solution should however be of high quality and of course feasible. Different optimization approaches for the CVRP could be adapted to the CVRPBB.

Construction Heuristics Construction heuristics can be used to construct solutions to Vehicle Routing Problems. Some of the heuristics presented for the CVRP can be easily extended, such that they are allowed to build only *bb*-feasible routes. In the case of Insertion heuristics, one can for example choose to accept the insertion of a customer i in route r at position p only if $feasbb(insert(i, r, p)) = true$. Since the decisions taken in construction heuristics are myopic, this could of course result in a solution infeasible because of the number of vehicles it uses.

Local Search In the same way Local Search could be adapted to accept the move to a neighboring solution only if this neighboring solution is feasible. However it is possible that for some solution Sol no bb-feasible solution Sol' exists in the neighborhood of Sol. Also it might be difficult to find an initial solution that is completely feasible. This means that it would be beneficial to allow the search to move to bb-infeasible solutions. Since however only the black box function is available to evaluate the bb-feasibility of a solution, there is no means to measure the degree of violation of F_{bb} for a given solution. Thus providing guidance to Local Search to move away from bb-infeasibility is impossible.

Ant Colony Optimization Since Ant Colony Optimization methods are built on repetitive executions of construction heuristics they can be adapted in the same way construction heuristics can. The issue with the ants building solutions using a too high number of vehicles remains.

Branch & Propagate Constraint Programming-based Branch & Propagate depends on propagators for the constraints appearing in the model. Since the constraints in F_{bb} are unknown only generic constraint propagators can be used. These generic propagators are based on a check method, which allows to test the feasibility of a given assignment. This can of course be implemented using the black box function. However these check methods can in the worst case, be called for every possible assignment, which, given the worst-case time complexity of the black box function becomes intractable.

Branch & Price In Branch & Price the VRP is reformulated as a Set Partitioning Problem. The goal is then to choose among a set of feasible routes the ones that can be combined into a VRP-feasible solution, such that this solution minimizes the objective. In the case of the CVRPBB this would mean to choose among a set of feasible (w.r.t. F_{wb} and F_{bb}) routes the ones that minimize the objective function. Column generation would be necessary to generate feasible routes, thus the problem of coming up with routes respecting F_{wb} and F_{bb} would be deferred to the pricing problem. The intuition is that it is simpler to generate a big set of feasible routes, and then select the ones that can be combined into a VRP-feasible solution, than to generate $|K|$ (or less) feasible routes that also correspond to a VRP-feasible solution.

Based on these reflections a combination of construction heuristics and Branch & Price seems appropriate.

5

CVRPBB AS SET PARTITIONING PROBLEM

All the methods presented in this chapter are based on the reformulation of the CVRPBB as a Set Partitioning/Covering Problem. The reformulation as a SPP is as follows:

$$\begin{aligned} Min \quad & \sum_{r\in\mathcal{R}} o_r x_r & (5.1)\\ s.t. \quad & \sum_{r\in\mathcal{R}} a_{ir} x_r = 1 \quad \forall i \in V\backslash 0 & (5.2)\\ & \sum_{r\in\mathcal{R}} x_r \leq K & (5.3)\\ & x_r \in \{0,1\} \quad \forall r \in \mathcal{R} & (5.4)\end{aligned}$$

$$a_{ir} = \begin{cases} 1 & \text{if customer } i \text{ is visited in route } r \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in V\backslash 0, \quad \forall r \in \mathcal{R}$$

Set $\mathcal{R}$ corresponds to the set of all feasible routes. The set of corresponding variables is denoted by $X_{\mathcal{R}}$. The cost o_r associated with route $r \in \mathcal{R}$ is given by $distance(r) = \sum_{s=0}^{|r.S|} c_{r[s]r[s+1]}$.

For efficiency reasons constraints 5.2 will be replaced by constraints of the form

$$\sum_{r \in \mathcal{R}} a_{ir} x_r \geq 1 \quad \forall i \in V \backslash 0$$

This results in a Set Covering Problem (SCP) in which a customer may be visited more than once. In the case of the CVRPBB it may be necessary to solve the Set Partitioning Problem (SPP) in order to obtain the real solution, where each customer is visited exactly once (note that, given that $\mathcal{R}$ corresponds to the set of all feasible routes this is not the case for the CVRP). Both problems are denoted $SPP(\mathcal{R})$ and $SCP(\mathcal{R})$ whereas their restricted versions over a restricted set of feasible routes $\mathcal{R}^*$ is denoted by $SPP(\mathcal{R}^*)$ and $SCP(\mathcal{R}^*)$.

5.1 NOTATIONS

In the following two functions $solveIP(SPP(\mathcal{R}^*))$ and $solveLP(Relax(SPP(\mathcal{R}^*)))$ will be used to solve problem $SPP(\mathcal{R}^*)$ and its linear programming (LP) relaxation $Relax(SPP(\mathcal{R}^*))$.
Function $solveIP(SPP(\mathcal{R}^*))$ evaluates to a solution sol with $sol \in Sol(SPP(\mathcal{R}^*))$ if problem $SPP(\mathcal{R}^*)$ can be feasibly solved. In the other case it evaluates to $sol = \perp$. Function $solveLP(Relax(SPP(\mathcal{R}^*)))$ allows to retrieve a solution $sol_{rel} \in Sol^*(Relax(SPP(\mathcal{R}^*)))$ and the associated dual costs π. Should problem $Relax(SPP(\mathcal{R}^*))$ be infeasible it evaluates to $sol_{rel} = \perp$ and $\pi = \perp$.

A solution $sol \in Sol(SPP(\mathcal{R}^*))$ (or $sol_{rel} \in Sol^*(Relax(SPP(\mathcal{R}^*)))$) corresponds to an assignment of values to the variables in $X_{\mathcal{R}^*}$. The value of variables $x_r \in X_{\mathcal{R}^*}$ in solution sol (or sol_{rel}) is given by $\mathcal{A}_{sol}(x_r)$ ($\mathcal{A}_{sol_{rel}}(x_r)$). Note that if $sol = \perp$ then $\mathcal{A}_{sol}(x_r) = \infty \quad \forall r \in \mathcal{R}^*$.
In the case of $sol \neq \perp$, $\mathcal{A}_{sol}(x_r) \in \{0,1\} \quad \forall r \in \mathcal{R}^*$, whereas in the case of $sol_{rel} \neq \perp$, $\mathcal{A}_{sol_{rel}}(x_r) \in \mathbb{R}_+ \quad \forall r \in \mathcal{R}^*$. The set of variables taking a non-zero value in solution sol (sol_{rel}) is given by
$\overline{\mathcal{Z}}_{sol} = \bigcup_{r \in \mathcal{R}^*} \{x_r | \mathcal{A}_{sol}(x_r) > 0\}$ ($\overline{\mathcal{Z}}_{sol_{rel}} = \bigcup_{r \in \mathcal{R}^*} \{x_r | \mathcal{A}_{sol_{rel}}(x_r) > 0\}$).
The same notations and properties hold for problem SCP.

Finally a solution $sol_{rel} \in Sol^*(Relax(SPP(\mathcal{R}^*)))$ can be integer. A function $isInteger(sol_{rel})$ is provided to test this. It evaluates to true ($isInteger(sol_{rel}) = true$) if and only if $\mathcal{A}_{sol_{rel}}(x_r) \in \{0,1\} \quad \forall x_r \in \mathcal{R}^*$.

6 PHEROMONE-BASED HEURISTIC COLUMN GENERATION

As stated previously, Branch & Price is appropriate for solving the CVRPBB. The CVRPBB may however be too complex to be solved in an exact way in a reasonable amount of time. Also, while Branching subdivides the original problem into smaller, easier subproblems, this may not be necessary for smaller problem sizes. The presented method, called Pheromone-based Heuristic Column Generation (ACO-HCG) corresponds to a pure Heuristic Column Generation method, it is *incomplete* and *constructive*.
A paper describing this method was published and presented at the CPAIOR'12 conference ([MDVH12]).

This chapter is structured as follows. In section 6.1 the principles of the proposed approach are illustrated and the high-level algorithm is presented. Then the so-called Collector Ants used to generate feasible routes are introduced in section 6.2. Section 6.3 explains how pheromones are used to guide the collector ants in their process. Finally sections 6.4 and 6.5 detail how feasible routes are post-optimized and how an integer solution to the CVRPBB is obtained.

6.1 PRINCIPLES

The idea of Pheromone-based Heuristic Column Generation (ACO-HCG) is to generate feasible routes using ants (as in Ant Colony Optimization). Feasible routes are collected in a set $\mathcal{R}^*$ and each iteration the LP relaxation of the Set Covering Problem formulated over this set is solved. The resulting solution is then used to update the pheromones which will be used by the ants during the next route generation.
The overall algorithm for the Pheromone-based Heuristic Column Generation for the CVRPBB is given in Figure 6.1.1. At each iteration a varying number of feasible routes is generated using so-called *Collector Ants* (lines 5 – 7). For more information on Collector Ants see section 6.2. The generated routes are post-optimized (section 6.4) and added to $\mathcal{R}^*$ resulting in an augmented problem formulation (line 8). The relaxation of the new Set Covering Problem is then solved optimally over $\mathcal{R}^*$, providing an optimal solution sol_{rel} and the associated dual costs π (line 9). The *strict* flag is used to set the Collector ants into a special *strict* mode (as opposed to *liberal* mode) until a feasible solution to the relaxation of the resulting Set Covering Problem has been found for a first time (line 11). Finally sol_{rel} is used to update the pheromone matrix of the ants (line 12), see section 6.3 for more details on this.
After a certain stopping criterion has been reached the integer Set Covering Problem is solved over the current set $\mathcal{R}^*$ (line 15). Some customer might be visited more than once in the resulting solution sol. This must be checked, and possibly fixed in order to obtain the final set of routes $\mathcal{R}_{sol}$ (line 16) (more details in section 6.5) which is then returned.
Note that the *bb*-feasibility information of routes that have been tested is stored in the feasibility store Ψ throughout the optimization procedure.

6.2 GENERATING FEASIBLE ROUTES

Heuristic executions, called Collector ants, generating feasible routes are proposed (see Algorithm 6.2.1). Collector ants are based on the savings-based ants from [RSD02], which themselves are based on the Savings heuristic (see section 2.4.1). Each of the ℓ ants starts from an initial state where each customer is visited in a route of its own (line 1). At each iteration the ant will build a set Ω of *wb*-feasible and pos-

Algorithm 6.1.1 : Pheromone-based Heuristic Column Generation

```
1  initialize $\tau, \pi, \Psi$;
2  $strict \leftarrow$ true; $\mathcal{R}^*, collected, \mathcal{R}_{sol} \leftarrow \emptyset$; $sol, sol_{rel}, \leftarrow \perp$;
3  while $\neg$ stopping criterion do
4      $collected \leftarrow \emptyset$;
5      repeat $\ell$ times
6          $collected \leftarrow collected \cup$ CollectorAnt($strict, \tau, \pi, \Psi$);
7      end
8      $\mathcal{R}^* \leftarrow \mathcal{R}^* \cup$ post-optimize($collected, \Psi$);
9      $sol_{rel}, \pi \leftarrow solveLP(Relax(SCP(\mathcal{R}^*)))$;
10     if $sol_{rel} \neq \perp$ then
11         $strict \leftarrow$ false;
12         $\tau \leftarrow$ updatePheromones($sol_{rel}, \tau$);
13     end
14 end
15 $sol \leftarrow solveIP(SCP(\mathcal{R}^*))$;
16 $\mathcal{R}_{sol} \leftarrow$ extractSolution($sol, \mathcal{R}^*, \Psi$);
17 return $\mathcal{R}_{sol}$
```

sibly *bb*-feasible route merges (lines 4 – 16). A merge m implicates two routes r_1 and r_2. Merge m is considered feasible if route $r = r_1 \times r_2$ is feasible. Note that in Algorithm 6.2.1 a merge m is considered to contain the following information: the arc (i, j) to be added to the current state and the associated savings value s_{ij}. One of the merges in Ω is selected and executed (line 17). The execution of a merge corresponds to replacing, in the current state, the routes implicated in the merge by the route resulting from the merge. The ant merges pairs of routes until no further merge is possible (lines 3 – 20).

To build the set Ω of potential merges, the ant will first gather the set of all *wb*-feasible merges M given the current state S (lines 2 and 18). The ant will also compute the attractiveness value for each of these merges as follows:

$$attractiveness(m, \tau) = s_{ij}^{\beta} \cdot \tau_{ij}^{\alpha}$$

where (i, j) is the arc to be introduced in merge m, s_{ij} is the savings value associated with the merge and τ_{ij} is the pheromone deposit on arc (i, j). The ant then considers all the merges in M by non-increasing

Algorithm 6.2.1 : Collector ant

```
  Input : $strict, \tau, \pi, \Psi$
  Output : $collected$
1 initialize $S$ to state with shuffle routes visiting all custs. $\in V \setminus \{0\}$;
2 $i \leftarrow 0$; $collected \leftarrow \emptyset$; $\Omega \leftarrow \emptyset$; $M \leftarrow$ wbFeasibleMerges($S$);
3 while $M \neq \emptyset$ do
4     foreach $m \in M$ by - attractiveness($m, \tau$) do
5         route $r \leftarrow$ getRouteFromMerge($m$);
6         if $strict \vee rc_r(\pi) < 0 \vee r \notin \Psi$ then
7             if $r \notin \Psi$ then $\Psi[r] \leftarrow feasbb(r)$;
8             if $\Psi[r]$ then
9                 $\Omega \leftarrow \Omega \cup m$; $i \leftarrow i + 1$;
10                $collected \leftarrow collected \cup r$;
11            end
12            if $i \geq \nu$ then break;
13        else
14            $\Omega \leftarrow \Omega \cup m$
15        end
16    end
17    select $m \in \Omega$; $S \leftarrow$ executeMerge($m$, $S$);
18    $M \leftarrow$ wbFeasibleMerges($S$);
19    $i \leftarrow 0$; $\Omega \leftarrow \emptyset$;
20 end
21 return $collected$
```

attractiveness value (line 4). An important decision for the ant is which merges to accept into Ω and which ones to refuse. The decision depends on the *strict* flag, the reduced cost of the route resulting from the tentative merge, the fact whether the resulting route is known to the feasibility store Ψ and finally the feasibility of the route (lines 6 – 15). The rationale behind this approach is to keep a balance between diversification (from one ant execution to the next), extending the feasibility store and intensification towards feasible routes. Note that, depending on the black box constraints, an infeasible route might contribute to a feasible merge. When the ants are in *strict* mode, only merges that are *bb*-feasible are accepted into Ω. The construction of Ω stops once ν *bb*-feasible routes have been included, or all possible merges in M have been considered. Then, the merge from Ω to be executed is se-

lected using roulette wheel selection based on the attractiveness of all the merges in Ω (line 17).
Since the focus is shifted on feasible routes, rather than on feasible solutions, the Collector ants will collect all the feasible routes they encounter while executing the Savings heuristic (line 10). This means that not only routes that were part of the current state S at some point are collected by the ants, but also routes resulting from merges that were included into Ω but never actually selected for execution.
Finally note that both routes r s.t. $rc_r(\pi) < 0$ and routes $rc_r(\pi) \geq 0$ are tested using black box function $feasbb$. The reasoning is that while routes r s.t. $rc_r(\pi) \geq 0$ may not improve the lower bound (i.e. the quality of $sol_{rel} \in Sol^*(Relax(SCP(\mathcal{R}^*)))$) when added to $\mathcal{R}^*$, they may very well improve $sol \in Sol^*(SCP(\mathcal{R}^*))$, given the highly combinatorial nature of the CVRPBB.

6.3 PHEROMONE UPDATE

Each collector ant disposes of the pheromone matrix τ, indicating the amount of pheromones on every arc (i, j). The pheromones influence the attractiveness of an arc, and thus the probability of a merge introducing the arc to enter Ω and to be executed by an ant. The pheromone matrix is updated based on solution $sol_{rel} \in Sol^*(Relax(SCP(\mathcal{R}^*)))$.
The idea behind the pheromones is to guide the ants towards routes of good quality and with higher probability of being bb-feasible. Since routes in sol_{rel} are known to be optimal (w.r.t. $Relax(SCP(\mathcal{R}^*))$), as well as bb-feasible, the idea is to get the ants to produce routes similar to those in sol_{rel}, in the hope of producing routes of similar qualities.
Once $sol_{rel} \neq \perp$ is available, the pheromone matrix is updated as follows:

$$\tau_{ij} = \rho\tau_{ij} + \sigma_{ij}\epsilon \quad \forall(i,j)(i,j \in V,\ i \neq j)$$

where ρ ($0 \leq \rho \leq 1$) is the trail persistence, ϵ is a small constant and σ_{ij} is the number of routes $r \in \mathcal{R}^*$ such that $\mathcal{A}_{sol_{rel}}(x_r) > 0$ and s.t. $i, j \in r.S$ and $pos(i,r) = pos(j,r) - 1$. Simpler put, σ_{ij} corresponds to the number of times arc (i, j) appears in sol_{rel}.
The first term thus corresponds to pheromone evaporation, while the second is the deposit. Note that the pheromone deposit of any arc $(i, j) \in A$ is not allowed to fall below τ_{min}.

6.4 POST-OPTIMIZATION OF FEASIBLE ROUTES

Collector ants are used to generate feasible routes. Before being added to the set of feasible routes $\mathcal{R}^*$, the generated routes are first post-optimized. This is done using local search. A local search is thus applied to each feasible route collected by the Collector ants. The goal is, given a route r to find a feasible route r' such that $r.S = r'.S$ and $distance(r') < distance(r)$. Two different local search methods have been considered : a Tabu Search (TS) and an Iterated Local Search-style method (ILS).

6.4.1 *Tabu Search*

The algorithm is given in 6.4.1. It corresponds to a deterministic tabu search without aspiration criterion and with a tabu list Ξ of infinite length. Every iteration, the neighborhood N of the current route r_{cur} is built (line 7) using neighborhood operator η. N corresponds to a set of neighboring routes. These neighboring routes are then analyzed one by one in order of non-decreasing distance. As soon as a neighboring route that hasn't been visited so far is found, the search moves to this new route (lines 9 and 10) which becomes the current route. The current route is added to the list of visited routes (line 11). If the current route improves the best route so far, the *bb*-feasibility of the current route is checked (lines 13 – 15). If it is found to be feasible the best known route and distance are updated (lines 16 – 17). The procedure stops once there are no unvisited neighbors in the current neighborhood or once a global stopping criterion has been reached.

This tabu search is applied twice, first with $\eta = \eta_{reloc}$ and then with $\eta = \eta_{2ex}$ on the resulting route. The rationale is that since the relocation moves have a smaller impact on a route, the probability that the *bb*-feasibility is retained is higher. Thus the relocation neighborhood is used first on route r in the hope of obtaining an improved route r'. The idea is that using the 2-exchange neighborhood on this route r' only few routes further improving r' will be discovered, and thus only few *bb*-feasibility tests (with higher probability to fail) will be necessary.

The relocation neighborhood is built using algorithm 6.4.2 while the 2-exchange neighborhood is built using algorithm 6.4.3. The corresponding operators were already defined in section 2.5. Note that both neighborhoods are complete.

Algorithm 6.4.1 : Tabu Search

```
Input : route $r$, $\Psi$, $\eta$
Output : route $r_{best}$
1  $N \leftarrow \emptyset$; $\Xi \leftarrow \emptyset$;
2  $r_{best} \leftarrow r$; $r_{cur} \leftarrow r$;
3  $bestDistance \leftarrow distance(r_{cur})$;
4  found $\leftarrow$ true;
5  while $found = true \wedge \neg$ stopping criterion do
6      found $\leftarrow$ false;
7      $N \leftarrow$ buildNeighborhood($r_{cur}, \eta$);
8      foreach $n \in N$ by distance(n) do
9          if $r \notin \Xi$ then
10             $r_{cur} \leftarrow n$;
11             $\Xi \leftarrow \Xi \cup r_{cur}$;
12             found $\leftarrow$ true;
13             if $distance(r_{cur}) < bestDistance$ then
14                 if $r_{cur} \notin \Psi$ then $\Psi[r_{cur}] \leftarrow feasbb(r_{cur})$;
15                 if $\Psi[r_{cur}]$ then
16                     $bestDistance \leftarrow distance(r_{cur})$;
17                     $r_{best} \leftarrow r_{cur}$
18                 end
19             end
20         end
21     end
22 end
23 return $r_{best}$
```

To construct the relocation neighborhood of route r every customer $i \in r.S$ and every possible position $p \neq pos(i, r)$ is considered (lines 1 – 2 in algorithm 6.4.2). In the case of the 2-exchange neighborhood every possible pair of arcs (i, j) and (u, v) is considered (lines 1 – 4 in algorithm 6.4.3). Note also that only *wb*-feasible routes are accepted into the respective neighborhoods (lines 4–6 in algorithm 6.4.2 and lines 6 – 8 in algorithm 6.4.3).

Algorithm 6.4.2 : Neighborhood construction using η_{reloc}

```
Input : route $r$
Output : $N_{reloc}$
1 foreach $i \in r.S$ do
2     foreach $p \in 1, \dots, |r.S| + 1$ do
3         $r' \leftarrow \eta_{reloc}(i, r, p, \{r\})$;
4         if $\bigwedge_{c \in F_{wb}} feas(r', c)$ then
5             $N_{reloc} \leftarrow N_{reloc} \cup r'$;
6         end
7     end
8 end
9 return $N_{reloc}$
```

6.4.2 *Iterated Local Search-style approach*

The basic idea of this approach is to collect a list of good (and possibly *wb*- and/or *bb*-infeasible) routes found over several iterations. Each iteration a hill-climbing method and a random perturbation are executed. The routes collected are then considered by increasing distance and tested for feasibility. The algorithm is given in 6.4.4.

Lines 5 – 11 correspond to the hill-climbing procedure. Here a complete neighborhood N is built by merging the complete *wb*-feasible N_{reloc} and the complete *wb*-feasible N_{2ex} neighborhoods (lines 6 and 7, see algorithms 6.4.2 and 6.4.3). The neighbors that do not improve the distance of the current route r_{cur} are removed from the neighborhood (line 8). The search then moves to the best route in the neighborhood (line 9) and stores this new route in list Ξ (line 10). Once no further improving moves can be found a random perturbation is applied to the current route (lines 12 – 20). If the current routes visits less than 4 customers, a random relocation move is applied (possibly resulting in a *wb*-infeasible route, lines 12 – 14). In the other case a random 4-exchange move is applied, removing arcs in set A_{out} and replacing them by arcs A_{in} (lines 15 –20, again possibly resulting in an *wb*-infeasible route). Hill-climbing and random perturbations are executed until a stopping criterion is met.

At this point the routes stored in list Ξ are then considered individually by non-decreasing distance (line 22). Each route is checked for *wb*-feasibility and *bb*-feasibility (lines 23 – 25). Once a feasible route

Algorithm 6.4.3 : Neighborhood construction using η_{2ex}

```
Input : route $r$
Output : $N_{2ex}$
1  foreach $i \in 0, \ldots, |r.S| - 2$ do
2      $j \leftarrow i + 1$;
3      foreach $u \in j + 1, \ldots, |r.S|$ do
4          $v \leftarrow u + 1$;
5          $r' \leftarrow \eta_{2ex}(r, \{(i,j),(u,v)\}, \{(i,u),(j,v)\}, \{r\})$;
6          if $\bigwedge_{c \in F_{wb}} feas(r', c)$ then
7              $N_{2ex} \leftarrow N_{2ex} \cup r'$;
8          end
9      end
10 end
11 return $N_{2ex}$
```

is found, this must be the best feasible route in Ξ. The procedure is stopped and this route is returned.

6.5 SOLVING THE INTEGER PROBLEM

The final solution is obtained by solving $SCP(\mathcal{R}^*)$, where $\mathcal{R}^*$ is the set of feasible routes that have been generated throughout the process. A time limit is imposed on this process, thus the resulting solution $sol \in Sol(SCP(\mathcal{R}^*))$ may not be optimal for the set $\mathcal{R}^*$. Furthermore some customers may be visited more than once in sol. Given the collector ant algorithm, it is not possible for a customer to be visited more than once in a same route. Also given the nature of the function $solveIP$ a customer i that is visited more than once in sol will never appear in a route r s.t. $r.S = \{i\}$. However, a customer might appear in different routes r s.t. $|r.S| > 1$ of a given solution. If this is the case, the solution must be fixed. Else it is sufficient to extract the set of selected routes from sol. Both is done using algorithm 6.5.1.

First the routes currently selected in sol are extracted into set $\mathcal{R}_{sol}$ (line 1). Then each customer i is considered in turn (line 2) and the number of visits to the customer in the solution is counted (line 4). If i is visited more than once, this situation needs to be fixed. Each of the routes in which customer i is visited are considered. Routes where re-

Algorithm 6.4.4 : Iterated Local Search

```
Input : route $r$, $\Psi$
Output : route $r_{best}$
$N \leftarrow \emptyset$; $\Xi \leftarrow \emptyset$;
$r_{best} \leftarrow r$; $r_{cur} \leftarrow r$;
$bestDistance \leftarrow distance(r_{cur})$;
while $\neg$ stopping criterion do
    repeat
        $N \leftarrow$ buildNeighborhood($r_{cur}, \eta_{reloc}$);
        $N \leftarrow N \cup$ buildNeighborhood($r_{cur}, \eta_{2ex}$);
        $N \leftarrow N \setminus \{r_{bad} \in N | distance(r_{bad}) \geq distance(r_{cur})\}$;
        $r_{cur} \leftarrow \text{argmin}_{r' \in N}\, distance(r')$;
        $\Xi \leftarrow \Xi \cup r_{cur}$;
    until $N \neq \emptyset$;
    if $|r_{cur}.S| < 4$ then
        select random $p, p' \in 1, \ldots, |r_{cur}.S|$ s.t. $p \neq p'$;
        $r_{cur} \leftarrow \eta_{reloc}(r_{cur}[p], r_{cur}, p', \{r_{cur}\})$;
    else
        select random $p_1, p_2, p_3, p_4 \in 1, \ldots, |r_{cur}.S|$ s.t.
        $p_1 \neq p_2 \neq p_3 \neq p_4$;
        $A_{out} \leftarrow \{(p_1, p_1 + 1), (p_2, p_2 + 1), (p_3, p_3 + 1), (p_4, p_4 + 1)\}$;
        $A_{in} \leftarrow \{(p_1, p_3 + 1), (p_4, p_2 + 1), (p_3, p_1 + 1), (p_2, p_4 + 1)\}$;
        $r_{cur} \leftarrow \eta_{4ex}(r_{cur}, A_{out}, A_{in}, \{r_{cur}\})$;
    end
end
foreach $r \in \Xi$ by $distance(r)$ do
    if $\bigwedge_{c \in F_{wb}} feas(r', c)$ then
        if $r_{cur} \notin \Psi$ then $\Psi[r_{cur}] \leftarrow feasbb(r_{cur})$ ;
        if $\Psi[r_{cur}]$ then
            $bestDistance \leftarrow distance(r_{cur})$;
            $r_{best} \leftarrow r_{cur}$;
            break;
        end
    end
end
return $r_{best}$
```

moving i results in a higher loss of distance are considered first (line 6). For each route r the route r' resulting from removing customer i from r is examined for feasibility (lines 8 – 9). If r' is feasible it is used to replace r in the set of selected routes (line 11). The process for customer i stops once all but one visits to customer i have been removed (lines 15 – 17) or all routes have been treated. If not all excess visits could be feasibly removed the fixing failed and the exact problem needs to be solved (lines 19 – 23). Note that the solution fixing algorithm is heuristic and is not guaranteed to find a way to fix a solution if one exists. Depending on the F_{bb} constraints, the order in which visits are removed from a route might have an impact on the bb-feasibility of the intermediary route. More clearly, assume that customers i and j must be removed from some route r $(i, j \in |r.S|)$. Assume further that removing customer i from route r results in $r' = remove(i, r)$, and that r' is infeasible. In such a case the heuristic fixing procedure will fail. It is possible however that removing customer j, resulting in route $r'' = remove(j, r)$ and then removing customer i from the resulting route $r''' = remove(i, r'')$ would have resulted in a feasible route r'''.

Algorithm 6.5.1 : Extract solution

```
Input : $sol, \mathcal{R}^*, \Psi$
Output : set of routes $\mathcal{R}_{sol}$
1  $\mathcal{R}_{sol} \leftarrow \{x_r | \mathcal{A}_{sol}(x_r) = 1\}$;
2  foreach $i \in V \backslash \{0\}$ do
3      done $\leftarrow$ false;
4      $visits \leftarrow |\{r \in \mathcal{R}_{sol} | i \in r.S\}|$;
5      if $visits > 1$ then
6          foreach $r \in \mathcal{R}_{sol}$ $s.t.$ $i \in r.S$ by $distance(remove(i, r))$ do
7              $r' \leftarrow remove(i, r)$;
8              if $\bigwedge_{c \in F_{wb}} feas(r', c)$ then
9                  if $r' \notin \Psi$ then $\Psi[r'] \leftarrow feasbb(r')$;
10                 if $\Psi[r']$ then
11                     $\mathcal{R}_{sol} \leftarrow \mathcal{R}_{sol} \backslash \{r\} \cup \{r'\}$;
12                     $visits \leftarrow visits - 1$;
13                 end
14             end
15             if $visits = 1$ then
16                 break;
17             end
18         end
19         if $visits > 1$ then
20             $sol \leftarrow solveIP(SPP(\mathcal{R}^*))$;
21             $\mathcal{R}_{sol} \leftarrow \{x_r | \mathcal{A}_{sol}(x_r) = 1\}$;
22             break;
23         end
24     end
25 end
26 return $\mathcal{R}_{sol}$
```

7 DIVE & GENERATE

This chapter presents an approach combining Heuristic Column Generation with the heuristic exploration of a Branching search tree. This method is based on the hope, that when solving problems that are more restricted than the original Set Covering Problem (corresponding to the original CVRPBB), the chances are higher that routes of negative reduced cost are also part of high quality integer solutions to this same restricted problem.
Restricted problems are created by fixing routes in the solution. To each node in the Branching search tree thus corresponds a Set Covering Problem formulated over a current set of feasible routes, and also a set of variables that are forced to appear in the solution to this problem. This problem is solved and the resulting solution is used to determine the set of variables to be fixed at the child nodes of the current node. Moreover Heuristic Column Generation is executed at selected nodes. The approach is called Dive & Generate (DING). It is an *incomplete* and *constructive* method based on Branch & Generate, which is itself based on Branch & Price.

The following chapter is structured as follows. First an overview of the Branch & Generate scheme is provided in section 7.1. Then section 7.2 explains how the Branch & Generate scheme is extended into Dive & Generate. In the latter section the overall algorithm is provided and the main design choices to be taken are underlined. These choices are then considered in their respective sections; the way an initial set of feasible routes is obtained is explained in section 7.3. Section 7.4 provides details on the restart policy. Section 7.5 explains how nodes in the Dive & Generate tree are explored and how decisions such as whether to generate columns, to prune the subtree or to compute the child nodes are taken. The algorithm used to generate feasible routes is detailed in section 7.6. Finally the branching heuristic and search strategy are covered in sections 7.7 and 7.8.

7.1 BRANCH & GENERATE

Branch & Generate has been proposed to quickly find feasible (integral) solutions to Set Covering (or Partitioning) Problems. The idea of Branch & Generate is to directly descend to leaf nodes in the Branch & Price tree in order to find a feasible solution and to generate new sets $\mathcal{I}$ (routes in the case of VRPs) only when considered necessary. Thus it combines a diving heuristic with column generation.

Branch & Generate has been proposed by R. Marsten but has never been officially published. Due to this the technique may appear under different names in the literature. It is used in [GBLJ03] to solve a Duty Scheduling Problem. A heuristic called *Rapid Branching* that can be seen as a Branch & Generate heuristic is developed in [BLR+12]. Branch & Generate is also acknowledged in [MO11], an overview of heuristics based on mathematical programming. The authors of [GKL+06] consider a Branch & Generate scheme for their Vehicle Routing and Staffing problem, but this under the name of *Fix & Price*. Finally the heuristics proposed in [JMS+10] are also in the spirit of Branch & Generate.

The idea of Branch & Generate is to create only one subproblem (as opposed to multiple ones as is done in Branch & Bound and Branch & Price) per original Set Covering Problem. This is done by fixing some (binary) variable(s) in the original problem to 1. Using this method, a tree, corresponding to one branch, i.e. a single path from root node to a leaf node, is constructed. As in Branch & Price the original problem is formulated over a restricted set of sets called $\mathcal{R}^*$. At some nodes column generation is used to enrich this $\mathcal{R}^*$.

In this section, a node i in the Branch & Generate tree will be designated by SCP_i. With each node SCP_i is associated $\mathcal{F}_i$, a set of variables forced to take value 1, an initial set $\mathcal{R}_i^*$ and a final set $\mathcal{R}_i^{*\prime}$. Naturally $\mathcal{F}_i \subseteq \mathcal{R}_i^*$. The associated problems are then called $SCP(\mathcal{R}_i^*, \mathcal{F}_i)$ and $SCP(\mathcal{R}_i^{*\prime}, \mathcal{F}_i)$ respectively. Note that if column generation is executed at node SCP_i then $\mathcal{R}_i^* \subseteq \mathcal{R}_i^{*\prime}$, in the other case $\mathcal{R}_i^* = \mathcal{R}_i^{*\prime}$.

As stated before, given some original problem $SCP(\mathcal{R}^*, \mathcal{F})$ only one subproblem $SCP(\mathcal{R}^*, \mathcal{F} \cup U)$ is created. This is done by choosing a set of variables U, s.t. $U \subseteq X_{\mathcal{R}^*} \backslash \mathcal{F}$ and forcing them to take value 1 in the subproblem. To do this, solution $sol_{rel} \in Sol^*(Relax(SCP(\mathcal{R}^*, \mathcal{F})))$ is considered and only variables $x_r \in \overline{\mathcal{Z}}_{sol_{rel}} \backslash \mathcal{F}$ are considered to be fixed to 1 in the subproblem. Thus a set U of non-zero variables, that are not already fixed (forced to 1) in the original problem, is chosen in sol_{rel}. In the subproblem the variables in U will be fixed, in addition to those already fixed in the original problem.
The process is repeated for each subproblem until reaching either a feasible solution or an infeasible subproblem. Branch & Generate only constructs a single path from a root node (original problem) to a leaf node (feasible solution or infeasible subproblem).

At some nodes SCP_j on this path, column generation is used to selectively generate new sets $\mathcal{I}$ (routes in the case of VRPs) s.t. $rc_{\mathcal{I}}(\pi^j) < 0$ (where π^j are the dual costs associated with
$sol_{rel} \in Sol^*(Relax(SCP(\mathcal{R}_j^*, \mathcal{F}_j))))$. The decision on whether to execute column generation depends on $LB_j = Obj(sol_{rel})$. A trust region is defined for the LB_j value, and once LB_j falls outside this region, new sets (or routes) are generated. The pricing, that is the generation of such new sets, is generally done in a heuristic manner, thus the problem $Relax(SCP(\mathcal{R}, \mathcal{F}_j))$ is not solved to optimality ($\mathcal{R}$ being the set of all possible sets).

7.2 PRINCIPLES

In this thesis Branch & Generate is combined with Backtracking, Heuristic Pruning and Restarts (called Dive & Generate (DING)). Note that Dive & Generate is quite similar to Rapid Branching [BLR+12] where the concepts of Backtracking and Heuristic Pruning are used as well. Rapid Branching combines Branch & Generate with a sophisticated branching strategy called Perturbation Branching and a Binary Search-based backtracking strategy, both of which are not considered here.

In the Dive & Generate tree, each non-leaf node SCP_i will have $|\overline{\mathcal{Z}}_{sol_{rel}} \backslash \mathcal{F}_i|$ children, with $sol_{rel} \in Sol^*(Relax(SCP(\mathcal{R}_i^{*\prime}, \mathcal{F}_i)))$. Thus, each non-leaf node will have one child node per variable (that is route) that can be fixed. The root node corresponds to SCP_0 with $\mathcal{F}_0 = \emptyset$. Set $\mathcal{R}_0^*$ is initialized such that $Relax(SCP(\mathcal{R}_0^*, \emptyset))$ and $SPP(\mathcal{R}_0^*, \emptyset)$ are feasible. The solution $sol \in Sol(SPP(\mathcal{R}_0^*, \emptyset))$ will be used to initialize a global upper bound UB.

As seen in Branch & Generate, heuristic column generation will be used at some nodes SCP_i. This is the case if at some non-root node SCP_i the lower bound $LB_i = Obj(sol_{rel})$ lies outside a trust region ($sol_{rel} \in Sol^*(Relax(SCP(\mathcal{R}_i^*, \mathcal{F}_i)))$). In the case of the root node, column generation will always be executed.
Generating new feasible routes at some node SCP_i means enriching set $\mathcal{R}_i^*$, resulting in set $\mathcal{R}_i^{*\prime}$. Thus the set of routes associated with the node SCP_i is augmented.
When exploring some node SCP_i the final set $\mathcal{R}_i^{*\prime}$ corresponds to the initial set of routes for the next node to be explored SCP_{i+1}. Thus if node SCP_{i+1} is explored directly after node SCP_i, $\mathcal{R}_{i+1}^* = \mathcal{R}_i^{*\prime}$. This means that if nodes $SCP_0, SCP_1, \ldots, SCP_w$ are visited in that order, then $\mathcal{R}_0^* \subseteq \mathcal{R}_0^{*\prime} \subseteq \mathcal{R}_1^* \subseteq \mathcal{R}_1^{*\prime} \subseteq \cdots \subseteq \mathcal{R}_w^* \subseteq \mathcal{R}_w^{*\prime}$, given that column generation may add new feasible routes at each node. This is usual in a Branch & Price context.
Let two solutions, $sol \in Sol^*(Relax(SCP(\mathcal{R}_0^{*\prime}, \emptyset)))$ and $sol_w \in Sol^*(Relax(SCP(\mathcal{R}_w^{*\prime}, \emptyset)))$ with $\mathcal{R}_0^{*\prime} \subseteq \mathcal{R}_w^{*\prime}$, then the following holds: $Obj(sol_w) \leq Obj(sol)$ and $0 \leq |\overline{\mathcal{Z}}_{sol} \cap \overline{\mathcal{Z}}_{sol_w}| \leq |\overline{\mathcal{Z}}_{sol}|$. Thus, as the set of routes has a higher cardinality for the latter problem, the non-zero variables in sol_w might be partially or completely different from the non-zero variables in sol.

The candidate routes considered for fixing (to compute the child nodes) at the root node will be decided based on the solution of $Relax(SCP(\mathcal{R}_0^{*\prime}, \emptyset)$. Should this initial set of candidates be a bad choice then time will be lost exploring the corresponding tree to no avail. It is for this reason that the construction and exploration of the Dive & Generate tree is stopped and restarted from time to time. Each restart possibly results in a fundamentally different tree.

Design Decisions

The following decisions must be taken when designing the Dive & Generate approach for the CVRPBB:

1. How to initialize $\mathcal{R}^*$? (Initial set of routes, section 7.3)
2. When to restart? (Restarts, section 7.4)
3. How are non-root nodes explored? (Node exploration, section 7.5)
   - when to prune?
   - when to generate new routes?
4. How to generate new routes? (Pricing, section 7.6)
5. How to order nodes in the tree? (Branching heuristic, section 7.7)
6. How to explore the Dive & Generate tree ? (Search strategy, section 7.8)

These concerns are addressed in the indicated sections.

Algorithm

The high-level Dive & Generate algorithm is given in 7.2.1. A feasibility store Ψ is used to store the *bb*-feasibility of tested routes.

First an initial set of feasible routes is generated (line 3) and the current and incumbent solutions, the dual costs associated with the current solution, as well as the upper bound UB are initialized, see section 7.3 on details on how this is done. Note that this initialization must ensure that $sol_0 \neq \perp$ and $sol_{inc} \neq \perp$. Lines 5 – 17 correspond to the construction and exploration of one Dive & Generate tree. First new feasible routes are generated using the dual costs π^0 (line 6, see section 7.6) and added to $\mathcal{R}^*$. Then the LP relaxation of the problem $SCP(\mathcal{R}^*, \emptyset)$ is solved (updating solution sol_0 and dual costs π^0) in line 7. Should sol_0 be an integer solution, it could possibly be used to update the upper bound UB and the incumbent solution sol_{inc} (line 8). Route generation and problem resolving are repeated until sol_0 becomes fractional or the overall stopping criterion is met.
Once the root solution sol_0 is available, one child node is created for

Algorithm 7.2.1 : Dive & Generate

```
1  $Frontier, \mathcal{R}^*, \Psi \leftarrow \emptyset$;
2  $sol_0, sol_{inc} \leftarrow \perp$; $UB \leftarrow \infty$;
3  initialize $\mathcal{R}^*, sol_0, \pi^0, sol_{inc}$ and $UB$ // see section 7.3
4  while $\neg$ stopping criterion do
5      repeat
6          $\mathcal{R}^* \leftarrow \mathcal{R}^* \cup$ doPricing($\emptyset, \pi^0, \Psi$) // see section 7.6
7          $sol_0, \pi^0 \leftarrow solveLP(Relax(SCP(\mathcal{R}^*, \emptyset)))$;
8          if $isInteger(sol_0)$ then update($sol_0, sol_{inc}, UB, \Psi, \mathcal{R}^*$);
9      until $\neg isInteger(sol_0) \vee$ stopping criterion;
10     if stopping criterion then break;
11     foreach $x_k \in X^*_{\mathcal{R}}$ s.t. $\mathcal{A}_{sol_0}(x_k) > 0$ do
12         $Frontier \leftarrow Frontier \cup \{x_k\}$;
13     end
14     while $Frontier \neq \emptyset \wedge \neg$ time to restart $\wedge \neg$ stopping criterion
       do
15         select $\mathcal{F} \in Frontier$;
16          /* perform exploration of node $(\mathcal{R}^*, \mathcal{F})$, see section
           7.5 */
17     end
18     if stopping criterion then break;
19     $\mathcal{R}^* \leftarrow \mathcal{R}^* \cup$ strictGeneration($\Psi$) // see section 7.6
20     $sol_0, \pi^0 \leftarrow solveLP(Relax(SCP(\mathcal{R}^*, \emptyset)))$;
21 end
22 return $\{r | \mathcal{A}_{sol_{inc}}(x_r) = 1\}$;
```

each non-zero variable in sol_0 (lines 11 – 13). Since $\mathcal{R}^*$ is global to the tree and the problem being solved is always a Set Covering Problem, it is sufficient to store the set of variables to fix to 1 in the frontier. Next the nodes in the frontier are explored either until the entire tree has been explored, or until the process should be restarted, or until the global stopping criterion is reached (lines 14 – 17). Of course the exploration of a node may add further nodes to the frontier and further

routes to $\mathcal{R}^*$. The exploration process of a node is explained in detail in section 7.5.
After the exploration of a Dive & Generate tree has finished, an intensified route generation procedure is started (line 18). Here feasible routes are generated, without taking any dual costs into account. The idea behind this is to enrich set $\mathcal{R}^*$ with feasible routes that might not improve the current solution, but might be necessary to find good solutions at a later point. Finally the LP relaxation of problem $SCP(\mathcal{R}^*, \emptyset)$ is solved to initialize sol_0 and π^0 for the next Dive & Generate iteration (line 20).

As said previously, should $sol_0 \in Sol^*(Relax(SCP(\mathcal{R}^*, \emptyset)))$ be integral, it can possibly be used to update the incumbent and upper bound. This is however only possible if no customer is visited more than once in sol_0. Method `update(`$sol_0, sol_{inc}, UB, \Psi, \mathcal{R}^*$`)` must thus check sol_0 and possibly fix it heuristically. This is done in a way that is conceptually similar to the one described in section 6.5. Each customer is considered individually. If the customer is visited more than once, each of the routes visiting it are considered one by one. The customer is removed from all but one of these routes. Of course this is only possible if the resulting routes are *wb*- and *bb*-feasible. If the heuristic fails, the incumbent sol_{inc} (and upper bound UB) cannot be updated. If it succeeds the incumbent sol_{inc} and upper bound UB are updated with the fixed solution and its cost. Note that new feasible routes discovered using the fixing process are added to $\mathcal{R}^*$.

7.3 INITIALIZATION OF $\mathcal{R}^*$

The method used to initialize $\mathcal{R}^*$ is exactly the same as the one used in Pheromone-based Heuristic Column Generation (chapter 6). A number ℓ of collector ants in *strict* mode (see 6.2.1) is repeatedly executed to accumulate routes in $\mathcal{R}^*$ (no pheromones are used) which are then post-optimized. Problems $Relax(SCP(\mathcal{R}^*, \emptyset))$ and $SCP(\mathcal{R}^*, \emptyset)$ are then solved and the resulting solutions are stored in sol_0 and sol_{inc}. These steps are repeated either until the overall stopping criterion is met, or until sol_0 and sol_{inc} are feasible and sol_0 is fractional. Solution sol_{inc} is fixed if necessary which allows to initialize UB with the cost of sol_{inc}.

7.4 RESTARTS

Let $sol_0 \in Sol^*(Relax(SCP(\mathcal{R}_0^{*\prime}, \emptyset)))$ be the solution at the root node of the current Dive & Generate tree, and let π^0 the corresponding dual costs. Throughout the tree traversal, every time new feasible routes are generated, the number imp_0 of routes r s.t. $rc_r(\pi^0) < 0$ is updated. Once imp_0 reaches a limit ϕ the tree exploration is aborted and restarted. Thus condition `time to restart` in algorithm 7.2.1 is fulfilled as soon as $imp_0 \geq \phi$.

7.5 NODE EXPLORATION

A decision scheme telling us when to prune a particular node SCP_i and when to generate new feasible routes at node SCP_i is needed. Let $\mathcal{R}_i^*$ the initial set of feasible routes associated with SCP_i.
If $Relax(SCP(\mathcal{R}_i^*, \mathcal{F}_i))$ turns out to be infeasible, node SCP_i is a leaf node. If the problem is feasible and solution sol_i is integral
($sol_i \in Sol^*(Relax(SCP(\mathcal{R}_i^*, \mathcal{F}_i)))$, then SCP_i is considered a leaf node as well. On the other hand if sol_i is fractional and the lower bound LB_i ($LB_i = Obj(sol_i)$) lies outside a trust region, column generation is entered. The trust region is defined based on the current global upper bound UB and a parameter θ ($0 \leq \theta \leq 1$). If $LB_i \geq \theta \cdot UB$ it is considered that LB_i lies outside the trust region.
Generating feasible routes at node SCP_i will result in final set $\mathcal{R}_i^{*\prime}$ with $\mathcal{R}_i^* \subseteq \mathcal{R}_i^{*\prime}$. Once feasible routes have been generated, the final lower bound LB_i' is computed as $LB_i' = Obj(sol_i')$) with $sol_i' \in Sol^*(Relax(SCP(\mathcal{R}_i^{*\prime}, \mathcal{F}_i))))$. If LB_i' is greater or equal to the global upper bound UB ($LB_i' \geq UB$) then SCP_i is pruned.
The intuition behind this is the following: either new routes of negative reduced cost have been added, but the new lower bound LB_i' does not sufficiently improve over LB_i to fall below UB, or the pricing heuristic failed to generate routes of negative reduced cost (not a proof of their non-existence!). In both cases the assumption is that LB_i' is a reasonable approximation of $Obj(Relax(SCP(\mathcal{R}, \mathcal{F}_i)))$, and that probably no solution improving UB will be found in the subtree of SCP_i. Again, if sol_i' is integer, SCP_i is considered a leaf node.

The complete (non-root) node exploration algorithm is given in 7.5.1. It takes number of inputs. Note that some of these are modified throughout the node exploration.

- $\mathcal{R}^*$ the current node's initial set of feasible routes
- $\mathcal{F}$ the set of variables fixed in the current node
- sol_{inc} the incumbent solution
- UB the global upper bound
- π^0, the dual costs associated with the final root node solution
- imp_0 the counter of routes of negative reduced cost w.r.t. π^0
- $Frontier$ the frontier of unexplored nodes
- Ψ the feasibility store

First the LP relaxation of the current problem is solved resulting in solution sol_{cur} and dual costs π^{cur} (line 2). If the subproblem turns out to be infeasible, the node is pruned (line 3). If solution sol_{cur} is integer and improves the upper bound UB, then the latter and the incumbent solution are possibly updated (as explained previously) and the exploration of the node finishes since it is a leaf node (lines 4 – 7). Next it is tested whether the lower bound LB_{cur} corresponding to sol_{cur} lies outside the trust region (line 9). If LB_{cur} lies outside the trust region then feasible routes r s.t. $rc_r(\pi^{cur}) < 0$ should be generated. The generated routes are inserted into $\mathcal{R}^*$ (line 11) and imp_0 is updated (line 12). Then the LP relaxation of the current subproblem formulated on the (possibly) augmented $\mathcal{R}^*$ is solved, updating sol_{cur} (line 13). If the resulting lower bound is equal to or exceeds the global upper bound the node is pruned (line 18). Finally the solution sol_{cur} is used to compute the child nodes which are added to the frontier. One node is created for each variable x_k taking a non-zero value in sol_{cur}, not already fixed by a constraint (i.e. not already in the current $\mathcal{F}$) (lines 20 – 22).

7.6 GENERATING FEASIBLE ROUTES

Two different column generation schemes are used in Dive & Generate. The `strictGeneration(...)` scheme is used to generate additional random feasible routes at root nodes. This corresponds to the execution of ℓ collector ants (no pheromones are used) in *strict* mode (see algorithm 6.2.1) and the post-optimization of the generated feasible routes.

Algorithm 7.5.1 : Dive & Generate - explore Node

```
Input : $\mathcal{F}, \pi^0, imp_0$
Inout : $\mathcal{R}^*, Frontier, \Psi, imp_0, sol_{inc}, UB$
1  $\mathcal{L} \leftarrow \emptyset$;
2  $sol_{cur}, \pi^{cur} \leftarrow solveLP(Relax(SCP(\mathcal{R}^*, \mathcal{F})))$;
3  if $sol_{cur} = \perp$ then return;
4  if $isInteger(sol_{cur}) \wedge Obj(sol_{cur}) < UB$ then
5      update($sol_{inc}, UB, \Psi, \mathcal{R}^*$);
6      return;
7  end
8  $LB_{cur} \leftarrow Obj(sol_{cur})$;
9  if $LB_{cur} \geq \theta \cdot UB$ then
10     $\mathcal{L} \leftarrow$ doPricing($\mathcal{F}, \pi^{cur}, \Psi$) // see section 7.6
11     $\mathcal{R}^* \leftarrow \mathcal{R}^* \cup \mathcal{L}$;
12     $imp_0 \leftarrow imp_0 + |\{r \in \mathcal{L} | rc_r(\pi^0) < 0\}|$;
13     $sol_{cur} \leftarrow solveLP(Relax(SCP(\mathcal{R}^*, \mathcal{F})))$;
14     if $isInteger(sol_{cur}) \wedge Obj(sol_{cur}) < UB$ then
15         update($sol_{inc}, UB, \Psi, \mathcal{R}^*$);
16         return;
17     end
18     if $LB_{cur} \geq UB$ then return;
19 end
20 foreach $x_k \in X_{\mathcal{R}^*} \backslash \mathcal{F}$ s.t. $\mathcal{A}_{sol_{cur}}(x_k) > 0$ do
21     $Frontier \leftarrow Frontier \cup \{x_k\}$;
22 end
23 return;
```

The second scheme (`doPricing(...)`) is used to generate feasible routes of negative reduced cost at different nodes SCP_i in the Dive & Generate tree. The goal is to generate new feasible routes r s.t. $rc_r(\pi^i) < 0$ where π^i are the dual costs associated with solution $sol_i \in Sol^*(Relax(SCP(\mathcal{R}_i^*, \mathcal{F}_i)))$. To do this, ℓ randomized executions of the savings heuristic are used. The idea is to test *bb*-feasibility only for routes r s.t. $rc_r(\pi^i) < 0$, since only these nodes may improve the current lower bound.

Since at node SCP_i the customers in routes r s.t. $x_r \in \mathcal{F}_i$ are already visited, no new routes visiting any customer $c \in \bigcup_{x_r \in \mathcal{F}_i} \{r.S\}$ should be generated. This can be easily handled by considering customers in set $\mathcal{F}_i$ as not belonging to the current problem instance.

The algorithm is given in 7.6.1. The initial one-customer routes are created only for customers $c \notin \bigcup_{x_r \in \mathcal{F}} \{r.S\}$ (line 1). Then the arc costs are updated w.r.t. the dual costs π^{cur} (line 2). The cost c_{ij} of each arc $(i,j) \in A$ is updated as

$$c_{ij}^{cur} = c_{ij} - \frac{\pi_i^{cur}}{2} - \frac{\pi_j^{cur}}{2}$$

in order to facilitate the generation of routes r s.t. $rc_r(\pi^{cur}) < 0$. These arc costs are then used in the remainder of the algorithm.
At each iteration the heuristic constructs a set M of wb-feasible merges in the current state (lines 3 and 19). The most attractive of these merges are then compiled into a list Ω (lines 5 – 17). One of the merges in Ω is finally selected and executed at the end of each iteration (line 18). To construct Ω, the merges in M are considered by decreasing savings value (based on the updated arc costs). The savings value s_m of a merge m corresponding to $r = r_1 \times r_2$ is computed as $s_m = c^{cur}_{0last(r_1)} + c^{cur}_{0first(r_2)} - c^{cur}_{last(r_1)first(r_2)}$.
If the current merge m would result in a route r s.t. $rc_r(\pi^{cur}) < 0$ (line 8) the route is tested for bb-feasibility, and if feasible is collected (line 11). Merges resulting in routes r s.t. $rc_r(\pi^{cur}) \geq 0$ as well as bb-infeasible merges are accepted into Ω by default. Routes that are bb-infeasible could be extended into bb-feasible routes (depending on constraints F_{bb} of course). In the same spirit routes of positive reduced cost could be extended into routes of negative reduced cost. The entire process is repeated until no further wb-feasible merges are possible given the current state S. Finally the feasible routes produced are post-optimized as explained in chapter 6 (line 22). (Of course the cost associated with the produced columns is computed based on the original arc costs.)

7.7 BRANCHING HEURISTIC

Different approaches are possible to order nodes in a Dive & Generate tree. At each non-leaf node SCP_i in the tree, child nodes are created. One child node SCP_r is created for each variable $x_r \in X_{\mathcal{R}_i^{*\prime}} \backslash \mathcal{F}_i$

Algorithm 7.6.1 : Dive & Generate - doPricing

```
Input : $\mathcal{F}, \pi^{cur}$
Inout : $\Psi$
Output : $collected$
1  initialize $S$ to state using shuffle routes visiting all customers in
   $V \backslash (\bigcup_{x_r \in \mathcal{F}_i} \{r.S\} \cup \{0\})$;
2  update arc costs w.r.t. $\pi^{cur}$;
3  $i \leftarrow 0$; $collected \leftarrow \emptyset$; $\Omega \leftarrow \emptyset$;
4  $M \leftarrow$ wbFeasibleMerges($S$);
5  while $M \neq \emptyset$ do
6      foreach $m \in M$ by - $s_m$ do
7          route $r \leftarrow$ getRouteFromMerge($m$);
8          if $rc_r(\pi^{cur}) < 0$ then
9              if $r \notin \Psi$ then $\Psi[r] \leftarrow feasbb(r)$;
10             if $\Psi[r]$ then
11                 $collected \leftarrow collected \cup r$;
12                 $i \leftarrow i + 1$;
13             end
14         end
15         if $i \geq \nu$ then break;
16         $\Omega \leftarrow \Omega \cup m$;
17     end
18     select $m \in \Omega$; $S \leftarrow$ executeMerge($m$, $S$);
19     $M \leftarrow$ wbFeasibleMerges($S$);
20     $i \leftarrow 0$; $\Omega \leftarrow \emptyset$;
21 end
22 $collected \leftarrow$ post-optimize($collected$, $\Psi$);
23 return $collected$
```

s.t. $\mathcal{A}_{sol_i}(x_r) > 0$, with $sol_i \in Sol^*(Relax(SCP(\mathcal{R}_i^{*\prime}, \mathcal{F}_i)))$. These child nodes are ordered in increasing order by the heuristic value $h(SCP_r)$.

Different possibilities to compute $h(SCP_r)$ exist:

- **Value-based ordering** The child nodes SCP_r are ordered by the value they take in the parent node solution, $h(SCP_r) = -\mathcal{A}_{sol_i}(x_r)$. This is the approach taken in [JMS+10].

- **Problem-specific ordering** The authors of [CRS09] propose problem-specific values for $h(SCP_r)$ for a complicated vehicle routing problem (to be used in a diving heuristic). While their computations are based on items to be delivered rather than on customers to be visited, they can be easily adapted to a classical Vehicle Routing Problem. They propose two problem-specific possible computations for $h(SCP_r)$:
  - $h(SCP_r) = \frac{distance(r)}{|V \setminus (\{0\} \cup \mathcal{F}_i) \cap r.S|}$
  - $h(SCP_r) = (1 - \mathcal{A}_{sol_i}(x_r)) \cdot \frac{distance(r)}{|V \setminus (\{0\} \cup \mathcal{F}_i) \cap r.S|}$

  Both of these heuristics weigh the distance of route r against the number of customers visited in r and not visited so far in the fixed routes.

7.8 SEARCH STRATEGY

The obvious search strategy for Branch & Generate-based methods is *Depth First Search* since the goal of the heuristic is to descend to leaf nodes as quickly as possible. Since however a bad branching decision at a low level in the Dive & Generate tree may lead to a lot of effort being lost in the exploration of a subtree without feasible leaf-node, *Limited Discrepancy Search* is a reasonable alternative.

Limited Discrepancy Search is a Discrepancy Search where a limit is imposed on the maximum number of allowed discrepancies. Thus, depending on the limit and the number of nodes in the search tree, not the entire tree is explored. It has also been decided to take discrepancies as early as possible in the search tree such as done in [PU11].
In Limited Discrepancy Search for Dive & Generate a new tree is constructed and explored for each value of allowed discrepancies. Thus once the tree corresponding to a number of allowed discrepancies d has been completely explored (or search is restarted as explained in section 7.4), the `strictGeneration(...)` scheme will be executed and construction of a new tree begins with the number of allowed discrepancies set to $d + 1$. Exploring a new tree also means that, when necessary, column generation will be executed at nodes in this new tree.
Let a node SCP_i^d in the tree with the number of allowed discrepancies set to d. With this node are associated a final set of routes $\mathcal{R}^{*\prime d}$ and a set of fixed routes $\mathcal{F}_i^d$. Now assume a node SCP_i^{d+1} in the tree

with the number of allowed discrepancies set to $d+1$. This node has associated a final set of routes $\mathcal{R}^{*\prime d+1}$ and a set of fixed routes $\mathcal{F}_i^{d+1}$. Assume that $\mathcal{F}_i^d = \mathcal{F}_i^{d+1}$. As set $\mathcal{R}^*$ is enriched throughout the exploration of each tree, we have $\mathcal{R}_i^{*\prime d} \subseteq \mathcal{R}_i^{*\prime d+1}$. Therefore it is possible that $sol_i^d \in Sol^*(Relax(SCP(\mathcal{R}_i^{*\prime d}, \mathcal{F}_i^d)))$ obtained at a node SCP_i^d is different from the solution $sol_i^{d+1} \in Sol^*(Relax(SCP(\mathcal{R}_i^{*\prime d+1}, \mathcal{F}_i^{d+1})))$ obtained at node SCP_i^{d+1}, even though $\mathcal{F}_i^d = \mathcal{F}_i^{d+1}$. Thus it is possible that the subtree of SCP_i^{d+1} will be fundamentally different from the subtree of SCP_i^d.

8 HEURISTIC BRANCH & PRICE

This chapter presents an approach combining Heuristic Column Generation with the heuristic exploration of a Branch & Price tree. To each node in this tree corresponds a Set Covering Problem formulated over a current set of feasible routes, as well as a set of route features that are forced or forbidden to appear in the solution to this restricted problem. This solution is then used to derive new subproblems. For each node two such subproblems are created by selecting a route feature in the current solution and then enforcing it in one subproblem and forbidding it in the other. Heuristic Column Generation is executed at all non-leaf nodes of the tree. The approach is *incomplete* and *constructive*.

The chapter presents itself as follows. First the basic principles of the Heuristic Branch & Price (HBP) method are presented in 8.1. This section provides the overall algorithm and the main design decisions to be taken are emphasized. The options chosen for these decisions are then explained in their respective sections. Section 8.2 elaborates on how the initial set of feasible routes is created. Section 8.3 gives an overview of the different actions and decisions taken during the exploration of a node in the Heuristic Branch & Price tree. Next, section 8.4 explains which kinds of features are considered for subproblem creation, and how the resulting constraints are enforced in the Set Covering Problem. The way the features are selected is detailed in section 8.5. The algorithms used to generate feasible routes are explained in section 8.6. Finally the search strategy used to explore the tree is covered in section 8.7.

8.1 PRINCIPLES

In practice when Branch & Price is applied to Vehicle Routing Problems the constraints added to an original problem $SCP(\mathcal{R}^*)$ in order to create subproblems, are not directly imposed on the variables in $X_{\mathcal{R}^*}$. The reasoning is that by selecting a variable $x_r \in X_{\mathcal{R}^*}$ and creating two subproblems, one with $x_r = 0$ and one with $x_r = 1$ (as is usually done in Branch & Bound) an unbalanced tree is created (see e.g. [Fei10]). This is because forbidding a route r (by fixing $x_r = 0$ ($x_r \in \mathcal{R}^*$)) often only very slightly restricts the number of feasible solutions to the problem. Also it is difficult to impose such a constraint in the pricing problem, the goal of which becomes then to generate all routes of negative reduced cost, except route r.
Branch & Price thus creates subproblems by selecting some route feature in the current fractional solution and creating one subproblem where this route feature is forced to appear in all feasible solutions, and one subproblem where this route feature is forbidden to appear in all feasible solutions. The same type of route feature is used throughout the construction of the heuristic Branch & Price tree.

In this section, a node i in the Branch & Price tree will be designated by SCP_i. With each node SCP_i are associated two sets of route features $\mathcal{Y}_i^-$ (forbidden route features) and $\mathcal{Y}_i^+$ (enforced route features) as well as an initial set $\mathcal{R}_i^*$ and a final set $\mathcal{R}_i^{*\prime}$ of routes. The associated problems are then called $SCP(\mathcal{R}_i^*, \mathcal{Y}_i^-, \mathcal{Y}_i^+)$ and $SCP(\mathcal{R}_i^{*\prime}, \mathcal{Y}_i^-, \mathcal{Y}_i^+)$ respectively. Heuristic Column Generation is executed at all non-leaf nodes, and thus for node SCP_i we have $\mathcal{R}_i^* \subseteq \mathcal{R}_i^{*\prime}$. Note that new feasible routes must be generated in such a way that they respect the enforced or forbidden route features.

When solving problem $SCP(\mathcal{R}_i^*, \mathcal{Y}_i^-, \mathcal{Y}_i^+)$ at node SCP_i the constraints $\mathcal{Y}_i^-$ and $\mathcal{Y}_i^+$ must be somehow enforced. To enforce or forbid route features, the Set Covering Problem is not modified directly. Rather, at each node SCP_i, routes not respecting the constraints corresponding to $\mathcal{Y}_i^-$ and $\mathcal{Y}_i^+$ are temporarily removed from $\mathcal{R}_i^*$.

Finally in the Heuristic Branch & Price tree, each non-leaf node will thus have two child nodes. One where a given route feature is enforced and one where the same feature is forbidden. With root node SCP_0 is associated problem $SCP(\mathcal{R}_0^*, \emptyset, \emptyset)$. Set $\mathcal{R}_0^*$ is initialized such that $Relax(SCP(\mathcal{R}_0^*, \emptyset, \emptyset))$ is feasible, the corresponding solution is fractional and such that $SPP(\mathcal{R}_0^*, \emptyset, \emptyset)$ is feasible. The solution
$sol \in Sol(SPP(\mathcal{R}_0^*, \emptyset, \emptyset))$ will be used to initialize a global upper bound UB.
Note that it is not strictly necessary to have a non-fractional solution at the root node in the current context, however imposing this allows to consider nodes SCP_i with an integral solution sol_{rel} as leaf nodes. ($sol_{rel} \in Sol^*(Relax(SCP(\mathcal{R}_i^*, \mathcal{Y}_i^-, \mathcal{Y}_i^+)))$).
Note that nodes SCP_i s.t. $Relax(SCP(\mathcal{R}_i^*, \mathcal{Y}_i^-, \mathcal{Y}_i^+))$ is infeasible, are considered as leaf nodes as well.

As for Dive & Generate, given a node SCP_i and the node SCP_{i+1} visited next, we have $\mathcal{R}_i^{*\prime} = \mathcal{R}_{i+1}^*$ and $\mathcal{R}_i^* \subseteq \mathcal{R}_i^{*\prime} \subseteq \mathcal{R}_{i+1}^* \subseteq \mathcal{R}_{i+1}^{*\prime}$ (i.e. same relationship as in the Dive & Generate tree). The same arguments for restarts as in the Dive & Generate approach could thus be made at this point. However Restarts are not included in Heuristic Branch & Price since the constraints imposed by fixing or forbidding route features are less restrictive than fixing a route. However once a tree has been completely explored, the exploration and construction of a new tree is started should the overall stopping criterion not have been reached yet.

Design Decisions

The following decisions must be taken when designing the heuristic Branch & Price approach for the CVRPBB:

1. How to initialize $\mathcal{R}^*$? (Initial set of routes, section 8.2)
2. How are nodes explored? (Node exploration, section 8.3)
   - when to prune?
3. Which type of route feature is used for branching? (Subproblem creation, section 8.4)
   - how is the set of routes adapted for compliance?

4. How are the features to branch on selected? (Branching heuristic, section 8.5)

5. How to generate new routes? (Pricing, section 8.6)

6. How to explore the Heuristic Branch& Price tree? (Search strategy, section 8.7)

These concerns are addressed in the indicated sections.

Algorithm

The high-level Heuristic Branch & Price algorithm is given in algorithm 8.1.1. A feasibility store Ψ is used to store the *bb*-feasibility information of tested routes.

First an initial set of feasible routes is generated (line 3) and the current and incumbent solutions, the dual costs associated with the current solution, as well as the upper bound UB are initialized, see section 8.2 on details on how this is done. Note that this initialization must ensure that $sol_0 \neq \perp$ and $sol_{inc} \neq \perp$. Lines 4 – 17 correspond to the construction and exploration of a Heuristic Branch & Price tree. First the dual costs π^0 are used to generate new feasible routes (line 7). These route are added to $\mathcal{R}^*$ and problem $Relax(SCP(\mathcal{R}^*, \emptyset, \emptyset))$ is solved, updating the current solution sol_0 and the associated dual costs π^0. If sol^0 is integral an attempt at updating the current best solution sol_{inc} and the global upper bound UB is made (the attempt succeeds if sol_0 can be fixed such that no customer is visited more than once) (line 8). These steps are repeated until sol_0 is fractional or the overall stopping criterion is reached.
Solution sol_0 is then used to select some feature f, appearing in one of its routes (line 11). Two child nodes are created, one where the use of f is forbidden (line 12) and one where f must be used (line 13). As $\mathcal{R}^*$ is global to the tree and the problem being solved is always a Set Covering Problem, it is sufficient to store the set of desired and undesired route features in the frontier. Next the nodes in the frontier are explored either until the entire tree has been explored, or the stopping criterion is reached (lines 14 – 17). The exploration process of a node is explained in detail in section 8.3. Finally once an entire tree has been explored feasible routes are generated at random, to enrich set $\mathcal{R}^*$ (line 19) and the root problem is solved once more (line 20).

Algorithm 8.1.1 : Heuristic Branch & Price

```
1  $Frontier, \mathcal{R}^*, \Psi \leftarrow \emptyset$;
2  $sol_0, sol_{inc} \leftarrow \bot$; $UB \leftarrow \infty$;
3  initialize $\mathcal{R}^*, sol_0, \pi^0, sol_{inc}$ and $UB$ // see section 8.2
4  while $\neg$ stopping criterion do
5      repeat
6          $\mathcal{R}^* \leftarrow \mathcal{R}^* \cup$ doPricing($\emptyset, \emptyset, \pi^0, \Psi$) // see section 7.6
7          $sol_0, \pi^0 \leftarrow solveLP(Relax(SCP(\mathcal{R}^*, \emptyset, \emptyset)))$;
8          if $isInteger(sol_0)$ then update($sol_0, sol_{inc}, UB, \Psi, \mathcal{R}^*$);
9      until $\neg isInteger(sol_0) \vee$ stopping criterion;
10     if stopping criterion then break;
11     Select feature $f$ in some route in $\{r | \mathcal{A}_{sol_0}(x_r) > 0\}$ // see
           sections 8.4 and 8.5
12     $Frontier \leftarrow Frontier \cup (\{f\}, \emptyset)$;
13     $Frontier \leftarrow Frontier \cup (\emptyset, \{f\})$;
14     while $Frontier \neq \emptyset \wedge \neg$ stopping criterion do
15         select $(\mathcal{Y}^-, \mathcal{Y}^+) \in Frontier$;
16             /* perform exploration of node $(\mathcal{R}^*, \mathcal{Y}^-, \mathcal{Y}^+)$, see
           section 8.3 */
17     end
18     if stopping criterion then break;
19     $\mathcal{R}^* \leftarrow \mathcal{R}^* \cup$ strictGeneration($\Psi$) // see section 8.6
20     $sol_0, \pi^0 \leftarrow solveLP(Relax(SCP(\mathcal{R}^*, \emptyset, \emptyset)))$;
21 end
22 return $\{r | \mathcal{A}_{sol_{inc}}(x_r) = 1\}$;
```

8.2 INITIALIZATION OF $\mathcal{R}^*$

The method used to initialize $\mathcal{R}^*$ is exactly the same as the one used in Pheromone-based Heuristic Column Generation (chapter 6) and Dive & Generate (chapter 7). A number ℓ of collector ants (no pheromones are used) in *strict* mode (see 6.2.1) is repeatedly executed to accumulate routes in $\mathcal{R}^*$ (no pheromones are used) which are then post-optimized.

Problems $Relax(SCP(\mathcal{R}^*,\emptyset,\emptyset))$ and $SCP(\mathcal{R}^*,\emptyset,\emptyset)$ are then solved and the resulting solutions are stored in sol_0 and sol_{inc}. These steps are repeated either until the overall stopping criterion is met, or until sol_0 and sol_{inc} are feasible and sol_0 is fractional. Solution sol_{inc} is fixed if necessary which allows to initialize UB with the cost of sol_{inc}.

8.3 NODE EXPLORATION

Again a decision scheme telling us when to prune a node is needed. Let $\mathcal{R}_i^*$ the initial set associated with node SCP_i.
If $Relax(SCP(\mathcal{R}_i^*,\mathcal{Y}_i^-,\mathcal{Y}_i^+))$ is infeasible, node SCP_i is a leaf node. The same holds if solution $sol_i \in Sol^*(Relax(SCP(\mathcal{R}_i^*,\mathcal{Y}_i^-,\mathcal{Y}_i^+)))$ is integral. If the problem is feasible and the solution fractional, column generation is executed.
In column generation, the dual costs π^i associated with sol_i are used in the generation of feasible routes respecting also the constraints corresponding to $\mathcal{Y}_i^-$ and $\mathcal{Y}_i^+$. These new routes are added to $\mathcal{R}_i^*$ to result in the final set $\mathcal{R}_i^{*\prime}$. If then, the new lower bound $LB_i' = Obj(sol_i')$ with $(sol_i' \in Sol^*(Relax(SCP(\mathcal{R}_i^{*\prime},\mathcal{Y}_i^-,\mathcal{Y}_i^+))))$ is greater or equal to the global upper bound ($LB_i' \geq UB$), node SCP_i is pruned. This means, it is assumed that LB_i' is a reasonable approximation of $Obj(Relax(SCP(\mathcal{R},\mathcal{Y}_i^-,\mathcal{Y}_i^+)))$ and that no solution improving UB will be found in the subtree of SCP_i. Again if sol_i' is integer, SCP_i is considered a leaf node.

The complete non-root node exploration algorithm is given in 8.3.1. It takes number of inputs. Some of these are modified throughout the node exploration.

- $\mathcal{R}^*$ the current node's initial set of feasible routes
- $\mathcal{Y}^-$ and $\mathcal{Y}^+$ the set of features forbidden and enforced in the current node
- sol_{inc} the incumbent solution
- UB the global upper bound
- *Frontier* the frontier of unexplored nodes
- Ψ the feasibility store

First a temporary set $\mathcal{R}^*_{temp}$ is created by removing from the current set of routes $\mathcal{R}^*$ the ones that do not comply with this node's constraints (line 2) (see section 8.4 on how this is done). Then the LP relaxation of the subproblem corresponding to the current node is solved resulting in solution sol_{cur} and dual costs π^{cur} (line 3). If the subproblem turns out to be infeasible (given the current $\mathcal{R}^*_{temp}$) we have reached a leaf node (line 4). If solution sol_{cur} is integral and improves the upper bound UB, then the latter and the incumbent solution may possibly be updated (lines 5 – 6) and the node is a leaf node (line 7).
Next, feasible routes r s.t. $rc_r(\pi^{cur}) < 0$ (and r feasible w.r.t. $\mathcal{Y}^-$ and $\mathcal{Y}^+$) should be generated (line 10). The generated routes are inserted into $\mathcal{R}^*$ and $\mathcal{R}^*_{temp}$ (lines 11 and 12). Then the LP relaxation of the current subproblem, formulated on the (possibly) augmented $\mathcal{R}^*_{temp}$, is solved, updating sol_{cur} (line 13). If the resulting lower bound exceeds the global upper bound the node is pruned (line 18). Finally the solution sol_{cur} is used to compute the child nodes which are added to the frontier. To do this a feature f is selected (line 19). The child nodes corresponding to the subproblems resulting from forbidding and enforcing f are then created, and added to the frontier (lines 20 and 21).

8.4 SUBPROBLEM CREATION

One single type of route feature is used to branch on in the entire heuristic Branch & Price tree. In this thesis two types of features are considered: customer pairs and arcs. Both are classical ways to create subproblems in the context of routing problems [BJN+94].

8.4.1 *Branching on customer pairs*

Here two customers i and j ($i, j \in V \backslash \{0\}$) are selected, and forced to be visited in different routes on one subproblem, and forced to be visited in a same route in the other subproblem. Thus sets $\mathcal{Y}^-$ and $\mathcal{Y}^+$ correspond to sets of customers pairs.

Algorithm 8.3.1 : Heuristic Branch & Price - explore Node

```
Input : $\mathcal{Y}^-, \mathcal{Y}^+$
Inout : $\mathcal{R}^*, Frontier, \Psi, sol_{inc}, UB$
1  $\mathcal{L} \leftarrow \emptyset$;
2  $\mathcal{R}^*_{temp} \leftarrow \mathcal{R}^* \setminus noncomply(\mathcal{Y}^-, \mathcal{Y}^+, \mathcal{R}^*)$;
3  $sol_{cur}, \pi^{cur} \leftarrow solveLP(Relax(SCP(\mathcal{R}^*_{temp})))$;
4  if $sol_{cur} = \perp$ then return;
5  if $isInteger(sol_{cur}) \wedge Obj(sol_{cur}) < UB$ then
6      update($sol_{inc}, UB, \Psi, \mathcal{R}^*$);
7      return;
8  end
9  $LB_{cur} \leftarrow Obj(sol_{cur})$;
10 $\mathcal{L} \leftarrow$ doPricing($\mathcal{Y}^-, \mathcal{Y}^+, \pi^{cur}, \Psi$) // see section 8.6
11 $\mathcal{R}^* \leftarrow \mathcal{R}^* \cup \mathcal{L}$;
12 $\mathcal{R}^*_{temp} \leftarrow \mathcal{R}^*_{temp} \cup \mathcal{L}$;
13 $sol_{cur} \leftarrow solveLP(Relax(SCP(\mathcal{R}^*_{temp})))$;
14 if $isInteger(sol_{cur}) \wedge Obj(sol_{cur}) < UB$ then
15     update($sol_{inc}, UB, \Psi, \mathcal{R}^*$);
16     return;
17 end
18 if $LB_{cur} \geq UB$ then return;
19 Select feature $f$ in some route in $\{r | \mathcal{A}_{sol_{cur}}(x_r) > 0\}$ // see
       sections 8.4 and 8.5
20 $Frontier \leftarrow Frontier \cup (\mathcal{Y}^- \cup \{f\}, \mathcal{Y}^+)$;
21 $Frontier \leftarrow Frontier \cup (\mathcal{Y}^-, \mathcal{Y}^+ \cup \{f\})$;
22 return;
```

Enforcing constraints

As previously seen, the constraints corresponding to $\mathcal{Y}^-$ and $\mathcal{Y}^+$ are enforced by temporarily removing routes not respecting these constraints from the current set of routes $\mathcal{R}^*$. Such routes are collected in a set by function $noncomply(\mathcal{Y}^-,\mathcal{Y}^+,\mathcal{R}^*) = noncomply(\mathcal{Y}^-,\mathcal{R}^*) \cup noncomply(\mathcal{Y}^+,\mathcal{R}^*)$. If $\mathcal{Y}^-$ and $\mathcal{Y}^+$ correspond to sets of customer pairs then the routes to be removed correspond to:

- $noncomply(\mathcal{Y}^-,\mathcal{R}^*) = \bigcup_{\{i,j\}\in\mathcal{Y}^-}\{r \in \mathcal{R}^* | i \in r.S \wedge j \in r.S\}$
  all routes visiting both i and j must be removed
- $noncomply(\mathcal{Y}^+,\mathcal{R}^*) = \bigcup_{\{i,j\}\in\mathcal{Y}^+}\{r \in \mathcal{R}^* | (i \in r.S \oplus j \in r.S)\}$
  all routes visiting either i or j but not both must be removed

8.4.2 *Branching on arcs*

Here an arc $(i,j) \in A$ is selected and is forbidden to appear in routes in one subproblem, and must appear in some route in the other subproblem. Thus sets $\mathcal{Y}^-$ and $\mathcal{Y}^+$ correspond to sets of arcs.

Enforcing constraints

The constraints corresponding to $\mathcal{Y}^-$ and $\mathcal{Y}^+$ are enforced by temporarily removing routes not respecting these constraints from the current set of routes $\mathcal{R}^*$. This is done using function
$noncomply(\mathcal{Y}^-,\mathcal{Y}^+,\mathcal{R}^*) = noncomply(\mathcal{Y}^-,\mathcal{R}^*) \cup noncomply(\mathcal{Y}^+,\mathcal{R}^*)$.
If $\mathcal{Y}^-$ and $\mathcal{Y}^+$ correspond to sets of arcs then the routes to be removed correspond to:

- $noncomply(\mathcal{Y}^-,\mathcal{R}^*) = \bigcup_{(i,j)\in\mathcal{Y}^-}\{r \in \mathcal{R}^* | (\exists p = 0 \ldots |r.S| \text{ s.t. } r[p] = i \wedge r[p+1] = j)$
  all routes visiting j immediately after i must be removed
- $noncomply(\mathcal{Y}^+,\mathcal{R}^*) =$
  $\bigcup_{(i,j)\in\mathcal{Y}^+ \text{s.t.} i\neq 0}\{r \in \mathcal{R}^* | (\exists p = 1 \ldots |r.S| \text{ s.t. } r[p] = i \wedge r[p+1] \neq j)\} \cup$
  $\bigcup_{(i,j)\in\mathcal{Y}^+ \text{s.t.} j\neq 0}\{r \in \mathcal{R}^* | (\exists p = 1 \ldots |r.S| \text{ s.t. } r[p] = j \wedge r[p-1] \neq i)\}$
  all routes visiting j not immediately after i and i not immediately before j must be removed (special case if i or j is the depot)

8.5 BRANCHING HEURISTICS

Different approaches are possible to select the feature to branch on at a non-leaf node SCP_i. They depend obviously on the type of feature used.

8.5.1 *Customer Pairs*

The idea given in [VBJN94] is used.
Let $sol_i \in Sol^*(Relax(SCP(R_i^{*\prime}, \mathcal{Y}_i^-, \mathcal{Y}_i^+)))$. The idea is then to select a pair $\{i, j\}$ $(i, j \in V \backslash \{0\})$ such that $\exists x_r \in X_{R_i^{*\prime}}$ s.t. $0 < \mathcal{A}_{sol_i}(x_r) < 1 \wedge i, j \in r.S$ and $\exists x_{r'} \in X_{R_i^{*\prime}}$ s.t. $0 < \mathcal{A}_{sol_i}(x_{r'}) < 1 \wedge i \in r.S \wedge j \notin r.S$. That is a pair of customers is selected such that in the current fractional solution there exists a route visiting both customers, and there exists a route visiting only one of the two. For each pair of customers $\{i, j\}$ value f_{ij} is computed as $\sum_{r \in R_i^{*\prime} s.t. i,j \in r.S} x_r$ and the pair minimizing $|f_{ij} - 0.5|$ is selected to branch on.

Alternatively, information from F_{wb} could be used as a branching heuristic. For the CVRPBB considered here, where F_{wb} corresponds to capacity constraints, the heuristic employed would be to select the pair $\{i, j\}$ maximizing the sum of the demands $q_i + q_j$. The reasoning is that these are the customers with the biggest impact (as far as can be told given the black box nature of $\mathcal{F}_{bb}$) since forcing them into one route leaves only a limited amount of vehicle capacity in that route. This approach has however been discarded due to the fact that the same pair of customers tend to be selected even over multiple restarts.

The node forcing a pair of customers is considered as the left-most node. The reasoning is that the decision of forcing a pair of customers to be visited in a same route has a stronger impact than forbidding this, and may lead to a leaf-node faster.

8.5.2 *Arcs*

Arcs are selected based on the current solution sol_i
$(sol_i \in Sol^*(Relax(SCP(R_i^{*\prime}, \mathcal{Y}_i^-, \mathcal{Y}_i^+))))$ using the methodology for arc selection explained in [Fei10].

Let $W_{ij} = \{r | 0 < \mathcal{A}_{sol_i}(x_r) < 1 \wedge \exists\, a \text{ s.t. } 0 \leq a \leq |r.S| \wedge r[a] = i \wedge r[a+1] = j\}$. An arc (i,j) is then selected such that $f_{ij} = \sum_{r \in W_{ij}} x_r$ and $0 < f_{ij} < 1$. That is, an arc (i,j) is chosen such that in the current solution we can find a route employing this arc, but we can also find a route employing arc (i,t) with $t \neq j$. Since there might be multiple candidate arcs, the arc (i,j) s.t. $|f_{ij} - 0.5|$ is minimized is chosen. Again the node enforcing arc (i,j) is considered the left-most node with the same reasoning as given previously for customer pairs.

8.6 GENERATING FEASIBLE ROUTES

Again two different schemes are used for generating new feasible routes. The `strictGeneration(...)` scheme corresponds to the execution of ℓ collector ants (no pheromones are used) in *strict* mode (see algorithm 6.2.1) and the post-optimization of the generated feasible routes.

The second scheme (`doPricing(...)`) is used to generate feasible routes of negative reduced cost at different nodes SCP_i in the Branch & Price tree. These routes must be feasible with respect to the route features forced or forbidden at the current node. The way this is handled depends on the route feature. Again ℓ executions of a randomized savings heuristic are used. Feasibility w.r.t. $\mathcal{F}_{bb}$ is tested only for routes r s.t. $rc_r(\pi^i) < 0$.

8.6.1 *Customer Pairs*

As seen before, the branching decisions correspond to forcing a pair of customers to be visited in the same route in one branch, and forcing them to be visited in different routes in the second branch. If a pair of customers is forbidden in a same route, this can be simply enforced by considering as infeasible the merges resulting in a route where both customers are visited. On the other hand, forcing a pair of customers to be visited in a same route is not as straight-forward. The fact that a pair of customers must be visited in a same route does not imply that the customers are visited in sequence. Thus there is no direct way to force the savings heuristic to produce such routes. The idea is thus to "fix" produced routes not respecting this type of constraints. Thus if the heuristic produces a route (*wb*-feasible, *bb*-feasible, and respecting constraints derived from $\mathcal{Y}^-$) visiting only one customer out of a forced customer pair, the routes will be fixed by trying to insert the remaining customer(s). The respective algorithms are given below.

Algorithm 8.6.1 describes how the randomized savings heuristic is used to generate *bb*-feasible routes of negative reduced cost and respecting the constraints corresponding to $\mathcal{Y}^-$ and $\mathcal{Y}^+$. The algorithm starts by creating a state where each customer is visited in a route of its own (line 1). Then the arc costs are updated w.r.t. the current dual costs as was described in section 7.6 (line 2). Next the set of *wb*-feasible merges is created (line 4). This set is then filtered to remove merges implying routes r_1 and r_2 such that $\exists i \in r_1.S, j \in r_2.S$ s.t. $\{i, j\} \in \mathcal{Y}^-$ (line 5). The remaining merges are then considered by decreasing savings value (given the updated arc costs) (line 7) to construct candidate list Ω. For each considered merge the reduced cost of the resulting route is computed (line 9). Should this reduced cost be found to be negative the *bb*-feasibility of the route is checked and if *bb*-feasible the route is stored (line 10). The process stops once ν *bb*-feasible merges have been introduced into candidate list Ω, or once all merges have been considered. All considered merges may enter the candidate list. The reasoning is that two routes of positive reduced cost (or a feasible and a *bb*-infeasible route) could be combined to a negative reduced cost route (and/or a *bb*-feasible route). Finally one merge in Ω is selected and executed (line 19) and the set of possible merges is recomputed (lines 20 and 21). The process stops once no further possible merges exist. At this point the set of collected *bb*-feasible routes must be analyzed, and some routes must possibly be fixed (line 23). Finally the feasible routes are post-optimized using either the TS (see algorithm 6.4.1) or ILS (see algorithm 6.4.4) post-optimization method (line 25).

The algorithm used for fixing routes is given in 8.6.2. Each route is considered individually (line 2). Then the set of customers that should be visited, but aren't is computed (line 3). Note that a customer j missing from route r may imply further missing customers. The missing customers are then considered one by one (line 5). Each customer is inserted in the best position given the (modified) arc costs that results in a feasible route w.r.t. F_{wb}. If all missing customers have been inserted, and if the resulting route r is of negative reduced cost (line 9) its feasibility w.r.t. F_{bb} is checked. If it is feasible, it is inserted into the collection of routes to return (line 12). Finally the set of accumulated routes is returned (line 16).

Two things should be pointed out. First the algorithm fixing routes might fail at fixing a route even though it could have been fixed. One could imagine a set of constraints F_{bb} where the order in which customers are inserted is important, or where several customers have to be inserted at once to result in a *bb*-feasible route.
Second, as the savings heuristic produces new routes by merging two routes, and as feasible routes are stored at each heuristic step, the routes produced will have very different lengths. Thus the routes to be fixed by inserting missing customers will have different lengths as well. It might be complicated for longer routes (or shorter depending on F_{bb}) to retain *bb*-feasibility while inserting further customers. However as there will also be shorter (or longer) routes to be fixed, the probability is high that some *bb*-feasible routes will result from this procedure.

8.6.2 *Arc*

Here the branching decisions correspond to forcing and forbidding an arc (i, j) to appear in feasible routes. The case where either $i = 0$ or $j = 0$ must be handled specifically.

Let arc (i, j) with $i, j \in V \backslash \{0\}$. Forcing the arc is simple. Instead of initializing the heuristic with an initial state with a separate route for i and j, one initial route $0 - i - j - 0$ is created. Forbidding the arc can be done by not considering any merge introducing (i, j) as part of the possible set of merges.
Now consider arc $(0, i)$. Forcing this arc is done by simply forbidding any merges introducing an arc (j, i) ($j \in V \backslash \{0\}$). On the other hand forbidding the arc is more complicated, since initially arc $(0, i)$ will be part of i's shuffle route. The idea is then to set the cost of arc $(0, i)$ to ∞, in order to guide the heuristic to execute merges removing $(0, i)$. Routes where arc $(0, i)$ is used are furthermore not checked for *bb*-feasibilty. The same approach is taken with forbidden arcs of the form $(i, 0)$.

Algorithm 8.6.3 describes how the randomized savings heuristic is used to generate *bb*-feasible routes of negative reduced cost when branching is done on customer arcs. First the current state is initial-

Algorithm 8.6.1 : Heuristic B&P - doPricing - branching on pairs

```
Input : $\mathcal{Y}^-, \mathcal{Y}^+, \pi_{cur}$
Inout : $\Psi$
Output : $collected$
1  initialize $S$ to state using shuffle routes visiting all customers;
2  update arc costs w.r.t. $\pi^{cur}$;
3  $i \leftarrow 0$; $collected \leftarrow \emptyset$; $\Omega \leftarrow \emptyset$;
4  M $\leftarrow$ wbFeasibleMerges($S$);
5  M $\leftarrow$ filter($\mathcal{Y}^-$,$M$);
6  while $M \neq \emptyset$ do
7      foreach $m \in M$ by - $s_m$ do
8          route $r \leftarrow$ getRouteFromMerge($m$);
9          if $rc_r(\pi^{cur}) < 0$ then
10             if $r \notin \Psi$ then $\Psi[r] \leftarrow feasbb(r)$;
11             if $\Psi[r]$ then
12                 $collected \leftarrow collected \cup r$;
13                 $i \leftarrow i + 1$;
14             end
15         end
16         if $i \geq v$ then break;
17         $\Omega \leftarrow \Omega \cup m$;
18     end
19     select $m \in \Omega$; $St \leftarrow$ executeMerge($m$, $S$);
20     M $\leftarrow$ wbFeasibleMerges($S$);
21     M $\leftarrow$ filter($\mathcal{Y}^-$,$M$);
22     $i \leftarrow 0$; $\Omega \leftarrow \emptyset$;
23 end
24 $collected \leftarrow$ fixInfeasible($collected$, $\mathcal{Y}^-, \mathcal{Y}^+, \pi^{cur}, \Psi$);
25 $collected \leftarrow$ post-optimize($collected$, $\Psi$);
26 return $collected$
```

ized with routes as described above (line 1). Then the cost of forbidden depot arcs is set to infinite (lines 2 and 3). Next the set of *wb*-feasible merges is computed (line 5). This set is then filtered to remove merges adding forbidden arcs or removing an enforced arc outgoing from the depot (line 6). Next the possible merges are considered by increasing savings value (line 8) to construct candidate list Ω. If the route does not use any forbidden arcs (incident to the depot) and if it has a

Algorithm 8.6.2 : Heuristic B&P - Fix routes w.r.t. pairs in $\mathcal{Y}^+$

```
   Input : $collected, \mathcal{Y}^-, \mathcal{Y}^+, \pi^{cur}, \Psi$
   Output : $fixed$
1  $fixed \leftarrow \emptyset$;
2  foreach $r \in collected$ do
3      $missing \leftarrow$ getMissingCustomers($r, \mathcal{Y}^+$);
4      $r' \leftarrow r$;
5      foreach $v \in missing$ do
6          select $pos \in 1, \dots, |r.S| + 1$ such that
           $\bigwedge_{c \in \mathcal{F}_{wb}} feas(c, insert(v, r', pos))$;
7          $r' \leftarrow insert(v, r', pos)$;
8      end
9      if $rc_{r'}(\pi^{cur}) < 0$ then
10         if $\bigwedge_{c \in \mathcal{F}_{wb}} feas(c, r')$ then
11             if $r' \notin \Psi$ then $\Psi[r'] \leftarrow feasbb(r')$;
12             if $\Psi[r']$ then $fixed \leftarrow fixed \cup r'$;
13         end
14     end
15 end
16 return $fixed$
```

negative reduced cost, its *bb*-feasibility is checked (lines 10 – 13). The process stops once ν *bb*-feasible merges have been introduced into candidate list Ω, or once all merges have been considered. All considered merges may enter the candidate list. The reasoning is that two routes of positive reduced cost (or a feasible and a *bb*-infeasible route) could be combined to a negative reduced cost route (and/or a *bb*-feasible route). Finally one merge in Ω is selected and executed (line 22) and the set of possible merges is recomputed (lines 23 and 24). The process stops once no further possible merges exist. Finally the feasible routes are post-optimized using either the TS (see algorithm 6.4.1) or ILS (see algorithm 6.4.4) post-optimization method (line 27). Note that the neighborhoods used need to be adapted in order to respect enforced and forbidden arcs.

Algorithm 8.6.3 : Heuristic B&P - doPricing - branching on arcs

```
Input : $\mathcal{Y}^-, \mathcal{Y}^+, \pi^{cur}, \Psi$
Output : $collected$
1  initialize $S$ to state with shuffle routes and non-shuffle routes
   using arcs in $\mathcal{Y}^+$;
2  foreach $(0, j) \in \mathcal{Y}^-$ do $c_{0j} \leftarrow \infty$;
3  foreach $(i, 0) \in \mathcal{Y}^-$ do $c_{i0} \leftarrow \infty$;
4  $i \leftarrow 0$; $collected \leftarrow \emptyset$; $\Omega \leftarrow \emptyset$;
5  M $\leftarrow$ wbFeasibleMerges($S$);
6  M $\leftarrow$ filter($\mathcal{Y}^-$,$\mathcal{Y}^+$,$M$);
7  while $M \neq \emptyset$ do
8      foreach $m \in M$ by - $s_m$ do
9          route $r \leftarrow$ getRouteFromMerge($m$);
10         if $\neg(\{0, r[1]\} \in \mathcal{Y}^- \vee \{r[|r.S|], 0\} \in \mathcal{Y}^-)$ then
11             if $rc_r(\pi^{cur}) < 0$ then
12                 if $r \notin \Psi$ then $\Psi[r] \leftarrow feasbb(r)$;
13                 if $\Psi[r]$ then
14                     $collected \leftarrow collected \cup r$;
15                     $i \leftarrow i + 1$;
16                 end
17             end
18         end
19         if $i \geq v$ then break;
20         $\Omega \leftarrow \Omega \cup m$;
21     end
22     select $m \in \Omega$; $St \leftarrow$ executeMerge($m$, $S$);
23     M $\leftarrow$ wbFeasibleMerges($S$);
24     M $\leftarrow$ filter($\mathcal{Y}^-$,$\mathcal{Y}^+$,$M$);
25     $i \leftarrow 0$; $\Omega \leftarrow \emptyset$;
26 end
27 $collected \leftarrow$ post-optimize($collected$, $\Psi$);
28 return $collected$
```

8.7 SEARCH STRATEGY

Both *Depth First Search* and *Limited Discrepancy Search* are considered for exploring the heuristic Branch & Price tree. Note that a *Best First* strategy might give good results too. It was however discarded due to its memory requirements. In the case of Limited Discrepancy Search discrepancies are taken as early as possible in the search tree. The same method of creating a new tree for each number of allowed discrepancies, such as done for Dive & Generate, is used. The explanations given in section 7.8 hold here as well.

9 DECOMPOSITION-BASED APPROACH

This chapter proposes a method that can be employed on top of any of the previously presented approaches. The idea is to randomly decompose the problem instance into smaller instances that are solved individually. The feasible routes identified during this process are then used to enrich an overall set of feasible routes $\mathcal{R}^*$.

9.1 PRINCIPLES

The idea to decompose a given problem instance into several detached problem instances is not new, see e.g. [Tai93]. The decomposition scheme used here is based on the one proposed in [BVH07] for the VRP with Time Windows. The authors propose to start from a feasible solution Sol_{init} (corresponding to a set of routes). Then a set of routes $T \subseteq Sol_{init}$ is selected, and a new problem instance is created where the goal is to find a high-quality solution Sol_d visiting all the customers in $\bigcup_{r \in T}\{r.S\}$, using at most $|T|$ vehicles. Once such a solution Sol_d is found, the original routes are replaced with the routes in Sol_d to create a new solution Sol' ($Sol' = (Sol_{init} \backslash T) \cup Sol_d$).

This latter decomposition scheme can be easily adapted to the CVRPBB. The idea is to start from an initial set of feasible routes $\mathcal{R}_i^*$ and an initial solution $sol_i \in SCP(\mathcal{R}_i^*)$ s.t. $sol_i \neq \perp$. As in [BVH07] a subset of routes T is selected based on sol_i and a new problem instance is created from T. Next, either ACO-HCG, DING or HBP are used to solve the resulting problem instance. At the end of this process, feasible routes for the new problem instance have been generated, and they are then added to $\mathcal{R}_i^*$ resulting in $\mathcal{R}_{i+1}^*$. A new solution $sol_{i+1} \in SCP(\mathcal{R}_{i+1}^*)$ is then computed to figure as initial solution of the next iteration.

Note that the steps consisting in the selection of set T, the creation of a new problem instance from set T and the resolution of the resulting problem can be seen as steps of a Large Neighborhood Search (LNS) in the broad sense. A part of the current solution (restricted to a set of routes) is destroyed, freeing a set of customers, and the resulting neighborhood is explored either using ACO-HCG, DING or HBP (in the case of [BVH07] the neighborhood is explored using LNS itself). However here, contrary to Large Neighborhood Search, the goal of the repair phase is not to repair the destroyed solution, but rather to generate new feasible routes visiting the free customers. Given the employed methods both augment to the same. The updated overall solution is thus not obtained by repairing the initial solution, but rather by resolving the Set Covering Problem on the augmented set of feasible routes.

The algorithm is given in 9.1.1. First the set $\mathcal{R}^*$ is initialized in such a way that problem $SCP(\mathcal{R}^*)$ is feasible (line 3). This problem is then solved, producing solution sol (line 5). A set of routes T is extracted from this solution (line 6) and a new problem graph is created. In this new problem graph the set of vertexes corresponds to the customers visited in the routes in set T and the depot. The set of arcs is reduced to arcs incident to these same vertexes (lines 7 – 9). Next a new problem, defined on this reduced graph, and with the number of vehicles reduced to $|K| = |T|$ is created (line 10). This problem is then solved (line 11). Here any of the approaches described in chapters 6,7,8 can be used. However instead of returning the solution found using these approaches, the set of feasible routes accumulated during the resolution process is returned. These routes are then added to the set of routes $\mathcal{R}^*$ and the resulting problem is solved the next iteration. The process is repeated until some stopping criterion is reached. To obtain a final solution, problem $SCP(\mathcal{R}^*)$ is solved and the routes chosen in this solution are extracted (and possibly fixed as explained in chapter

6) (lines 14 –15).

Algorithm 9.1.1 : Decomposition for the VRPBB

```
Input : CVRPBB defined on G=(V,A)
1  $\mathcal{R}^*, \Psi, \mathcal{L}, T, V', A' \leftarrow \emptyset$;
2  $sol \leftarrow \perp$;
3  initialize $\mathcal{R}^*$ s.t. $SCP(\mathcal{R}^*)$ is feasible;
4  while $\neg$ stopping criterion do
5      $sol \leftarrow solveIP(SCP(\mathcal{R}^*))$;
6      select set $T \subseteq \{r | \mathcal{A}_{sol_{init}}(x_r) = 1\}$;
7      $V' \leftarrow \bigcup_{r \in T}\{r.S\} \cup \{0\}$;
8      $A' \leftarrow V' \times V'$;
9      create problem graph $G' = (V', A')$;
10     create problem CVRPBB(G',|T|);
11     $\mathcal{L} \leftarrow solve(CVRPBB(G', |T|), \Psi)$;
12     $\mathcal{R}^* \leftarrow \mathcal{R}^* \cup \mathcal{L}$;
13 end
14 $sol \leftarrow solveIP(SCP(\mathcal{R}^*))$;
15 $\mathcal{R}_{sol} \leftarrow$ extractSolution($sol, \mathcal{R}^*, \Psi$);
16 return $\mathcal{R}_{sol}$
```

Two design choices remain to be made:

- How is the initial set of routes generated? (Initial set of routes, section 9.2)
- How is set T selected? (Subproblem creation, section 9.3)

9.2 INITIALIZATION OF $\mathcal{R}^*$

The method used to initialize $\mathcal{R}^*$ is exactly the same that is used in Pheromone-based Heuristic Column Generation (section 6). A number ℓ of collector ants in *strict* mode (see 6.2.1) is repeatedly executed to accumulate routes in $\mathcal{R}^*$ and problem $SCP(\mathcal{R}^*)$ is solved after each execution. As soon as a feasible solution to the problem can be found the process is stopped.

9.3 SELECTION OF SET OF ROUTES T

The approach described in [BVH07] is used. As in the Sweep heuristic the polar coordinates (θ, ρ) of customers w.r.t. the depot are computed. A wedge W described by a pair of angles α and β is selected. A customer i is said to belong to wedge W if $\theta_i \in [\alpha, \beta]$ (or $\theta_i \in [\alpha, 359] \cup [0, \beta]$ iff $\alpha > \beta$). Once a wedge has been selected, all customers belonging to this wedge are collected in set C_W. The set of routes T is then extracted from the current solution sol using C_W as follows: $T = \{r | \mathcal{A}_{sol}(x_r) = 1 \wedge C_W \cap r.S \neq \emptyset\}$. The principle is depicted in Figure 9.1.
To select W angle α is chosen randomly, angle β is then chosen as the smallest angle, s.t. the number of customers in the corresponding set T is greater or equal to parameter N.

Due to the combinatorial nature of the problem it is possible that good solutions may include routes where distant customers are visited in a same route. If no such route appears in the initial solution sol no such routes will ever be produced due to the way customers are selected based on spatial proximity. In order to alleviate this, a diversification mechanism is employed. At each iteration, if the best solution so far has not improved, the number of wedges $\#wed$ is increased by one. Each of the wedges will have a size greater or equal to N. Set C_W then corresponds to the set of customers belonging to all $\#wed$ wedges. As soon as the best solution has improved $\#wed$ is set back to 1.

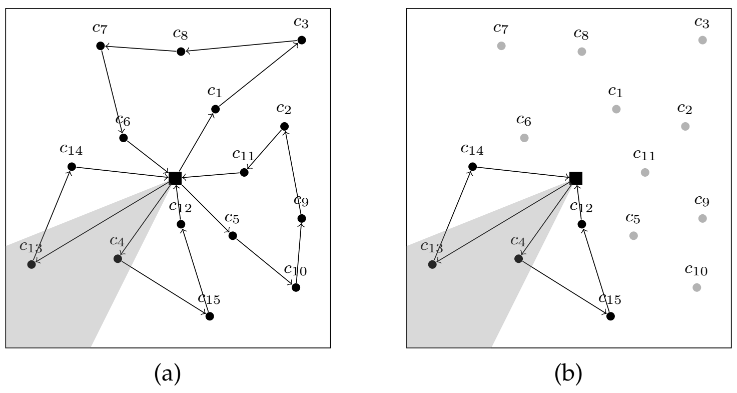


Figure 9.1: A wedge selected in a CVRPBB solution (a). The routes (and their customers) extracted using this wedge are shown in (b).

10 IMPLEMENTATION

The approaches presented in Part II of this thesis have been implemented as a coherent system in C++ which can be downloaded under the LGPL license at `http://becool.info.ucl.ac.be/resources/VRPBB`. The approaches have all been implemented in such a way that they can be considered as the construction and exploration of a Branch & Bound tree. The implementation follows the object-oriented design principle. The proposed approaches are included in the system via the template design pattern. This means they are implemented by overriding the functions provided in different partially or completely abstract classes. The approach-specific functions are then automatically called by the system (following the Hollywood principle). The most important high-level classes are depicted, regrouped in packages by responsibility in Fig. 10.1. The links between packages and classes indicate the communication streams.

In the following, high-level classes of the system and approach-independent implementations of different parts of the system are described.

10.1 BRANCH & BOUND TREE

A node in the B&B Tree corresponds to an instance of the `BBTreenode` class. Upon the visit of a node three actions are performed:

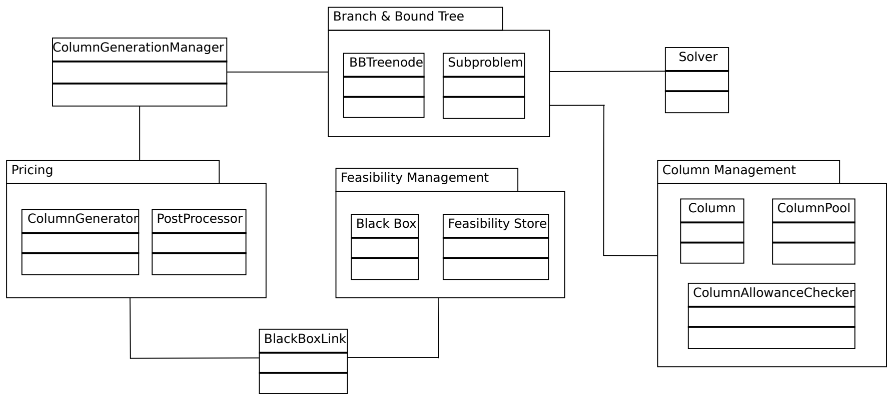


Figure 10.1: Visualization of the most important classes and their responsibilities. Links indicate communication streams.

- Opening action: The subproblem associated with the current node is solved and the current solution is updated accordingly.
- Visiting action: The decision whether to execute column generation is taken (and column generation is possibly executed). Furthermore the decisions whether to backtrack up to the parent node, to backtrack up to the root node, or to compute the children of the current node is taken.
- Closing action: The closing action is empty in the class `BBTreenode`.

The exploration of the subtree of a given node is implemented in class `DFSTreenode`, which is the base class (i.e. superclass) of class `BBTreenode`. It takes the list of computed child nodes and explores them in order, by performing the *opening*, *visiting* actions, possibly the exploration of the subtree, and the *closing* action. Note that this class could be modified easily to ensure a non-Depth First Search-based tree exploration.

Each instance of `BBTreenode` is associated with an instance of the `Subproblem` class. The decision to separate treenode and subproblem in different classes was made in order not to limit the possible uses of the system.

As previously explained the approaches proposed in part II of this thesis are implemented by deriving the (i.e. inheriting from or subclassing) basic system classes and overriding their default behavior. Derived classes of `BBTreenode` are allowed to override the *opening*, *visiting* and *closing* actions. However more fine-grained behavior modifications can be obtained by overriding:

- the computation and ordering of child nodes and their associated subproblems
- the decision whether to enter the column generation phase
- actions taken before or after the column generation phase
- the decision whether to backtrack
- the decision whether to backtrack up to the root node (restart)

10.2 COLUMN POOL

All along the exploration of the Branch & Bound tree, feasible routes are generated and stored in the column pool. A class `Column` is used to represent a route r along with its cost $distance(r)$.
The class `ColumnWrapper` is used to envelop a column with additional information such as an unique identifier. It can be subclassed in order to associate further information with a `Column` instance.
An abstract class `ColumnPool` stores `ColumnWrapper` instances along with the associated `Column` instance. The `ColumnPool` class can be subclassed in order to provide a concrete implementation of the column pool. In this work it has been decided to implement the column pool using a map from the standard C++ library. The route r is used as key and the value is the corresponding `ColumnWrapper` instance. Finally the `ColumnPool` can be asked to return the ids associated with columns that do, or do not, pass a test implemented in the `ColumnAllowanceChecker` class. Again behavior of this class can be overridden by inheritance in order to comply with different subproblem restrictions.

10.3 SOLVER AND SOLUTIONS

Columns (i.e. feasible routes) correspond to variables and coefficients in the Set Covering/Partitioning Problem. IBM CPLEX is used to model and solve the SCP/SPP and their continuous relaxations. The link between columns in the column pool and the variables in the solver is maintained using the unique identifier associated with each `Column` instance via the `ColumnWrapper`. This allows to modifiy the bounds of the variables in the solver, in order to fix or forbid certain columns in the SCP/SPP. Each time the SCP/SPP (or their continuous relaxation) is solved, the selected columns, their value in the solution, and the dual costs associated with the constraints are stored in an instance of the `Solution` class. A function of this class allows to verify whether some customer is visited more than once in the represented solution (assuming the latter is integer). If this is the case an attempt at heuristically fixing this situation is made by the `Solution` instance itself.

10.4 COLUMN GENERATION

New classes can be derived from the abstract `ColumnGenerator` class in order to implement different methods of generating feasible routes. Since all heuristics used in the proposed approaches are based on a randomized version of the savings heuristic (a heuristic constructing routes by concatenating smaller routes), a template class called `RandomizedSavings` has been implemented. Its behavior can be modified by overriding:

- the computation of the initial state
- the computation of the attractiveness of a merge
- the decision to consider a given merge for execution
- the decision to verify the feasibility of a given merge
- the decision to register the route resulting from a given merge as column

The `ColumnGenerationManager` class is responsible for setting up the column generation heuristic, executing it and calling the post-optimization methods on the produced routes. Of course this class can be adapted by subclassing to allow setting up approach-specific heuristics.

10.5 BLACK BOX AND FEASIBILITY STORE

Finally the class `BlackBoxLink` provides an interface to the black box function $feasbb$, used to check the feasibility of a route r w.r.t F_{bb}. It also

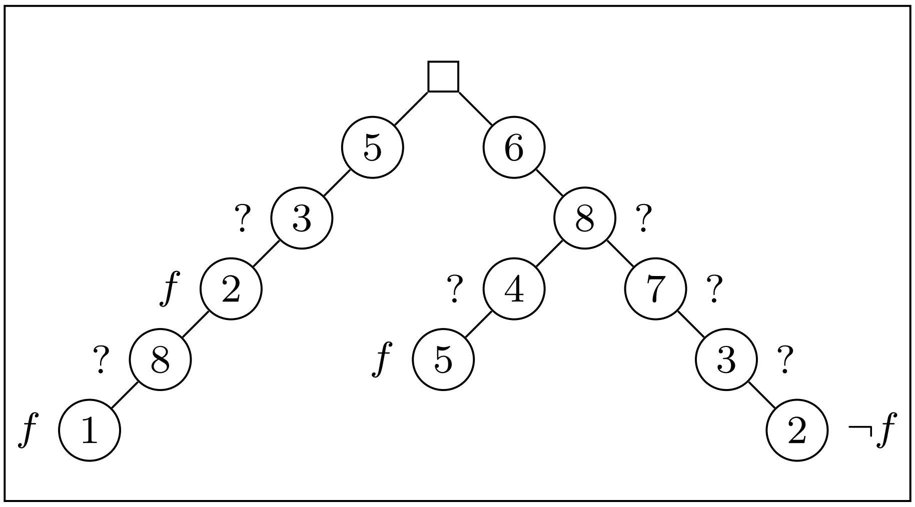


Figure 10.2: Feasibility Store implementation. The shown store contains feasible routes $\langle 5,3,2\rangle$, $\langle 5,3,2,8,1\rangle$,$\langle 6,8,5,4\rangle$ and infeasible route $\langle 6,8,7,3,2\rangle$. The feasibility status of the other routes contained in the store remains unknown.

provides an interface to the class `FeasibilityStore` where the feasibility of routes already tested using *feasbb* is stored. This class represents a tree. With each node, except the root node in this tree is associated a customer. The customers encountered on the path from a given node c up to the root node constitute a route, where the customer associated with node c constitutes the end-point of the route. It is with these end-points that the feasibility information of the route corresponding to the path up to the root node is associated. The concept is illustrated in Fig. 10.2. This store contains 12 routes. Route $\langle 5,3,2,8,1\rangle$ is known to be feasible, which is why f (for *feasible*) is associated with the corresponding node 1. However not all of subroutes of this route (appearing in the store) are known to be feasible. For example route $\langle 5,3\rangle$ hasn't been tested for feasibility so far, thus its feasibility status is unknown (denoted by ?). On the other hand, route $\langle 5,3,2\rangle$ is already known to be feasible. Note that depending on the black box constraints F_{bb}, it is possible for a feasible route to contain an infeasible subroute.

In the implementation of the proposed optimization approaches a given route r is always checked for feasibility (in the black box and the feasibility store) in both directions. That is, first the feasibility store is searched for the feasibility status of route r. Should the status of r be unknown the store is searched for $reverse(r)$. If the feasibility status for both r and $reverse(r)$ is unknown, route r is verified using $feasbb$. If it is feasible, route r is stored as feasible in the feasibility store. If it is infeasible, $reverse(r)$ is checked using $feasbb$. If this should be infeasible as well, route r is stored as infeasible in the feasibility store, else it is stored as feasible. Note that this means that a route is stored as feasible in the store as soon as it has been proven to be feasible in one direction. It is stored as infeasible once it has been proven infeasible in both directions. For Heuristic Branch & Price where branching is done on arcs, routes are checked in only one direction. This is to comply with the fact that branching fixes or forbids arcs (as opposed to edges). Thus only routes that are found to be feasible will be added to the feasibility store in this case.

10.6 IMPLEMENTATION OF THE PROPOSED APPROACHES

Pheromone-based Heuristic Column Generation (ACO-HCG)
Each iteration in the ACO-HCG approach corresponds to the exploration of a root node in the Branch & Bound Tree. Column Generation is always entered and a restart is always decided. The behavior of the system is modified using subclasses of the classes `BBTreenode`, `RandomizedSavings` and `ColumnGenerationManager`. The subclasses of the latter implement the *strict* and *liberal* Collector ants described in chapter 6. Furthermore class `PheromoneManager` has been added to provide the pheromone matrix. The subclass of `ColumnGenerationManager` updates the pheromone matrix w.r.t. the current solution and calls the collector ants and post-optimization methods.

Dive & Generate (DING)
Adaptation of the `BBTreenode` class for DING is straight-forward given the description provided in chapter 7. The `Subproblem` class needs to be adapted in order to include the set $\mathcal{F}$ of fixed variables (i.e. columns, i.e. feasible routes). The `RandomizedSavings` class is subclassed, modifying its behavior in terms of all points underlined in section 7.6.

Implementation of Heuristic Branch & Price (HBP)
Again the adaptation of `BBTreenode` is straight-forward. As however the setup of the `ColumnAllowanceChecker` depends on the type of branching decisions taken, separate adaptations of template classes

`BBTreenode` and `ColumnAllowanceChecker` are necessary for the HBP where the branching is done on customer pairs and the one where branching is done on arcs. The same holds obviously for the adaptations of the `Subproblem` class. In the same spirit different specializations of class `ColumnGeneratioManager` are needed. In the case of the HBP where branching is done on arcs, certain arcs need to be forbidden. This is done by adapting the arc costs using the `DistanceManager` class. The pricing for both cases is handled by different classes, each subclassing `RandomizedSavings`, and implementing the behavior as described in chapter 8.

Decomposition-based approach

The decomposition-based approach is executed on top of any of the proposed approaches. For facility of implementation a new instance is created from the customers appearing in set T. A mapping between the new and the original instance is done when accessing the black box and the feasibility store. Note that the feasibility store remains global to the approach and is not reinitialized for each new set T.

Part III

APPLICATIONS

11 PARAMETER SETTING WITH IRACE

The method proposed in chapter 6 makes use of multiple parameters. However as the number of possible configurations of values for these parameters is huge, it is impossible to experimentally validate or invalidate all these configurations. It is for this reason that an automatic algorithm configuration procedure called iterated racing was used to determine high-performing parameter configurations for the Pheromone-based Heuristic Column Generation approach. This was done in joint work with researchers of the Iridia department at the Université Libre de Bruxelles. The results have been published in [MLISD13]. Two optimized configurations have been obtained, one per concrete application of the Vehicle Routing Problem with Black Box Feasibility. These concrete applications, the Three-dimensional Loading Capacitated Vehicle Routing Problem (3L-CVRP) and the Multi-Pile Vehicle Routing Problem (MP-VRP), are introduced in detail later in this thesis (chapters 12 and 13).

11.1 PRINCIPLES

The Iterated F-Race method considers all possible parameter value configurations and uses statistical information to exclude bad configurations early on. In order to use the method, the configurable parameters and their possible values need to be clearly defined. This is done in section 11.4. The iterated F-Race procedure itself is described in section 11.2. The automatic configuration procedure was done separately for each of the sets of black box constraints F_{bb} (corresponding to the side-problems from 3L-CVRP and MP-VRP) considered in this thesis. To do this training instances were used, they are described in section 11.3. The "optimized" configurations were then used to do a parametrical analysis (on each set of black box constraints) to evaluate the impact of each parameter individually. The "optimal" configurations obtained and their experimental evaluation are given in the applications section 11.6.

11.2 ITERATED F-RACE

A good overview of the Iterated F-Race approach can be found in [BYBS10]; the explanations given here are based on this paper. A description of the irace package implementing the Iterated F-Race approach, used here, can be found in [LIDLSB11].
Iterated F-Race is based on three concepts: the Racing approach, F-race and Iterative Racing.
The goal of the Racing approach is, given an initial set of candidate configurations Θ_0 and a set of training problem instances I, to identify a subset of configurations in Θ_0 that outperform the other configurations. The approach proceeds in steps, each step k resulting in a set $\Theta_k \subseteq \Theta_{k-1}$. Step k consists in applying all the configurations in set Θ_{k-1} to problem instance $i_k \in I$ and then discarding the configurations for which enough statistical evidence of their sub-optimality has been gathered over steps 1 to k. The "surviving" configurations are gathered in set Θ_k. The procedure stops once a predefined computation budget has been reached. Such a computation budget can for example correspond to a maximum number of evaluations (one evaluation being the application of some configuration to some problem instance).
The F-race approach prescribes how the statistical evidence for discarding sub-optimal configurations is computed. It uses a non-parametrical Friedman test (see [BYBS10]) to verify whether the configurations have performed differently in the past (in a statistical significant way). If this is the case, each configuration is compared to the current best configuration, and those that have performed significantly worse are discarded.
Finally Iterated F-Race applies the racing procedure iteratively to sets

of configurations. With each parameter value in the space of possible parameter values is associated a probability. Initially this probability is the same for each parameter value. A set of configurations is then sampled from the parameter space using these probabilities. F-Race is then applied to this set, returning a set of elite configurations. The probabilities associated with the parameter values appearing in these elite configurations are increased. Using these adapted probabilities a new set of configurations is sampled, serving, together with the set of elite configurations from the previous iteration, as the initial set for the next F-Race procedure. The overall procedure is stopped once a given computation budget has been used up.

11.3 TRAINING INSTANCES

As mentioned before the method is applied on a set of representative training instances. This set is different from the set of instances for which a high-performing configuration should be produced. The idea behind this is to avoid overtuning the configuration to a small set of instances, but rather to achieve a configuration performing well even on previously unknown instances. A set of such training instances was created by perturbing the original set of unseen instances (corresponding to the benchmark instances from the literature). Only slight perturbations were allowed in order to not destroy the underlying problem structure. Each customer property was perturbed with a probability of 95%. A perturbation replaces the value of the property by $\textit{current value} + r \cdot \max_{\text{prop}}$, where r is a number selected uniformly at random in the interval $[-\delta, \delta]$ and $\max_{\text{prop}}$ is the maximal value for the considered property in the original instance. Different values for δ were considered ($\delta \in \{0.05, 0.1, 0.15\}$ for MP-VRP, and $\delta \in \{0.1, 0.15\}$ for 3L-CVRP). In the case of the MP-VRP instances the demand of a customer corresponds to the number of items demanded per type (see chapter 13). The following customer properties were considered for perturbation: x-coordinate and demand of one randomly selected type of item. For the 3L-CVRP, the following customer properties were considered: x-coordinate, demand, randomly selected dimension of randomly selected item, fragility of randomly selected item (see chapter 12). Five different combinations of these properties were considered. The time limits used on the perturbed instances correspond to those used on the original instances (1800 CPU seconds for MP-VRP and 1800, 3600 or 7200 CPU seconds for 3L-CVRP).

11.4 CONSIDERED PARAMETERS AND THEIR RANGES

The numerical and configurational parameters optimized appear in different parts of the Pheromone-based Heuristic Column Generation approach. For each part, the parameters are explained next. An overview, as well as the parameter ranges and types is given in table 11.1. The ranges used were based on an initial intuitive parameter configuration.

11.4.1 *Parameters of the collector ant algorithm*

Collector ants are used to generate *wb*- and *bb*-feasible routes. To do this they execute a randomized version of the savings heuristic. All possible merges are considered and added, based on their attractiveness to a candidate list. Once ν *bb*-feasible merges haven been gathered in the candidate list one of the candidate merges is selected and executed.

The following parameters have been considered for configuration:

$\underline{\nu}$	the number of *bb*-feasible merges that need to be gathered in the candidate list Ω before one of the merges in the list is selected and executed
$\underline{\alpha,\beta}$	the exponential factors used for the pheromones (α) and the savings (β) value to compute the attractiveness of a merge
$\underline{op}$	the operator used to compute the attractiveness of a merge. op corresponds either to a multiplication or summation.

11.4.2 *Parameters of the main algorithm*

The main algorithm repetitively executes ℓ collector ants to generate *wb*- and *bb*-feasible routes, post-optimizes these routes and adds them to the set of feasible routes $\mathcal{R}^*$. It then solves problem $Relax(SCP(\mathcal{R}^*))$

and uses the resulting solution to update the pheromone matrix. In the following, besides numerical parameters, configurational parameters modifying the described behavior of the algorithm are considered:

$\underline{\ell}$	the number of ants executed per iteration of Pheromone-based Heuristic Column Generation
$\underline{\rho, \epsilon}$ and $\underline{\tau_{min}}$	the trail persistence, pheromone update constant and minimum pheromone deposit used in the update of pheromones
strictness	indicating whether to use *strict* or *liberal* ants (see section 6.2, the initialization of $\mathcal{R}^*$ is always done with *strict* ants)
post-opt	indicating whether to use the *Tabu Search* (*TS*) or *Iterated Local Search* (*ILS*) for post-optimization
useint and $\underline{f}$	parameter *useint* indicates whether instead of solving problem $Relax(SCP(\mathcal{R}^*))$, problem $SCP(\mathcal{R}^*)$ should be solved (possibly not to optimality), and the corresponding integer solution be used for the pheromone update. Note that when problem $SCP(\mathcal{R}^*)$ is solved, the corresponding dual costs π are considered zero. Parameter *useint* can take 3 possible values: 1. *always* : problem $SCP(\mathcal{R}^*)$ is solved all the time, problem $Relax(SCP(\mathcal{R}^*))$ is never solved 2. *never* : problem $Relax(SCP(\mathcal{R}^*))$ is solved all the time, problem $SCP(\mathcal{R}^*)$ is never solved (except to obtain the final solution) 3. *freq* : problem $SCP(\mathcal{R}^*)$ is solved instead of problem $Relax(SCP(\mathcal{R}^*))$ every f iterations

11.5 AUTOMATIC PARAMETER SETTING

The automatic configuration of ACO-HCG was done with irace using the parameter domains given in Table 11.1, and the described set of training instances. The computational budget corresponded to 5000 runs of ACO-HCG. irace was executed two times, once for the MP-VRP training instances and another time for the 3L-CVRP training instances. Thus, two automatic configurations of ACO-HCG result. For

Table 11.1: Parameters considered for automatic configuration ([MLISD13])

Parameter	Domain	Description
ν	$[10, 50] \in \mathbb{N}$	#*bb*-feasible merges in Ω
α	$[0, 20] \in \mathbb{N}$	exp. factor for τ in merge attractiveness (Eq. 21)
β	$[0, 20] \in \mathbb{N}$	exp. factor for s in merge attractiveness (Eq. 21)
op	$\{+, \cdot\}$	use addition/multiplication operator in Eq. 21
ℓ	$[1, 10] \in \mathbb{N}$	#ants executed per iteration
ρ	$[0, 1] \in \mathbb{R}$	trail persistence
ϵ	$[0.0, 1.0] \in \mathbb{R}$	pheromone update constant
$\tau_{\min}$	$[0, 1] \in \mathbb{R}$	lower bound on pheromone level
strictness	{ *strict*, *liberal* }	*strict* / *liberal* ants
post-opt	{ *ILS*, *TS* }	*ILS* / *TS* for post-optimization
useint	$\{never, always, freq\}$	solve $SCP(\mathcal{R}^*)$ instead of $SCP(Relax(\mathcal{R}^*))$
f	$[2, 10] \in \mathbb{N}$	if *useint* $==$ *freq*, solve $SCP(\mathcal{R}^*)$ instead of $SCP(Relax(\mathcal{R}^*))$ every f iterations

the experiments in this chapter ACO-HCG was compiled using gcc 4.4.6 and CPLEX 12.4 was used as the MILP solver. Experiments were run on a single core of an AMD Opteron 6272 CPU (2.1 GHz, 16 MB L2/L3 cache size) running under Cluster Rocks Linux version 6/CentOS 6.3, 64bits. Table 11.2 shows the automatic configurations and the two manual configurations that were originally used. The manual configurations were based on preliminary experiments and standard ACO parameters, and were found to be competitive with existing approaches.

Some notable differences between the settings of the automatic configurations and the manual ones are that the former have larger values of β, a larger number of ants (ℓ), a lower pheromone trail persistence

(ρ), they solve $SCP(\mathcal{R}^*)$ instead of $SCP(Relax(\mathcal{R}^*))$ in some iterations (f), and the ants are *strict* instead of *liberal*.

The quality of a given configuration is measured in terms of mean relative percentage deviation (%-deviation) from the best known solution cost reached using the same black box functions (including results from [MDVH12]). For the MP-VRP this corresponds to the %-deviation from the best solution costs presented in [TDHI09], while for the 3L-CVRP it corresponds to the %-deviation of the best solution costs presented in [Bor12]. The automatic and manual configurations of ACO-HCG are compared in Fig. 11.1. Each point in the plot shows the mean %-deviation over 20 independent runs (with different random seeds) on the same test instance. The two configurations perform equally on the same instance if the point is on the diagonal, the automatic configuration performs better if the point is under the diagonal, and the manual configuration performs better if the point is above the diagonal. Moreover, the symbols denote whether the differences observed are statistically significant (decided using a Wilcoxon signed-rank test with confidence level 95%).

For the MP-VRP (Fig. 11.1a), the improvement of the automatic configuration over the manual one is considerable. In particular, all the differences are statistically significant. Moreover, for many instances, the automatic configuration obtains an average result that is better than the best known solution among those considered. For the 3L-CVRP (Fig. 11.1b), the improvement is smaller. Nonetheless, the automatic configuration is never worse than the manual configuration and it is significantly better on a few instances. In Tables 11.3 and 11.4 the manual and automatic configuration are compared to the best known solutions in literature using all kinds of loading algorithms (and thus possibly obtained using loading algorithms different from the ones used as black box functions in this thesis). The automatic configuration is able to find new best solutions for 14 out of the 21 MP-VRP instances and 8 out of the 27 3L-CVRP instances.

11.6 EXPERIMENTAL ANALYSIS OF THE ACO-HCG PARAMETERS

Based on the automatic configuration obtained in the previous section several of the parameters of ACO-HCG are analysed in a systematic

Table 11.2: Parameter configurations of ACO-HCG.

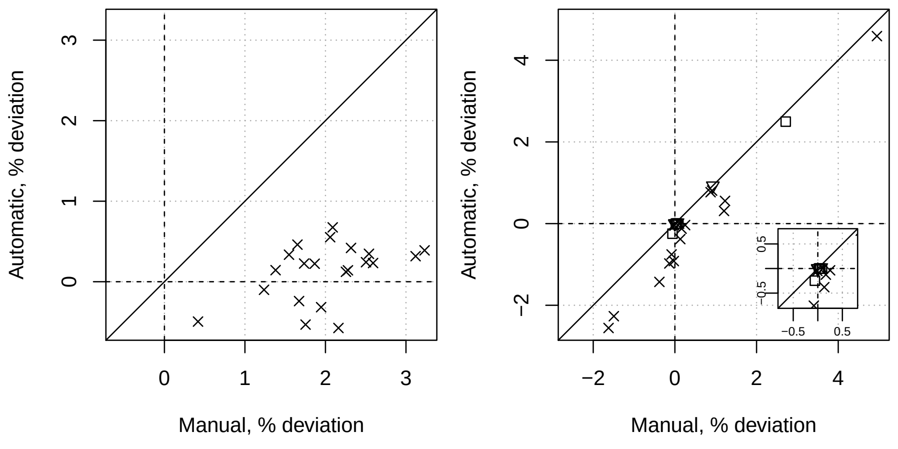
Problem	**Config.**	ν	ℓ	α	β	ϵ	ρ	$\tau_{\min}$	useint (f)	*strict.*	*post-opt*	*op*
MP-VRP	Manual	13	1	5	5	0.15	0.95	0.20	never	liberal	TS	mult
	Automatic	10	9	1	10	0.69	0.34	0.79	$f = 7$	strict	TS	mult
3L-CVRP	Manual	13	5	5	5	0.15	0.95	0.20	never	liberal	TS	mult
	Automatic	41	10	3	9	0.66	0.45	0.29	$f = 6$	strict	TS	mult

manner. Note that a fully-factorial experimental design is intractable given the number of parameters and the computation time required by each run. Instead the high-performing parameter configuration automatically obtained in the previous section is used and parameter settings that disable or replace one algorithmic component at a time are considered.

Pheromone information (α). By setting $\alpha = 0$, the influence of the pheromone information is disabled. The result is a noticeable deterioration in quality in most MP-VRP instances (Fig 11.2a) and some 3L-CVRP instances (Fig 11.2e). This suggests that the pheromone information plays a positive role in the performance of the algorithm.

Savings heuristic (β). By setting $\beta = 0$ the use of the savings heuristic value s in the attractiveness equation is disabled. The savings heuristic guides the ants to build cost-efficient routes, and, hence, disabling it leads to a substantial quality deterioration in both problems (MP-VRP, Fig. 11.2b, and 3L-CVRP, Fig. 11.2f). For small instances of the 3L-CVRP, however, the differences are typically minor. This is due to the high value for parameter ν. In fact, for small instances with only few customers, setting ν, that is, the number of feasible merges in candidate list Ω, to a high value, results in most possible merges being included in Ω. These are then checked for feasibility and added to $\mathcal{R}^*$. However, for large instances, the setting of parameter ν excludes many interesting merges that would have a high heuristic value if $\beta \neq 0$. In summary, the savings heuristic remains essential for the generation of high-quality routes.

Learning mechanism (ρ). By setting $\rho = 0$, the pheromones are reset at every iteration, and only the amount deposited in the current iteration has an effect. Hence, this setting disables the learning mechanism of ACO and forces the ants to focus on the solution found in



(a) MPVRP (b) 3L-CVRP

Figure 11.1: Comparison between manual and automatic configuration. Each point gives the mean %-deviation from the best-bb solution over 20 runs with different random seed on the same test instance. The symbols denote whether there is a statistically significant difference (X) or not (□), or all the runs obtained the same cost (▽).

the current iteration. Given that the results do not show a clear effect of the learning mechanism (Fig. 11.2d and 11.2h), but that completely disabling the pheromone information ($\alpha = 0$, as discussed above) does deteriorate quality, it can be concluded that the pheromone information provides a diversification mechanism rather than learning the best arcs over time.

Strict vs. liberal ants. Whereas *strict* ants may only execute merges resulting in *bb*-feasible routes, *liberal* ants may also execute merges resulting in only *wb*-feasible routes. The rationale of *liberal* ants is that *bb*-infeasible routes might be merged with *bb*-feasible routes in order to produce *bb*-feasible routes. This, of course, depends on the black box at hand. In the case of the MP-VRP, such a situation cannot arise and, hence, *strict* ants produce much better results (Fig. 11.2c). While for the 3L-CVRP an infeasible and a feasible route can theoretically be concatenated to produce a feasible route, such a situation does not seem to occur frequently in the benchmark instances available (Fig. 11.2g). Thus, the choice of *strict* ants rather than *liberal* ants in the automatic configuration seems to be justified.

Sum vs. multiplication in attractiveness equation. The results clearly worsen when using a sum ($op = +$) in the attractiveness equation. Since the values of the savings heuristic are much larger than the pheromone values, summing both neglects the effect of the pheromones. In fact, the plots (Fig. 11.3a and 11.3e) are almost identical to those where the pheromone information is disabled ($\alpha = 0$, Fig 11.2a and 11.2e). Therefore, the use of multiplication is recommended.

Solving SCP$(\mathcal{R}^*)$ instead of Relax(SCP$(\mathcal{R}^*)$). Parameter *useint* controls whether the solution used to update the pheromone information is obtained by solving $SCP(\mathcal{R}^*)$ or $Relax(SCP(\mathcal{R}^*))$. For the MP-VRP, while never using the integer solution does not have a significant effect on most instances, always using it slightly deteriorates the quality in some instances (see Fig. 11.3b) For the 3L-CVRP, it does not matter whether the solutions of $SCP(\mathcal{R}^*)$ or $Relax(SCP(\mathcal{R}^*))$ are used to update the pheromones (see Fig. 11.3f). Thus it does not matter whether the pheromones are deposed in such a way that the collector ants are more likely to construct new routes similar to the optimal routes in the solution $sol \in Sol(Relax(SCP(\mathcal{R}^*)))$ (having a zero reduced cost).

Post-optimization using TS vs. ILS When comparing TS vs. ILS as the post-optimization method, we observe that, in the MP-VRP, solution quality does deteriorate in some instances when using ILS instead of TS, and in the 3L-CVRP, a slight deterioration can be observed on a couple of instances (Fig. 11.3c and 11.3g). This indicates that in most cases the TS is able to find the improvements that can be found using ILS, or possibly that in the majority of cases neither TS nor ILS can find a feasible improvement to a given feasible route.

Number of ants Finally, if the number of ants is set to one ($\ell = 1$), results improve slightly on a few instances and get slightly worse in others for the MP-VRP (Fig. 11.3d). In the case of the 3L-CVRP, setting $\ell = 1$ slightly deteriorates quality in a majority of the instances (Fig. 11.3h). This could indicate that in the case of the 3L-CVRP the collector ants converge too rapidly towards a set of routes, the pheromone matrix being updated each time only one ($\ell = 1$) ant has finished generating feasible routes.

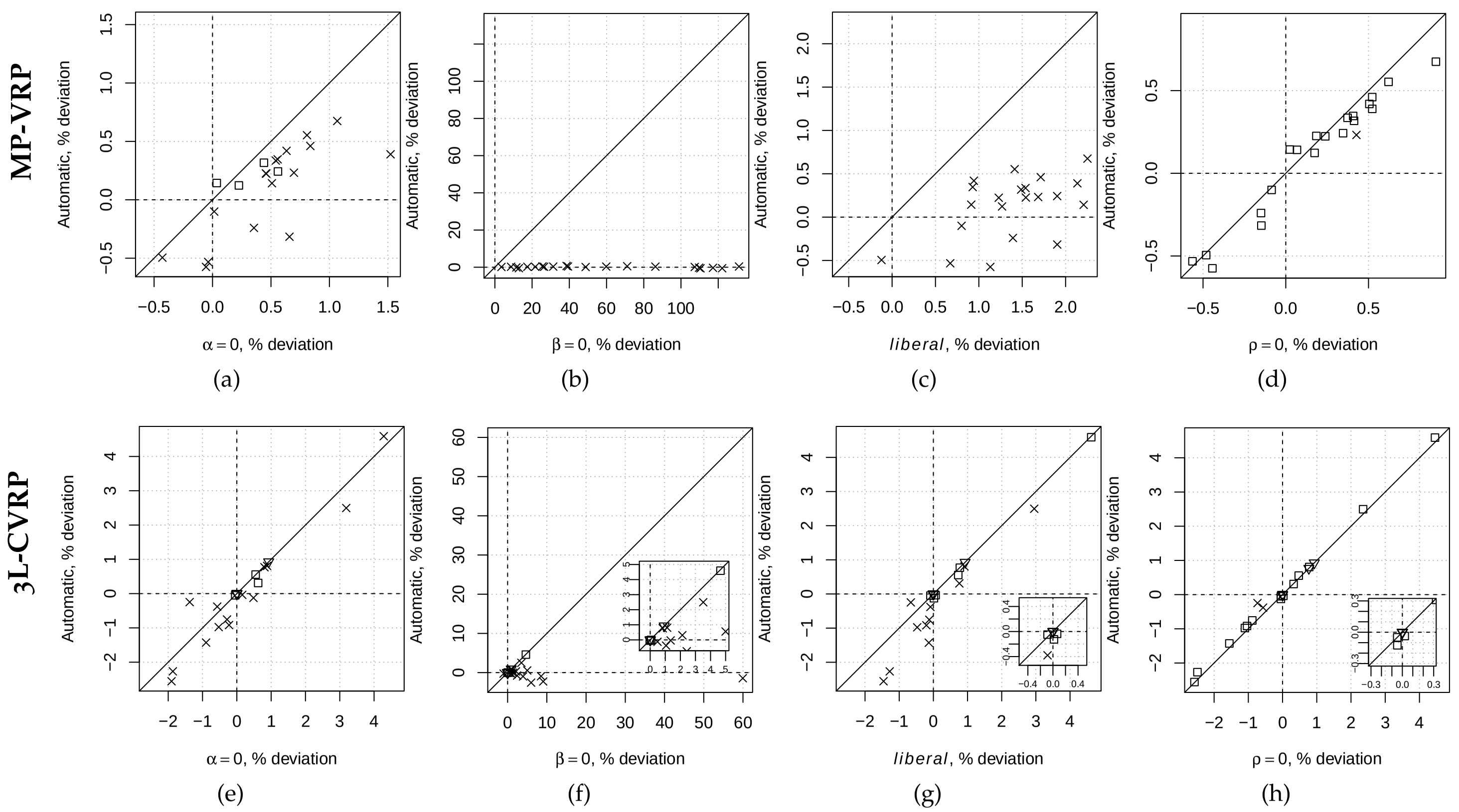


Figure 11.2: Each plot shows the effect of changing one parameter of the "automatic" configuration. Results for the MP-VRP are in the upper row; results for 3L-CVRP in the lower row. Each point gives the mean %-deviation from the best-bb solution over 20 runs with different random seed on the same test instance. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

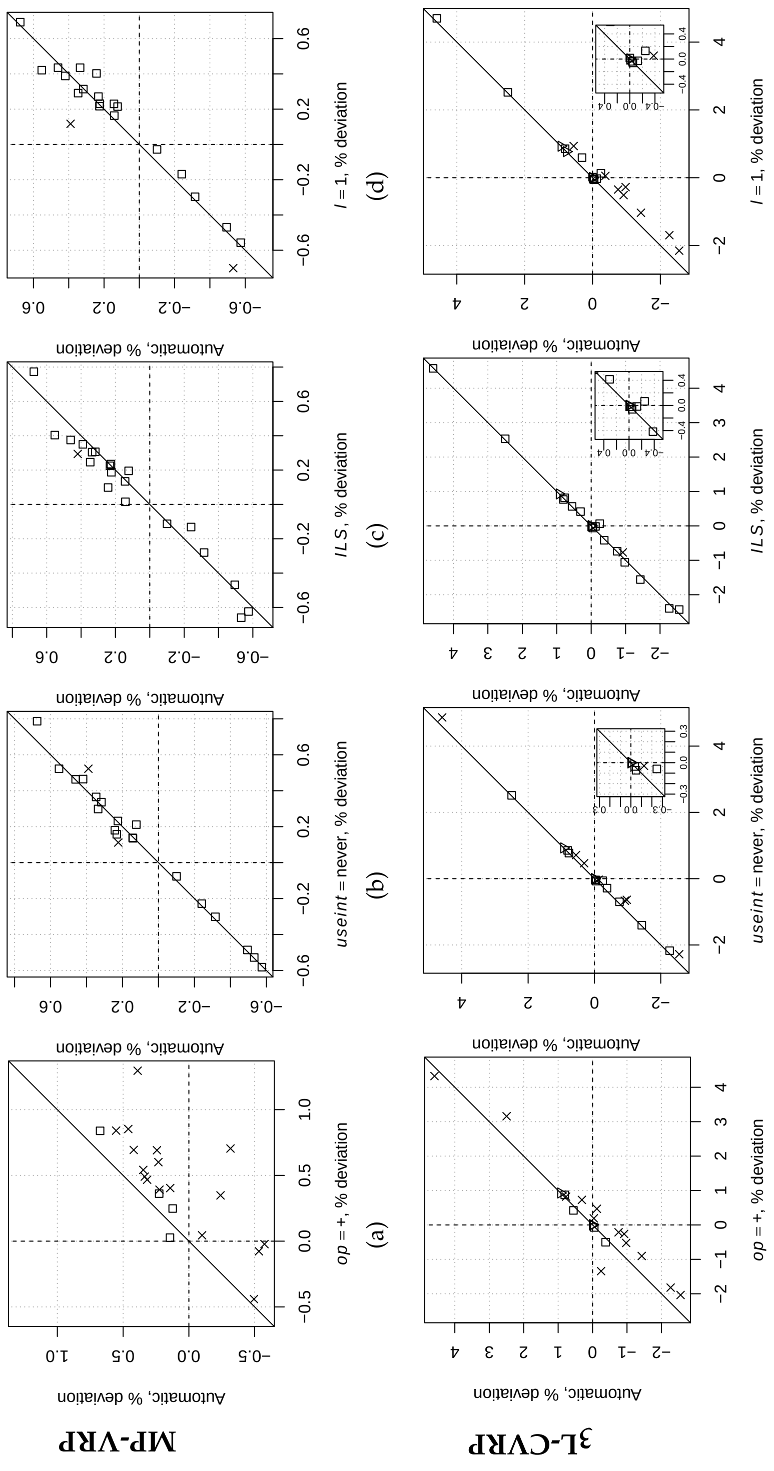


Figure 11.3: Each plot shows the effect of changing one parameter of the "automatic" configuration. Results for the MP-VRP are in the upper row; results for 3L-CVRP in the lower row. Each point gives the mean %-deviation from the best-bb solution over 20 runs with different random seed on the same test instance. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

	Best	**Manual Conf.**				**Automatic Conf.**			
	z_{best}	z_{min}	z_{avg}	sec_{tt}	g_{avg}	z_{min}	z_{avg}	sec_{tt}	g_{avg}
CMT01-1	587.29	590.45	599.38	1820	2.06	**587.29**	*590.54*	1833	0.55
CMT01-2	615.12	617.82	628.99	1807	2.25	**615.11**	*615.88*	1811	0.12
CMT01-3	623.45	623.91	632.06	1807	1.38	**623.44**	*624.35*	1806	0.14
CMT02-1	978.66	976.70	984.47	1811	0.59	**974.47**	*975.53*	1848	-0.32
CMT02-2	897.62	901.91	911.51	1826	1.55	**897.51**	*900.63*	1825	0.34
CMT02-3	888.38	895.49	903.79	1813	1.73	889.26	*890.39*	1814	0.23
CMT03-1	1188.18	1198.09	1218.03	1833	2.51	**1180.21**	*1184.93*	1819	-0.27
CMT03-2	1218.96	1229.23	1241.71	1833	1.87	1219.13	*1221.68*	1926	0.22
CMT03-3	1156.84	1170.63	1186.82	1836	2.59	**1154.11**	*1159.52*	1814	0.23
CMT04-1	1624.98	1631.73	1660.10	1881	2.16	**1607.63**	*1615.64*	1900	-0.57
CMT04-2	1552.27	1564.80	1578.23	1864	1.67	**1543.37**	*1548.54*	1940	-0.24
CMT04-3	1541.81	1563.87	1578.30	1862	2.37	**1541.35**	*1545.32*	1899	0.23
CMT05-1	2035.77	2052.01	2074.49	1950	1.90	**2019.80**	*2027.90*	2029	-0.39
CMT05-2	1833.41	1866.50	1892.65	1909	3.23	**1832.18**	*1840.55*	2022	0.39
CMT05-3	1948.84	1974.25	1996.54	1946	2.45	**1946.30**	*1952.23*	1907	0.17
CMT06-1	2240.57	2242.65	2292.50	2072	2.32	**2238.56**	*2249.97*	2111	0.42
CMT06-2	2070.04	2107.45	2142.76	2021	3.51	2089.17	*2096.93*	2000	1.30
CMT06-3	2154.19	2169.29	2189.78	1870	1.65	**2153.45**	*2164.11*	1847	0.46
CMT07-1	1136.55	1151.85	1160.34	1855	2.09	1140.14	*1144.25*	1835	0.68
CMT07-2	1217.45	1226.17	1232.51	1855	1.24	**1214.58**	*1216.23*	1854	-0.10
CMT07-3	1157.67	1171.37	1188.00	1894	2.62	**1152.16**	*1161.79*	1847	0.36
AVG					2.08				0.19

Table 11.3: Comparison of "manual" and "automatic" configuration with best-known results on MP-VRP. $z_{min/avg}$=best and average solution value, sec_{tt}=total execution time in seconds, gap_{avg}=average relative percentage deviation w.r.t. best published solution. All results over 20 independent runs. The relative percentage deviation for one run is computed as $100 \cdot \frac{z - z_{\text{best}}}{z_{\text{best}}}$ where z is the solution value for the given run and z_{best} the best published solution value. For the automatic configuration results in **bold** indicate a tie or improvement over the best published solution value, results in *italic* indicate a tie or improvement over the average solution value obtained using the manual configuration. Note that small variations such as these for instances CMT01-2 and CMT01-3 are to be attributed to rounding.

	Best	**Manual Conf.**				**Automatic Conf.**			
	z_{best}	z_{min}	z_{avg}	sec_{tt}	g_{avg}	z_{min}	z_{avg}	sec_{tt}	g_{avg}
3l-cvrp01	291.00	302.02	302.13	1800	3.82	302.02	*302.02*	1800	3.79
3l-cvrp02	334.96	334.96	334.96	1800	0.00	**334.96**	*334.96*	1800	0.00
3l-cvrp03	392.46	385.53	392.23	1800	-0.06	**385.53**	*391.49*	1800	-0.25
3l-cvrp04	437.19	437.19	437.19	1800	0.00	**437.19**	*437.19*	1800	0.00
3l-cvrp05	443.61	447.73	447.73	1800	0.93	447.73	*447.73*	1800	0.93
3l-cvrp06	498.16	498.16	498.27	1800	0.02	**498.16**	*498.16*	1800	0.00
3l-cvrp07	768.85	769.68	769.68	1800	0.11	769.68	*769.68*	1800	0.11
3l-cvrp08	805.35	845.50	851.01	1800	5.67	845.50	*848.12*	1800	5.31
3l-cvrp09	630.13	630.13	630.67	1800	0.09	**630.13**	*630.13*	1800	0.00
3l-cvrp10	817.38	826.66	827.52	3601	1.24	826.66	*826.66*	3603	1.14
3l-cvrp11	778.10	776.19	778.03	3600	-0.01	**776.19**	*777.72*	3600	-0.05
3l-cvrp12	612.25	612.25	612.88	3600	0.10	**612.25**	*612.25*	3600	0.00
3l-cvrp13	2645.95	2661.62	2670.06	3600	0.91	2661.62	*2667.32*	3601	0.81
3l-cvrp14	1368.42	1392.06	1405.56	3602	2.71	1385.00	*1402.58*	3604	2.50
3l-cvrp15	1341.14	1336.21	1343.34	3611	0.16	**1336.21**	*1339.46*	3618	-0.13
3l-cvrp16	698.61	698.61	698.61	3600	0.00	**698.61**	*698.61*	3600	0.00
3l-cvrp17	866.40	866.40	866.84	3600	0.05	**866.40**	*866.40*	3600	0.00
3l-cvrp18	1207.72	1205.11	1222.25	3612	1.20	**1205.11**	*1211.46*	3614	0.31
3l-cvrp19	741.74	741.31	743.59	7203	0.25	**741.31**	*741.47*	7203	-0.04
3l-cvrp20	586.92	581.13	586.12	7236	-0.14	**577.39**	*581.19*	7215	-0.98
3l-cvrp21	1042.72	1080.24	1089.93	7272	4.53	1075.16	*1080.22*	7217	3.60
3l-cvrp22	1147.80	1154.12	1161.89	7220	1.23	1148.82	*1154.18*	7211	0.56
3l-cvrp23	1119.05	1104.06	1118.14	7229	-0.08	**1101.47**	*1110.60*	7213	-0.76
3l-cvrp24	1096.88	1112.49	1117.60	7225	1.89	1107.15	*1111.89*	7225	1.37
3l-cvrp25	1407.36	1389.35	1402.02	7346	-0.38	**1373.24**	*1387.27*	7240	-1.43
3l-cvrp26	1430.15	1553.04	1568.21	7291	9.65	1544.40	*1555.92*	7216	8.79
3l-cvrp27	1455.27	1493.33	1503.80	7316	3.33	1483.59	*1489.56*	7235	2.36
AVG					1.38				1.03

Table 11.4: Comparison of "manual" and "automatic" configuration with best-known results on 3L-CVRP. $z_{min/avg}$=best and average solution value, sec_{tt}=total execution time in seconds, gap_{avg}=average relative percentage deviation w.r.t. best published solution. All results over 20 independent runs. The relative percentage deviation for one run is computed as $100 \cdot \frac{z - z_{\text{best}}}{z_{\text{best}}}$ where z is the solution value for the given run and z_{best} the best published solution value. For the automatic configuration results in **bold** indicate a tie or improvement over the best published solution value, results in *italic* indicate a tie or improvement over the average solution value obtained using the manual configuration.

12

VRP WITH THREE-DIMENSIONAL LOADING CONSTRAINTS

The Three-dimensional Loading Capacitated Vehicle Routing Problem (3L-CVRP) was first introduced in [GILM06]. To verify the feasibility of a route a three-dimensional loading problem needs to be solved. Customers request a set of three-dimensional boxes. These boxes must then be loaded into the back of the delivery truck, which is itself rectangular and limited in three dimensions as well as in maximal weight. A feasible route must respect the capacity constraint, and a feasible loading, respecting typical bin-packing constraints as well as some complicating real-world constraints, must exist.

12.1 PROBLEM DESCRIPTION

The 3L-CVRP is defined on top of the basic CVRP. A limited, homogeneous fleet of vehicles is available to perform the deliveries of items to customers. Each vehicle corresponds to a rectangular container of width W, height H and length L. The loading space is accessible only from the rear of the vehicle. The vehicles also have a maximum capacity (read maximum weight) limited to Q. Each customer $i \in V\backslash\{0\}$ demands a set I_i of m_i items of total weight q_i. Each item $I_{iz}(z = 1, ..., m_i\, , \, i \in V\backslash\{0\})$ is a box of width w_{iz}, height h_{iz} and length l_{iz}, and is either fragile or non-fragile. See Figure 12.1 for the

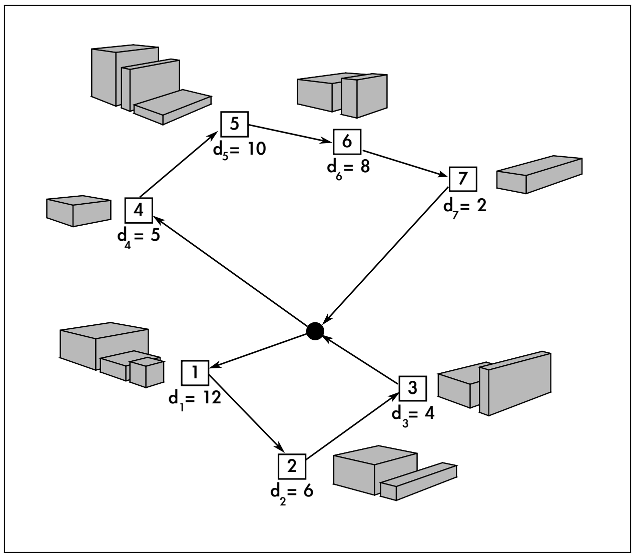


Figure 12.1: 3L-CVRP instance with routes. Demand of customer i is denoted d_i

depiction of an 3L-CVRP instance with routes. The goal is to find a set of at most $|K|$ routes each starting and ending at the depot, visiting every customer exactly once such that the total cost is minimized and the following conditions hold for every route r:

- $\sum_{i \in r.S} q_i \leq Q$ (capacity constraint)
- a feasible orthogonal loading exists for $\bigcup_{i \in r.S} \bigcup_{z=1}^{m_i} \{I_{iz}\}$

A loading of a route r is the assignment of coordinates to the lower left corner of each item $I_{iz}(z = 1, ..., m_i), i \in r.S$. The origin of the coordinate system is assumed in the lower left back corner of the vehicle. The following conditions must be fulfilled over the items $\bigcup_{i \in r.S} \{I_i\}$ for a loading to be considered feasible:

- *Containment* All items fit completely into the loading space of the vehicle.
- *Non-overlapping* None of the items overlap in all three dimensions at the same time.
- *Fragility* No non-fragile item is placed on top of a fragile item.
- *Support* An item must be supported from below by at least $\alpha\%$ of its bottom surface.

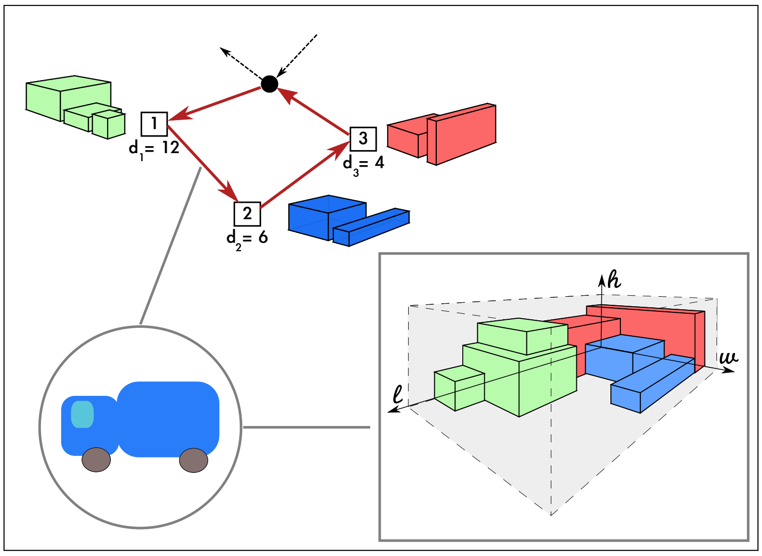


Figure 12.2: Feasible loading for a 3L-CVRP route

- *LIFO* When visiting a customer $i \in r.S$ no item $I_{jz}(z = 1, \ldots, m_j)$ s.t. $pos(j,r) > pos(i,r)$ may be placed between the items of i and the rear of the vehicle, nor between the items of i and the top of the vehicle.

Note that items may be rotated by 180° but may not be flipped. A feasible loading for a given route is depicted in Figure 12.2.

12.2 EXISTING APPROACHES

In [GILM06] the authors propose a deterministic Tabu Search (TS) allowing visits to capacity- and loading-infeasible solutions. The loading subproblem is addressed using a Tabu Search itself using two greedy heuristics adapted from well-known loading heuristics.
Two initial solutions are computed. They are computed using adaptations of the savings and an insertion heuristic. The produced solutions may violate the capacity and the loading constraints. The solutions with the smallest number of infeasible routes is used as the initial solution. A measure of infeasibility is included in the objective function as the excess weight and excess length. This means that all loading produced must respect all the loading constraints, except for the *containment* constraint, which may be violated in one dimension. The weights

used for these measures in the objective function are adapted throughout the search. The search constructs a restricted neighborhood using η_{reloc}. The routes implicated in a move are reoptimized using 4-opt insertion. The best of these moves, taking into account the objective of the resulting solution and a penalty term used to enhance diversification, is chosen. Each time a new best solution is found, the size of the neighborhood N_{reloc} is increased during the next iteration. Finally, when a solution improving the total distance of the incumbent is encountered, and if this new solution is capacity-feasible but not feasible in terms of loading, the excess length of the loading is recomputed, while allowing more iterations to the Tabu Search handling the loading. The TS algorithm is tested on benchmark instances generated from known CVRP benchmark instances.

The authors in [TZK09] propose a Guided Tabu Search (GTS). They propose a bundle of packing heuristics to consider the loading problem. The 6 heuristics are always applied in increasing order of complexity. This is stopped as soon as one of the heuristics is able to produce a feasible loading.
An initial solution is constructed using an adapted parallel insertion heuristic. Customers are inserted in the route that currently has the lowest non-occupied volume. All routes in the initial solution must be completely feasible, however the number of vehicles in the initial solution might exceed the actual fleet size. The GTS uses three restricted neighborhoods, one of which is selected at random each iteration. The neighborhoods, including only feasible solutions (but possibly using more vehicles than available), are constructed using η_{reloc}, η_{swap} and a combination of η_{cross} and η_{2ex}. The best neighbor is chosen. The guiding mechanism works by selecting a long (cost-intensive) arc in the current best solution, and penalizing it over a given number of iterations by doubling its cost. This penalty may however be overridden if a new best solution using the penalized arc is found. Using the best solutions found over all runs the authors are able to improve the costs over 21 instances out of 27 compared to [GILM06].

In [FDHI10] (ACO) an adapted version of the Savings-based ants introduced in [RSD02] is proposed. The loading subproblem is considered by using lower bounds derived from bin-packing, and if these do not prove infeasibility, the heuristics from [GILM06] are applied repeatedly (perturbing the input sequence of items to be loaded each time). The attractiveness of a merge (in the savings heuristic executed by the ants) is adapted to the 3L-CVRP by introducing a notion of the loading compactness of the route resulting from that merge. Solutions produced by the ants must be feasible except for the number of vehicles used. The solutions are post-optimized using a local search. This local search explores neighborhoods constructed using η_{reloc} and η_{swap}. The local search is allowed to violate capacity constraints at a penalty. A further term is introduced in the objective function used for the local search if the number of vehicles used in the current solution exceeds the fleet size. It aims at prioritizing solutions with a high capacity- and volume-utilization of the vehicles. Finally an elitist scheme is used to update the pheromone matrix. The ACO is able to outperform the TS on average in 26 out of 27 instances and to beat the GTS on average in 23 out of 27 instances .

A Hybrid Tabu Search (HTS) is presented in [Bor12]. A tree traversal algorithm with a limited number of backtracks is proposed to handle the loading problem. An initial solution is generated by choosing the best among several solutions generated using a randomized savings heuristic. The produced solutions may not violate any constraints, but may use more vehicles than available. The HTS is split in two phases, one aiming at reducing the number of vehicles in the initial solution, and the other aiming at minimizing the total distance. At each iteration, complete neighborhoods constructed using η_{reloc} and η_{swap} are explored to select a small set of the best neighbors, one of which is chosen at random. In the first phase the number of vehicles is used as objective function. The quality of a move is measured using the total volume loaded in the lighter of the implied routes. The search stops as soon as a solution, using as many vehicles as available (or less) is found. This solution is then used as initial solution for the second tabu search, using the total distance as objective. In this search only entirely feasible solutions may be visited. The HTS is able to improve on average 20 out of the 27 benchmark instances when compared to [FDHI10].

In [ZQLW12] a two-stage tabu search (2STS) is proposed. The authors enhance well-known packing heuristics which are embedded in a local search. An initial solution is constructed using an adapted version of the savings heuristic. The initial solution may violate the capacity and loading constraints. The first phase of their tabu search is employed if the initial solution is infeasible. The objective function used penalizes excess capacity and excess length of a loading (as done in [GILM06]). A fixed number of neighbors is randomly generated using η_{2ex} and η_{swap}. The best of the generated neighbors is then selected. Phase one stops as soon as a feasible solution has been found. This solution is then used as initial solution for the second phase. The second phase explores only feasible solutions. Neighbors are generated as before, this time using additionally η_{reloc}, η_{cross} and an operator splitting a single route in two routes. The 2STS improves on average 20 out of the 27 instances over [FDHI10].

The authors in [RTS11] propose a heuristic Branch & Bound (BB) method for the 3L-CVRP. They use a previously published Container Loading algorithm to solve the loading side-problem. This algorithm is based on a partial tree search and greedy packing heuristics. The proposed Branch & Bound model corresponds to a relaxation of the 3L-CVRP where the loading constraint is replaced with a lower bound based on the volume of items to be loaded. It is iteratively used to extend partial solutions into complete solutions, using lower and upper bounds from the VRP literature. The minimum cost routes that are generated using this method are checked for feasibility using the container loading procedure. The authors improve the best result from [FDHI10] in 14 out of 27 cases.

A Honey Bees Mating approach (HBM) is proposed in [RZMS13]. The loading problem is attacked using a set of six loading heuristics. Each bee corresponds to a solution. The population is initialized to feasible VRP solutions, which also take the total allowed volume per route into account, using an Greedy Randomized Adaptive Search (GRASP). The existence of a feasible loading is not considered at this stage. The population evolves until a maximum number of generations has been reached. The solutions in a restricted candidate list are then considered by increasing total distance and loading feasibility is checked using the heuristics. If none of the solutions is feasible, a new honey bee procedure is started. The HBM is able to improve 18 out of the 27 instances when compared to the average results obtained in [FDHI10].

12.3 3L-CVRP AS A VRPBB

The 3L-CVPR can be decomposed into a routing and a loading problem. The set of white box constraints F_{wb} corresponds to the capacity constraints. The unknown black box constraints F_{bb} to be respected by every route, correspond to the constraint that a feasible three-dimensional loading must exist for the items to be delivered on the route.

For this thesis the algorithm described in [Bor12] has been reimplemented. The author proposes a tree traversal method to solve the loading problem. Each node in the tree corresponds to a partial loading, and the branching decision consists in deciding which potential placement (i.e, item and position pair) to execute next. Heuristic decisions are used to prune branches. Since this loading algorithm is of high complexity (polynomial in the number of items per route), only a limited number of nodes may be visited before aborting the procedure. Note that this approach is not complete. The loading check provided for the CVRPBB here is thus not exact, i.e. feasible routes may be considered infeasible.

12.3.0.1 *Problem-specific knowledge in existing approaches*

The TS and the 2STS use the excess length of a loading to measure the violation of the loading constraints. Both GTS and HTS use the volume of the demanded items at some point in the initialization or optimization procedure. The ACO algorithm uses a measure of loading compactness in the attractiveness of route merges. It also uses the volume information in the local search post-optimizing solutions constructed by the ants. The BB also uses the volume as a lower bound. Finally HBM also uses the volume, to restrict the set of solutions considered in the bees process to solutions where none of the routes exceed the total volume allowed by the vehicles.

12.4 PROBLEM INSTANCES

The benchmark instances used for the 3L-CVRP are the ones presented in [GILM06]. They were generated based on well-known Capacitated VRP instances from the literature. The loading volume of the vehicles was fixed to $W = 25$, $H = 30$ and $L = 60$. For each customer the number of items was drawn uniformly at random from the range $1, \ldots, 3$. For each item, its width (resp. height or length) was generated uniformly at random from the interval $[0.2W, 0.6W]$ (resp. $[0.2H, 0.6H]$ or $[0.2L, 0.6L]$). The minimum percentage of supported area α was fixed to 75%. The authors then proceeded to compute the minimum number of vehicles necessary to load all the customers' items using the greedy loading heuristics presented in the same paper. The number of vehicles reached for each instance using this approach was then set as the maximum number of vehicles for the given instance. It should be noted that the maximum numbers of vehicles indicated in the paper are incorrect. The correct numbers of vehicles can be found in the instance files that can be downloaded from `http://www.or.deis.unibo.it/research.html`. To the best of our knowledge all papers presented in the previous section use the correct numbers.

12.5 EXPERIMENTAL RESULTS

The experiments presented in this section have all been conducted on an AMD Opteron 6284 SE CPU (2.7 Ghz, 16MB cache size). The system has been compiled using gcc 4.4.7 and uses CPLEX 12.4 as MILP solver and the Boost Libraries 1.53 for random number generation. Results have been obtained over 10 independent runs. The solutions corresponding to the results presented in this section can be downloaded from `http://becool.info.ucl.ac.be/resources/VRPBB`.

In this section two types of best known results from the literature are considered. The *all-best* results correspond to the overall best known results from the literature for the 3L-CVRP. The *bb-best* results on the other hand correspond to results from the literature obtained using the same black box algorithm as the one used here. In the case of the 3L-CVRP the *bb-best* results correspond to the ones presented in [Bor12].

12.5.1 *Pheromone-based Heuristic Column Generation (ACO-HCG)*

The parameter configuration used for ACO-HCG has been elaborated using an automatic algorithm configuration procedure. The process and the resulting configuration are described in detail in chapter 11. Note that the experiments in chapter 11 and the experiments for the present chapter were conducted on different architectures. A comparison of the results achieved on both architectures over the same number of runs (using the same seeds for the random number generator) is given in Table A.2 in the appendix. It shows that the variations are not very important, the average results (over 10 runs) vary by 0.05%

when averaged over all instances. This can be attributed to the varying architecture but also to the non-determinism inherent to CPLEX.

The time limits from [GILM06] are used. The limit is imposed on the route generation process. After the allowed time has been used up, the current ant iteration is finished, the generation of new routes is stopped and the final solution is computed. For instances with $0 \leq n \leq 25$, the route generation time is limited to 1800 CPU seconds, for instances with $25 < n < 50$ it is limited to 3600 CPU seconds, and to 4200 CPU seconds for instances with $n \geq 50$. A time limit of $1.5 \cdot n$ is imposed on CPLEX when solving problem $SCP(\mathcal{R}^*)$. If it is necessary to solve problem $SPP(\mathcal{R}^*)$, this time limit is used in combination with a limit on the number of discovered solutions, set to 10.

Table 12.1 presents the results obtained on the 3L-CVRP benchmark instances. Column z_{min} and z_{avg} indicate the best and average solution value over 10 runs. Column $\%RSD$ gives the relative standard deviation over 10 runs, while column sec_{tt} indicates the average total execution time. Finally in column $\%g_{avg}^{bb}$ the average relative deviation w.r.t the *bb-best* result is given. The g_{avg}^{bb} value is computed as $100 \cdot \frac{z - z_{bb}}{z_{bb}}$ where z is the solution cost obtained using ACO-HCG and z_{bb} is the solution cost of the *bb-best* solution. A negative g_{avg}^{bb} value indicates that ACO-HCG improves the *bb-best* result on average (over 10 runs) .

A look at the $\%g_{avg}^{bb}$ column shows that the proposed ACO-HCG approach is competitive in terms of solution quality. Over the 10 runs, on average the *bb-best* result is obtained or even improved for 18 out of 27 instances. It should be noted however that most variations are in the range of a few percent. The most notable average deterioration w.r.t. the *bb-best* result is for instance 3l-cvrp08, a small-scale instance, while the biggest improvement is obtained on 3l-cvrp26, a large-scale instance with clustered customers.

In terms of execution times, ACO-HCG is outperformed by the Hybrid Tabu Search (HTS) presented in [Bor12]. HTS uses both a limit on stable iterations and a limit on total execution time, whereas in ACO-HCG only a time limit is used. However, preliminary experiments have shown that even when using a limit on the number of stable iterations with ACO-HCG, the total execution time needed for all but the smallest instances remains notably superior to the one needed in [Bor12].

When compared to the state-of-the-art HTS is very time-efficient. Execution times of the state-of-the-art approaches for the 3L-CVRP are recapitulated in Table A.1 in the appendix.

Analysis of CPU time allocation and Feasibility in generated routes

In this section the percentage of total execution time spent in different parts of the algorithm is analyzed, as well as the number of *bb*-feasible routes found, compared to the total number of routes tested for feasibility w.r.t. F_{bb}.

Figure 12.3 gives a visual representation of the percentage of CPU time allocated to different tasks in ACO-HCG. A detailed view of this is given in Table 12.2. The tasks under consideration are:

- solving problem $SCP(\mathcal{R}^*)$ to obtain an integral solution ($\%IP$)
- solving problem $Relax(SCP(\mathcal{R}^*))$, repeated each iteration of ACO-HCG ($\%LP$)
- testing routes for *bb*-feasibility ($\%BB$)
- retrieving or adding routes to the Feasibility store Ψ ($\%ST$)
- generating new routes, thus executing the collector ants modulo the time to check feasibility of routes ($\%GEN$)
- post-optimizing the feasible routes generated by the collector ants ($\%PO$)
- other parts of ACO-HCG ($\%OTH$)

It is clear from Fig. 12.4 that the majority of the CPU time is allocated to the generation of feasible routes and to testing the *bb*-feasibility of routes. However there are notable differences between different instances. First, for the larger instances ($n \geq 50$) most time is spent in the black box (testing feasibility w.r.t. F_{bb} of routes). For the smaller instances, we can identify two sets, those where the majority of the time is spent in the generation of routes, and those where the majority of the time is spent in the black box. In the first set there are instances where the capacity constraints (that is the constraints in F_{wb}) are very hard (instances 3l-cvrp02, 3l-cvrp16 and 3l-cvrp17). Only a handful of

Instance			**ACO-HCG Automatic Configuration**				
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_{tt}	$\%g^{bb}_{avg}$
3l-cvrp01	4	15	302.02	302.02	0.0	1800.0	0.00
3l-cvrp02	5	15	334.96	334.96	0.0	1800.0	0.00
3l-cvrp03	4	20	385.53	390.12	1.0	1800.3	-0.64
3l-cvrp04	6	20	437.19	437.19	0.0	1800.0	0.00
3l-cvrp05	6	21	447.73	447.73	0.0	1800.2	0.93
3l-cvrp06	6	21	498.16	498.16	0.0	1800.1	0.00
3l-cvrp07	6	22	769.68	769.68	0.0	1800.4	0.00
3l-cvrp08	6	22	845.50	845.50	0.0	1800.6	4.27
3l-cvrp09	8	25	630.13	630.13	0.0	1800.0	0.00
3l-cvrp10	8	29	826.66	826.66	0.0	3603.2	0.77
3l-cvrp11	8	29	776.19	777.91	0.1	3601.5	-3.20
3l-cvrp12	9	30	612.25	612.25	0.0	3600.3	-0.38
3l-cvrp13	8	32	2661.62	2665.48	0.1	3601.6	0.74
3l-cvrp14	9	32	1385.00	1398.84	0.4	3605.3	2.22
3l-cvrp15	9	32	1336.22	1341.02	0.2	3624.1	-0.01
3l-cvrp16	11	35	698.61	698.61	0.0	3600.3	0.00
3l-cvrp17	14	40	866.40	866.40	0.0	3600.1	0.00
3l-cvrp18	11	44	1205.11	1209.21	0.4	3606.2	0.12
3l-cvrp19	12	50	741.31	741.31	0.0	7204.6	-0.06
3l-cvrp20	18	71	577.85	580.84	0.4	7217.9	-1.21
3l-cvrp21	17	75	1075.97	1079.47	0.2	7220.4	-0.99
3l-cvrp22	18	75	1147.43	1154.64	0.3	7217.8	0.60
3l-cvrp23	17	75	1102.39	1110.24	0.7	7221.1	-1.80
3l-cvrp24	16	75	1107.71	1110.20	0.1	7228.3	-0.53
3l-cvrp25	22	100	1370.70	1387.14	0.6	7256.1	-1.44
3l-cvrp26	26	100	1541.49	1548.15	0.3	7212.9	-3.26
3l-cvrp27	23	100	1483.66	1491.09	0.2	7253.1	-2.53
AVG					0.2		-0.24

Table 12.1: Results on 3L-CVRP using ACO-HCG in the automatic configuration, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_{tt} = average total execution time, $\%g^{bb}_{avg}$=average relative percentage deviation w.r.t. *bb-best* solution.

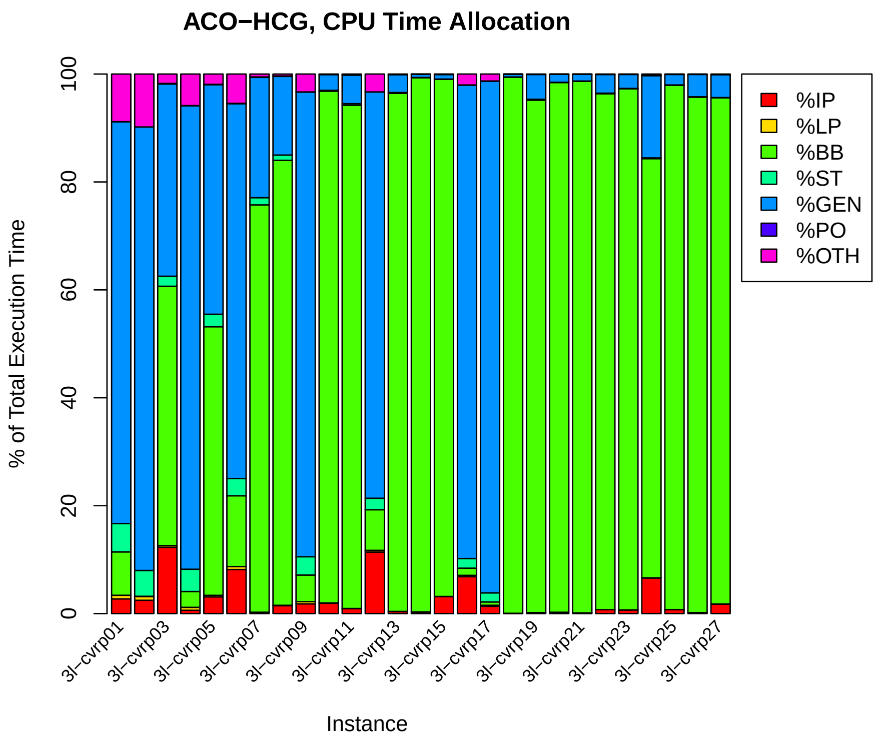


Figure 12.3: ACO-HCG: CPU Time Allocation

routes are actually *wb*-feasible (feasible w.r.t. the capacity constraints), and the time spent testing these routes for *bb*-feasibility is negligible. Then in this set, there are also instances where the best solution to $Relax(SCP(\mathcal{R}^*))$ is reached early on. The same solution will be obtained over several iterations. Thus the same solution will be used to update the pheromones. The collector ants will converge to reproduce the routes appearing in this solution. The *bb*-feasibility of these routes will be verified in the Feasibility store Ψ, where they are stored as feasible. This explains the higher $\%ST$ value for these instances.

Figure 12.4 shows the percentage of routes (averaged over 10 runs) that were tested for feasibility w.r.t. F_{bb}, and that were actually found to be feasible. Again this value varies significantly from instance to instance. It is however to be noted, that the percentage of feasible routes is higher for instances with tight capacity constraints (see above). This could be explained by the fact that in the set of *wb*-feasible routes the majority is also *bb*-feasible. It can also be noted that instances 3l-cvrp04, 3l-cvrp06, 3l-cvrp09 and 3l-cvrp12 have a higher percentage of positive tests.

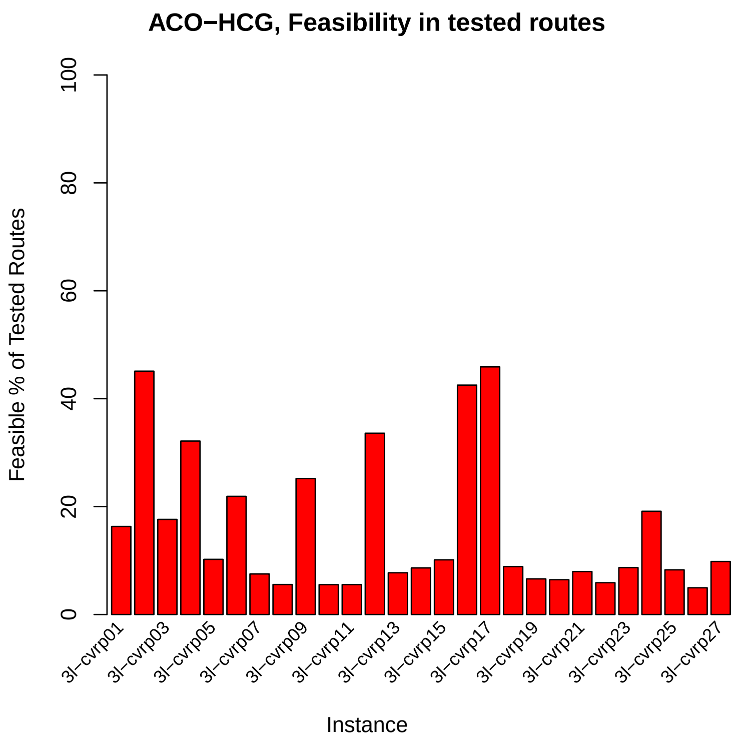


Figure 12.4: ACO-HCG: % of tested routes that were *bb*-feasible

The exact numbers of feasible and tested routes are given in Table 12.2. In columns *#Feasible routes* and $|\Psi|$ the exact number of routes in the final $\mathcal{R}^*$ and the final number of routes in Feasibility store Ψ is given. In the case of ACO-HCG, $|\Psi|$ corresponds to the total number of routes tested for *bb*-feasibility. The numbers of feasible and tested routes vary of course in function of the problem size.

12.5.2 *Decomposition-based approach using ACO-HCG*

The decomposition-based approach described in chapter 9 has been tested with ACO-HCG. Parameter N has been set to $\lceil \frac{n}{5} \rceil$. The time allowed to solve each subproblem was fixed to $min(300, \overline{n})$ CPU seconds, where $\overline{n}$ corresponds to the number of customers in the subproblem. Explicit results comparing the performance of ACO-HCG and the Decomposition-based approach using ACO-HCG (DECOMP ACO-HCG) can be found in table 12.3. ACO-HCG outperforms DECOMP ACO-HCG on a majority of instances. In only one instance is DECOMP ACO-HCG able to outperform ACO-HCG on average. It seems that solving subproblems does not allow to generate routes interesting for the global problem. Possibly too much time is lost solving simple problems which do not allow to contribute further feasible routes to the pool of feasible routes. A plot comparing the performance of DECOMP ACO-HCG to ACO-HCG can be found in the appendix (Fig. A.1).

12.5.3 *ACO-HCG without Post-optimization of feasible routes*

In chapter 11 a parameter configuration for ACO-HCG has been elaborated using an automatic algorithm configuration procedure. For the post-optimization two parameters were considered: *TS* and *ILS*, corresponding to the choice of doing the post-optimization of feasible routes either using Tabu Search or Iterated Local Search. However the choice to do no post-optimization at all was not considered. Thus in order to establish whether the post-optimization procedure actually improves ACO-HCG's results, experiments were run using ACO-HCG's automatic configuration (see 11) where the post-optimization was switched off (i.e. neither Tabu Search nor Iterated Local Search are performed). The resulting approach is denoted ACO-HCG-NOPO. The corresponding results are given in Table A.3 in the appendix. Furthermore the exact CPU Time Allocation percentages along with the number of tested and feasible routes is given in Table A.4 in the ap-

Instance	**% Total Execution Time**						**Routes in Ψ**	
	$\%IP$	$\%LP$	$\%BB$	$\%ST$	$\%GEN$	$\%PO$	#Feasible routes	$\|\Psi\|$
3l-cvrp01	2.7	0.7	8.0	5.3	74.5	0.0	774.1	4750.8
3l-cvrp02	2.5	0.7	0.1	4.8	82.2	0.0	406.3	902.7
3l-cvrp03	12.3	0.3	48.1	1.9	35.6	0.1	2261.5	12916.0
3l-cvrp04	0.6	0.5	2.9	4.1	85.9	0.0	1156.1	3597.1
3l-cvrp05	3.1	0.3	49.8	2.3	42.5	0.1	1773.2	17382.3
3l-cvrp06	8.2	0.5	13.1	3.2	69.5	0.1	1821.2	8318.1
3l-cvrp07	0.1	0.1	75.5	1.3	22.3	0.1	1628.8	21756.7
3l-cvrp08	1.5	0.1	82.5	1.0	14.6	0.1	1944.7	35173.0
3l-cvrp09	1.8	0.4	4.9	3.4	86.1	0.0	1365.8	5423.1
3l-cvrp10	2.0	0.0	94.8	0.2	2.9	0.1	3311.9	60714.2
3l-cvrp11	0.9	0.0	93.3	0.3	5.3	0.1	4032.3	73603.1
3l-cvrp12	11.4	0.3	7.5	2.1	75.3	0.0	2631.8	7844.0
3l-cvrp13	0.4	0.0	96.0	0.2	3.3	0.1	3677.5	47700.8
3l-cvrp14	0.3	0.0	99.0	0.0	0.6	0.0	4506.7	53065.0
3l-cvrp15	3.2	0.0	95.8	0.1	0.8	0.1	5285.1	52672.8
3l-cvrp16	6.8	0.3	1.3	1.8	87.7	0.0	3020.3	7111.1
3l-cvrp17	1.4	0.2	0.6	1.7	94.9	0.0	2584.5	5632.3
3l-cvrp18	0.1	0.0	99.4	0.0	0.5	0.0	3481.6	39726.2
3l-cvrp19	0.2	0.0	95.0	0.2	4.6	0.0	6490.0	98943.2
3l-cvrp20	0.2	0.0	98.2	0.1	1.5	0.0	6953.6	108634.4
3l-cvrp21	0.1	0.0	98.5	0.0	1.2	0.0	6974.5	88005.4
3l-cvrp22	0.7	0.0	95.6	0.1	3.5	0.1	8604.0	147775.3
3l-cvrp23	0.7	0.0	96.6	0.1	2.6	0.0	8319.5	96452.1
3l-cvrp24	6.6	0.0	77.7	0.1	15.2	0.1	17296.4	90581.1
3l-cvrp25	0.7	0.0	97.2	0.0	1.9	0.0	8399.3	102284.9
3l-cvrp26	0.2	0.0	95.5	0.1	4.1	0.0	7589.7	153888.7
3l-cvrp27	1.8	0.0	93.8	0.1	4.2	0.1	10633.5	108480.4

Table 12.2: ACO-HCG : CPU Time Allocation, # feasible routes, # routes in Feasibility Store ($|\Psi|$)

pendix.

Table A.3 shows that it is of advantage to execute the post-optimization of feasible routes, this in terms of best solutions as well as in terms of average solution value. The gaps (best solution and average solution value) between ACO-HCG and ACO-HCG-NOPO are however not very large, they generally lie below 1%. A look at Table A.4 reveals that ACO-HCG-NOPO tests generally less routes than ACO-HCG. During the post-optimization in ACO-HCG several perturbations of each feasible route are tested if necessary. This is of course not done in ACO-HCG-NOPO. However one could expect ACO-HCG-NOPO to use the time won by not performing *bb*-feasibility tests during post-optimization to execute more iterations in general in which new feasible routes are discovered and tested. This is not the case, Table A.4 shows that ACO-HCG-NOPO spends more time on tasks $\%IP$ and $\%LP$ than ACO-HCG. This is due to the fact that ACO-HCG-NOPO ends up with a higher number of feasible routes in the column pool than ACO-HCG. In ACO-HCG once a feasible route has been found it is post-optimized, and only the resulting route (not the intermediate feasible perturbations) will be added to the column pool. In ACO-HCG-NOPO no post-optimization is done. Thus each route discovered by the collector ants is directly added to the column pool, and thus the feasible perturbations of feasible routes already in the column pool produced by the collector ants will also be added to the column pool, in contrary to ACO-HCG (assuming those perturbations were encountered in post-optimization with ACO-HCG). Thus the time won by not systematically testing perturbations of feasible routes will be lost again when solving the more complicated Set Covering Problem. Note also that ACO-HCG-NOPO spends slightly more time in the black box than ACO-HCG for a number of instances. This could be due to variations between the routes tested with ACO-HCG and ACO-HCG-NOPO.

Instance			**ACO-HCG**			**DECOMP ACO-HCG**				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
3l-cvrp01	4	15	302.02	302.02	1800.0	302.02	302.02	1808.5	0.00	0.00
3l-cvrp02	5	15	334.96	334.96	1800.0	334.96	334.96	1806.9	0.00	0.00
3l-cvrp03	4	20	385.53	390.12	1800.3	388.09	389.06	1808.2	0.66	-0.27
3l-cvrp04	6	20	437.19	437.19	1800.0	437.19	437.19	1807.5	0.00	0.00
3l-cvrp05	6	21	447.73	447.73	1800.2	447.73	447.73	1814.4	0.00	0.00
3l-cvrp06	6	21	498.16	498.16	1800.1	498.16	498.16	1809.9	0.00	0.00
3l-cvrp07	6	22	769.68	769.68	1800.4	769.68	769.68	1810.4	0.00	0.00
3l-cvrp08	6	22	845.50	845.50	1800.6	845.50	845.74	1810.4	0.00	0.03
3l-cvrp09	8	25	630.13	630.13	1800.0	630.13	630.13	1815.5	0.00	0.00
3l-cvrp10	8	29	826.66	826.66	3603.2	826.66	829.58	3618.2	0.00	0.35
3l-cvrp11	8	29	776.19	777.91	3601.5	776.19	779.93	3616.2	0.00	0.26
3l-cvrp12	9	30	612.25	612.25	3600.3	612.25	612.25	3610.8	0.00	0.00
3l-cvrp13	8	32	2661.62	2665.48	3601.6	2665.33	2670.36	3625.5	0.14	0.18
3l-cvrp14	9	32	1385.00	1398.84	3605.3	1418.51	1434.37	3623.6	2.42	2.54
3l-cvrp15	9	32	1336.22	1341.02	3624.1	1351.99	1358.68	3616.5	1.18	1.32
3l-cvrp16	11	35	698.61	698.61	3600.3	698.61	698.61	3617.8	0.00	0.00
3l-cvrp17	14	40	866.40	866.40	3600.1	866.40	866.40	3614.5	0.00	0.00
3l-cvrp18	11	44	1205.11	1209.21	3606.2	1221.56	1231.39	3632.1	1.37	1.83
3l-cvrp19	12	50	741.31	741.31	7204.6	741.31	743.95	7232.2	0.00	0.36
3l-cvrp20	18	71	577.85	580.84	7217.9	584.67	587.47	7236.2	1.18	1.14
3l-cvrp21	17	75	1075.97	1079.47	7220.4	1089.27	1102.47	7252.3	1.24	2.13
3l-cvrp22	18	75	1147.43	1154.64	7217.8	1154.71	1161.22	7286.1	0.63	0.57
3l-cvrp23	17	75	1102.39	1110.24	7221.1	1121.92	1125.59	7255.2	1.77	1.38
3l-cvrp24	16	75	1107.71	1110.20	7228.3	1109.93	1115.74	7246.3	0.20	0.50
3l-cvrp25	22	100	1370.70	1387.14	7256.1	1406.73	1422.03	7278.4	2.63	2.52
3l-cvrp26	26	100	1541.49	1548.15	7212.9	1564.27	1578.15	7270.0	1.48	1.94
3l-cvrp27	23	100	1483.66	1491.09	7253.1	1501.74	1512.80	7288.9	1.22	1.46
AVG					4210.2			4230.1	0.60	0.68

Table 12.3: Comparison of ACO-HCG in automatic configuration (ACO-HCG) with Decomposition-based approach using ACO-HCG in automatic configuration (DECOMP ACO-HCG). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time. $g^{min/avg}$= relative percentage deviation of DECOMP ACO-HCG w.r.t. ACO-HCG , computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with DECOMP ACO-HCG and $z^D_{min/avg}$ the results obtained with ACO-HCG.

12.5.4 *Dive & Generate (DING)*

Dive & Generate has been executed over 10 independent runs using the same time limits as in ACO-HCG. The set-up of DING is described below.

12.5.4.1 *Set-up of DING*

Restarts In DING the exploration and construction of the current tree is aborted and restarted once ϕ feasible routes with negative reduced cost w.r.t. the root solution have been generated. Here $\phi = 5 \cdot K_{init}$ has been used, where K_{init} corresponds to the number of routes in the initial non-integral solution.

Post-optimization The post-optimization of generated feasible routes has been done using *Tabu Search*.

Branching heuristic At each non-leaf node of the Dive & Generate tree a set of child nodes with associated subproblems is created. A subproblem is created by fixing a route in the current non-integral solution. The corresponding child nodes are ordered using *Value-based ordering*.

Trust region At each non-leaf node SCP_i of the Dive & Generate tree column generation is possibly executed. The decision on whether to do so, depends on the lower bound LB_i and the global upper bound UB. If $LB_i \geq \theta \cdot UB$, then column generation is entered. Parameter θ has been fixed to 1.

Column Generation The randomized savings heuristic constructs at each step a list Ω of potential route merges to execute. The construction of list Ω is stopped as soon as ν merges resulting in *bb*-feasible routes have been added. Parameter ν has been fixed to 13.
Finally the randomized savings heuristic is executed $\tilde{\ell}$ times per column generation execution. Here $\tilde{\ell}$ is computed based on parameter ℓ (fixed to 5) and the percentage of customers already visited in a fixed route ($1 \leq \tilde{\ell} \leq \ell$).

Search Strategy Both Depth First Search and Limited Discrepancy Search have been used. In the case of Limited Discrepancy Search the maximum number of allowed discrepancies is set to K_{init}.

12.5.4.2 *Comparison with ACO-HCG*

In the following we will individually compare Dive & Generate using Depth First Search (DING-DFS) and Dive & Generate using Limited Discrepancy Search (DING-LDS) with ACO-HCG. This is done by computing for each instance and both for ACO-HCG and the DING variant under consideration, g^{bb}_{avg}, the average relative deviation w.r.t. the *bb-best* solution. The resulting values are then visualized in a scatter plot where each point corresponds to one problem instance. The x-coordinate of the point will correspond to g^{bb}_{avg} for ACO-HCG and the y-coordinate to g^{bb}_{avg} for the method under consideration. DING and ACO-HCG perform equally on the same instance if the point is on the diagonal, DING performs better if the point is under the diagonal, and ACO-HCG performs better if the point is above the diagonal. Statistical significance has been determined using a Wilcoxon signed-rank test with confidence level 95%. The resulting plots are visualized in Fig. 12.5. The explicit results for DING-DFS and DING-LDS are given in Tables A.5 and A.6 in the appendix.

The figures show that both DING-DFS and DING-LDS are outperformed by ACO-HCG on a majority of instances. This difference in performance is however more evident with DING-DFS. Fig. 12.5 (b) shows that the differences in performance between ACO-HCG and DING-LDS is similar in all instances (the points follow a line) and not that important (the points are close to the diagonal). The same observations do not hold for DING-DFS. The difference in performance between ACO-HCG and DING-DFS varies significantly from one problem instance to the next. Also, as the plot shows the difference in performance is quite significant for some instances. While DING-LDS is able to improve the *bb-best* solution on average for a handful of instances, DING-DFS does this for only one single instance (in a statistically significant way). Finally neither DING-DFS nor DING-LDS allow to improve upon the results obtained with ACO-HCG on any instance.
While ACO-HCG considers any route as a potential candidate to be added to the set of feasible routes $\mathcal{R}^*$, DING does so only for routes

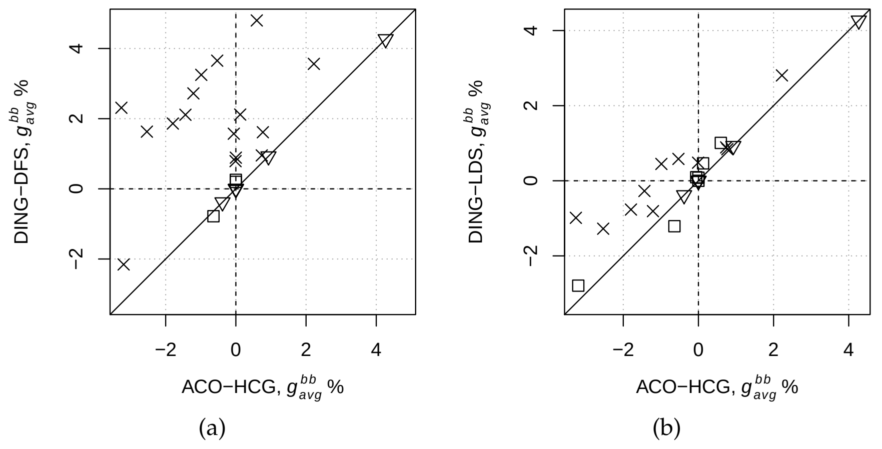


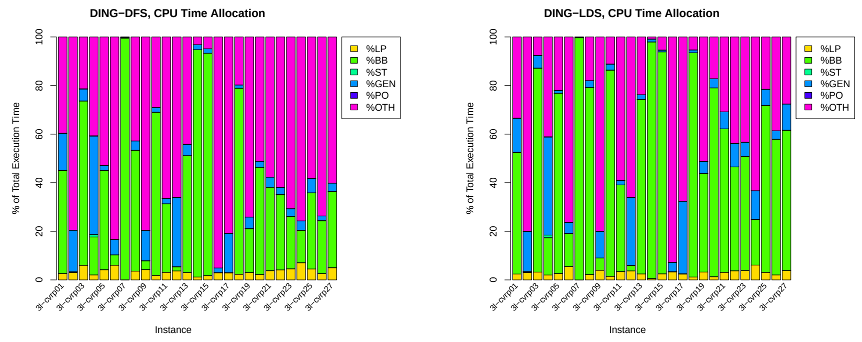
Figure 12.5: Comparison of DING and ACO-HCG. Each point gives the average relative deviation from the *bb-best* solution over 10 independent runs. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

that have a negative reduced cost w.r.t the current non-integral solution. These routes are possibly not the ones that are necessary to obtain a high-quality integer solution. Furthermore only the routes that actually appear in a relaxed solution can ever be part of an integer solution (as this solution is obtained by fixing routes). Thus only if some of the routes appearing in the relaxed solutions are also part of a high-quality integer solution can DING perform well. Given the highly combinatorial nature of the problem under consideration, this might not be the case. Plots comparing DING-DFS and DING-LDS with ACO-HCG in the manual configuration from 11 (denoted $\overline{\text{ACO-HCG}}$ in this chapter) can be found in the appendix (Fig. A.2). DING-LDS and $\overline{\text{ACO-HCG}}$ perform very similar (more so than DING-LDS and ACO-HCG), however the majority of the results is not statistically significant. DING-DFS still performs worse than $\overline{\text{ACO-HCG}}$, but is now able to improve the g^{bb}_{avg} value obtained with $\overline{\text{ACO-HCG}}$ on one instance (DING-LDS improves over $\overline{\text{ACO-HCG}}$ in 3 instances)

12.5.4.3 *Analysis of CPU time allocation and Feasibility in generated routes*

In this section the percentage of CPU time spent in different parts of the algorithm is analyzed, as well as the number of *bb*-feasible routes

found, compared to the total number of routes tested for feasibility w.r.t. F_{bb}.
Figures 12.6 gives a visual representation of the percentage of the total CPU time allocated to different tasks in DING-DFS and DING-LDS. A detailed view of this is given in Tables 12.4 and 12.5. The tasks under consideration are the same as for ACO-HCG. In DING problem SCP($\mathcal{R}^*$) is only solved at the beginning of the process to initialize the upper bound. Task $\%IP$ has been omitted in both figures and tables since the percentage of time spent solving $SCP(\mathcal{R}^*)$ is negligible.



(a) (b)

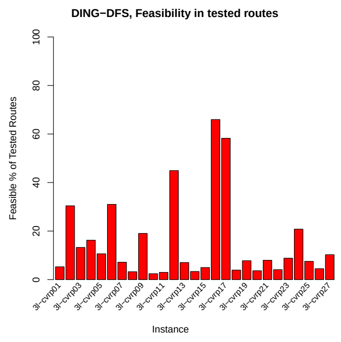
Figure 12.6: DING: CPU Time Allocation

DING spends most of the allocated CPU time while testing the *bb*-feasibility of routes and in the "Other" DING parts. In DING these latter correspond mainly to the construction and exploration of the DING tree structure. Note that the repartition of time between those tasks depends a lot on the problem instance.
Compared to ACO-HCG, DING spends less time testing the *bb*-feasibility of routes in most instances. However, for some instances, e.g. 3l-cvrp01, it is to be noted that while ACO-HCG only spent a small fraction of time in the black box (testing routes for *bb*-feasiblity), DING spends almost half of the total execution time here. This is to be explained by the absence of pheromones in DING. In ACO-HCG the best non-integral solution for this instance is discovered early on. Thus pheromones are always put on the same arcs, leading the collector ants to converge and to reproduce the same set of routes. This doesn't happen in DING since the concept of pheromones doesn't appear. On the

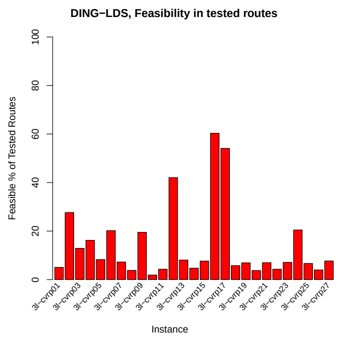
other hand, for some instances (e.g. 3-cvrp02 and 3l-cvrp16), DING spends almost no time at all in the black box, while ACO-HCG did. In these instances most of the CPU time is the used up in the maintenance of the tree structure and some time in the route generation. The fraction of time then allocated to each of these tasks again varies from instance to instance. For some of these instances only very little time is spent in the generation of routes, this is the case for those instances where ACO-HCG spent almost no time in the black box. It seems that for these instances, the lower bound only seldom falls outside the trust region in non-leaf nodes, and thus column generation is only very rarely executed.

When comparing DING-DFS and DING-LDS it turns out that, especially on large scale instances (with one notable exception), DING-LDS spends more time in the black box and generating routes than DING-DFS, but less time in the maintenance of the tree structure. These large-scale instances typically have a high number of vehicles, and thus the Dive & Generate tree is deeper and broader than for smaller instances. Using DFS the search tends to stay in the deeper levels of the tree. At those deeper levels, if column generation is executed, the randomized savings heuristic will be executed less times as a high fraction of customers is already visited in the fixed routes (see parameter $\tilde{\ell}$). Also if a bad decision has been taken by fixing a route at a lower level, it is possible that many infeasible nodes are visited, and thus most of the effort is put into the creation of new sibling nodes. On the other hand using LDS, the search backtracks to lower levels of the tree in a regular fashion, thus modifying decisions taken near the root of the tree. The lower bounds found here usually do not allow to prune nodes. Also, when route generation is entered, here a higher number of randomized savings heuristic executions is performed.

Figure 12.7 shows the percentage of routes tested for feasibility w.r.t. F_{bb} that are found to be feasible. The percentages are similar for DING-DFS and DING-LDS, but generally lower (again with some notable exceptions) when compared to ACO-HCG. The exact numbers are given in Tables 12.4 and 12.5. In columns *#Feasible routes* and $|\Psi|$, the exact number of routes in the final $\mathcal{R}^*$ and the final number of routes in Feasibility store Ψ is given. In the case of DING, $|\Psi|$ corresponds to the total number of routes tested for *bb*-feasibility. The number of feasible and tested routes varies of course in function of the problem size. A closer look at the tables reveals that the number of routes tested



(a)



(b)

Figure 12.7: DING: % of tested routes that were *bb*-feasible

with DING-DFS and DING-LDS is, on most of the smaller and middle-scale instances, higher than for ACO-HCG. This is however not the case for larger instances, where ACO-HCG also clearly spends more time in the black box feasibility test. While DING tests a higher number of routes, the number of feasible routes detected with ACO-HCG is higher than the one with DING-DFS and DING-LDS for all but the smallest instances. Finally the number of *bb*-feasible routes is higher for DING-LDS than DING-DFS on a majority of instances. Again this can be explained by the fact that DING-LDS visits different parts of the search tree. If high-level decisions in DING-DFS leave a subset of customers to visit for which only few *bb*-feasible routes are possible, a lot of time will be spent trying to generate other *bb*-feasible routes that do not exist.

Instance	**% Total Execution Time**					**Routes in Ψ**	
	$\%LP$	$\%BB$	$\%ST$	$\%GEN$	$\%PO$	#Feasible routes	$\|\Psi\|$
3l-cvrp01	2.7	42.4	0.1	15.1	0.1	1673.4	31724.4
3l-cvrp02	3.1	0.2	0.1	17.1	0.0	594.1	1954.4
3l-cvrp03	6.0	67.6	0.0	4.9	0.1	3649.6	27593.7
3l-cvrp04	2.1	15.6	1.1	40.5	0.1	2928.8	18041.1
3l-cvrp05	4.2	40.9	0.0	2.0	0.0	2085.7	19617.8
3l-cvrp06	6.0	4.3	0.0	6.3	0.0	1678.8	5416.7
3l-cvrp07	0.0	99.5	0.0	0.3	0.1	2383.5	33374.4
3l-cvrp08	3.6	49.8	0.0	3.7	0.0	1596.5	49420.3
3l-cvrp09	4.2	3.6	0.0	12.4	0.0	1150.7	6035.0
3l-cvrp10	1.9	67.2	0.0	1.9	0.0	1936.3	93833.3
3l-cvrp11	3.1	28.2	0.0	2.1	0.0	1378.4	61295.6
3l-cvrp12	3.7	1.7	0.0	28.6	0.0	1297.8	2893.6
3l-cvrp13	3.0	48.1	0.0	4.7	0.0	2440.5	34981.4
3l-cvrp14	1.2	93.6	0.0	2.0	0.0	3575.6	107722.6
3l-cvrp15	1.8	91.5	0.0	1.8	0.0	4603.6	92982.0
3l-cvrp16	2.9	0.1	0.0	1.9	0.0	830.7	1300.5
3l-cvrp17	2.9	0.1	0.0	16.2	0.0	863.2	1510.0
3l-cvrp18	2.3	76.6	0.0	1.3	0.0	2494.6	65401.3
3l-cvrp19	3.0	18.1	0.0	4.8	0.0	2624.0	37541.9
3l-cvrp20	2.2	44.2	0.0	2.5	0.0	3730.7	123048.0
3l-cvrp21	3.8	34.4	0.0	4.1	0.0	4004.3	54022.7
3l-cvrp22	4.1	31.0	0.0	3.0	0.0	3636.1	102377.8
3l-cvrp23	4.6	21.7	0.0	3.1	0.0	3450.5	42512.8
3l-cvrp24	7.1	13.4	0.0	3.8	0.0	5544.2	28225.9
3l-cvrp25	4.5	31.3	0.0	6.0	0.0	4838.7	69186.0
3l-cvrp26	2.6	21.8	0.0	1.9	0.0	2990.6	68700.6
3l-cvrp27	5.0	31.4	0.0	3.4	0.0	4934.2	61925.0

Table 12.4: DING-DFS : CPU Time Allocation, # feasible routes, # routes in Feasibility Store ($|\Psi|$)

Instance	**% Total Execution Time**					**Routes in Ψ**	
	%LP	*%BB*	*%ST*	*%GEN*	*%PO*	#Feasible routes	$\lvert\Psi\rvert$
3l-cvrp01	2.4	49.9	0.2	14.0	0.1	1775.4	35196.2
3l-cvrp02	3.1	0.2	0.1	16.6	0.0	612.5	2215.5
3l-cvrp03	3.2	83.9	0.1	5.1	0.1	4164.1	32342.3
3l-cvrp04	2.0	15.3	1.1	40.3	0.1	2932.2	18100.4
3l-cvrp05	2.6	74.2	0.0	1.1	0.1	2800.9	33874.0
3l-cvrp06	5.5	13.6	0.0	4.6	0.1	2746.2	13651.6
3l-cvrp07	0.0	99.6	0.0	0.3	0.1	2370.0	32913.4
3l-cvrp08	2.2	76.9	0.0	2.8	0.1	2256.1	59586.2
3l-cvrp09	4.0	5.0	0.0	11.0	0.0	1419.2	7281.7
3l-cvrp10	1.5	84.9	0.0	2.5	0.0	2560.5	136271.9
3l-cvrp11	3.4	35.7	0.0	1.8	0.0	2152.3	60067.0
3l-cvrp12	3.7	2.3	0.0	27.9	0.0	1496.3	3557.9
3l-cvrp13	2.5	71.8	0.0	1.9	0.1	3608.4	44961.5
3l-cvrp14	0.6	97.4	0.0	1.2	0.0	3826.3	82093.7
3l-cvrp15	2.5	91.4	0.0	0.7	0.0	5356.5	70693.5
3l-cvrp16	3.3	0.1	0.0	3.9	0.0	943.2	1566.1
3l-cvrp17	2.4	0.1	0.0	29.9	0.0	991.3	1864.4
3l-cvrp18	1.2	92.3	0.0	1.2	0.0	3756.7	66166.7
3l-cvrp19	3.2	40.6	0.0	4.9	0.0	4102.9	59967.9
3l-cvrp20	1.3	77.7	0.0	3.7	0.0	4896.9	132615.0
3l-cvrp21	3.1	59.0	0.0	7.0	0.0	4948.3	73408.5
3l-cvrp22	3.8	42.7	0.0	9.6	0.0	5428.8	133397.0
3l-cvrp23	3.9	46.9	0.0	5.8	0.0	5175.2	76396.3
3l-cvrp24	6.1	18.9	0.0	11.7	0.0	6303.0	31119.1
3l-cvrp25	3.1	68.7	0.0	6.6	0.0	6760.7	102354.9
3l-cvrp26	2.1	55.9	0.0	3.4	0.0	5157.9	133668.8
3l-cvrp27	3.8	57.8	0.0	10.7	0.0	7358.7	95972.9

Table 12.5: DING-LDS : CPU Time Allocation, # feasible routes, # routes in Feasibility Store ($\lvert\Psi\rvert$)

12.5.4.4 *Results obtained when solving $SCP(\mathcal{R}^*)$*

In ACO-HCG the final solution is obtained by solving the integer Set Covering Problem over the set of accumulated feasible routes $\mathcal{R}^*$. In DING the final solution corresponds to the best solution found in the Dive & Generate tree. Possibly better solutions could be obtained for DING, if the final solution were obtained in the same way as in ACO-HCG, that is by solving problem $SCP(\mathcal{R}^*)$ where $\mathcal{R}^*$ is the set of feasible routes accumulated during the DING execution (the resulting approach is denoted DING*). The results corresponding to this are given in the appendix (Tables A.7 and A.8). A look at these tables reveals, that both DING*-DFS and DING*-LDS improve over DING-DFS and DING-LDS. However, the improvements obtained with DING*-LDS are slightly more important. Indeed, DING*-LDS improves the results obtained with DING-LDS by up to 1.17%, while the maximal improvement (over DING-DFS) obtained with DING*-DFS is of 1.03%.
On average the performance of DING*-LDS is still worse than that of ACO-HCG. Finally the time needed to solve $SCP(\mathcal{R}^*)$ on DING-LDS is bigger than the time needed to solve $SCP(\mathcal{R}^*)$ for DING-DFS, again especially for large-scale instances, this is to be expected as the $\mathcal{R}^*$ (the set of feasible routes) is larger for DING-LDS (and thus DING*-LDS) than for DING*-DFS.

12.5.5 *Decomposition-based approach using DING-LDS*

The decomposition-based approach described in chapter 9 has been tested with DING-LDS. Parameter N has been set to $\lceil \frac{n}{5} \rceil$. The time to solve each subproblem to $min(300, \overline{n})$ CPU seconds, where $\overline{n}$ corresponds to the number of customers in the subproblem. Explicit results comparing the performance of DING-LDS and the Decomposition-based approach using DING-LDS (DECOMP DING-LDS) can be found in Table 12.6. The average solution value is improved by the decomposition-based approach in 16 instances. Plots comparing the performance of DECOMP DING-LDS to DING-LDS and to ACO-HCG can be found in the appendix (Figs. A.3). It seems that the DING method profits more from the decomposition scheme than does ACO-HCG.

Instance			**DING-LDS**			**DECOMP DING-LDS**				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
3l-cvrp01	4	15	302.02	302.02	1800.1	302.02	302.02	1836.2	0.00	0.00
3l-cvrp02	5	15	334.96	334.96	1800.0	334.96	334.96	1883.2	0.00	0.00
3l-cvrp03	4	20	385.53	387.88	1800.2	385.53	387.84	1880.4	0.00	-0.01
3l-cvrp04	6	20	437.19	437.19	1800.0	437.19	437.19	1883.4	0.00	0.00
3l-cvrp05	6	21	447.73	447.73	1800.2	447.73	447.73	1861.8	0.00	0.00
3l-cvrp06	6	21	498.16	498.16	1800.1	498.16	498.16	1894.4	0.00	0.00
3l-cvrp07	6	22	769.68	769.68	1802.1	769.68	769.68	1884.2	0.00	0.00
3l-cvrp08	6	22	845.50	845.50	1800.7	845.50	845.50	1917.7	0.00	0.00
3l-cvrp09	8	25	630.13	630.13	1800.1	630.13	630.13	1926.4	0.00	0.00
3l-cvrp10	8	29	826.66	827.25	3600.5	826.66	826.92	3725.9	0.00	-0.04
3l-cvrp11	8	29	776.19	781.17	3600.2	776.19	776.97	3746.5	0.00	-0.54
3l-cvrp12	9	30	612.25	612.25	3600.1	612.25	612.25	3770.7	0.00	0.00
3l-cvrp13	8	32	2665.33	2669.47	3601.1	2661.62	2668.03	3753.5	-0.14	-0.05
3l-cvrp14	9	32	1384.09	1406.81	3600.8	1392.08	1402.78	3791.8	0.58	-0.29
3l-cvrp15	9	32	1341.73	1347.60	3603.0	1340.66	1344.64	3735.1	-0.08	-0.22
3l-cvrp16	11	35	698.61	698.61	3600.2	698.61	698.61	3752.3	0.00	0.00
3l-cvrp17	14	40	866.40	867.10	3600.2	866.40	866.40	3700.2	0.00	-0.08
3l-cvrp18	11	44	1207.44	1213.32	3604.8	1205.96	1216.73	3790.8	-0.12	0.28
3l-cvrp19	12	50	741.31	742.44	7201.0	741.31	742.27	7383.2	0.00	-0.02
3l-cvrp20	18	71	580.38	583.19	7213.0	579.66	581.76	7305.6	-0.12	-0.25
3l-cvrp21	17	75	1086.18	1095.10	7202.1	1084.86	1087.53	7353.4	-0.12	-0.69
3l-cvrp22	18	75	1147.93	1159.37	7203.8	1145.18	1152.68	7355.7	-0.24	-0.58
3l-cvrp23	17	75	1113.45	1121.88	7202.3	1103.34	1113.32	7340.6	-0.91	-0.76
3l-cvrp24	16	75	1111.63	1122.62	7200.4	1107.15	1108.72	7381.9	-0.40	-1.24
3l-cvrp25	22	100	1382.84	1403.54	7209.7	1384.31	1389.84	7343.2	0.11	-0.98
3l-cvrp26	26	100	1566.11	1584.56	7201.6	1543.70	1563.43	7395.8	-1.43	-1.33
3l-cvrp27	23	100	1494.29	1510.34	7205.5	1492.15	1497.23	7345.4	-0.14	-0.87
AVG					4202.0			4331.1	-0.11	-0.28

Table 12.6: Comparison of Dive & Generate with Limited Discrepancy Search (DING-LDS) with Decomposition-based approach using DING-LDS (DECOMP DING-LDS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time. $g^{min/avg}$= relative percentage deviation of DECOMP DING-LDS w.r.t. DING-LDS, computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with DECOMP DING-LDS and $z^D_{min/avg}$ the results obtained with DING-LDS.

12.5.6 *Heuristic Branch & Price (HBP)*

Heuristic Branch & Price has been executed over 10 independent runs using the same time limits as in ACO-HCG. The set-up of HBP is described below.

12.5.6.1 *Set-up of HBP*

Post-optimization The post-optimization of generated feasible routes has been done using *Tabu Search*.

Column Generation The randomized savings heuristic constructs at each step a list Ω of potential route merges to execute. The construction of list Ω is stopped as soon as ν merges resulting in *bb*-feasible routes have been added. Parameter ν has been fixed to 13.
Finally the randomized savings heuristic is executed $\ell = 5$ times per column generation execution.

Search Strategy Both Depth First Search and Limited Discrepancy Search have been used. In the case of Limited Discrepancy Search: for Heuristic Branch & Price, branching on arcs, the maximum number of allowed discrepancies has been set to n, the number of customers in the considered instance. For Heuristic Branch & Price, branching on pairs, the maximum number of allowed discrepancies has been set to $n/2$.

12.5.6.2 *Comparison with ACO-HCG*

In the following we will individually compare the different Heuristic Branch & Price variants presented earlier with ACO-HCG. We consider two different branching schemes: branching on arcs (HBPA) and branching on customer pairs (HBPP). Both of these are combined with a Depth-First Search (HBPA-DFS and HBPP-DFS) and a Limited Discrepancy Search (HBPA-LDS and HBPP-LDS), resulting in a total of 4 different configurations. The comparisons with ACO-HCG are done by computing for each instance and both for ACO-HCG and the method under consideration, g^{bb}_{avg}, the average relative deviation w.r.t. the *best-bb* solution. The resulting values are then visualized in a scatter plot where each point corresponds to one problem instance. The x-coordinate of the point will correspond to g^{bb}_{avg} for ACO-HCG and the y-coordinate to g^{bb}_{avg} for the method under consideration. HBP and

ACO-HCG perform equally on the same instance if the point is on the diagonal, the HBP performs better if the point is under the diagonal, and ACO-HCG performs better if the point is above the diagonal. Statistical significance has been determined using a Wilcoxon signed-rank test with confidence level 95%. The resulting plots are visualized in Fig. 12.8. The explicit results for the 4 configurations are given in Tables A.9, A.10, A.11 and A.12 in the appendix.

As for Dive & Generate, Heuristic Branch & Price is clearly outperformed by ACO-HCG.
It is difficult to establish clear differences between HBPA and HBPP. HBPA-DFS and HBPP-DFS appear to perform very similarly. The tables show that HBPP-DFS improves HBPA-DFS on 11 out of 27 instances, but that HBPA-DFS wins on the larger instances. It should also be noted that both HBPP-DFS and HBPP-LDS are able to improve on average the *bb-best* solution on one instance by several percent, which is not the case for HBPA-DFS and HBPA-LDS. As HBPP-LDS performs even better on this particular (small-scale) instance, a possible explanation could be, that the constraints F_{bb} are such that forcing or forbidding a pair of customers to appear in a same route has a particularly strong impact in this instance.
A clear difference can be seen when comparing HBPA-DFS and HBPA-LDS. With HBPA-LDS, the relative deviation obtained w.r.t. *bb-best* is lower than the one obtained with HBPA-DFS on a majority of instances. This is particularly true for the large-scale instances. Such a result is to be expected, as Limited Discrepancy Search ensures a more balanced exploration of the Heuristic Branch & Price tree. In contrary to HBPA, the choice of the search strategy does not show a significant impact on HBPP, where HBPP-DFS and HBPP-LDS achieve similar results. In HBPP the selected pair of customers is first forced to appear in a same route (left branch) and then forbidden from doing so (right branch). With the Limited Discrepancy Search used here, the discrepancies are taken as early as possible in the search tree. Thus Limited Discrepancy Search will from time to time choose to forbid a pair of customers, as high as possible in the search tree. However forbidding a pair of customers to appear in a same route probably doesn't restrict the set of feasible solutions by a lot (depending of course also on the F_{bb} and F_{wb} constraints), especially not if the number of customers is high. Forbidding an arc on the contrary has a bigger impact.
Finally another explanation for the weakness of HBPP is the rather

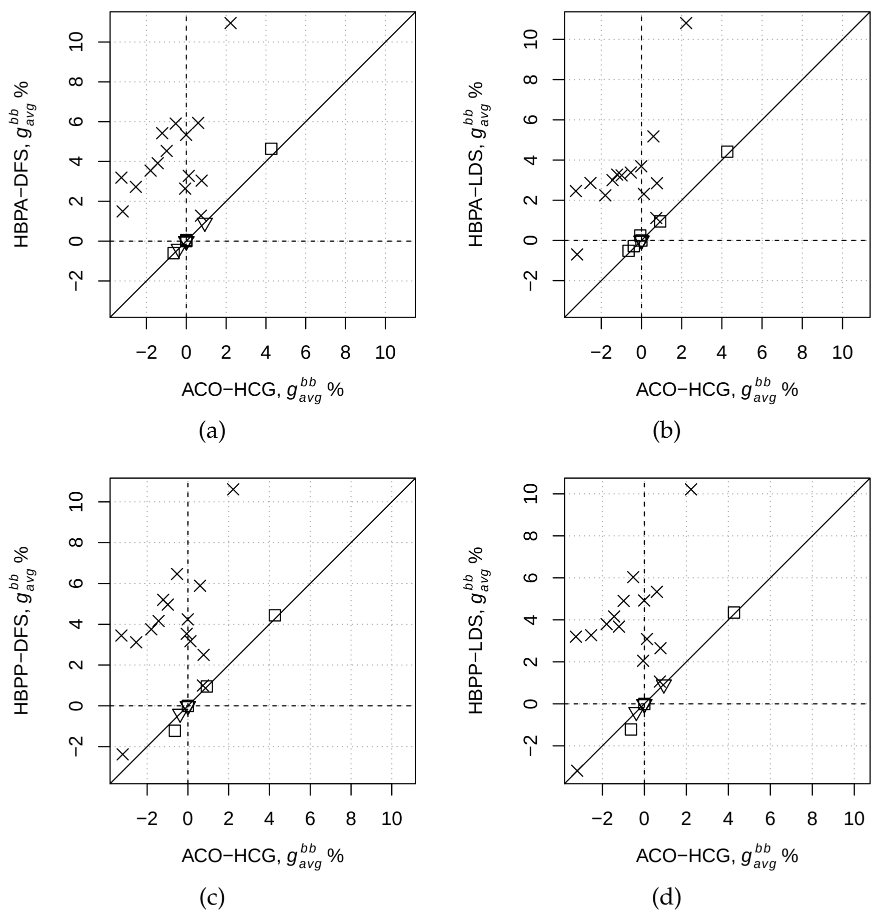
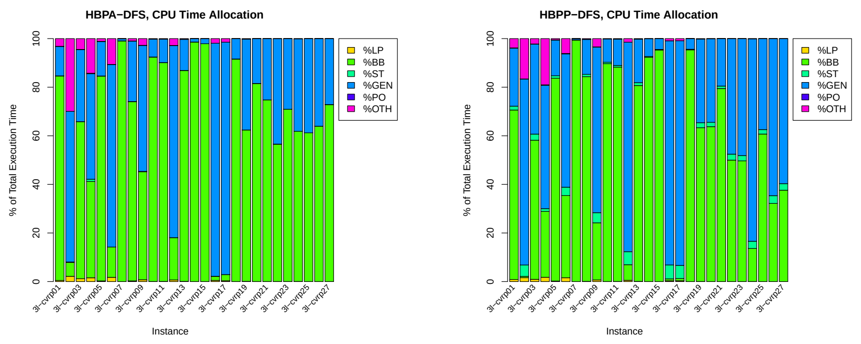


Figure 12.8: Comparison of HBP and ACO-HCG. Each point gives the average relative deviation from the *bb-best* solution over 10 independent runs. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

weak pricing mechanism used. Obviously the way the randomized savings heuristic is used to identify new feasible routes of negative reduced cost is not optimal.
Plots comparing HBPA and HBPP with ACO-HCG in the manual configuration from chapter 11 (denoted $\overline{\text{ACO-HCG}}$ in this chapter) can be found in the appendix (Figs. A.4). Neither HBPA nor HBPA are able to improve results of $\overline{\text{ACO-HCG}}$.

12.5.6.3 *Analysis of CPU time allocation and Feasibility in generated routes*

In this section the percentage of CPU time spent in different parts of the algorithm is analyzed. Figures 12.9 give a visual representation of the percentage of the total execution time allocated to different tasks in HBPA-DFS and HBPP-DFS. The plots for HBPA-LDS and HBPP-LDS have been omitted for conciseness, but can be found in the appendix (Figures A.5). They are however very similar to those given here. A detailed view of the time allocation is given in Tables 12.7 and 12.8 (and in the appendix for HBPA-LDS and HBPP-LDS, see Tables A.13 and A.14) . The tasks under consideration are the same as for ACO-HCG. Task $\%IP$ has been omitted in both figures and tables since the percentage of time spent solving $SCP(\mathcal{R}^*)$ is negligible.



(a) (b)

Figure 12.9: HBP: CPU Time Allocation

A look at the plots in Fig. 12.9 reveals that in terms of time allocation both HBPA and HBPP spend the majority of their time generating routes, and testing feasibility of routes w.r.t. F_{bb}. The time spent in the black box is higher than with DING on most instances, although this is not the case on larger scale instances when comparing HBPP-DFS with DING-LDS. Also the time used to generate routes is clearly higher and the time spent in the maintenance of the tree structure lower in HBP than in DING. It should be noted that with HBPA and HBPP column generation is executed at every non-leaf node, whereas in DING column generation is executed only if the current lower bounds falls outside a trust region, which explains why HBP spends more time generating and testing routes. Furthermore the set of customers to be visited is not reduced in the route generation as it is in DING.

When comparing HBPA and HBPP it is clear that HBPA spends significantly more time in the black box. On the other hand HBPP spends more time in the Feasibility store Ψ (%ST). The explanation for this last observation is, that in HBPA routes discovered during pricing are only checked for *bb*-feasibility in one direction. This entails that infeasible routes will not be stored as such, since they haven't been proven to be infeasible in both directions, a requirement of the Feasibility Store. A closer look reveals however that the time HBPP spends in the Feasibility store doesn't make up for the difference in time spent in the black box between HBPA and HBPP spend. The remaining difference can again be attributed to the weak pricing mechanism of HBPP.

In Figure 12.10, the percentage of routes tested for *bb*-feasibility that have been found to be feasible is illustrated for HBPP-DFS (the same plot for HBPP-LDS has been omitted for conciseness, but is very similar. It can be found in the appendix, Fig. A.6). Since a same route may be tested multiple times in HBPA, the plot for HBPA is not provided, as the values obtained are not directly comparable to those obtained for HBPP. It can be seen that the percentage of feasible routes among the tested ones for HBPP-DFS is lower on most instances compared to DING and ACO-HCG. At the same time the number of tested routes is lower, in some cases significantly lower, than for DING. Compared to ACO-HCG it very much depends on the problem instance. For some instances the number of tested routes is lower in HBPP-DFS (especially for some of the larger instances) and on others it is higher in HBPP-DFS (especially for some of the smaller instances). A possible explanation for this is that in the larger instances the pricing heuristic simply has trouble producing routes of negative reduced cost. A stronger guidance w.r.t. the dual costs would possibly produce better results.

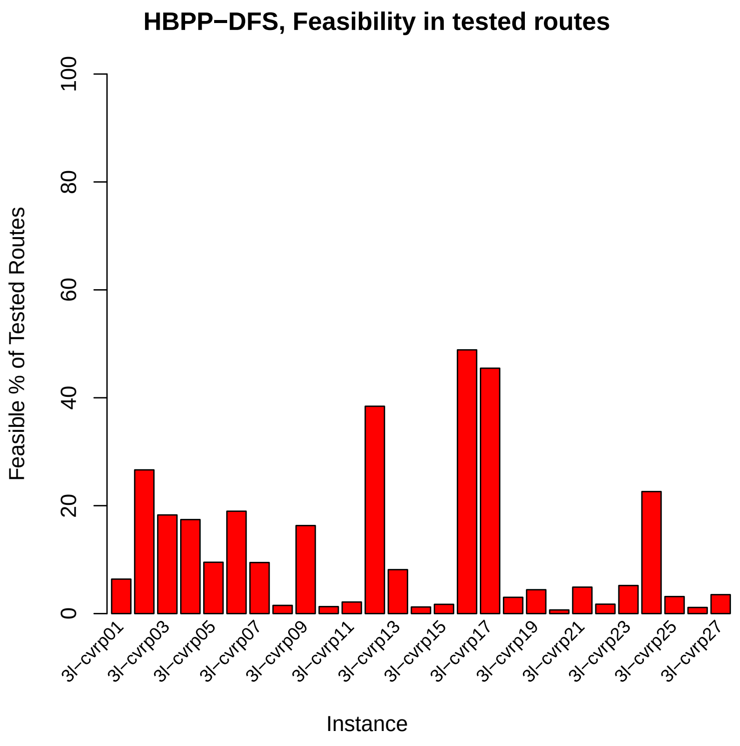


Figure 12.10: HBPP-DFS: % of tested routes that were *bb*-feasible

Finally, the exact numbers of feasible routes and routes stored in Feasibility store Ψ, for HBPA-DFS and HBPP-DFS are given in Tables 12.7 and 12.8. Note that for HBPP, $|\Psi|$ corresponds to the total number of routes checked for *bb*-feasibility. For HBPA, it corresponds to the routes that have been positively tested for *bb*-feasibility in the tree, or the ones generated to initialize $\mathcal{R}^*$ or in enforced route generation upon the start of a new tree. For most instances the number of feasible routes discovered using HBPA is somewhat, though not significantly lower. Again this can be explained by the fact that HBPA checks routes for feasibility only in one direction. Thus it misses on opportunities to discover new feasible routes (routes infeasible in one direction but feasible in the other), an opportunity used by HBPP. The tables for

HBPA-LDS and HBPP-LDS (see appendix, Tables A.13 and A.14) show that, especially for the large-scale instances, the number of routes in Ψ is higher for HBPA-LDS than for HBPA-DFS. At the same time the number of feasible routes is also higher with HBPA-LDS.
With HBPP-LDS the number of feasible routes found is higher than with HBPP-DFS on all instances. In terms of $|\Psi|$, it depends on the instance. On some instances HBPP-LDS tested more routes than HBPP-DFS, on some less. Thus on some instances, even when testing less routes HBPP-LDS managed to achieve a higher number of feasible routes than HBPP-DFS. This is probably to be attributed to characteristics of the given problem instance.

12.5.6.4 *Results obtained when solving $SCP(\mathcal{R}^*)$*

As for DING, the final solution obtained in HBP is discovered in the Heuristic Branch & Price tree. Possibly better results could be obtained if one were to compute a final solution by solving problem $SCP(\mathcal{R}^*)$ over the set of feasible routes $\mathcal{R}^*$ accumulated during the HBP execution (the corresponding approach is denoted HBP*). The results corresponding to this are given in the appendix (tables A.15 to A.18). As for DING, it can be observed that HBP* improves over HBP. The improvements are bigger with HBPA*-LDS and HBPP*-LDS than with HBPA*-DFS and HBPP*-DFS. An interesting observation is that the highest improvements are obtained with HBPP*-LDS. Solving $SCP(\mathcal{R}^*)$ allows to improve the results of HBPP-LDS up to 4.03%. Also the results obtained with HBPP*-LDS improve (on average) over those obtained with HBPP*-DFS in 16 out of 27 instances. Furthermore the results of HBPP*-LDS improve (on average) over those of HBPA*-LDS in 14 out of 27 instances. The execution times needed to solve $SCP(\mathcal{R}^*)$ are similar for all configurations.

12.5.7 ***Comparison of all methods***

Table 12.9 compares the results obtained on all considered approaches (except the decomposition-based approach) using the configurations described in this section with the results obtained using the Hybrid Tabu Search (HTS) presented in [Bor12]. It is the loading algorithm described in this paper, that has been reimplemented as black box function and is used in the experiments described in this section. Columns z_{min} and z_{avg} give the best and the average solution cost determined

Instance	**% Total Execution Time**					**Routes in Ψ**	
	$\%LP$	$\%BB$	$\%ST$	$\%GEN$	$\%PO$	#Feasible routes	$\|\Psi\|$
3l-cvrp01	0.5	84.2	0.1	12.1	0.1	1753.1	14934.6
3l-cvrp02	2.2	5.7	0.2	62.0	0.0	867.1	2525.9
3l-cvrp03	1.2	64.6	0.0	29.6	0.1	2880.6	10886.7
3l-cvrp04	1.6	39.6	0.9	43.4	0.2	2808.1	17322.5
3l-cvrp05	0.3	84.3	0.0	14.2	0.1	1996.6	10183.2
3l-cvrp06	1.7	12.5	0.0	75.1	0.0	2051.2	5417.1
3l-cvrp07	0.0	99.1	0.0	0.8	0.1	1605.8	13852.5
3l-cvrp08	0.3	73.8	0.0	24.9	0.0	1121.3	6041.0
3l-cvrp09	0.8	44.5	0.2	51.8	0.1	2164.4	13123.9
3l-cvrp10	0.0	92.4	0.0	7.4	0.0	1131.8	3498.7
3l-cvrp11	0.1	90.1	0.0	9.6	0.0	1586.2	5064.7
3l-cvrp12	0.7	17.4	0.0	79.1	0.0	2002.5	5397.4
3l-cvrp13	0.1	86.8	0.0	12.8	0.0	2574.8	18556.6
3l-cvrp14	0.0	98.6	0.0	1.4	0.0	1043.2	4259.7
3l-cvrp15	0.0	97.9	0.0	2.1	0.0	1276.7	4527.5
3l-cvrp16	0.5	1.7	0.0	95.9	0.0	2290.9	5344.6
3l-cvrp17	0.3	2.5	0.1	95.6	0.0	2430.2	5948.1
3l-cvrp18	0.0	91.6	0.0	8.3	0.0	1135.0	6675.5
3l-cvrp19	0.1	62.3	0.0	37.4	0.0	2848.1	20831.2
3l-cvrp20	0.0	81.5	0.0	18.4	0.0	1597.9	10137.2
3l-cvrp21	0.0	74.8	0.0	25.2	0.0	1759.6	9922.2
3l-cvrp22	0.0	56.5	0.0	43.3	0.0	1748.6	10029.7
3l-cvrp23	0.0	70.9	0.0	29.0	0.0	1713.9	7816.0
3l-cvrp24	0.1	61.8	0.0	38.1	0.0	2284.7	4956.2
3l-cvrp25	0.0	61.2	0.0	38.7	0.0	2017.4	13635.4
3l-cvrp26	0.0	65.4	0.0	34.6	0.0	1723.0	15323.6
3l-cvrp27	0.0	75.9	0.0	24.0	0.0	1871.2	10432.6

Table 12.7: HBPA-DFS : CPU Time Allocation, # feasible routes, # routes in Feasibility Store ($|\Psi|$)

Instance	**% Total Execution Time**					**Routes in Ψ**	
	%LP	*%BB*	*%ST*	*%GEN*	*%PO*	#Feasible routes	$\lvert\Psi\rvert$
3l-cvrp01	0.8	69.8	1.6	23.9	0.1	1583.0	24814.6
3l-cvrp02	1.6	0.5	4.7	76.6	0.0	619.2	2325.0
3l-cvrp03	0.9	57.3	2.5	37.0	0.1	2204.5	12069.4
3l-cvrp04	1.8	27.1	1.2	50.7	0.2	2792.2	16027.2
3l-cvrp05	0.2	83.5	1.0	14.7	0.1	1852.5	19501.1
3l-cvrp06	1.6	33.8	3.4	54.9	0.2	2874.0	15154.3
3l-cvrp07	0.0	99.4	0.0	0.5	0.1	1566.9	16591.1
3l-cvrp08	0.1	84.3	1.0	14.4	0.0	841.7	55667.6
3l-cvrp09	0.7	23.5	4.2	68.2	0.1	2132.8	13071.6
3l-cvrp10	0.0	89.7	0.6	9.6	0.0	1021.8	80189.1
3l-cvrp11	0.0	88.2	0.7	10.8	0.0	1992.3	92833.7
3l-cvrp12	0.5	6.4	5.4	86.2	0.0	1811.8	4717.2
3l-cvrp13	0.1	80.6	1.2	17.9	0.1	2352.3	28909.1
3l-cvrp14	0.0	92.3	0.4	7.3	0.0	1031.8	88901.5
3l-cvrp15	0.0	95.3	0.2	4.4	0.0	1054.0	64104.0
3l-cvrp16	0.3	0.8	5.7	92.2	0.0	1826.4	3736.6
3l-cvrp17	0.3	0.9	5.4	92.4	0.0	2206.3	4852.4
3l-cvrp18	0.0	95.4	0.3	4.4	0.0	963.4	34380.2
3l-cvrp19	0.0	63.3	2.0	34.5	0.0	2578.7	61575.6
3l-cvrp20	0.0	63.8	1.8	34.4	0.0	1517.9	233785.4
3l-cvrp21	0.0	79.4	1.0	19.5	0.0	1787.6	38050.9
3l-cvrp22	0.0	50.0	2.4	47.5	0.0	1695.9	98111.1
3l-cvrp23	0.0	49.7	2.2	48.1	0.0	2031.6	39846.0
3l-cvrp24	0.0	13.6	2.9	83.3	0.0	2674.1	12093.1
3l-cvrp25	0.0	60.7	1.8	37.4	0.0	2201.5	72562.0
3l-cvrp26	0.0	33.1	3.1	63.7	0.0	1756.3	143999.8
3l-cvrp27	0.0	44.6	2.3	52.9	0.0	2134.7	58269.5

Table 12.8: HBPP-DFS : CPU Time Allocation, # feasible routes, # routes in Feasibility Store ($\lvert\Psi\rvert$)

over 10 runs for all methods and for HTS. Note that the values in the HTS z_{min} column correspond to the *bb-best* values considered in the previous parts of this section. Columns sec_{tt} give the total execution times. They are only indicated for HTS and ACO-HCG as the values for DING and HBP are very similar to those of ACO-HCG. ACO-HCG is able to improve over the best solution from HTS in 14 out of 27 instances, and able to find the best solution from HTS in 8 instances. Thus in terms of best solution HTS outperforms ACO-HCG in only 5 instances out of 27. In terms of average solution value, ACO-HCG improves the HTS results in 24 out of 27 instances. DING-LDS performs very similar to ACO-HCG, but DING-DFS performs slightly less good. Both are able to improve the best solution from HTS in 8, respectively 12 instances. In terms of average solution DING-LDS improves the HTS value in 24 instances, but DING-DFS only in 11. The HBP methods perform worst. They improve the best HTS solution only on few instances, and the average HTS value on a handful. It is very clear that ACO-HCG and DING-LDS outperform the remaining methods, especially on large scale instances. As noted previously HTS is very time-efficient.

Table 12.10 shows, in columns $\%g^{allb}_{avg}$, the average relative deviation from the overall best known solution from the state-of-the-art for the 3L-CVRP (this doesn't include results published in [MDVH12] or [MLISD13]). The relative deviation for one run is computed as $100 \cdot \frac{z-z^{allb}}{z^{allb}}$, where z corresponds to the solution cost obtained using the methods presented in this thesis and z^{allb} corresponds to the best known solution cost. The table indicates in column z^{allb} for each instance the best known solution cost, and in columns z^{min} the best solution cost found over all runs using the corresponding method. Negative values for $\%g^{allb}_{avg}$ indicate that the corresponding method improves the best known solution on average. ACO-HCG improves or ties with the best known solution in 16 out of 27 instances. It is also able to improve on average the best known in 6 instances. The results obtained with the other methods are less good. While they all allow to tie with or even improve the best known solution in a handful instances, the average relative deviation from the best known is higher than for ACO-HCG. Note that for ACO-HCG, this average gap w.r.t. the best known solution, averaged over all instances, amounts to only 0.9%, while it amounts to 1.3% for DING-LDS.

Instance	HTS			ACO-HCG			DING-DFS		DING-LDS		HBPA-DFS		HBPA-LDS		HBPP-DFS		HBPP-LDS	
	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	z_{min}	z_{avg}	z_{min}	z_{avg}	z_{min}	z_{avg}	z_{min}	z_{avg}	z_{min}	z_{avg}
3l-cvrp01	302.02	302.02	72.3	**302.02**	*302.02*	1800.0	**302.02**	*302.02*	**302.02**	*302.02*	**302.02**	*302.02*	**302.02**	*302.02*	**302.02**	*302.02*	**302.02**	*302.02*
3l-cvrp02	334.96	334.96	0.9	**334.96**	*334.96*	1800.0	**334.96**	*334.96*	**334.96**	*334.96*	**334.96**	*334.96*	**334.96**	*334.96*	**334.96**	*334.96*	**334.96**	*334.96*
3l-cvrp03	392.63	401.44	182.0	**385.53**	*390.12*	1800.3	**385.53**	*389.56*	**385.57**	*387.88*	**388.09**	*390.25*	**385.53**	*390.61*	**385.53**	*387.84*	**385.53**	*387.84*
3l-cvrp04	437.19	437.54	16.1	**437.19**	*437.19*	1800.0	**437.19**	*437.19*	**437.19**	*437.19*	**437.19**	*437.19*	**437.19**	*437.19*	**437.19**	*437.19*	**437.19**	*437.19*
3l-cvrp05	443.61	451.03	182.6	447.73	*447.73*	1800.2	447.73	*447.73*	447.73	*447.73*	447.73	*447.73*	447.73	*447.82*	447.73	*447.82*	447.73	*447.73*
3l-cvrp06	498.16	498.38	23.6	**498.16**	*498.16*	1800.1	**498.16**	*498.16*	**498.16**	*498.16*	**498.16**	*498.16*	**498.16**	*498.16*	**498.16**	*498.16*	**498.16**	*498.16*
3l-cvrp07	769.68	772.49	133.1	**769.68**	*769.68*	1800.4	**769.68**	*769.68*	**769.68**	*769.68*	**769.68**	*770.06*	**769.68**	*769.68*	**769.68**	*769.68*	**769.68**	*769.68*
3l-cvrp08	810.89	821.35	139.1	845.50	845.50	1800.6	845.50	845.50	845.50	845.50	845.50	848.49	845.50	846.69	845.50	846.86	845.50	846.16
3l-cvrp09	630.13	645.81	24.3	**630.13**	*630.13*	1800.0	**630.13**	*631.38*	**630.13**	*630.13*	**630.13**	*630.13*	**630.13**	*630.13*	**630.13**	*630.13*	**630.13**	*630.13*
3l-cvrp10	820.35	827.29	175.1	826.66	*826.66*	3603.2	826.66	833.59	826.66	*827.25*	829.24	845.28	835.74	843.71	828.75	840.87	828.75	842.09
3l-cvrp11	803.61	815.62	136.4	**776.19**	*777.91*	3601.5	**778.10**	*786.29*	**776.19**	*781.17*	**787.04**	*815.61*	**778.24**	*798.03*	**776.19**	*784.50*	**776.19**	*778.00*
3l-cvrp12	614.59	630.46	14.0	**612.25**	*612.25*	3600.3	**612.25**	*612.25*	**612.25**	*612.25*	**612.25**	*612.25*	**612.25**	*612.84*	**612.25**	*612.25*	**612.25**	*612.25*
3l-cvrp13	2645.95	2694.81	268.4	2661.62	*2665.48*	3601.6	2665.33	*2671.16*	2665.33	*2669.47*	2665.33	*2679.80*	2670.50	*2675.48*	2665.33	*2672.30*	2670.50	*2674.04*
3l-cvrp14	1368.42	1413.59	311.6	1385.00	*1398.84*	3605.3	1401.64	1417.16	1384.09	*1406.81*	1497.31	1518.29	1477.98	1516.36	1437.55	1513.65	1428.57	1508.20
3l-cvrp15	1341.14	1355.50	311.5	**1336.22**	*1341.02*	3624.1	1342.32	1351.72	1341.73	*1347.60*	1358.00	1412.71	1349.73	1390.63	1356.08	1398.00	1364.67	1407.16
3l-cvrp16	698.61	705.05	3.4	**698.61**	*698.61*	3600.3	**698.61**	*700.40*	**698.61**	*698.61*	**698.61**	*698.61*	**698.61**	*698.61*	**698.61**	*698.61*	**698.61**	*698.61*
3l-cvrp17	866.40	917.96	2.5	**866.40**	*866.40*	3600.1	**866.40**	*874.09*	**866.40**	*867.10*	**866.40**	*866.40*	**866.40**	*866.40*	**866.40**	*866.40*	**866.40**	*866.40*
3l-cvrp18	1207.72	1228.98	309.5	**1205.11**	*1209.21*	3606.2	1209.81	1233.29	**1207.44**	*1213.32*	1225.62	1247.29	1226.40	1235.68	1226.42	1245.96	1225.46	1245.03
3l-cvrp19	741.74	753.87	416.5	**741.31**	*741.31*	7204.6	**741.31**	*753.39*	**741.31**	*742.44*	**741.31**	761.29	**741.31**	*743.58*	746.21	767.94	743.62	756.96
3l-cvrp20	587.95	596.42	427.0	**577.85**	*580.84*	7217.9	**583.29**	603.94	**580.38**	*583.19*	602.15	619.82	594.50	607.20	601.55	618.53	593.12	609.58
3l-cvrp21	1090.22	1107.00	443.4	**1075.97**	*1079.47*	7220.4	1100.87	1125.64	**1086.18**	*1095.10*	1113.60	1139.60	1103.02	1125.50	1108.16	1144.34	1109.38	1143.71
3l-cvrp22	1147.80	1171.49	423.5	**1147.43**	*1154.64*	7217.8	1167.28	1202.89	1147.93	*1159.37*	1190.92	1215.92	1185.80	1207.22	1184.46	1215.38	1184.05	1209.08
3l-cvrp23	1130.54	1135.46	425.8	**1102.39**	*1110.24*	7221.1	**1119.59**	1151.58	**1113.45**	*1121.88*	1138.47	1170.57	**1128.80**	1155.91	1158.35	1172.95	1158.13	1173.47
3l-cvrp24	1116.13	1128.82	411.1	**1107.71**	*1110.20*	7228.3	1131.51	1156.91	**1111.63**	*1122.62*	1164.36	1182.02	1120.76	1153.84	1168.77	1188.30	1154.37	1183.50
3l-cvrp25	1407.36	1428.82	453.0	**1370.70**	*1387.14*	7256.1	1420.01	1437.07	**1382.84**	*1403.54*	1444.22	1462.44	1421.81	1449.56	1444.22	1465.93	1444.22	1465.93
3l-cvrp26	1600.35	1625.31	430.6	**1541.49**	*1548.15*	7212.9	**1594.20**	1637.32	**1566.11**	*1584.56*	1635.49	1651.40	1601.09	1639.66	1645.05	1655.50	1614.04	1651.56
3l-cvrp27	1529.86	1550.85	435.0	**1483.66**	*1491.09*	7253.1	**1519.41**	1554.80	**1494.29**	*1510.34*	1541.37	1571.54	1541.37	1573.55	1548.00	1577.50	1548.00	1579.90
AVG			228.6			4210.2												

Table 12.9: Comparison of proposed methods with HTS ([Bor12]). The best known results in [Bor12] correspond to the *bb-best* results for 3L-CVRP. $z_{min/avg}$ corresponds to the best/average solution value over 10 runs. sec_{tt} corresponds to the average total execution time. sec_{tt} is indicated for HTS and ACO-HCG only. Remaining execution times are equal (give or take 5 seconds) to ACO-HCG. Values in **bold** in the z_{min} columns indicate that the corresponding approach is a tie with or improves the *bb-best*. Values in *italic* in the z_{avg} column indicate that the corresponding approach improves or ties with the average results published in [Bor12].

Instance	Best known	ACO-HCG		DING-DFS		DING-LDS		HBPA-DFS		HBPA-LDS		HBPP-DFS		HBPP-LDS	
	z_{allb}	z_{min}	$\%g_{avg}^{allb}$	z_{min}	$\%g_{avg}^{allb}$	z_{min}	$\%g_{avg}^{allb}$	z_{min}	$\%g_{avg}^{allb}$	z_{min}	$\%g_{avg}^{allb}$	z_{min}	$\%g_{avg}^{allb}$	z_{min}	$\%g_{avg}^{allb}$
3l-cvrp01	291.00	302.02	3.8	302.02	3.8	302.02	3.8	302.02	3.8	302.02	3.8	302.02	3.8	302.02	3.8
3l-cvrp02	334.96	**334.96**	0.0	**334.96**	0.0	**334.96**	0.0	**334.96**	0.0	**334.96**	0.0	**334.96**	0.0	**334.96**	0.0
3l-cvrp03	392.63	**385.53**	-0.6	**385.53**	-0.8	**387.88**	-1.21	**388.09**	-0.6	**385.53**	-0.5	**385.53**	-1.2	**385.53**	-1.2
3l-cvrp04	437.19	**437.19**	0.0	**437.19**	0.0	**437.19**	0.0	**437.19**	0.0	**437.19**	0.0	**437.19**	0.0	**437.19**	0.0
3l-cvrp05	443.61	447.73	0.9	447.73	0.9	447.73	0.9	447.73	0.9	447.73	0.9	447.73	0.9	447.73	0.9
3l-cvrp06	498.16	**498.16**	0.0	**498.16**	0.0	**498.16**	0.0	**498.16**	0.0	**498.16**	0.0	**498.16**	0.0	**498.16**	0.0
3l-cvrp07	768.85	769.68	0.1	769.68	0.1	769.68	0.1	769.68	0.2	769.68	0.1	769.68	0.1	769.68	0.1
3l-cvrp08	805.35	845.50	5.0	845.50	5.0	845.50	5.0	845.50	5.4	845.50	5.1	845.50	5.2	845.50	5.1
3l-cvrp09	630.13	**630.13**	0.0	**630.13**	0.0	**630.13**	0.0	**630.13**	0.0	**630.13**	0.0	**630.13**	0.0	**630.13**	0.0
3l-cvrp10	820.35	826.66	0.8	826.66	1.61	826.66	0.8	829.24	3.0	835.74	2.8	828.75	2.5	828.75	2.6
3l-cvrp11	786.19	**776.19**	-1.1	**778.10**	0.0	**776.19**	-0.6	787.04	**3.7**	**778.24**	1.5	**776.19**	-0.2	**776.19**	-1.0
3l-cvrp12	612.25	**612.25**	0.0	**612.25**	0.0	**612.25**	0.0	**612.25**	0.0	**612.25**	0.1	**612.25**	0.0	**612.25**	0.0
3l-cvrp13	2645.95	2661.62	0.7	2665.22	1.0	2665.33	0.9	2665.33	1.3	2670.50	1.1	2665.33	1.0	2670.50	1.1
3l-cvrp14	1368.42	1385.00	2.2	1401.64	3.6	1384.09	2.8	1497.31	11.0	1477.98	10.8	1437.55	10.6	1428.57	10.2
3l-cvrp15	1341.14	**1336.22**	0.0	1342.32	0.8	1341.73	0.5	1358.00	5.3	1349.73	3.7	1356.08	4.2	1364.67	4.9
3l-cvrp16	698.61	**698.61**	0.0	**698.61**	0.3	**698.61**	0.0	**698.61**	0.0	**698.61**	0.0	**698.61**	0.0	**698.61**	0.0
3l-cvrp17	866.40	**866.40**	0.0	**866.40**	0.9	**866.40**	0.1	**866.40**	0.0	**866.40**	0.0	**866.40**	0.0	**866.40**	0.0
3l-cvrp18	1207.72	**1205.11**	0.1	1209.81	2.1	**1207.44**	0.5	1225.62	3.3	1226.40	2.3	1226.42	3.2	1225.46	3.1
3l-cvrp19	741.74	**741.31**	-0.1	**741.31**	1.6	**741.31**	0.1	**741.31**	2.6	**741.31**	0.2	746.21	3.5	743.62	2.1
3l-cvrp20	587.95	**577.85**	-1.2	**583.29**	2.7	**580.38**	-0.8	602.15	5.4	594.50	3.3	601.55	5.2	593.12	3.7
3l-cvrp21	1042.72	1075.97	3.5	1100.87	8.0	1086.18	5.0	1113.60	9.3	1103.02	7.9	1108.16	9.7	1109.38	9.7
3l-cvrp22	1147.80	**1147.43**	0.6	1167.28	4.8	1147.93	1.0	1190.92	5.9	1185.80	5.2	1184.46	5.9	1184.05	5.3
3l-cvrp23	1130.54	**1102.39**	-1.8	**1119.59**	1.9	**1113.45**	-0.8	1138.47	3.5	**1128.80**	2.2	1158.35	3.8	1158.13	3.8
3l-cvrp24	1096.88	1107.71	1.2	1131.51	5.5	1111.63	2.4	1164.36	7.8	1120.76	5.2	1168.77	8.3	1154.37	7.9
3l-cvrp25	1407.36	**1370.70**	-1.4	1420.01	2.1	**1382.84**	-0.3	1444.22	3.9	1421.81	3.0	1444.22	4.2	1444.22	4.2
3l-cvrp26	1430.15	1541.49	8.3	1594.20	14.5	1566.11	10.8	1635.49	15.5	1601.09	14.6	1645.05	15.8	1614.04	15.5
3l-cvrp27	1455.27	1483.66	2.5	1519.41	6.8	1494.29	3.78	1541.37	8.0	1541.37	8.1	1548.00	8.4	1548.00	8.6
AVG			0.9		2.5		1.3		3.7		3.0		3.5		3.3

Table 12.10: Comparison of proposed methods with overall best known results (*all-best*) for 3L-CVRP. z_{min} corresponds to the best solution value (over 10 runs for the proposed methods). $\%g_{avg}^{allb}$ corresponds to the average relative percentage deviation from the best known result. Negative $\%g_{avg}^{allb}$ values indicate that the best known solution is improved on average. Values in **bold** in the z_{min} columns indicate that the corresponding approach is a tie with or improves the best known.

13 MULTI-PILE VEHICLE ROUTING PROBLEM

The Multi-Pile Vehicle Routing Problem (MP-VRP) was first introduced in [DFH+07] and is derived from a real-world problem encountered at an Austrian wood-products retailer. To verify the feasibility of a route, a one-dimensional loading problem has to be solved. In this problem customers request a given quantity for each of the available types of chipboards. For each customer, the requested chipboards of the same type are then loaded onto pallets, which themselves must be loaded in the back of the delivery truck, which is divided in mutiple piles. The decision on how the chipboards are loaded onto the pallets is heuristic and deterministic. Finally a set of items, each corresponding to one or several loaded pallets, must be delivered to each customer, and a feasible loading for these items on the multiple piles of the truck must exist. The problem that must be solved to achieve this corresponds to a one-dimensional loading problem (which is actually a generalization of a well-known scheduling problem).

13.1 PROBLEM DESCRIPTION

The MP-VRP is defined on top of the basic CVRP. A homogeneous fleet of vehicles is available to perform the deliveries of items to customers. The number of vehicles is not limited, i.e. $|K| = |V \backslash \{0\}|$. Each vehicle corresponds to a container of width W, height H and length L. The

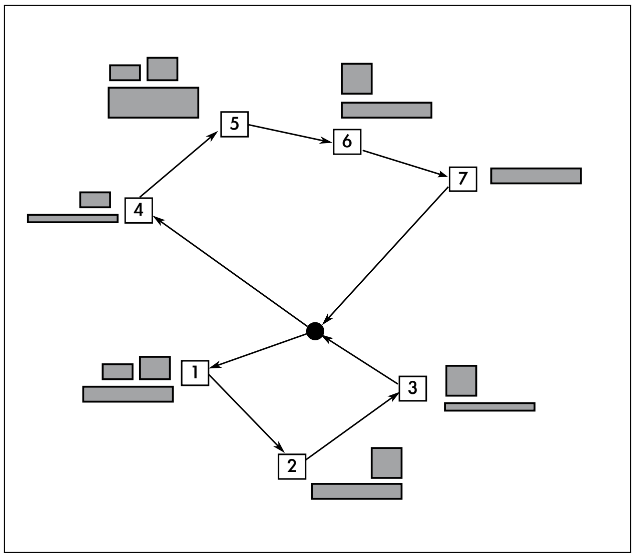


Figure 13.1: MP-VRP instance with routes

capacity Q of the vehicles is considered infinite ($Q = \infty$). The length of each vehicle is divided in p symmetric piles, each of width W. The height of those piles is limited by the height H of the vehicle. Each customer i $(i \in V \backslash \{0\})$ demands a set I_i of m_i items each of length $l_{iz} \in \{L/p, L\}$, width W and height $h_{iz}(z = 1, ..., m_i)$. That is, all items take up the entire width of the vehicle, and take up either the length of one pile or the length of the vehicle. See Figure 13.1 for the depiction of an MP-VRP instance with routes. The goal is to find a set of routes each starting and ending at the depot, visiting every customer exactly once such that the total cost is minimized and the following condition holds for every route r:
a feasible loading exists for $\bigcup_{i \in r.S} \bigcup_{z=1}^{m_i} \{I_{iz}\}$ on the p piles.

A loading of a route r is the assignment of a pile and a coordinate (height) to the lower left corner of each item $I_{iz}(z = 1, ..., m_i), i \in r.S$. The following conditions must be fulfilled over the items $\bigcup_{i \in r.S} \{I_i\}$ for a loading to be considered feasible:

- *Containment* All items fit completely into the loading space of the vehicle.
- *Non-overlapping* None of the items overlap.

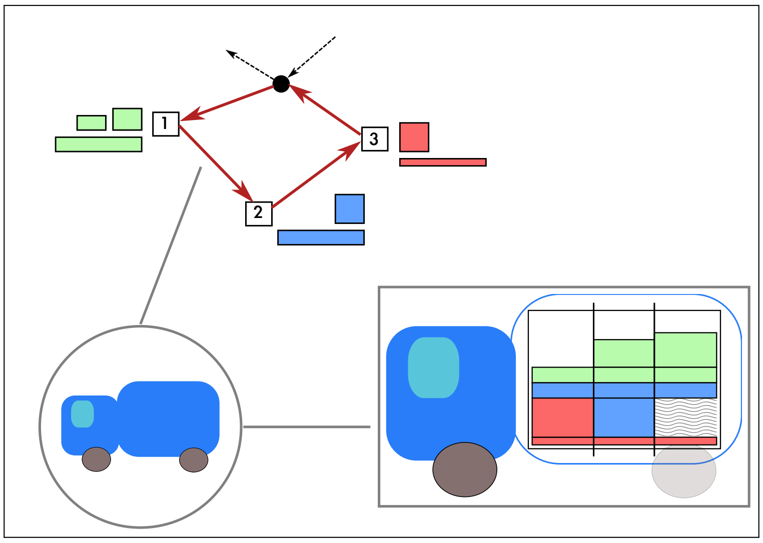


Figure 13.2: Feasible loading for a MP-VRP route

- *Sequential Loading* When visiting a customer $i \in r.S$ no item $I_{jz}(z = 1, ..., m_j)$ s.t. $pos(j, r) > pos(i, r)$ may be placed between the items of customer i and the top of their allocated pile.

Note that there is no constraint on the support of items from below. A feasible loading can contain holes, which will later on be filled out with bulk material. A feasible loading for a given route is depicted in Figure 13.2.

13.2 EXISTING APPROACHES

Doerner et al. introduced the problem in [DFH+07] and in the same paper propose a Tabu Search (TS) algorithm as well as an Ant Colony Optimization (ACO) approach.
For both TS and ACO they propose a heuristic and a basic dynamic programming approach to solve the loading problem. The heuristic is based on a branch and bound method for the $P||C_{max}$ scheduling problem, of which the loading problem encountered in the MP-VRP is a generalization. It is this heuristic that was used in both the TS and ACO.

The ACO approach is derived from the savings-based ants [RSD02]. At each iteration a colony of ants uses the savings heuristics to construct feasible solutions (i.e. respecting the loading constraints). To compute the attractiveness of a merge m introducing arc (i, j) the ants use two pheromones matrices, one dedicated to the total distance, and one to the total packing height (summation of packing heights reached in all the vehicles). Also, two types of heuristic information intervene in the attractiveness of a merge. The heuristic information associated with the distance is the classical savings value s_{ij}. The heuristic information associated with the loading is obtained by combining the minimum height a loading for the items of customers i and j can reach, as well as the amount of bulk material necessitated in such a loading. The solutions constructed by the ants are post-optimized using η_{reloc}, η_{swap} and η_{2ex} neighborhoods. Finally the F best solutions in terms of total distance and loading are used to update the respective pheromone matrices.

The TS approach is deterministic. The initial solution is computed using a simple savings heuristic, accepting only merges leading to feasible routes. The tabu search is however allowed to move towards infeasible solutions. A measure of infeasibility is included in the objective function as the excess height over all the vehicles in a solution. The weight accorded to this measure is adapted throughout the search in order to force it into feasible regions. The restricted neighborhood is constructed using the η_{reloc} operator. The routes implied in this move are then post-optimized using 4-opt insertion. The best of these moves, taking into account the objective of the resulting solution and a penalty term used to enhance diversification, is chosen. Finally to enhance intensification, each time a new best solution is discovered, the size of the neighborhood is increased during the next iteration.

Both algorithms are tested on a set of randomly generated benchmark instances as well as on real-world instances. The ACO algorithm beats the TS in 17 out of the 21 random instances and 4 out of the 5 real-world instances, both while taking (significantly in the case of the real world instances) less time to converge.

In [TDHI09] the authors propose a Variable Neighborhood Search (VNS) for the MP-VRP. They use several heuristics to provide lower and upper bounds on the height of a giving loading, as well as a dynamic programming approach to exactly solve the loading problem.

A feasible solution is constructed using the savings heuristic. This solution is then used as initial solution for their VNS, in which they consider neighborhoods of increasing size. All neighborhoods are constructed by exchanging route segments. The size of these segments increases over the different neighborhoods. At each iteration, first a random neighboring solution from the current neighborhood is selected. This solution is accepted if the infeasibility of the loading problem is not proved using lower bounds. This solution is then reoptimized using 2-opt moves. The loading feasibility of the ensuing locally optimal solution is verified using upper bounds. Here, since the upper bounds can overestimate the height of a loading, it is acceptable that the height computed with the upper bounds exceed the height of the vehicle by a certain percentage. If the local optimum improves the best known solution so far, but the loading feasibility has not been proved using the upper bounds on the loading height, the exact dynamic programming approach is used to determine feasibility. If the local optimum should be proven not to be feasible, a repairing procedure is applied in the hopes to be able to end up with a feasible and still new best solution.

The authors also present a branch & cut approach for the problem. First they run their VNS algorithm. The resulting solution is used to initialize the upper bound. They also use their pool of routes proven infeasible to initialize cuts corresponding to loading feasibility constraints. Subtour elimination cuts are generated throughout the procedure.

Using the VNS approach the authors obtain good results when compared to ACO and TS, improving 19 out of the 21 random instances and 4 out of the 5 real-world instances. The Branch & Cut approach is tested on partial instances, due to its time complexity and the size of the original instances. They are able to show that their VNS solved more than half of the considered instances to optimality.

13.3 MP-VRP AS A VRPBB

It is easy to see that the MP-VRP can be decomposed into a routing and a loading problem. The set of white box constraints F_{wb} is empty, since no capacity constraints need to be considered. The unknown black box constraints F_{bb} to be respected by every route, correspond to the constraint that a feasible loading must exist for the items to be delivered on the route.

The authors from [TDHI09] kindly provided us with their implementation for the Loading Feasibility Check. First feasibility is tested by computing an upper bound on the height of the loading. This is done using the heuristics presented in [DFH+07]. If the upper bound cannot prove the feasibility the exact dynamic programming method is used. It establishes a dominance criterion based on the heights of the different piles, which are sorted in non-decreasing order by height. Dominance between two partial loadings is then established by comparing the height of the piles. The authors furthermore introduce a pruning mechanism using a set of lower bounds stemming from the $P||C_{max}$. The upper bound heuristic is of temporal complexity linear in the number of items to be loaded. The exact algorithm runs in a time exponential in the number of items, and thus also in the size of the route. This is why in this thesis a time limit (set to 5 CPU seconds) is enforced on the execution of the exact algorithm. The loading check provided for the VRPBB here is thus not exact, i.e. feasible routes may be considered infeasible.

13.3.1 *Problem-specific knowledge in existing approaches*

The ACO algorithm uses loading-specific heuristic information as well as a loading-specific pheromone matrix. The TS only uses loading-specific information to measure infeasibility in the objective function. The VNS uses specific knowledge about its feasibility checks. Lower and upper bounds are used strategically to reduce computation time. The exact method is used only when unavoidable.

13.4 PROBLEM INSTANCES

The benchmark instances used for the MP-VRP are those presented in [DFH+07]. The instances were generated based on well-known Capacitated VRP instances from the literature. The problem instances define four types of chipboards. Each customer demands a given number of chipboards of each type. Three different classes of customers were defined, customers in the first class ordering a small number of chipboards, in the second class ordering a medium amount and in the third class ordering a large amount of chipboards. For each of these classes and each type of chipboard an interval delimiting the possible number of chipboards ordered was then defined. For each customer, the number of chipboards demanded per type is then drawn uniformly at random from the interval corresponding to the customer's class and the chipboard type. For each original CVRP instance three MP-VRP instances were created by considering different distributions of customers in one of the three customer classes. The vehicle height was fixed to $H = 200$ and the number of piles to $p = 3$. The height of the different types of chipboards varies between 1 and 5, the height of a pallet is 5. The instances can be downloaded from `http://prolog.univie.ac.at/research/VRPandBPP/`.

As previously pointed out, solving the loading problem consists of packing the different chipboards unto pallets and then loading these pallets into the truck. Both the construction of pallets and loading are part of the black box in the case of this thesis.

13.5 EXPERIMENTAL RESULTS

The experiments presented in this section have all been conducted on an AMD Opteron 6284 SE CPU (2.7 Ghz, 16MB cache size). The system has been compiled using gcc 4.4.7 and uses CPLEX 12.4 as MILP solver. Results have been obtained over 10 independent runs. The solutions corresponding to the results presented in this section can be downloaded from `http://becool.info.ucl.ac.be/resources/VRPBB`.

In this section two types of best known results from the literature are considered. First the *all-best* results, which correspond to the overall best known results from the literature for the MP-VRP. The *bb-best* results on the other hand correspond to results from the literature obtained using the same black box algorithm as the one used here. In the case of the MP-VRP the *bb-best* results correspond to the ones presented in [TDHI09]. Note that the results taken as *bb-best* results have been obtained over different configuration settings and correspond to the overall best results the authors obtained in the course of their experiments.

13.5.1 *Pheromone-based Heuristic Column Generation (ACO-HCG)*

The parameter configuration used for ACO-HCG has been elaborated using an automatic algorithm configuration procedure. The process and the resulting configuration are described in detail in chapter 11. Note that the experiments in chapter 11 and the experiments for this chapter were conducted on different architectures. A comparison of the results achieved on both architectures over the same number of runs is given in table B.1 in the appendix. It shows that the variations are more important than for the 3L-CVRP. Indeed while for this problem, the best solution found can be also be improved on 4 instances, the average solution cost deteriorates on all instances but one. The variation on the average solution cost, averaged over all instances corresponds to 0.34%. These variations can be explained by the different architecture and the non-determinism in CPLEX. The results presented in chapter 11 seem to improve more significantly over these presented in this chapter due to the higher number of runs used in chapter 11.

The time limits from [TDHI09] are used. The limit is imposed on the route generation process. After the allowed time has been used up, the generation of new routes is stopped and the final solution is computed. The route generation time limit corresponds to 1800 CPU seconds for all instances.

Table 13.1 presents the results obtained on the MP-VRP benchmark instances. Column z_{min} and z_{avg} indicate the best and average solution value over 10 runs. Column $\%RSD$ gives the relative standard deviation over 10 runs, while column sec_{tt} indicates the average total execution time. Finally in column $\%g^{bb}_{avg}$ the average relative deviation w.r.t the *bb-best* result is given. The relative deviation w.r.t. *bb-best* is computed as $100 \cdot \frac{z - z_{bb}}{z_{bb}}$ where z is the solution cost obtained using ACO-HCG and z_{bb} is the solution cost of the *bb-best* solution. A negative value in column $\%g^{bb}_{avg}$ indicates that ACO-HCG on average (over 10 runs) improves the *bb-best* result.

A look at the $\%g^{bb}_{avg}$ column shows that the proposed ACO-HCG approach is able to improve the *bb-best* result on average for 2 out of 21 instances. It should be noted that the average deviation from the *bb-best* result never exceeds 1.65%. The most notable average deterioration w.r.t. the *bb-best* result is for instance CMT05-2, a large-scale instance where there is a similar number of customers in each class. The biggest improvement is obtained on CMT02-1, a small-scale instance where most customers either demand few or many items.

In terms of execution times, ACO-HCG is similar to the Variable Neighborhood Search presented in [TDHI09]. The latter approach uses a limit on stable iterations and a limit on total execution time (1800 CPU seconds). In ACO-HCG only a time limit is used, which is imposed on the route generation phase. A look at the sec_{tt} column in Table 13.1 suggests thus that up to 200 CPU seconds (instances CMT05) are necessary to extract a final integer solution from the set of collected feasible routes $\mathcal{R}^*$.

Analysis of CPU time allocation and Feasibility in generated routes

In this section the percentage of total execution time spent in different parts of the algorithm is analyzed, as well as the number of *bb*-feasible routes found, compared to the total number of routes tested for feasibility w.r.t. F_{bb}.

Figure 13.3 gives a visual representation of the percentage of the CPU time allocated to different tasks in ACO-HCG. A detailed view of this is given in table 13.2. The tasks under consideration are:

- solving problem $SCP(\mathcal{R}^*)$ to obtain an integral solution ($\%IP$)
- solving problems $Relax(SCP(\mathcal{R}^*))$, repeated each iteration of ACO-HCG ($\%LP$)
- testing routes for *bb*-feasibility ($\%BB$)
- retrieving or adding routes to the Feasibility store Ψ ($\%ST$)
- generating new routes, thus executing the collector ants modulo the time to check feasibility of routes ($\%GEN$)
- post-optimizing feasible routes generated by the collector ants ($\%PO$)
- other parts of ACO-HCG ($\%OTH$)

Instance			**ACO-HCG Automatic Configuration**				
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_{tt}	$\%g_{avg}^{bb}$
CMT01-1	1	50	587.81	590.10	0.2	1823.6	0.48
CMT01-2	2	50	615.11	618.50	0.6	1804.7	0.55
CMT01-3	3	50	623.44	624.76	0.3	1804.4	0.21
CMT02-1	1	75	975.56	976.16	0.1	1855.3	-0.50
CMT02-2	2	75	898.38	902.93	0.3	1837.9	0.59
CMT02-3	3	75	889.26	890.85	0.2	1817.8	0.28
CMT03-1	1	100	1183.76	1186.61	0.1	1809.1	-0.13
CMT03-2	2	100	1219.15	1222.52	0.1	1893.8	0.29
CMT03-3	3	100	1157.22	1161.15	0.3	1818.6	0.37
CMT04-1	1	150	1617.12	1630.01	0.6	1934.3	0.31
CMT04-2	2	150	1547.30	1553.76	0.3	1984.4	0.10
CMT04-3	3	150	1541.41	1548.00	0.4	1883.7	0.40
CMT05-1	1	199	2024.91	2041.67	0.4	2037.5	0.29
CMT05-2	2	199	1849.76	1863.53	0.5	2038.8	1.64
CMT05-3	3	199	1952.32	1961.75	0.5	1985.1	0.66
CMT06-1	1	120	2250.76	2258.60	0.3	1914.9	0.80
CMT06-2	2	120	2085.16	2103.69	0.4	1863.9	1.63
CMT06-3	3	120	2152.84	2168.93	0.4	1848.3	0.68
CMT07-1	1	100	1139.85	1149.18	0.4	1845.5	1.11
CMT07-2	2	100	1214.95	1217.20	0.1	1843.4	-0.02
CMT07-3	3	100	1153.81	1167.71	0.9	1836.4	0.87
AVG					0.4		0.51

Table 13.1: Results on MP-VRP using ACO-HCG in the automatic configuration, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_{tt} = average total execution time, $\%g_{avg}^{bb}$=average relative percentage deviation w.r.t. *bb-best* solution.

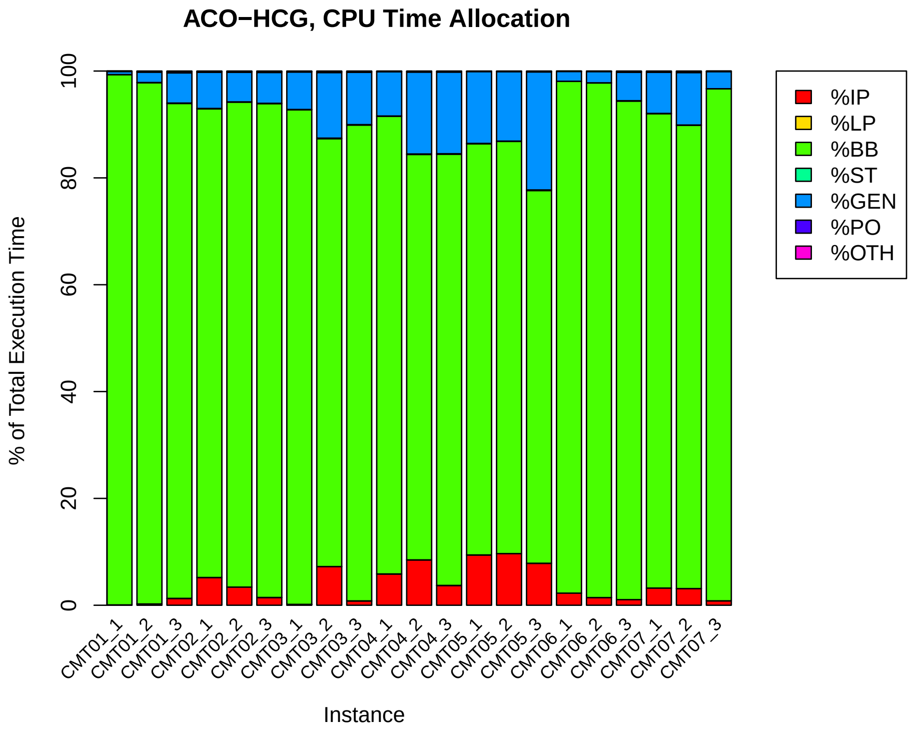


Figure 13.3: ACO-HCG: CPU Time Allocation

Clearly the majority of the CPU time is allocated to testing the *bb*-feasibility of routes for all instances. The remaining time is allocated to the generation of feasible routes and the resolution of the integer problem SCP($\mathcal{R}^*$). Note that in the automatic configuration of ACO-HCG this latter problem is solved every 6 iterations. The percentages of time allocated to the different tasks vary however from instance to instance. In the larger instances (CMT04 and CMT05) more time is spent in the generation of new feasible routes and in solving the integer problem.
There are different possible explanations for this. For once the data structures to be maintained for the collector ants grow with the problem size. Of course the number of possible initial single-customer routes and therefore possible route merges grows. In the MP-VRP the

feasibility of small routes, visiting only a few customers is often easily established. Given the initial number of single-customer routes many iterations will be spent considering mainly merges resulting in such routes. Also the collector ants continue merging routes until no further feasible merge can be found. It can take quite a few ant steps up to a state where only a few routes remain. Finally, the absence of capacity constraints means that no merges can be eliminated due to infeasibility w.r.t. F_{wb}. The high number of ant steps and the size of the data structures to be maintained explain that more time is spent in the route generation (and less in the black box feasibility) for large-scale instances.
There are also different possible causes to the fact that a higher time percentage is spent solving the integer problems. In the automatic configuration problem $SCP(\mathcal{R}^*)$ is solved every 6 iterations. In the case of smaller instances, this means that the problem is solved iteratively over an increasing number of feasible routes, each time providing an upper bound to the solver for the next time the problem is solved. In the large-scale instances, ACO-HCG doesn't reach 6 iterations, thus the problem must be solved from scratch, without upper bound. The fact that the number of vehicles is not limited in the MP-VRP means also that the problem is less restricted and thus harder to solve.

Figure 13.4 shows the percentage of routes (averaged over 10 runs) that were tested for feasibility w.r.t. F_{bb} that were actually found to be feasible. This value varies between roughly 8 and 18 %. No clear trend linked to problem size or class can be distinguished, if it isn't that instance CMT01-1 allows to find a higher percentage of feasible routes than the remaining instances.

The exact numbers are given in table 13.2. It gives the exact percentage of total execution time used up by each task in ACO-HCG. In columns *#Feasible routes* and $|\Psi|$ the exact number of routes in the final $\mathcal{R}^*$ and the final number of routes in Feasibility store Ψ is given. In the case of ACO-HCG, $|\Psi|$ corresponds to the total number of routes tested for *bb*-feasibility. An interesting discovery is that the number of routes tested is lowest for instance CMT01-1, the instance where the highest percentage of feasible routes was found and the highest percentage of time is spent in the black box. It seems that checking feasibility for routes for this instance takes a particular long time per route. Instances CMT01-2 and CMT01-3 both have the same number of customers, but take significantly less time to complete one ACO-

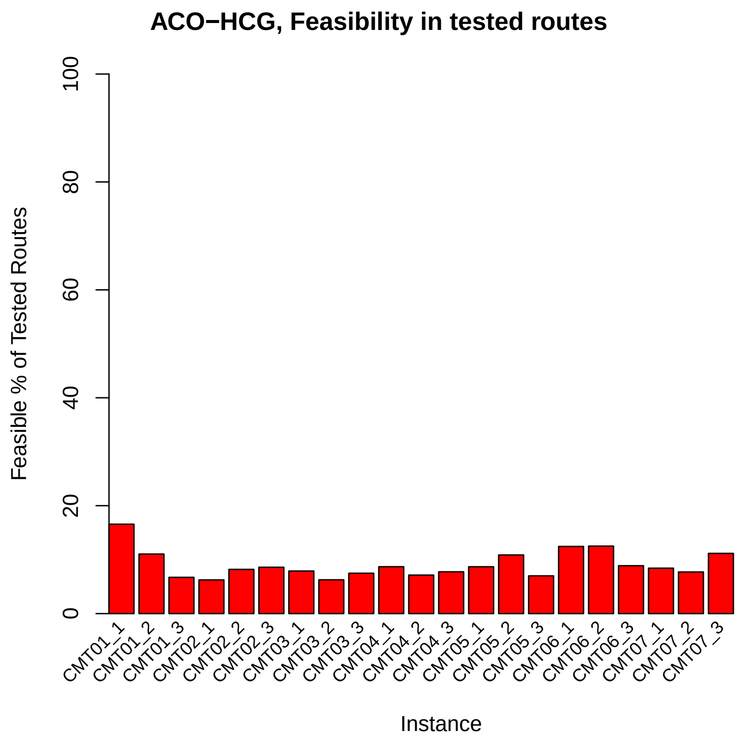


Figure 13.4: ACO-HCG: % of tested routes that were *bb*-feasible

HCG iteration. Note that the state-of-the-art approaches for the MP-VRP make use of the knowledge that the loading problem to be solved per route is symmetric, which can't be exploited in the corresponding VRPBB.

13.5.2 *Decomposition-based approach using ACO-HCG*

The decomposition-based approach described in chapter 9 has been tested with ACO-HCG. Parameter N has been set to $\lceil \frac{n}{5} \rceil$. The time to solve each subproblem to $min(300, \overline{n})$ CPU seconds, where $\overline{n}$ corresponds to the number of customers in the subproblem. Explicit results comparing the performance of ACO-HCG and the Decomposition-

Instance	**% Total Execution Time**						**Routes in Ψ**	
	$\%IP$	$\%LP$	$\%BB$	$\%ST$	$\%GEN$	$\%PO$	#Feasible routes	$\|\Psi\|$
CMT01-1	0.1	0	99.3	0.0	0.6	0.1	1871.6	11413.9
CMT01-2	0.2	0	97.6	0.0	1.9	0.2	3519.5	31936.4
CMT01-3	1.3	0	92.6	0.1	5.6	0.3	4956.2	74060.1
CMT02-1	5.2	0	87.7	0.1	6.8	0.2	4998.8	80304.9
CMT02-2	3.4	0	90.8	0.1	5.5	0.2	4387.9	53723.0
CMT02-3	1.5	0	92.4	0.1	5.8	0.2	5255.0	61384.9
CMT03-1	0.2	0	92.6	0.1	7.0	0.1	4220.5	53843.0
CMT03-2	7.2	0	80.1	0.1	12.3	0.2	5639.8	90278.5
CMT03-3	0.8	0	89.1	0.1	9.8	0.2	5262.6	70472.9
CMT04-1	5.9	0	85.7	0.1	8.3	0.1	3600.3	41736.6
CMT04-2	8.5	0	75.9	0.1	15.3	0.1	5011.1	70853.9
CMT04-3	3.7	0	80.7	0.1	15.3	0.1	5017.6	65451.1
CMT05-1	9.4	0	77.0	0.1	13.4	0.1	4056.0	47128.9
CMT05-2	9.7	0	77.1	0.1	13.0	0.1	4267.3	39407.1
CMT05-3	7.8	0	69.8	0.1	22.1	0.1	5128.7	73556.7
CMT06-1	2.3	0	95.8	0.0	1.8	0.1	3106.2	25076.0
CMT06-2	1.4	0	96.3	0.0	2.1	0.1	3607.1	28952.6
CMT06-3	1.0	0	93.3	0.1	5.3	0.2	6300.0	71371.4
CMT07-1	3.2	0	88.8	0.1	7.7	0.2	4189.1	50300.6
CMT07-2	3.1	0	86.7	0.1	9.8	0.2	4929.8	64325.1
CMT07-3	0.8	0	95.8	0.0	3.2	0.1	2667.7	24187.3

Table 13.2: ACO-HCG : CPU Time Allocation, # feasible routes, # routes in Feasibility Store ($|\Psi|$)

based approach using ACO-HCG (DECOMP ACO-HCG) can be found in table 13.3. Again ACO-HCG outperforms DECOMP ACO-HCG, this time on all instances. Interestingly the decomposition-based approach does not seem to bring an advantage to larger-scale instances. Instances CMT05 are the largest instances for the MP-VRP, however the decomposition-based approach does not perform better here than for smaller instances. A plot comparing the performance of DECOMP ACO-HCG to ACO-HCG can be found in the appendix (Fig. B.1).

13.5.3 *ACO-HCG without Post-optimization of feasible routes*

In chapter 11 a parameter configuration for ACO-HCG has been elaborated using an automatic algorithm configuration procedure. For the post-optimization two parameters were considered: *TS* and *ILS*, thus doing the post-optimization of feasible routes either using Tabu Search or Iterated Local Search. However the choice to do no post-optimization at all was not considered. Thus in order to establish whether the post-optimization procedure actually improves ACO-HCG's results, experiments were run using ACO-HCG's automatic configuration (see 11) where the post-optimization was switched off (i.e. neither Tabu Search nor Iterated Local Search are performed). The resulting approach is denoted ACO-HCG-NOPO. The corresponding results are given in Table B.2 in the appendix. Furthermore the exact CPU Time Allocation percentages along with the number of tested and feasible routes is given in table B.3 in the appendix.

Table B.2 shows that it is of advantage to execute the post-optimization of feasible routes, this in terms of best solutions as well as in terms of average solution value. It can however be noted that for a few instances a better best solution can be found with ACO-HCG-NOPO. In terms of average solution value the results obtained with ACO-HCG-NOPO are up to 1.2% worse than these obtained with ACO-HCG. The observations for the MP-VRP differ from these of the 3L-CVRP. While in the 3L-CVRP, ACO-HCG tested more routes thanACO-HCG-NOPO, Table B.3 shows that in MP-VRP it very much depends on the instance. However the observation that ACO-HCG-NOPO ends up with more feasible routes in its pool and thus more time is needed to solve the Set Covering Problem (column %*IP*) holds. Finally Table B.3 reveals

Instance			**ACO-HCG**			**DECOMP ACO-HCG**				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
CMT01-1	1	50	587.81	590.10	1823.60	587.81	594.13	1840.7	0.00	0.68
CMT01-2	2	50	615.11	618.50	1804.70	616.74	619.36	1817.8	0.26	0.14
CMT01-3	3	50	623.44	624.76	1804.40	623.44	624.44	1824.8	0.00	-0.05
CMT02-1	1	75	975.56	976.16	1855.30	976.63	977.00	1831.4	0.11	0.09
CMT02-2	2	75	898.38	902.93	1837.90	903.12	906.80	1847.6	0.53	0.43
CMT02-3	3	75	889.26	890.85	1817.80	893.93	897.50	1834.8	0.53	0.75
CMT03-1	1	100	1183.76	1186.61	1809.10	1184.51	1191.49	1841.3	0.06	0.41
CMT03-2	2	100	1219.15	1222.52	1893.80	1224.52	1228.38	1851.7	0.44	0.48
CMT03-3	3	100	1157.22	1161.15	1818.60	1159.15	1165.26	1834.7	0.17	0.35
CMT04-1	1	150	1617.12	1630.01	1934.30	1630.34	1646.30	1845.0	0.82	1.00
CMT04-2	2	150	1547.30	1553.76	1984.40	1555.89	1568.01	1854.4	0.56	0.92
CMT04-3	3	150	1541.41	1548.00	1883.70	1556.62	1566.98	1857.4	0.99	1.23
CMT05-1	1	199	2024.91	2041.67	2037.50	2045.13	2057.05	1904.5	1.00	0.75
CMT05-2	2	199	1849.76	1863.53	2038.80	1855.87	1874.20	1875.9	0.33	0.57
CMT05-3	3	199	1952.32	1961.75	1985.10	1987.30	1995.63	1879.5	1.79	1.73
CMT06-1	1	120	2250.76	2258.60	1914.90	2256.84	2273.05	1824.6	0.27	0.64
CMT06-2	2	120	2085.16	2103.69	1863.90	2106.20	2116.31	1837.1	1.01	0.60
CMT06-3	3	120	2152.84	2168.93	1848.30	2171.35	2187.73	1836.7	0.86	0.87
CMT07-1	1	100	1139.85	1149.18	1845.50	1145.55	1153.41	1859.7	0.50	0.37
CMT07-2	2	100	1214.95	1217.20	1843.40	1219.26	1222.76	1836.8	0.35	0.46
CMT07-3	3	100	1153.81	1167.71	1836.40	1165.88	1177.37	1868.7	1.05	0.83
AVG					1880.07			1847.9	0.55	0.63

Table 13.3: Comparison of ACO-HCG in automatic configuration (ACO-HCG) with Decomposition-based approach using ACO-HCG in automatic configuration (DECOMP ACO-HCG). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time. $g^{min/avg}$= relative percentage deviation of DECOMP ACO-HCG w.r.t. ACO-HCG , computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with DECOMP ACO-HCG and $z^D_{min/avg}$ the results obtained with ACO-HCG.

that for the MP-VRP, ACO-HCG-NOPO spends more time Generating Routes and less in the black box than ACO-HCG.

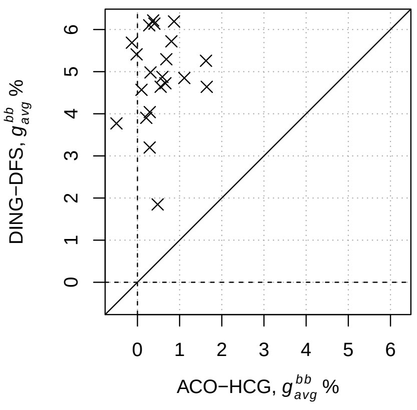
13.5.4 *Dive & Generate (DING)*

Dive & Generate has been executed over 10 independent runs using the same set-up as for the 3L-CVRP, with one difference. The randomized savings heuristic is executed $\ell = 1$ times per column generation execution.

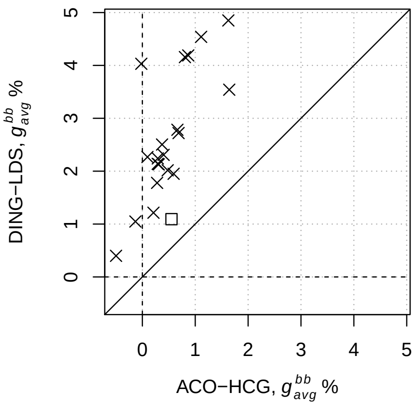
13.5.4.1 *Comparison with ACO-HCG*

In the following we will individually compare Dive & Generate using Depth First Search (DING-DFS) and Dive & Generate using Limited Discrepancy Search (DING-LDS) with ACO-HCG. This is done by computing for each instance and both for ACO-HCG and the method under consideration the mean relative deviation w.r.t. the *best-bb* solution (denoted by g^{bb}_{avg}). The resulting values are then visualized in a scatter plot where each point corresponds to one problem instance. The x-coordinate of the point will correspond to g^{bb}_{avg} for ACO-HCG and the y-coordinate to the g^{bb}_{avg} for the method under consideration. DING and ACO-HCG perform equally on the same instance if the point is on the diagonal, DING performs better if the point is under the diagonal, and ACO-HCG performs better if the point is above the diagonal. Statistical significance has been determined using a Wilcoxon signed-rank test with confidence level 95%. The resulting plots are visualized in Fig. 13.5. The explicit results for DING-DFS and DING-LDS are given in the appendix (tables B.4 and B.5).

The figures show clearly that both DING-DFS and DING-LDS are outperformed by ACO-HCG on all instances. However, DING-LDS gets better results than DING-DFS on all but one instance. While ACO-HCG considers any route as potential candidate to be added to the set of feasible routes $\mathcal{R}^*$, DING does so only for routes that have a negative reduced cost w.r.t the current non-integral solution. These routes are possibly not the ones that are necessary to obtain a high-quality integer solution. Furthermore only the routes that actually appear in a relaxed solution can ever be part of an integer solution (as this solution is obtained by fixing routes). Thus only if some of the routes appearing in the relaxed solutions are also part of a high-quality integer solution can DING perform well. Finally in the MP-VRP, as the number of vehicles is not limited, there can be a great number of such routes at a time. This results in a Dive & Generate tree with a high



(a)



(b)

Figure 13.5: Comparison of DING and ACO-HCG. Each point gives the mean %-deviation from the *bb-best* solution over 10 independent runs. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

branching factor and thus in a potentially huge tree. A lot of time will be lost exploring one branch. It is for this reason that DING-LDS allows to improve the performance w.r.t. DING-DFS. Discrepancies are taken as early as possible and thus completely different parts of the Dive & Generate tree can be explored.

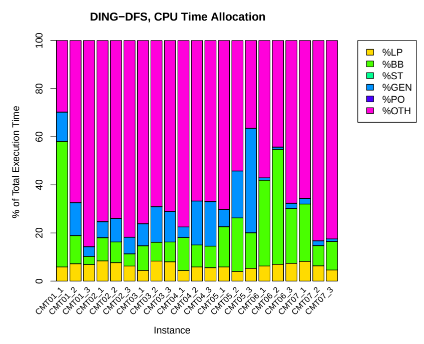
Plots comparing DING-DFS and DING-LDS with ACO-HCG in the manual configuration from chapter 11 (denoted $\overline{\text{ACO-HCG}}$ in this chapter) can be found in the appendix (Fig. B.2).

DING-LDS performs still worse than $\overline{\text{ACO-HCG}}$ but is able to statistically significantly improve the results obtained with $\overline{\text{ACO-HCG}}$ on one instance. Also, as for the 3L-CVRP, the results on many instances are not statistically significant. DING-DFS still performs worse than $\overline{\text{ACO-HCG}}$.

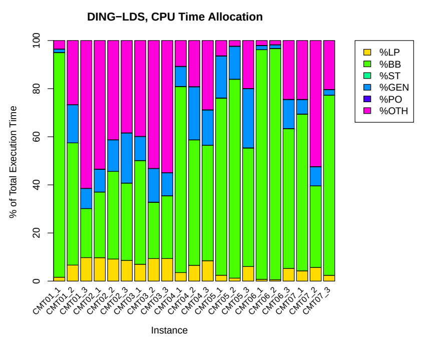
13.5.4.2 *Analysis of CPU time allocation and Feasibility in generated routes*

In this section the percentage of CPU time spent in different parts of the algorithm is analyzed, as well as the number of *bb*-feasible routes found, compared to the total number of routes tested for feasibility w.r.t. F_{bb}.

Figures 13.6 gives a visual representation of the percentage of the total execution time allocated to different tasks in DING-DFS and DING-



(a)



(b)

Figure 13.6: DING: CPU Time Allocation

LDS. A detailed view of this is given in Tables 13.4 and 13.5. The tasks under consideration are the same as for ACO-HCG. In DING problem SCP($\mathcal{R}^*$) is only solved at the beginning of the process to initialize the upper bound. Task $\%IP$ has been omitted in both figures and tables since the percentage of time spent solving $SCP(\mathcal{R}^*)$ is negligible.

A clear difference can be established between DING-DFS and DING-LDS. Whereas with DING-LDS a majority of time is spent testing routes for *bb*-feasibility and generating routes, a bigger portion of time is spent in the "Other" tasks for DING-DFS. DING-DFS spends clearly less time in the black box and for most instances also in the generation of new routes. In Dive & Generate column generation is executed only if the current lower bound falls outside a trust region, and an enforced route generation is executed upon restarts. The fact that only a small fraction of the CPU time is allocated to route generation in DING-DFS suggests that column generation is actually only executed seldom. As the number of vehicles is not limited in MP-VRP the resulting Dive & Generate tree is potentially very broad but also very deep. DING-DFS seems to visit a great number of nodes, at which column generation is executed but where no routes of negative reduced cost can be found. Given the nature of Depth First Search and the CPU time allocations a probable conclusion is that the search gets stuck in a specific branch, visiting numerous nodes where no routes of negative reduced cost are found. Furthermore, symmetries are not handled in Dive & Generate. The fact that the set of feasible routes $\mathcal{R}^*$ is enriched at some nodes,

might not be sufficient to avoid the visit to symmetric parts of the search tree.
Limited Discrepancy Search avoids the scenario of the search getting stuck in a specific branch. Discrepancies are taken as early as possible in the search tree, and drastically different parts of the search tree are thus explored. Also, the search will always first reascend to the root. This makes it impossible for the search to get stuck at leaf node level.
In terms of CPU time allocation, DING-LDS (and thus also DING-DFS) spends significantly less time in the black box and generally more time in route generation and the "Other" tasks when compared to ACO-HCG. Remember that these latter correspond to the maintenance of a tree structure in DING, not present in ACO-HCG.

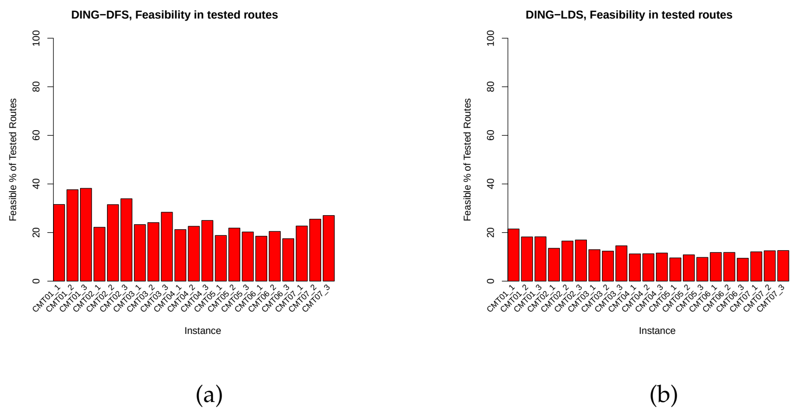


Figure 13.7: DING: % of tested routes that were *bb*-feasible

Figures 13.7 show the percentage of routes tested for feasibility w.r.t. F_{bb} that are found to be feasible. Interestingly, the percentages for DING-DFS are consistently higher than those for DING-LDS. Both DING-DFS and DING-LDS show a higher percentage of feasible routes than ACO-HCG. The exact numbers are given in Tables 13.4 and 13.5. In columns #*Feasible routes* and $|\Psi|$ the exact number of routes in the final $\mathcal{R}^*$ and the final number of routes in Feasibility store Ψ is given. In the case of DING, $|\Psi|$ corresponds to the total number of routes tested for *bb*-feasibility. The number of *bb*-feasible routes is lower with DING-DFS than with ACO-HCG for most instances. On the other hand DING-LDS detects more *bb*-feasible routes than ACO-HCG on the larger instances. In terms of the number of tested routes, ACO-

HCG performed more tests than DING-DFS and DING-LDS on most instances. In general DING-LDS tests more routes for *bb*-feasibility than DING-DFS, and also finds more *bb*-feasible routes.

13.5.4.3 *Results obtained when solving* $SCP(\mathcal{R}^*)$

In ACO-HCG the final solution is obtained by solving the integer Set Covering Problem over the set of accumulated feasible routes $\mathcal{R}^*$. In DING the final solution corresponds to the best solution found in the Dive & Generate tree. Possibly better solutions could be obtained for DING, if the final solution were obtained in the same way as in ACO-HCG, that is by solving problem $SCP(\mathcal{R}^*)$ where $\mathcal{R}^*$ is the set of feasible routes accumulated during the DING execution (the corresponding approach is denoted DING*). The results corresponding to this are given in the appendix (Tables B.6 and B.7). The improvements are similar to those obtained in the case of the 3L-CVRP. Both the results of DING-DFS and DING-LDS are improved by solving DING*-DFS and DING*-LDS, but the improvements obtained with DING*-LDS are more substantial (up to 3.02% in terms of average solution value). The results of DING*-LDS are still worse than those obtained with ACO-HCG. Finally the time needed to solve $SCP(\mathcal{R}^*)$ on DING-LDS is higher than the time needed to solve $SCP(\mathcal{R}^*)$ for DING-DFS, again especially for large scale instances, where up to 276.7 CPU seconds are necessary.

13.5.5 *Decomposition-based approach using DING-LDS*

The decomposition-based approach described in chapter 9 has been tested with DING-LDS. Parameter N has been set to $\lceil \frac{n}{5} \rceil$. The time to solve each subproblem to $min(300, \overline{n})$ CPU seconds, where $\overline{n}$ corresponds to the number of customers in the subproblem. Explicit results comparing the performance of DING-LDS and the Decomposition-based approach using DING-LDS (DECOMP DING-LDS) can be found in table 13.6. The decomposition-based approach allows to improve most of the DING-LDS results! In 10 instances DECOMP DING-LDS reaches better average solution values. However again, no clear link with problem size can be seen. Plots comparing the performance of DECOMP DING-LDS to DING-LDS and to ACO-HCG can be found in the appendix (Figs. B.3).

Instance	% Total Execution Time					Routes in Ψ	
	$\%LP$	$\%BB$	$\%ST$	$\%GEN$	$\%PO$	#Feasible routes	$\|\Psi\|$
CMT01-1	5.9	52.2	0	12.2	0.1	3229.8	10373.7
CMT01-2	7.2	11.7	0	13.7	0.0	2754.7	7420.1
CMT01-3	6.9	3.4	0	4.0	0.0	2077.8	5568.5
CMT02-1	8.4	9.6	0	6.6	0.0	3372.9	16550.0
CMT02-2	7.7	8.6	0	9.8	0.0	3063.4	9879.5
CMT02-3	6.3	5.1	0	6.9	0.0	2285.7	6834.8
CMT03-1	4.5	10.2	0	9.1	0.0	2270.4	11580.4
CMT03-2	8.3	7.9	0	14.7	0.0	3904.1	16863.0
CMT03-3	8.0	8.2	0	12.7	0.0	3436.3	12878.2
CMT04-1	4.4	13.7	0	4.4	0.0	2205.6	12065.8
CMT04-2	5.9	9.1	0	18.2	0.0	3079.0	13518.3
CMT04-3	5.6	9.0	0	18.5	0.0	3260.4	12618.3
CMT05-1	5.9	16.7	0	7.2	0.0	3624.4	19784.3
CMT05-2	4.0	22.2	0	19.5	0.0	3779.3	17803.7
CMT05-3	5.3	14.8	0	43.4	0.1	5121.4	24791.0
CMT06-1	6.3	35.7	0	0.9	0.1	3878.0	20982.5
CMT06-2	7.0	47.9	0	0.8	0.1	5858.7	28427.2
CMT06-3	7.4	22.8	0	2.2	0.1	5431.8	33649.3
CMT07-1	8.2	23.8	0	2.3	0.1	4894.1	21763.9
CMT07-2	6.4	8.4	0	2.0	0.0	3038.4	13795.5
CMT07-3	4.6	11.9	0	1.0	0.0	2163.2	10036.3

Table 13.4: DING-DFS : CPU Time Allocation, # feasible routes, # routes in Feasibility Store ($|\Psi|$)

Instance	**% Total Execution Time**					**Routes in Ψ**	
	$\%LP$	$\%BB$	$\%ST$	$\%GEN$	$\%PO$	#Feasible routes	$\|\Psi\|$
CMT01-1	1.6	93.4	0.0	1.4	0.1	2603.1	12118.2
CMT01-2	6.7	50.8	0.0	15.8	0.1	3923.3	21826.2
CMT01-3	9.8	20.4	0.0	8.3	0.1	3667.1	20255.6
CMT02-1	9.7	27.3	0.0	9.5	0.1	3900.1	29050.4
CMT02-2	9.2	36.4	0.0	13.1	0.1	4231.3	25736.0
CMT02-3	8.6	32.2	0.0	20.8	0.1	4560.3	26943.3
CMT03-1	7.0	43.1	0.0	10.0	0.1	3809.5	29422.4
CMT03-2	9.4	23.4	0.0	14.0	0.1	4421.7	36041.0
CMT03-3	9.4	26.1	0.0	9.5	0.1	4256.9	29260.6
CMT04-1	3.5	77.3	0.0	8.3	0.1	5019.3	45406.9
CMT04-2	6.6	52.1	0.1	22.0	0.1	6113.0	54602.0
CMT04-3	8.5	48.0	0.1	14.6	0.1	6134.2	53226.8
CMT05-1	2.4	73.6	0.1	17.4	0.1	5523.8	57944.3
CMT05-2	1.2	82.6	0.1	13.6	0.1	5746.4	53212.0
CMT05-3	6.1	49.2	0.1	24.6	0.1	6484.1	66825.6
CMT06-1	0.7	95.4	0.0	1.7	0.1	3848.9	33003.7
CMT06-2	0.6	96.1	0.0	1.5	0.1	3882.0	33761.9
CMT06-3	5.3	58.1	0.1	12.0	0.1	6470.5	69284.0
CMT07-1	4.3	65.1	0.0	6.0	0.1	4413.7	36622.5
CMT07-2	5.7	33.9	0.0	7.9	0.1	4128.7	33509.8
CMT07-3	2.4	74.9	0.0	2.3	0.1	3246.6	25903.2

Table 13.5: DING-LDS : CPU Time Allocation, # feasible routes, # routes in Feasibility Store ($|\Psi|$)

Instance			**DING-LDS**			**DECOMP DING-LDS**				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
CMT01-1	1	50	594.54	599.13	1857.30	591.66	600.49	1950.5	-0.48	0.23
CMT01-2	2	50	616.74	621.84	1800.50	622.83	629.79	1907.7	0.99	1.28
CMT01-3	3	50	623.91	631.02	1800.10	623.91	626.31	1844.6	0.00	-0.75
CMT02-1	1	75	976.63	984.97	1800.10	976.60	980.14	1870.1	0.00	-0.49
CMT02-2	2	75	907.50	915.13	1800.70	907.87	916.17	1884.6	0.04	0.11
CMT02-3	3	75	889.88	904.18	1800.70	889.26	899.58	1873.0	-0.07	-0.51
CMT03-1	1	100	1189.45	1200.64	1800.70	1198.68	1219.38	1891.6	0.78	1.56
CMT03-2	2	100	1234.40	1246.28	1800.20	1224.05	1230.43	1860.3	-0.84	-1.27
CMT03-3	3	100	1174.49	1185.81	1800.20	1169.89	1178.21	1835.7	-0.39	-0.64
CMT04-1	1	150	1629.61	1659.57	1815.00	1661.65	1678.14	1891.8	1.97	1.12
CMT04-2	2	150	1573.16	1587.47	1800.90	1571.69	1590.65	1900.8	-0.09	0.20
CMT04-3	3	150	1568.01	1577.40	1800.30	1580.02	1606.12	1929.3	0.77	1.82
CMT05-1	1	199	2052.35	2079.31	1809.60	2069.10	2088.64	2010.7	0.82	0.45
CMT05-2	2	199	1875.58	1898.33	1814.00	1883.52	1916.46	1884.7	0.42	0.96
CMT05-3	3	199	1992.71	2003.07	1801.60	1986.91	2010.83	1890.0	-0.29	0.39
CMT06-1	1	120	2298.87	2333.72	1825.30	2292.80	2315.93	1923.6	-0.26	-0.76
CMT06-2	2	120	2130.10	2170.45	1832.70	2101.97	2146.68	1952.6	-1.32	-1.10
CMT06-3	3	120	2190.97	2212.80	1803.30	2197.15	2243.98	1888.6	0.28	1.41
CMT07-1	1	100	1177.31	1188.15	1800.10	1147.41	1157.03	1937.3	-2.54	-2.62
CMT07-2	2	100	1236.68	1266.53	1801.10	1220.09	1230.26	1896.5	-1.34	-2.86
CMT07-3	3	100	1190.39	1206.16	1803.40	1163.88	1182.07	1934.2	-2.23	-2.00
AVG					1807.99			1902.8	-0.18	-0.17

Table 13.6: Comparison of Dive & Generate with Limited Discrepancy Search (DING-LDS) with Decomposition-based approach using DING-LDS (DECOMP DING-LDS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time. $g^{min/avg}$= relative percentage deviation of DECOMP DING-LDS w.r.t. DING-LDS, computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with DECOMP DING-LDS and $z^D_{min/avg}$ the results obtained with DING-LDS.

13.5.6 *Heuristic Branch & Price (HBP)*

Heuristic Branch & Price has been executed over 10 independent runs using the same set-up as for the 3L-CVRP, with one difference. The randomized savings heuristic is executed $\ell = 1$ times per column generation execution.

13.5.7 *Comparison with ACO-HCG*

In the following we will individually compare the different Heuristic Branch & Price variants with ACO-HCG. We consider two different branching schemes: branching on arcs (HBPA) and branching on customer pairs (HBPP). Both of these are combined with a Depth-First Search (HBPA-DFS and HBPP-DFS) and a Limited Discrepancy Search (HBPA-LDS and HBPP-LDS), resulting in a total of 4 different configurations. The comparisons with ACO-HCG are done by computing for each instance and both for ACO-HCG and the method under consideration the mean relative deviation w.r.t. the *best-bb* solution denoted by g_{avg}^{bb}. The resulting values are then visualized in a scatter plot where each point corresponds to one problem instance. The x-coordinate of the point will correspond to g_{avg}^{bb} for ACO-HCG and the y-coordinate to g_{avg}^{bb} for the method under consideration. HBP and ACO-HCG perform equally on the same instance if the point is on the diagonal, HBP performs better if the point is under the diagonal, and ACO-HCG performs better if the point is above the diagonal. Statistical significance has been determined using a Wilcoxon signed-rank test with confidence level 95%. The resulting plots are visualized in Fig. 13.8. The explicit results are given in the appendix in tables B.8, B.9, B.10 and B.11.

As for Dive & Generate, Heuristic Branch & Price is clearly outperformed by ACO-HCG. Both HBPA-LDS and HBPP-LDS perform slightly better than their DFS counterparts, although this is clearer for HBPA. This is to be expected, as Limited Discrepancy Search ensures a more balanced exploration of the Heuristic Branch & Price tree. The performance of HBPA-DFS and HBPP-DFS depends on the instance. In 9 out 21 instances HBPP-DFS actually outperforms HBPA-DFS. HBPP-LDS is outperformed by HBPP-DFS on 5 instances, while it outperforms HBPA-LDS on 3 instances. Finally HBPA-LDS outperforms HBPA-DFS on 18 out of the 21 instances. None of the methods is able to improve the *bb-best* result on average for any instance. Also the methods perform generally worse than DING. The different performances can't be clearly linked to problem size or problem class. The

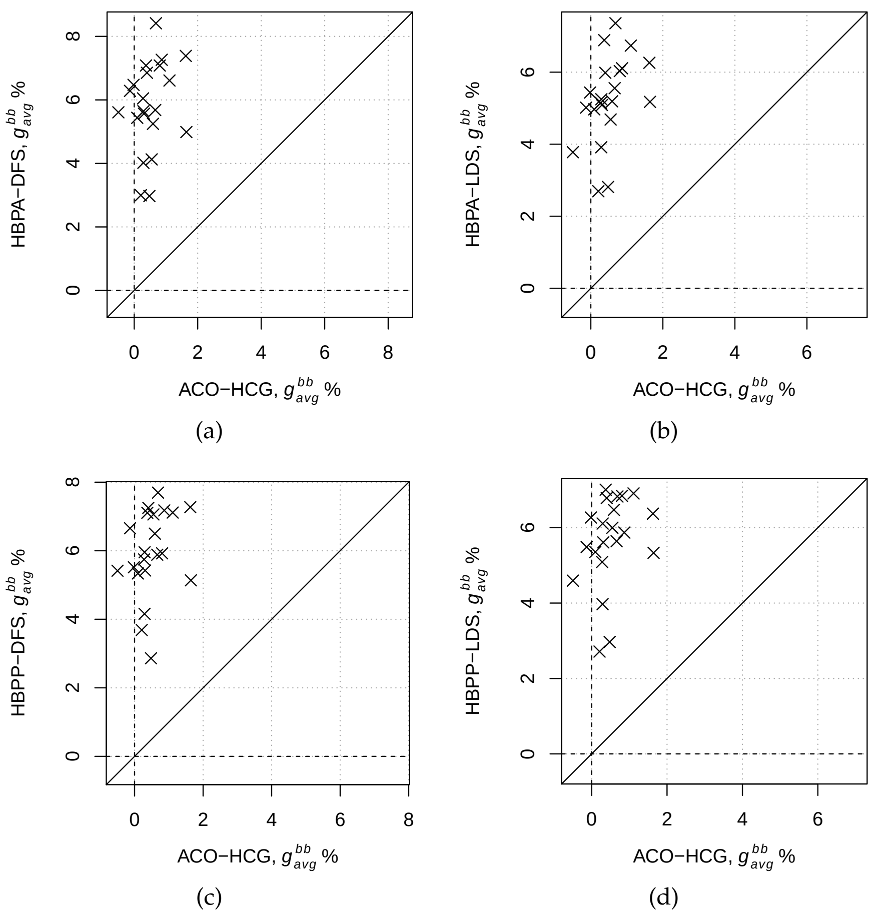
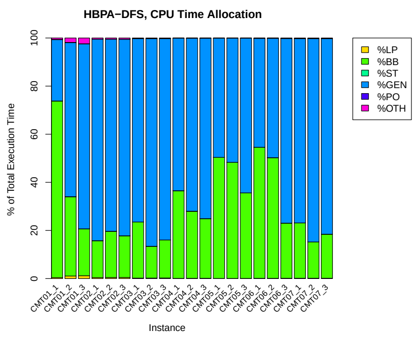


Figure 13.8: Comparison of HBP and ACO-HCG. Each point gives the mean %-deviation from the *bb-best* solution over 10 independent runs. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

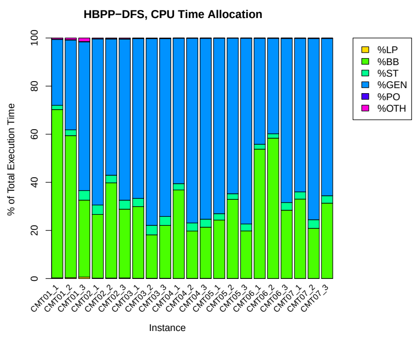
fact that HBPP-DFS performs better than HBPA-DFS on almost half the instances comes as a surprise, as it was clearly outperformed by HBPA-DFS on larger instances in the case of the 3L-CVRP. A closer look at CPU time allocation and the number of generated routes is necessary to explain this.
Plots comparing HBP with ACO-HCG in the manual configuration from chapter 11 (denoted $\overline{\text{ACO-HCG}}$ in this chapter) can be found in the appendix (Fig. B.4). Neither HBPA nor HBPP are able to improve the results of $\overline{\text{ACO-HCG}}$ on any instance.

13.5.7.1 *Analysis of CPU time allocation and Feasibility in generated routes*

In this section the percentage of total execution time spent in different parts of the algorithm is analyzed. Figures 13.9 give a visual representation of the percentage of the total execution time allocated to different tasks in HBPA-DFS and HBPP-DFS. The plots for HBPA-LDS and HBPP-LDS have been omitted for conciseness, but can be found in the appendix (Fig. B.5). They are however very similar to those given here. A detailed view of the time allocation is given in tables 13.7 and 13.8 (and tables B.12 and B.13 in the appendix for the LDS versions). The tasks under consideration are the same as for ACO-HCG.Task $\%IP$ has been omitted in both figures and tables since the percentage of time spent solving $SCP(\mathcal{R}^*)$ is negligible.



(a)



(b)

Figure 13.9: HBP: CPU Time Allocation

A look at the Figures 13.9 reveals that in terms of time allocation both HBPA and HBPP spend the majority of their time generating routes, and testing feasibility of routes w.r.t. F_{bb}. The time spent in the black box is lower compared to DING-LDS (and to ACO-HCG) but higher than with DING-DFS, whereas the time used to generate routes is higher than DING (remember that with HBPA and HBPP column generation is executed at every non-leaf node, whereas in DING column generation is executed only if the current lower bounds falls outside a trust region.). In DING only customers not already visited in a fixed route, are considered to appear in newly generated routes. This means that the problem of generating routes of negative reduced cost is more quickly more restricted as it is for HBP. This could mean that DING-LDS has more facility in finding routes of negative reduced cost than HBP and thus more routes are tested for *bb*-feasibility in DING-LDS (remember the observation that with DING-DFS column generation is executed only rarely).

When comparing HBPA and HBPP, it is visible that HBPP spends (on most instances) more time in the black box than HBPA. This is the case for all except for the largest instances (CMT05). This is surprising given that the opposite was true for the 3L-CVRP. A possible explanation for this higher quantity of time allocated to the tests for *bb*-feasibility would be that the tree exploration is restarted more often for HBPP than for HBPA. This would mean that the enforced route generation were executed more often. The data however does not support this explanation.
There remain thus two possible explanations: either HBPP actually tests more routes for *bb*-feasibility than HBPA, or it spends more time testing a lower number of routes. As previously explained in chapter 10, in general a route is tested for *bb*-feasiblity in both directions. Given that in HBPA the branching is performed on arcs, and thus the direction of the route matters w.r.t. the current subproblem, routes are only checked in one direction for HBPA. This entails that infeasible routes will not be stored as such, since they haven't been proven to be infeasible in both directions, a requirement of the Feasibility store. Thus it should actually be HBPA spending more time in the black box as the same route will possibly be checked several times, and thus more routes will be tested. On the other hand the fact that HBPP checks routes both ways may also explain why it is HBPP that spends more time in the black box for the MP-VRP. That is, if a given route is infeasible, in HBPP it will be reversed and checked for *bb*-feasibility again. In the MP-VRP, routes are symmetric however, and a route infeasible one

way will also be infeasible the other way. The loading black box uses an upper bound to first easily show the infeasibility w.r.t. F_{bb} of a route. If the upper bound fails to do this, an exact algorithm (with time limit set to 5 sec) is used to prove the feasibility w.r.t. F_{bb} of a route. Possibly the time limit of the exact algorithm is reached often in HBPP. If the time limit is reached without the route being proven bb-feasible, then HBPP will reverse the route and reexecute the check for bb-feasibility. Based on the nature of the black box algorithm, in that case the time limit of the exact algorithm will be reached again.

This latter explanation is however not sufficient to explain the difference in time allocated to the black box in HBPA and HBPP, since Tables 13.7 and 13.8 show that HBPP manages to accumulate a higher number of feasible routes compared to HBPA. This was already the case for the 3L-CVRP, however there it was attributed to the fact that HBPP checks routes in two directions, whereas HBPA doesn't. This same argument doesn't hold here, as the side-problem for the MP-VRP is symmetric. The only explanation that remains is thus that HBPP is able to generate more routes of negative reduced cost that are feasible, than HBPA. A possible explanation for this is the observation that for HBPA applied to the MP-VRP there is a tendency to first branch on arcs that are outgoing from the depot (depending of course on the values in the current non-integral solution). Fixing such an arc possibly does not allow to restrict the set of possible routes as significantly as fixing a pair of customers to a same route. Thus more restrictive problems are only encountered deeper in the tree, and it is only deeper in the tree that HBPA finds routes of negative reduced cost with ease.

Finally, HBPP spends more time in the Feasibility store Ψ ($\%ST$). The explanation for this last observation is the same as for the 3L-CVRP. The fact that HBPA adds less routes to the Feasibility Store.

In Figure 13.10, the percentage of routes tested for bb-feasibility that have been found to be feasible is illustrated for HBPP-DFS (the same plot for HBPP-LDS has been omitted for conciseness, but is quite similar and can be found in the appendix B.6). Since a same route may be tested multiple times in HBPA, the plot for HBPA is not provided, as the values obtained are not directly comparable to those obtained for HBPP.

It can be seen that the percentage of feasible routes among the tested ones for HBPP-DFS is higher on most instances compared to ACO-HCG and similar to DING-LDS (and lower to DING-DFS). At the same

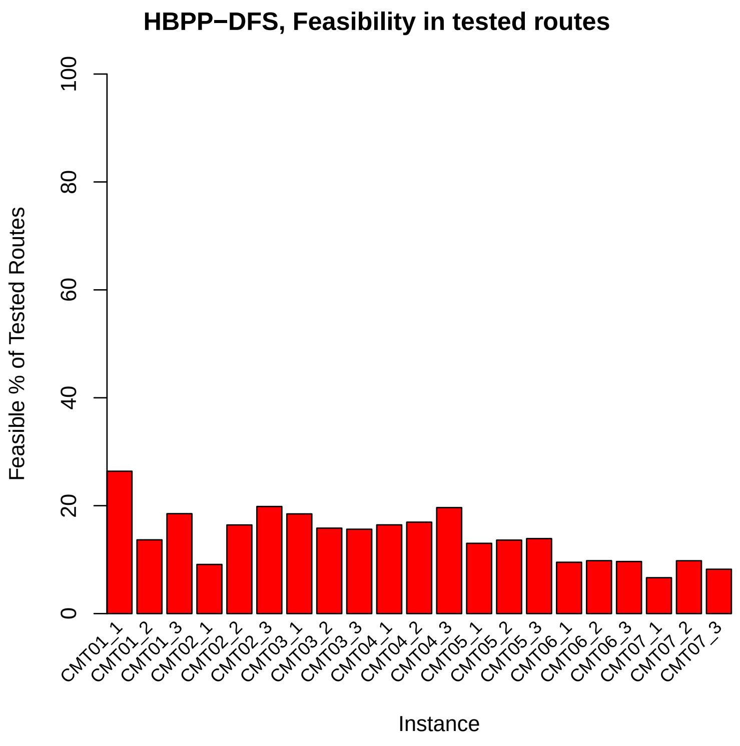


Figure 13.10: HBPP-DFS: % of tested routes that were *bb*-feasible

time the number of tested routes (and also of feasible routes) is lower, in some cases significantly lower, than for DING. The same holds for ACO-HCG. Thus even if a considerable amount of time is spent testing *bb*-feasibility of routes, it is still lower than for DING and ACO-HCG, who consequently discover a higher number of feasible routes. This can be explained by the simplistic pricing heuristic used in HBPP.

Finally, the exact numbers of feasible routes and routes stored in Feasibility store Ψ, for HBPA-DFS and HBPP-DFS are given in tables 13.7 and 13.8. Note that for HBPP, $|\Psi|$ corresponds to the total number of routes checked for *bb*-feasibility. For HBPA, it corresponds to the routes that have been positively tested for *bb*-feasibility in the tree, or the ones generated to initialize $\mathcal{R}^*$, or the ones generated in enforced route generation upon the construction of a new tree. As stated before, HBPP generally obtains a higher number of feasible routes than HBPA. HBPA-DFS has less routes in store and also less feasible routes than HBPA-LDS. HBPP-DFS has sometimes a higher, sometimes a lower amount of tested routes than HBPP-LDS. The same holds for the number of feasible routes. It should be noted however that on some instances HBPP-DFS tests more routes than HBPP-LDS while achieving a lower count of feasible routes. The opposite scenario can be observed as well on selected instances.

13.5.7.2 *Results obtained when solving $SCP(\mathcal{R}^*)$*

As for DING the final solution obtained in HBP is discovered in the Heuristic Branch & Price tree. Possibly better results could be obtained if one were to compute a final solution by solving problem $SCP(\mathcal{R}^*)$ over the set of feasible routes $\mathcal{R}^*$ accumulated during the HBP execution (approach denoted by HBP*). The results corresponding to this are given in the appendix (tables B.14 to B.17). Again the observations are similar to those made for the 3L-CVRP. HBP* allows to improve over the results obtained with HBP. The improvements are bigger for HBPA*-LDS and HBPP*-LDS than for HBPA*-DFS and HBPP*-DFS. Again highest improvements are obtained with HBPP*-LDS. Solving $SCP(\mathcal{R}^*)$ allows to improve the results of HBPP-LDS up to 2.24% (less than for DING-LDS). Also the results obtained with HBPP*-LDS improve (on average) over those obtained with HBPP*-DFS in 19 out of 21 instances. Furthermore the results obtained with HBPP*-LDS improve (on average) over those obtained with HBPA*-LDS in 9 out of 21 instances. The execution times needed to solve $SCP(\mathcal{R}^*)$ are similar for all configurations.

13.5.8 *Comparison of all methods*

Table 13.9 compares the results obtained on all considered approaches using the configurations described in this section with the best results obtained using the Variable Neighborhood Search in its best configuration (VNS) presented in [TDHI09]. It is the loading algorithm described in this paper that has been used as black box function and is used in the experiments described in this section. Columns z_{min} and z_{avg} give the best and the average solution cost determined over 10

Instance	**% Total Execution Time**					**Routes in Ψ**	
	$\%LP$	$\%BB$	$\%ST$	$\%GEN$	$\%PO$	#Feasible routes	$\lvert\Psi\rvert$
CMT01-1	0.4	73.4	0	25.5	0.1	2179.6	8037.8
CMT01-2	1.1	33.0	0	64.1	0.1	2557.2	9539.4
CMT01-3	1.2	19.5	0	76.9	0.1	2708.8	10977.7
CMT02-1	0.4	15.3	0	83.7	0.0	2143.1	5818.6
CMT02-2	0.4	19.2	0	79.8	0.0	2331.4	7893.9
CMT02-3	0.5	17.3	0	81.7	0.0	2349.0	8181.1
CMT03-1	0.2	23.3	0	76.3	0.0	2094.2	6342.2
CMT03-2	0.2	13.2	0	86.4	0.0	2068.3	6718.2
CMT03-3	0.2	15.8	0	83.7	0.0	2394.7	8018.7
CMT04-1	0.1	36.4	0	63.4	0.0	2092.5	7309.0
CMT04-2	0.1	27.9	0	71.9	0.0	2261.4	9174.9
CMT04-3	0.1	24.8	0	74.9	0.0	2363.7	9265.0
CMT05-1	0.0	50.3	0	49.5	0.0	2315.2	9715.1
CMT05-2	0.1	48.2	0	51.6	0.1	2431.9	11013.4
CMT05-3	0.1	35.5	0	64.3	0.0	2471.3	12506.9
CMT06-1	0.1	54.5	0	45.3	0.0	2483.5	9140.6
CMT06-2	0.1	50.1	0	49.6	0.0	2243.2	8310.7
CMT06-3	0.1	22.9	0	76.8	0.0	2351.6	9328.3
CMT07-1	0.1	23.0	0	76.6	0.0	2266.7	6165.3
CMT07-2	0.1	15.1	0	84.5	0.0	2146.3	6146.2
CMT07-3	0.1	18.3	0	81.3	0.0	2348.9	6257.5

Table 13.7: HBPA-DFS : CPU Time Allocation, # feasible routes, # routes in Feasibility Store ($\lvert\Psi\rvert$)

Instance	**% Total Execution Time**					**Routes in Ψ**	
	%*LP*	%*BB*	%*ST*	%*GEN*	%*PO*	#Feasible routes	$\lvert\Psi\rvert$
CMT01-1	0.3	70.0	1.8	27.4	0.1	2980.2	11268.3
CMT01-2	0.4	59.1	2.4	37.3	0.1	3566.6	27800.2
CMT01-3	0.6	32.0	4.0	61.8	0.1	4197.1	23648.9
CMT02-1	0.2	26.5	3.9	69.0	0.1	2685.3	34209.7
CMT02-2	0.2	39.6	3.2	56.6	0.1	3438.5	23275.8
CMT02-3	0.3	28.6	3.7	67.0	0.1	3564.2	19707.5
CMT03-1	0.1	29.8	3.4	66.5	0.1	2977.0	18325.0
CMT03-2	0.1	18.0	3.9	77.7	0.1	2985.0	20252.4
CMT03-3	0.1	22.0	3.7	74.0	0.1	2965.0	20646.5
CMT04-1	0.1	36.8	2.6	60.4	0.1	2839.7	17355.9
CMT04-2	0.1	19.7	3.4	76.8	0.1	2730.4	16704.5
CMT04-3	0.1	21.3	3.3	75.2	0.1	3267.1	16791.3
CMT05-1	0.1	24.3	2.7	72.9	0.1	2794.0	21710.8
CMT05-2	0.1	32.9	2.3	64.6	0.1	2567.9	18948.5
CMT05-3	0.1	19.7	2.9	77.2	0.1	2936.5	21268.2
CMT06-1	0.0	53.7	2.0	44.0	0.1	2587.4	32282.7
CMT06-2	0.0	58.3	1.8	39.7	0.1	2750.1	35429.1
CMT06-3	0.1	28.4	3.2	68.2	0.1	3030.6	41290.3
CMT07-1	0.0	33.0	3.0	63.8	0.1	2011.4	33331.4
CMT07-2	0.1	20.8	3.6	75.3	0.1	2200.1	30135.3
CMT07-3	0.0	31.3	3.1	65.4	0.1	2214.9	36986.0

Table 13.8: HBPP-DFS : CPU Time Allocation, # feasible routes, # routes in Feasibility Store ($\lvert\Psi\rvert$)

runs for all methods and for VNS. Columns sec_{tt} give the total execution times. They are only indicated for VNS and ACO- HCG as the values for DING and HBP are very similar to those of ACO-HCG.
ACO-HCG is able to improve the best solution from VNS in 14 out of 21 instances. Furthermore it improves the average solution in 15 instances. Small variations as these on instance CMT02-1 for example (615.11 vs 615.12) are to be attributed to rounding. The remaining methods clearly perform worse than ACO-HCG. DING is able to improve the best solution from VNS in 3 instances, HBP never achieves an improvement. None of DING or HBP is able to improve the average results of VNS in a consistent way, as ACO-HCG does. In terms of execution times, no exact numbers are given in [TDHI09]. However the time limit used is similar, and ACO-HCG exceeds the VNS time limit at most by 238 CPU seconds. The execution times are thus comparable.
In general DING, HBPA and HBPP perform worse for the MP-VRP than for the 3L-CVRP. This can be explained by the fact that the MP-VRP instances are larger (higher number of customers). Obviously DING and HBP do not scale well, which is to be expected due to the fact that the size of their respective search trees grows with the number of customers and vehicles (the latter of which is unlimited in the MP-VRP).

Table 13.10 shows, in columns $\%g_{avg}^{allb}$, the average relative deviation from the overall best known solution from the state-of-the-art for the MP-VRP (this doesn't include results published in [MDVH12] or [MLISD13]). The relative deviation for one run is computed as $100 \cdot \frac{z - z_{allb}}{z_{allb}}$, where z corresponds to the solution cost obtained using the methods presented in this thesis and z_{allb} corresponds to the best known solution cost. The table furthermore indicates in column z_{allb} for each instances the best known solution cost, and in columns z_{min} the cost of the best solution found over all runs for the presented methods. Negative values for$\%g_{avg}^{allb}$ indicate that the corresponding method improves the best known solution on average.

ACO-HCG improves or ties with the best known solution in 11 out of 21 instances. It is able to improve on average the best known in 2 instances. The results obtained with the other methods are less good. Only DING-LDS allows to improve the best known solution one one instance. Note that for ACO-HCG, the average gap w.r.t. the best known

Instance	VNS			ACO-HCG			DING-DFS		DING-LDS		HBPA-DFS		HBPA-LDS		HBPP-DFS		HBPP-LDS	
	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	z_{min}	z_{avg}	z_{min}	z_{avg}	z_{min}	z_{avg}	z_{min}	z_{avg}	z_{min}	z_{avg}
CMT01-1	590.55	591.80	1800	**587.81**	*590.10*	1823.6	**590.08**	598.14	594.54	599.13	**590.24**	604.74	598.24	603.81	594.88	604.10	598.14	604.72
CMT01-2	615.12	640.07	1800	**615.11**	*618.50*	1804.7	631.19	643.67	616.74	*621.84*	630.37	640.50	634.93	643.94	627.26	658.54	630.13	652.02
CMT01-3	623.45	634.46	1800	**623.44**	*624.76*	1804.4	628.94	647.79	623.91	*631.02*	633.24	642.14	631.68	640.25	633.03	646.43	631.68	640.38
CMT02-1	982.38	986.12	1800	**975.56**	*976.16*	1855.3	983.07	1018.05	**976.63**	*984.97*	1008.81	1036.08	993.97	1018.12	1000.02	1034.18	991.88	1026.13
CMT02-2	900.03	901.01	1800	**898.38**	902.93	1837.9	919.97	941.38	907.50	915.13	921.24	944.71	933.69	944.22	927.61	955.95	943.53	955.71
CMT02-3	889.29	892.11	1800	**889.26**	*890.85*	1817.8	903.61	942.57	889.88	904.18	916.07	942.10	926.99	934.13	921.57	939.43	914.48	933.59
CMT03-1	1191.83	1197.44	1800	**1183.76**	*1186.61*	1809.1	1212.75	1255.76	**1189.45**	1200.64	1230.50	1262.85	1232.23	1247.77	1231.14	1267.26	1210.47	1253.36
CMT03-2	1219.83	1222.37	1800	**1219.15**	1222.52	1893.8	1251.43	1268.20	1234.40	1246.28	1273.55	1287.75	1261.79	1282.92	1268.38	1291.56	1268.12	1293.43
CMT03-3	1161.03	1162.49	1800	**1157.22**	*1161.15*	1818.6	1190.55	1228.74	1174.49	1185.81	1208.34	1238.78	1215.68	1236.53	1202.59	1239.06	1206.35	1237.90
CMT04-1	1632.10	1635.78	1800	**1617.12**	*1630.01*	1934.3	1672.16	1705.93	**1629.61**	1659.57	1690.84	1715.42	1682.22	1707.66	1689.00	1713.11	1690.84	1716.13
CMT04-2	1559.05	1562.83	1800	**1547.30**	*1553.76*	1984.4	1586.18	1623.18	1573.16	1587.47	1610.61	1636.66	1612.69	1629.38	1612.69	1635.11	1612.69	1635.40
CMT04-3	1543.13	1547.90	1800	**1541.41**	1548.00	1883.7	1599.27	1636.48	1568.01	1577.40	1608.01	1647.48	1610.17	1634.05	1633.95	1653.62	1617.00	1646.42
CMT05-1	2038.75	2049.25	1800	**2024.91**	*2041.67*	2037.5	2065.69	2100.89	2052.35	2079.31	2097.42	2117.61	2085.40	2115.48	2091.13	2120.45	2098.10	2116.59
CMT05-2	1834.23	1839.54	1800	1849.76	1863.53	2038.8	1900.76	1918.47	1875.58	1893.33	1876.02	1924.84	1899.32	1928.29	1876.02	1927.59	1898.15	1931.19
CMT05-3	1958.42	1965.85	1800	**1952.32**	*1961.75*	1985.1	2008.70	2040.91	1992.71	2003.07	2030.21	2059.56	2030.56	2057.08	2038.80	2063.51	2030.21	2058.76
CMT06-1	2247.34	2254.28	1800	2250.76	2258.60	1914.9	2312.36	2368.72	2298.87	2333.72	2335.93	2399.26	2337.28	2375.67	2328.45	2373.29	2318.86	2393.91
CMT06-2	2089.68	2102.64	1800	**2085.16**	2103.69	1863.9	2130.06	2178.89	2130.10	2170.45	2159.63	2222.91	2147.80	2199.64	2145.30	2220.54	2147.80	2201.92
CMT06-3	2155.23	2183.12	1800	**2152.84**	*2168.93*	1848.3	2213.72	2268.24	2190.97	2212.80	2287.59	2335.43	2231.28	2312.68	2216.30	2319.96	2234.47	2301.26
CMT07-1	1136.58	1136.61	1800	1139.85	1149.18	1845.5	1171.28	1191.66	1177.31	1188.15	1185.63	1211.69	1186.52	1213.12	1185.63	1217.40	1175.08	1215.07
CMT07-2	1224.61	1228.61	1800	**1214.95**	*1217.20*	1843.4	1253.61	1283.33	1236.68	1266.53	1248.98	1296.27	1236.68	1283.61	1239.03	1284.62	1248.98	1293.77
CMT07-3	1158.98	1165.13	1800	**1153.81**	1167.71	1836.4	1192.92	1229.35	1190.39	1206.16	1223.04	1241.75	1199.89	1228.38	1196.21	1240.75	1199.47	1225.63
AVG			<1800			1880.1												

Table 13.9: Comparison of proposed methods with VNS ([TDHI09]). $z_{min/avg}$ corresponds to the best/average solution value over 10 runs. sec_{tt} corresponds to the average total execution time. sec_{tt} is indicated for VNS and ACO-HCG only. Remaining execution times are equal (give or take 5 seconds) to ACO-HCG. Values in **bold** in the z_{min} columns indicate that the corresponding approach is a tie with or improves the best solution of the VNS. Values in *italic* in the z_{avg} column indicate that the average over 10 runs obtained with the corresponding approach is a tie with or improves the average results published in [TDHI09].

solution, averaged over all instances, amounts to only 0.5%. This value is higher, up to 6% for the other methods.

Instance	**Best known**	**ACO-HCG**		**DING-DFS**		**DING-LDS**		**HBPA-DFS**		**HBPA-LDS**		**HBPP-DFS**		**HBPP-LDS**	
	z_{min}	z_{min}	g_{avg}	z_{min}	g_{avg}	z_{min}	g_{avg}	z_{min}	g_{avg}	z_{min}	g_{avg}	z_{min}	g_{avg}	z_{min}	g_{avg}
CMT01-1	587.29	587.81	0.5	590.08	1.9	594.54	2.0	590.24	3.0	598.24	2.8	594.88	2.9	598.14	3.0
CMT01-2	615.12	**615.11**	0.6	631.19	4.6	616.74	1.1	630.37	4.1	634.93	4.7	627.26	7.1	630.13	6.0
CMT01-3	623.45	**623.44**	0.2	628.94	3.9	623.91	1.2	633.24	3.0	631.68	2.7	633.03	3.7	631.68	2.7
CMT02-1	978.66	**975.56**	-0.3	983.07	4.0	**976.63**	0.6	1008.81	5.9	993.97	4.0	1000.02	5.7	991.88	4.9
CMT02-2	897.62	898.38	0.6	919.97	4.9	907.5	2.0	921.24	5.2	933.69	5.2	927.61	6.5	943.53	6.5
CMT02-3	888.38	889.26	0.3	903.61	6.1	889.88	1.8	916.07	6.0	926.99	5.1	921.57	5.7	914.48	5.1
CMT03-1	1188.18	**1183.76**	-0.1	1212.75	5.7	1189.45	1.1	1230.50	6.3	1232.23	5.0	1231.14	6.7	1210.47	5.5
CMT03-2	1218.96	1219.15	0.3	1251.43	4.0	1234.40	2.2	1273.55	5.6	1261.79	5.2	1268.38	6.0	1268.12	6.1
CMT03-3	1156.84	1157.22	0.4	1190.55	6.2	1174.49	2.5	1208.34	7.1	1215.68	6.9	1202.59	7.1	1206.35	7.0
CMT04-1	1624.98	**1617.12**	0.3	1672.16	5.0	1629.61	2.1	1690.84	5.6	1682.22	5.1	1689.00	5.4	1690.84	5.6
CMT04-2	1552.27	**1547.30**	0.1	1586.18	4.6	1573.16	2.3	1610.61	5.4	1612.69	5.0	1612.69	5.3	1612.69	5.4
CMT04-3	1541.81	**1541.41**	0.4	1599.27	6.1	1568.01	2.3	1608.01	6.9	1610.17	6.0	1633.95	7.3	1617.00	6.8
CMT05-1	2035.77	**2024.91**	0.3	2065.69	3.2	2052.35	2.1	2097.42	4.0	2085.40	3.9	2091.13	4.2	2098.10	4.0
CMT05-2	1833.41	1849.76	1.6	1900.76	4.6	1875.58	3.5	1876.02	5.0	1899.32	5.2	1876.02	5.1	1898.15	5.3
CMT05-3	1948.84	1952.32	0.7	2008.70	4.7	1992.71	2.8	2030.21	5.7	2030.56	5.6	2038.80	5.9	2030.21	5.6
CMT06-1	2240.57	2250.76	0.8	2312.36	5.7	2298.87	4.2	2335.93	7.1	2337.28	6.0	2328.45	5.9	2318.86	6.8
CMT06-2	2070.04	2085.16	1.6	2130.06	5.3	2130.10	4.9	2159.63	7.4	2147.80	6.3	2145.30	7.3	2147.80	6.4
CMT06-3	2154.19	**2152.84**	0.7	2213.72	5.3	2190.97	2.7	2287.59	8.4	2231.28	7.4	2216.30	7.7	2234.47	6.8
CMT07-1	1136.55	1139.85	1.1	1171.28	4.9	1177.31	4.5	1185.63	6.6	1186.52	6.7	1185.63	7.1	1175.08	6.9
CMT07-2	1217.45	**1214.95**	0.0	1253.61	5.4	1236.68	4.0	1248.98	6.5	1236.68	5.4	1239.03	5.5	1248.98	6.3
CMT07-3	1157.67	**1153.81**	0.9	1192.92	6.2	1190.39	4.2	1223.04	7.3	1199.89	6.1	1196.21	7.2	1199.47	5.9
AVG			0.5		4.9		2.6		5.8		5.3		6.0		5.6

Table 13.10: Comparison of proposed methods with overall best known results (all-best) for MP-VRP. z_{min} corresponds to the best solution value (over 10 runs for the proposed methods). $\%g_{avg}^{allb}$ corresponds to the average relative percentage deviation from the best known result. Negative $\%g_{avg}^{allb}$ values indicate that the best known solution is improved on average. Values in **bold** in the z_{min} columns indicate that the corresponding approach is a tie with or improves the best known.

14

VRPBB WITH MINMAX OBJECTIVE

The MinMax Capacitated Vehicle Routing Problem is a basic CVRP with a different objective. The goal is to find a solution such that the distance of the longest (read highest distance) route is minimized. Different problems with this objective function have been considered in the literature, such as the CVRP [GLT97] or the selective VRP where not all customers must be visited [VMdCM11]. The objective also often appears in disaster management applications, where equal fairness to all customers is of importance [CVH08].

14.1 PROBLEM DESCRIPTION

The goal of the MinMax CVRP is to devise a solution $Sol = \{r_1, \dots, r_m\}$ amounting to a set of routes such that:

1. $|Sol| \leq |K|$
   the solution does not use more vehicles than available
2. $\bigcap_{r_i \in Sol} r_i.S = \emptyset$ and $\bigcup_{r_i \in Sol} r_i.S = V \backslash \{0\}$
   each customer is visited exactly once
3. $\sum_{j \in r.S} q_j \leq Q \qquad \forall r \in Sol,$
   the sum of the customer demands on a route does not exceed the capacity Q

4. min $\text{argmax}_{r \in Sol}\, distance(r)$
   the distance of the longest route is minimized

14.2 ADAPTATION OF ACO-HCG TO THE MINMAX OBJECTIVE

To adapt the approach presented in chapter 6 to the MinMax objective, three modifications are necessary. They are described below.

14.2.1 *MinMax VRPBB as a Set Covering Problem*

The Pheromone-based Heuristic Column Generation method proposed in chapter 6 solves the CVRPBB as a Set Covering/Partitioning Problem. In this classical formulation a cost, the distance, is associated with each route, and the goal is to select the set of routes such that the Set Covering/Partitioning constraints are respected and such that the total cost (i.e. distance) is minimized. Since in the case of the MinMax VRPBB the objective is a different one, the Set Partitioning formulation needs to be adapted as follows:

$$
\begin{aligned}
Min \quad & z && (14.1)\\
s.t. \quad & \sum_{r \in \mathcal{R}} a_{ir} x_r = 1 \quad \forall i \in V \backslash 0 && (14.2)\\
& \sum_{r \in \mathcal{R}} x_r \leq K && (14.3)\\
& o_r x_r \leq z \quad \forall r \in \mathcal{R} && (14.4)\\
& x_r \in \{0,1\} \quad \forall r \in \mathcal{R} && (14.5)
\end{aligned}
$$

$$
a_{ir} = \begin{cases} 1 & \text{if customer } i \text{ is visited in route } r \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in V \backslash 0, \quad \forall r \in \mathcal{R}
$$

As before set $\mathcal{R}$ corresponds to the set of all *wb*- and *bb*-feasible routes. The cost o_r associated with route $r \in \mathcal{R}$ is given by $distance(r) = \sum_{s=0}^{|r.S|} c_{r[s]r[s+1]}$. An additional variable z is introduced here. It is the value of this variable that must be minimized. Constraints 14.5 force z to take a value at least as high as the cost of each of the selected routes. As furthermore the value of z is minimized it will take exactly the value of the highest cost selected route. This problem formulated over a set of feasible routes $\mathcal{R}^*$ is called $MMSCP(\mathcal{R}^*)$.

14.2.2 *Solving the intermediary problem*

All of the presented approaches rely on the fact that the linear relaxation of the original problem, possibly augmented with additional constraints, is solved from time to time. In the case of the MinMax VRPBB it is important to have a tight global upper bound at all times (as is explained in the next section). In the case of the Pheromone-based Heuristic Column Generation not the linear relaxation but the integer problem will be solved, although possibly not to optimality for efficiency reasons, in order to obtain an upper bound. A similar approach is used in [VMdCM11].

14.2.3 *Generating Feasible Columns*

The Pheromone-based Heuristic Column Generation method uses a version of randomized savings heuristic to generate feasible routes. It starts from an initial state where each customer is visited in a route of its own, and then iteratively selects a pair of routes in the current state to be merged.

The heuristic computes a list of candidate merges containing the most attractive *wb*-feasible merges in the current state. The heuristic information used in the computation of this attractiveness is the savings value, i.e. the amount of distance that can be saved when executing the merge. This measure is however not appropriate when considering a MinMax objective. Using the savings value, nothing guarantees that the routes are built to be equally long. This is why the heuristic computes for each merge the amount by which the distance of the currently longest route in the current state is increased. This measure is then used as heuristic value in the computation of the attractiveness of each merge.

The heuristic evaluates the *bb*-feasibility of all or only of some of the merges in the candidate list, depending on the approach. If an upper bound UB is given on the objective value of the current MinMax VRPBB (by a previous solution for example), then we know that *bb*-

feasibility of routes r s.t. $distance(r) \geq UB$ need not be evaluated, since the route can never be part of a solution sol s.t. $Obj(sol) < UB$.

14.3 EXPERIMENTAL RESULTS

In this section the results obtained on both the 3L-CVRP and the MP-VRP with MinMax objective are presented. The time limits used are those already used in the previous experiments for the 3L-CVRP and the MP-VRP. Results have been obtained over 10 independent runs.

14.3.1 *MinMax 3L-CVRP*

The Pheromone-based Heuristic Column Generation has been applied using the automatic configuration (except for parameters *useint* and f as problem $MMSCP(\mathcal{R}^*)$ is solved every iteration for the MinMax). No automated parameter setting has been conducted fro the MinMax 3L-CVRP. The automatic configuration has been chosen for simplicity, but it is potentially not high-performing. The results are given in Table 14.1.

A look at Table 14.1 reveals that for the large-scale instances the relative standard deviation is quite high, up to 22.5% for instance 3l-cvrp25. An interesting observation is that for these instances, the cost of the final solution corresponds to the cost of the first feasible solution (combined from feasible routes generated completely at random). That is, the initial solution is never improved throughout the process. This is due to the fact that the time limits allowed for solving problem $MMSCP(\mathcal{R}^*)$ are reached before the current upper bound is improved. The lower bounds of problem $MMSCP(\mathcal{R}^*)$ are very weak, and thus do not allow to prune important parts of the search tree. This weakness is however reinforced by the fact, that in the current implementation, problem $MMSCP(\mathcal{R}^*)$ is solved from scratch each time. This could be easily remedied by using the current best solution to warm start CPLEX.

14.3.2 *MinMax MP-VRP*

As the number of vehicles is not limited in the MP-VRP benchmark instances, the objective of the problem will always correspond to the distance from the depot to the most distant customer and back. There-

Instance			**ACO-HCG-MINMAX**			
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_{tt}
3l-cvrp01	4	15	86.17	86.17	0.0	1801.0
3l-cvrp02	5	15	75.77	75.77	0.0	1800.3
3l-cvrp03	4	20	106.84	109.23	0.8	1803.0
3l-cvrp04	6	20	83.11	84.25	1.4	1801.2
3l-cvrp05	6	21	99.41	99.41	0.0	1807.2
3l-cvrp06	6	21	106.89	106.89	0.0	1802.1
3l-cvrp07	6	22	164.79	164.79	0.0	1808.2
3l-cvrp08	6	22	169.86	169.86	0.0	1806.3
3l-cvrp09	8	25	96.71	96.71	0.0	1801.3
3l-cvrp10	8	29	137.53	137.53	0.0	3635.0
3l-cvrp11	8	29	137.53	137.53	0.0	3626.3
3l-cvrp12	9	30	81.62	82.07	1.8	3603.4
3l-cvrp13	8	32	509.03	509.03	0.0	3648.0
3l-cvrp14	9	32	237.74	237.74	0.0	3633.3
3l-cvrp15	9	32	237.74	237.74	0.0	3633.9
3l-cvrp16	11	35	77.41	77.41	0.0	3611.0
3l-cvrp17	14	40	74.56	74.59	0.2	3607.1
3l-cvrp18	11	44	259.67	259.67	0.0	3654.1
3l-cvrp19	12	50	87.86	87.86	0.0	7286.4
3l-cvrp20	18	71	52.95	52.95	0.0	7360.2
3l-cvrp21	17	75	100.12	126.99	14.3	7440.7
3l-cvrp22	18	75	89.31	120.60	16.3	7426.5
3l-cvrp23	17	75	122.83	142.34	7.5	7418.4
3l-cvrp24	16	75	133.76	160.62	12.1	7386.8
3l-cvrp25	22	100	102.25	134.56	22.5	7495.0
3l-cvrp26	26	100	117.05	117.05	0.0	7455.8
3l-cvrp27	23	100	132.87	152.38	8.3	7521.7
AVG					3.1	4284.2

Table 14.1: Results for the 3L-CVRP with MinMax objective. $z_{min/avg}$= best/average distance of the longest route,$\%RSD$=relative standard deviation,sec_{tt}=average total execution time.

fore the number of vehicles has been limited to $n/5$ in these instances. The Pheromone-based Heuristic Column Generation has been applied using the automatic configuration (except for parameters *useint* and f as problem $MMSCP(\mathcal{R}^*)$ is solved every iteration for the MinMax). No automated parameter setting has been conducted fro the MinMax MP-VRP. The automatic configuration has been chosen for simplicity, but it is potentially not high-performing. The results are given in Table 14.2.

Clearly the limit on the number of vehicle is not sufficiently restrictive to forbid single-customers routes for most instances. We see that for all instances except those of class CMT02, the cost of the best solution and the average solution are the same. Over the 10 runs, the same best solution is produced. In these solutions the routes setting the solution cost (i.e. the longest route) correspond to one customer routes. Finally instances CMT03, CMT04 and CMT05 share a set of customers, among which we find the most distant customer for the three instance classes at a distance of 99.86/2 from the central depot.

Instance			ACO-HCG-MINMAX			
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_{tt}
CMT01-1	1	50	87.86	87.86	0.0	1883.4
CMT01-2	2	50	87.86	87.86	0.0	1857.0
CMT01-3	3	50	87.86	87.86	0.0	1842.0
CMT02-1	1	75	88.45	88.48	0.1	1962.7
CMT02-2	2	75	86.60	86.81	0.2	1982.0
CMT02-3	3	75	86.90	87.02	0.2	1947.7
CMT03-1	1	100	99.86	99.86	0.0	1959.9
CMT03-2	2	100	99.86	99.86	0.0	2060.7
CMT03-3	3	100	99.86	99.86	0.0	2008.8
CMT04-1	1	150	99.86	99.86	0.0	1982.2
CMT04-2	2	150	99.86	99.86	0.0	1964.1
CMT04-3	3	150	99.86	99.86	0.0	1960.1
CMT05-1	1	199	99.86	99.86	0.0	2140.2
CMT05-2	2	199	99.86	99.86	0.0	1923.7
CMT05-3	3	199	99.86	99.86	0.0	2139.8
CMT06-1	1	120	198.56	198.56	0.0	2134.0
CMT06-2	2	120	198.56	198.56	0.0	2111.3
CMT06-3	3	120	198.56	198.56	0.0	2051.0
CMT07-1	1	100	117.05	117.05	0.0	1827.3
CMT07-2	2	100	117.05	117.05	0.0	1895.1
CMT07-3	3	100	117.05	117.05	0.0	1859.4
AVG					0.0	1975.8

Table 14.2: Results for the MP-VRP with MinMax objective. $z_{min/avg}$= best/average distance of the longest route,$\%RSD$=relative standard deviation,sec_{tt}=average total execution time.

Part IV

CONCLUSIONS AND FUTURE WORK

15 CONCLUSION

The focus of this thesis is on Vehicle Routing Problems with a complicated side-problem. The objective was the design of optimization approaches for such Vehicle Routing Problems, that work independently of the side-problem. Existing approaches for such Vehicle Routing Problems are often ad-hoc, and make extensive use of the knowledge about the nature of the side-problem. The advantage of the approaches proposed in this thesis is that they do not necessitate laborious adaptations once the side-problem changes, as is the case for existing ad-hoc approaches.

In order to formalize the objective, this thesis proposed the Vehicle Routing Problem with Black Box Feasibility (VRPBB, chapter 4). In this problem, each route must verify a set of hidden constraints F_{bb}. In order to test the feasibility w.r.t. F_{bb}, a black box function, into which no insight is possible, is introduced. This problem has not be considered in the VRP literature so far.

Three incomplete optimization approaches for the VRPBB were proposed in this thesis. All of them are based on the reformulation of the VRPBB as a Set Partitioning Problem.

- Pheromone-based Heuristic Column Generation (chapter 6), combines ideas from Ant Colony Optimization and Heuristic Column Generation in a novel way. In order to allow the approach to learn which arcs are more likely to appear in feasible and efficient routes, pheromones are deposed on arcs appearing in solutions to the Linear Programming relaxation of the Set Partitioning Problem.
- Dive & Generate (chapter 7) is derived from the Branch & Generate primal heuristic. The idea of Branch & Generate is to perform a dive in a Branch & Price tree, where new columns are generated only when considered necessary. In Dive & Generate, this idea is extended with backtracking which allows to perform multiple dives, heuristic pruning and restarts. Aside from the usual Depth-First Search, a Limited Discrepancy Search is considered to explore the resulting tree.
- Heuristic Branch & Price (chapter 8) performs a classical Branch & Price, but with heuristic pruning and an incomplete tree exploration. Two different well-known ranching decisions are considered. Both Depth-First Search and Limited Discrepancy Search are considered.

These methods have been implemented into a coherent system which is available for download under the LGPL license at `http://becool.info.ucl.ac.be/resources/VRPBB`.

While these approaches were all designed and implemented under a Vehicle Routing perspective, the concepts could be easily applied to different problems, under the condition that Dantzig-Wolfe decomposition is applicable and the black box constraints concern columns individually. In all of the methods the pricing heuristic would have to be adapted to the problem at hand, but, as for Branch & Price, both the Dive & Generate and Heuristic Branch & Price methods are generic except for the pricing subproblem and the branching decisions. The Pheromone-based Heuristic Column Generation method furthermore requires an underlying graph structure on which the pheromones can be deposed. This graph structure can be explicit or implicit. If it is explicit, as for the Vehicle Routing Problem, the problem is defined on a graph and the pheromones can be deposed on this explicit graph. If it is implicit, as for example for a Generalized Assignment Problem, pheromones can be deposed on (variable,value) pairs from the original problem formulation.

One of the disadvantages of the presented methods, especially of the Pheromone-based Heuristic Column Generation approach, are the numerous parameters and design options that need to be set. The use of irace, a publicly available implementation of the Iterated F-Race method, allowed to elaborate an optimized configuration and parameter setting (chapter 11). The resulting configurations showed that the intuitive parameter setting used initially was emphasizing the wrong parameters. By using the optimized configurations and drastically modifying parameter values or design choices one at a time, precious insights into the role and impact of the different components of the method were gained. This shows the utility of methods such as Iterate F-Race, not only for parameter tuning, but also for improving the understanding of the approach under consideration.

In order to evaluate the proposed methods, they have been tested on two concrete instantiations of the VRPBB for which benchmark instances and best known results were available. Both applications, the 3L-CVRP (chapter 12) and the MP-VRP (chapter 13), combine Vehicle Routing and Loading. A route is only feasible if a solution to a corresponding loading problem can be found. The existing approaches for these problems all use some kind of knowledge about the loading problem or the nature of the solution process for this loading problem. The comparison with the state of the art (ad-hoc) approaches for both problems was done in two steps: (a) the average solution value and best solution value were compared to the approach proposing the loading algorithms that were used as black box functions in this thesis; (b) the best solution value was compared to the best known solution from the literature.
Among all methods proposed in this thesis, Pheromone-based Heuristic Column Generation works best for both the 3L-CVRP and the MP-VRP. In step (a) it is able to outperform the considered ad-hoc approach in terms of best and average solution value on a majority of instances. In step (b), it is able to detect new best solutions for 9 out of 21 (MP-VRP) and 10 out of 27 (3L-CVRP) instances. While the Dive & Generate and Heuristic Branch & Price methods perform well for the 3L-CVRP, they perform less good for the MP-VRP. This is possibly linked to the size of the MP-VRP instances. The complete solutions from the experiments can be downloaded from http://becool.info.ucl.ac.be/resources/VRPBB.

The experimental results show that the generic methods proposed in this thesis are able to perform as well, if not better, than existing methods specifically tailored to the problem at hand. In order to adapt the generic methods only a feasibility check needs to be provided. Existing methods necessitate the design and implementation effort, either of lower and upper bounds, ways to measure the violation of the side-problem, specific ways to guide the approach w.r.t the loading side-problem, or a complete new approach. The generic methods proposed in this thesis allow to reach similar results with a smaller design and implementation effort.

15.1 FUTURE WORK

The work presented in this thesis could be extended in many different directions. Especially the Dive & Generate and Heuristic Branch & Price methods leave room for additional improvements and experiments, but the Pheromone-based Heuristic Column Generation could be further developed as well.

- **Relaxation of the limit on the fleet size** Depending on the F_{bb} constraints, even finding an initial non-integral feasible solution might be difficult, if the number of available vehicles is limited. This is problematic for all of the proposed approaches in this thesis. A simple way to circumvent this problem is to initially allow the use of a higher number of vehicles than available. The number of allowed vehicles can then be gradually lowered throughout the optimization procedure, until reaching the correct fleet size.
- **Association Rule Learning** As has been shown in chapter 11, the pheromones in the Pheromone-based Heuristic Column Generation do not actually allow to learn which arcs are more likely to appear in feasible routes, but rather function as a diversification mechanism. Other learning mechanisms, such as association rule learning should be considered to establish a set of rules allowing to evaluate the probability of a given route to be feasible. If such an approach could be developed, it would be very promising for further research on the VRPBB.
- **Pricing algorithms** In both Dive & Generate and the Heuristic Branch & Price, adapted versions of the savings heuristic are used to do the pricing, i.e. to generate routes of negative reduced cost.

This is a rather naïve approach, and substantial improvements could be done on this basis. One could imagine a simplified Label setting algorithm to solve a Resource-constrained shortest path problem. Only a small set of neighbors would be considered for each customer. A more promising pricing heuristic would be the perturbation of routes selected in the current non-integral solution. Customers currently not visited could be added to, or swapped with customers currently visited in, the route.

- **Beam Search** Most Branch & Price applications for Vehicle Routing Problems use Best First Search to explore the Branch & Price tree. In this thesis Depth First Search and Limited Discrepancy Search have been considered, both for the Dive & Generate and Heuristic Branch & Price trees. Following the Best First principle could possibly allow to improve the results obtained with these methods. It avoids the pitfalls of Depth First Search, while using a strong guidance from the lower bounds. In this thesis, Best First Search has not been considered due to its memory requirements. Beam Search corresponds to a Best First Search where only a fixed number of nodes is kept in the frontier at all times. A compromise between completeness and memory consumption could be made by using Beam Search.
- **irace on remaining methods** The Iterated F-Race procedure has been applied only to the Pheromone-based Heuristic Column Generation method so far. It might however also be useful to apply Iterated F-Race to Dive & Generate and Heuristic Branch & Price, especially as the parameter values chosen for these methods haven't been determined experimentally. Furthermore the type of branching decision used in Heuristic Branch & Price could be considered a design choice, and it would be left to the Iterated F-Race procedure to determine whether it is preferable to branch on arcs or on customer pairs. This could provide further insight into the reasons why one method performs better or worse on some problem instances.
- **Symmetry Handling** In the current Dive & Generate and Heuristic Branch & Price methods, symmetries in the corresponding trees are not handled. While more complex mechanisms could be imagined, a simple Tabu List of visited nodes could already allow to avoid revisiting a subset of symmetric branches.
- **Branching heuristic** The branching heuristics used in the Heuristic Branch & Price method are simple. More sophisticated heuristics,

based on past performances could be used. One possibility would for example be Pseudo-cost branching ([Ach05]). Here route features are attributed a score, and the feature maximizing this score is selected to branch on next. The score is computed based on the number of times that branching on this feature resulted in a feasible solution in the past, and the objective cost of this solution. In the Heuristic Branch & Price method, pseudo-cost branching could be used as is, or adapted by computing the score of a feature based on the number of times that branching on this feature occurred on a path to an integral solution.

- **Multiple Column Generation iterations and stabilization procedures** In the current versions of Dive & Generate and Heuristic Branch & Price only at most a single Column Generation iteration is performed at each node. One could consider performing multiple iterations, possibly only at very promising nodes. Observations of the evolution of the dual costs in the Pheromone-based Heuristic Column Generation method show that the dual costs vary significantly from one iteration to the next, even if the cost of the dual solution doesn't change. This implies that if multiple column generation iterations were performed in Dive & Generate or Heuristic Branch & Price, stabilization procedures, such as these mentioned in [Fei10], would have to be introduced.

- **Other applications** This thesis considered two concrete instantiations of the Vehicle Routing Problem with Black Box Feasibility, both combining routing and loading. Future work should consider other applications, such as for example Vehicle Routing Problems with Driver Break Scheduling ([PGDDR10]), or Vehicle Routing Problems with Pallet Packing ([ZTK12]). As previously stated the approaches have been designed and implemented for Vehicle Routing Problems, but could be adapted to other problems, such as the Generalized Assignment Problem or Airline Crew Scheduling. This would demand the implementation of problem-specific pricing heuristics and branching decisions (for Dive & Generate and Heuristic Branch & Price).

- **Comparison with exact methods** In [TDHI09] the authors propose an exact approach to solve the MP-VRP. This exact approach is tested on a set of reduced (small-scale) instances. Pheromone-based Heuristic Column Generation and the other methods should be evaluated on these instances. This would allow to determine how close the solution cost obtained using Pheromone-based Heuristic Column Generation (and the other methods) comes to optimal solution cost.

Part V

APPENDIX

A

3L-CVRP: EXPERIMENTAL RESULTS - ADDITIONAL TABLES AND PLOTS

a.0.1 *Execution Times for the 3L-CVRP in the state-of-the-art*

Reference	$Minsec_{tt}$	$Maxsec_{tt}$	$Avgsec_{tt}$
[GILM06]	1800	7200	4200
[FDHI10]	12	10483.9	1793.1
[TZK09]	11.5	9915.7	2415.9
[Bor12]	0.9	453	228.6
[RTS11]	-	-	805.3
[RZMS13]	98.85	3354	754.21
[ZQLW12]	36.7	8511	2252.2

Table A.1: Minimal, Maximal and Average reported total execution times in the 3L-CVRP state-of-the-art

A.1 PHEROMONE-BASED HEURISTIC COLUMN GENERATION

A.1.1 *Comparison of architectures over 10 runs*

Majorana : architecture used in in chapter 11, AMD Opteron 6272 CPU (2.1 GHz, 16 MB cache size), gcc 4.4.6, CPLEX 12.4 and Boost Libraries v. 1.41

Claus : architecture used in in chapter 12: AMD Opteron 6284 SE CPU (2.7 Ghz, 16MB cache size), gcc 4.4.7, CPLEX 12.4 and Boost Libraries v. 1.53

Instance			***Majorana***			***Claus***				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	$\%g^{min}$	$\%g^{avg}$
3l-cvrp01	4	15	302.02	302.02	1800.00	302.02	302.02	1800.0	0.00	0.00
3l-cvrp02	5	15	334.96	334.96	1800.00	334.96	334.96	1800.0	0.00	0.00
3l-cvrp03	4	20	388.09	391.04	1800.20	385.53	390.12	1800.3	-0.66	-0.24
3l-cvrp04	6	20	437.19	437.19	1800.00	437.19	437.19	1800.0	0.00	0.00
3l-cvrp05	6	21	447.73	447.73	1800.00	447.73	447.73	1800.2	0.00	0.00
3l-cvrp06	6	21	498.16	498.16	1800.00	498.16	498.16	1800.1	0.00	0.00
3l-cvrp07	6	22	769.68	769.68	1800.00	769.68	769.68	1800.4	0.00	0.00
3l-cvrp08	6	22	845.50	848.47	1800.00	845.50	845.50	1800.6	0.00	-0.35
3l-cvrp09	8	25	630.13	630.13	1800.00	630.13	630.13	1800.0	0.00	0.00
3l-cvrp10	8	29	826.66	826.66	3603.50	826.66	826.66	3603.2	0.00	0.00
3l-cvrp11	8	29	776.19	777.34	3600.60	776.19	777.91	3601.5	0.00	0.07
3l-cvrp12	9	30	612.25	612.25	3600.20	612.25	612.25	3600.3	0.00	0.00
3l-cvrp13	8	32	2661.62	2667.17	3600.90	2661.62	2665.48	3601.6	0.00	-0.06
3l-cvrp14	9	32	1385.00	1401.45	3606.55	1385.00	1398.84	3605.3	0.00	-0.19
3l-cvrp15	9	32	1336.21	1340.07	3618.90	1336.22	1341.02	3624.1	0.00	0.07
3l-cvrp16	11	35	698.61	698.61	3600.00	698.61	698.61	3600.3	0.00	0.00
3l-cvrp17	14	40	866.40	866.40	3600.00	866.40	866.40	3600.1	0.00	0.00
3l-cvrp18	11	44	1205.11	1207.29	3613.10	1205.11	1209.21	3606.2	0.00	0.16
3l-cvrp19	12	50	741.31	741.63	7203.20	741.31	741.31	7204.6	0.00	-0.04
3l-cvrp20	18	71	577.39	581.25	7217.30	577.85	580.84	7217.9	0.08	-0.07
3l-cvrp21	17	75	1075.16	1079.69	7217.10	1075.97	1079.47	7220.4	0.08	-0.02
3l-cvrp22	18	75	1148.82	1153.00	7214.10	1147.43	1154.64	7217.8	-0.12	0.14
3l-cvrp23	17	75	1101.47	1111.26	7212.90	1102.39	1110.24	7221.1	0.08	-0.09
3l-cvrp24	16	75	1107.15	1112.29	7221.80	1107.71	1110.20	7228.3	0.05	-0.19
3l-cvrp25	22	100	1373.43	1387.54	7241.10	1370.70	1387.14	7256.1	-0.20	-0.03
3l-cvrp26	26	100	1544.40	1557.82	7218.60	1541.49	1548.15	7212.9	-0.19	-0.62
3l-cvrp27	23	100	1483.59	1489.16	7240.00	1483.66	1491.09	7253.1	0.00	0.13
AVG					4208.52			4210.2	-0.03	-0.05

Table A.2: $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, $g^{min/avg}$= relative percentage deviation w.r.t. *Majorana*, computed as $100 \cdot \frac{z_{min/avg} - z^{Maj}_{min/avg}}{z^{Maj}_{min/avg}}$ where $z_{min/avg}$ are the results obtained on *Claus* and $z^{Maj}_{min/avg}$ the results obtained on *Majorana*

a.1.2 *Comparison of ACO-HCG and ACO-HCG-NOPO*

Instance			ACO-HCG			ACO-HCG-NOPO				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
3l-cvrp01	4	15	302.02	302.02	1800.0	302.02	302.02	1800.1	0.00	0.00
3l-cvrp02	5	15	334.96	334.96	1800.0	334.96	334.96	1800.0	0.00	0.00
3l-cvrp03	4	20	385.53	390.12	1800.3	392.24	393.61	1801.7	1.74	0.89
3l-cvrp04	6	20	437.19	437.19	1800.0	437.19	437.19	1800.0	0.00	0.00
3l-cvrp05	6	21	447.73	447.73	1800.2	447.73	447.73	1800.4	0.00	0.00
3l-cvrp06	6	21	498.16	498.16	1800.1	498.16	498.16	1800.4	0.00	0.00
3l-cvrp07	6	22	769.68	769.68	1800.4	769.68	769.68	1800.4	0.00	0.00
3l-cvrp08	6	22	845.50	845.50	1800.6	845.50	848.77	1800.8	0.00	0.39
3l-cvrp09	8	25	630.13	630.13	1800.0	630.13	630.13	1800.1	0.00	0.00
3l-cvrp10	8	29	826.66	826.66	3603.2	826.66	827.33	3607.9	0.00	0.08
3l-cvrp11	8	29	776.19	777.91	3601.5	776.19	777.74	3602.7	0.00	-0.02
3l-cvrp12	9	30	612.25	612.25	3600.3	612.25	612.25	3600.9	0.00	0.00
3l-cvrp13	8	32	2661.62	2665.48	3601.6	2666.10	2669.26	3602.5	0.17	0.14
3l-cvrp14	9	32	1385.00	1398.84	3605.3	1396.60	1407.55	3606.0	0.84	0.62
3l-cvrp15	9	32	1336.22	1341.02	3624.1	1340.78	1346.95	3631.3	0.34	0.44
3l-cvrp16	11	35	698.61	698.61	3600.3	698.61	698.61	3600.6	0.00	0.00
3l-cvrp17	14	40	866.40	866.40	3600.1	866.40	866.40	3600.2	0.00	0.00
3l-cvrp18	11	44	1205.11	1209.21	3606.2	1207.44	1217.10	3618.7	0.19	0.65
3l-cvrp19	12	50	741.31	741.31	7204.6	746.86	747.44	7204.7	0.75	0.83
3l-cvrp20	18	71	577.85	580.84	7217.9	580.07	582.68	7215.3	0.38	0.32
3l-cvrp21	17	75	1075.97	1079.47	7220.4	1078.65	1089.70	7229.8	0.25	0.95
3l-cvrp22	18	75	1147.43	1154.64	7217.8	1147.93	1158.32	7227.5	0.04	0.32
3l-cvrp23	17	75	1102.39	1110.24	7221.1	1104.25	1115.19	7225.6	0.17	0.45
3l-cvrp24	16	75	1107.71	1110.20	7228.3	1109.93	1116.24	7251.3	0.20	0.54
3l-cvrp25	22	100	1370.70	1387.14	7256.1	1392.26	1403.82	7303.7	1.57	1.20
3l-cvrp26	26	100	1541.49	1548.15	7212.9	1552.86	1564.28	7229.5	0.74	1.04
3l-cvrp27	23	100	1483.66	1491.09	7253.1	1490.47	1499.35	7267.7	0.46	0.55
AVG					4210.2			4215.9	0.29	0.35

Table A.3: Comparison of ACO-HCG in automatic configuration (ACO-HCG) with ACO-HCG in automatic configuration without Post-optimization of routes (ACO-HCG-NOPO). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time. $g^{min/avg}$= relative percentage deviation of ACO-HCG-NOPO w.r.t. ACO-HCG , computed as $100 \cdot \frac{z_{min/avg} - z^{D}_{min/avg}}{z^{D}_{min/avg}}$ where $z_{min/avg}$ are the results obtained with ACO-HCG-NOPO and $z^{D}_{min/avg}$ the results obtained with ACO-HCG.

Instance	**% Total Execution Time**						**Routes in Ψ**	
	%IP	*%LP*	*%BB*	*%ST*	*%GEN*	*%PO*	#Feasible routes	$\lvert\Psi\rvert$
3l-cvrp01	5.0	0.9	9.8	4.6	68.0	0	2042.2	4391.8
3l-cvrp02	4.0	0.8	0.1	4.5	79.9	0	862.2	868.9
3l-cvrp03	20.9	0.5	48.0	1.4	27.4	0	5836.5	10397.4
3l-cvrp04	1.1	0.8	3.4	3.8	82.8	0	2870.3	3289.6
3l-cvrp05	5.4	0.4	54.3	1.9	36.2	0	4613.5	13994.2
3l-cvrp06	19.0	0.8	14.1	2.6	57.7	0	5147.9	6918.7
3l-cvrp07	0.2	0.1	81.9	0.9	16.5	0	3550.9	16600.7
3l-cvrp08	3.6	0.1	79.3	0.9	15.5	0	4540.6	25805.2
3l-cvrp09	3.2	0.5	6.2	3.1	83.5	0	2891.0	4896.9
3l-cvrp10	2.6	0.0	94.8	0.1	2.4	0	7223.0	45213.4
3l-cvrp11	1.6	0.0	93.3	0.2	4.6	0	9685.9	55415.6
3l-cvrp12	15.6	0.4	9.0	1.8	70.0	0	5204.1	6801.2
3l-cvrp13	0.5	0.0	96.4	0.1	2.8	0	7389.5	34379.1
3l-cvrp14	0.3	0.0	99.1	0.0	0.5	0	6954.4	38064.9
3l-cvrp15	2.6	0.0	96.7	0.0	0.6	0	8859.8	37529.2
3l-cvrp16	9.9	0.3	1.4	1.6	84.4	0	5705.6	5939.6
3l-cvrp17	2.1	0.3	0.7	1.5	93.9	0	4730.9	5070.0
3l-cvrp18	0.1	0.0	99.4	0.0	0.4	0	5718.0	30428.5
3l-cvrp19	0.2	0.0	95.7	0.1	3.9	0	10034.3	71768.8
3l-cvrp20	0.2	0.0	98.3	0.0	1.4	0	10866.8	79554.1
3l-cvrp21	0.2	0.0	98.7	0.0	1.0	0	9566.9	62571.1
3l-cvrp22	0.9	0.0	95.4	0.1	3.6	0	12554.9	107494.4
3l-cvrp23	0.6	0.0	96.8	0.0	2.5	0	11833.1	68598.0
3l-cvrp24	10.3	0.0	72.2	0.1	16.7	0	29043.6	64069.2
3l-cvrp25	1.0	0.0	97.3	0.0	1.6	0	11612.0	73739.2
3l-cvrp26	0.6	0.0	94.7	0.1	4.6	0	12810.8	132618.5
3l-cvrp27	2.5	0.0	92.7	0.0	4.6	0	16173.2	91587.4

Table A.4: ACO-HCG-NOPO : CPU Time Allocation, # feasible routes, # routes in Feasibility Store ($\lvert\Psi\rvert$)

A.1.3 *Comparison of ACO-HCG and DECOMP ACO-HCG*

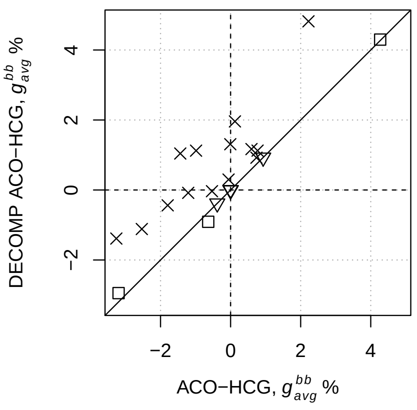


Figure A.1: Comparison of DECOMP ACO-HCG and ACO-HCG. Each point gives the average relative deviation from the *bb-best* solution over 10 independent runs. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

A.2 DIVE & GENERATE

a.2.1 *Comparison with $\overline{\text{ACO-HCG}}$*

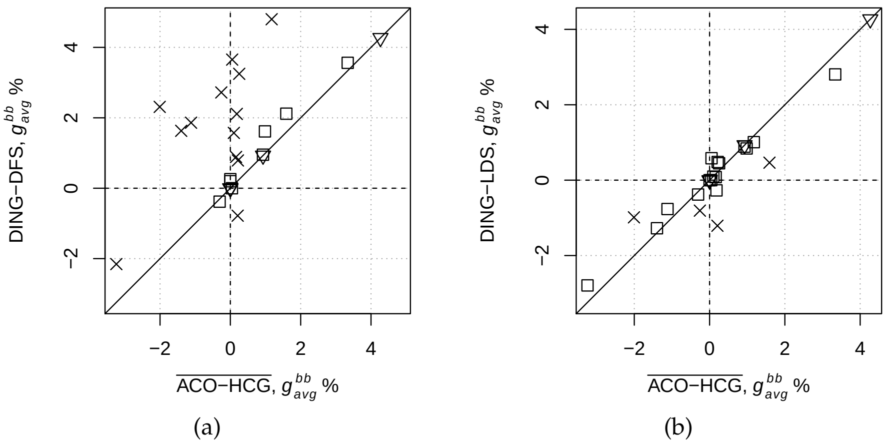


Figure A.2: Comparison of DING and $\overline{\text{ACO-HCG}}$, ACO-HCG in the manual configuration. Each point gives the average relative deviation from the *bb-best* solution over 10 independent runs. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

a.2.2 *Explicit Results for DING*

Instance			DING-DFS					
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_h	sec_{tt}	$\%g^{bb}_{avg}$
3l-cvrp01	4	15	302.02	302.02	0.0	12.2	1800.1	0.00
3l-cvrp02	5	15	334.96	334.96	0.0	0.5	1800.1	0.00
3l-cvrp03	4	20	385.53	389.56	0.8	300.3	1800.2	-0.78
3l-cvrp04	6	20	437.19	437.19	0.0	2.7	1800.0	0.00
3l-cvrp05	6	21	447.73	447.73	0.0	102.1	1800.1	0.93
3l-cvrp06	6	21	498.16	498.16	0.0	28.6	1800.1	0.00
3l-cvrp07	6	22	769.68	769.68	0.0	121.1	1801.6	0.00
3l-cvrp08	6	22	845.50	845.50	0.0	240.0	1800.1	4.27
3l-cvrp09	8	25	630.13	631.38	0.6	38.7	1800.1	0.20
3l-cvrp10	8	29	826.66	833.59	1.2	1620.2	3600.3	1.61
3l-cvrp11	8	29	778.10	786.29	2.1	1289.8	3600.2	-2.16
3l-cvrp12	9	30	612.25	612.25	0.0	80.2	3600.1	-0.38
3l-cvrp13	8	32	2665.33	2671.16	0.2	1292.4	3600.4	0.95
3l-cvrp14	9	32	1401.64	1417.16	1.3	2448.5	3600.4	3.56
3l-cvrp15	9	32	1342.32	1351.72	0.4	2755.8	3600.8	0.79
3l-cvrp16	11	35	698.61	700.40	0.3	613.7	3600.6	0.26
3l-cvrp17	14	40	866.40	874.09	0.9	543.0	3600.6	0.89
3l-cvrp18	11	44	1209.81	1233.29	1.0	2801.1	3600.5	2.12
3l-cvrp19	12	50	741.31	753.39	1.0	3236.2	7200.5	1.57
3l-cvrp20	18	71	583.29	603.94	2.0	3795.1	7200.7	2.72
3l-cvrp21	17	75	1100.87	1125.64	1.3	3719.0	7200.7	3.25
3l-cvrp22	18	75	1167.28	1202.89	1.4	2878.5	7200.7	4.80
3l-cvrp23	17	75	1119.59	1151.58	1.2	3344.1	7200.7	1.86
3l-cvrp24	16	75	1131.51	1156.91	1.4	3843.5	7200.6	3.65
3l-cvrp25	22	100	1420.01	1437.07	1.0	3915.8	7200.8	2.11
3l-cvrp26	26	100	1594.20	1637.32	1.2	2200.0	7201.0	2.31
3l-cvrp27	23	100	1519.41	1554.80	1.5	4103.1	7201.2	1.63
AVG					0.8	1678.7	4200.5	1.34

Table A.5: Explicit results on 3L-CVRP using Dive & Generate with Depth First Search, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_h = Time at which $UB = z_{min}$, sec_{tt} = average total execution time, $\%g^{bb}_{avg}$=average relative percentage deviation w.r.t. *bb-best* solution.

Instance			**DING-LDS**					
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_h	sec_{tt}	$\%g^{bb}_{avg}$
3l-cvrp01	4	15	302.02	302.02	0.0	12.4	1800.1	0.00
3l-cvrp02	5	15	334.96	334.96	0.0	0.2	1800.0	0.00
3l-cvrp03	4	20	385.53	387.88	0.6	369.4	1800.2	-1.21
3l-cvrp04	6	20	437.19	437.19	0.0	2.6	1800.0	0.00
3l-cvrp05	6	21	447.73	447.73	0.0	152.7	1800.2	0.93
3l-cvrp06	6	21	498.16	498.16	0.0	16.1	1800.1	0.00
3l-cvrp07	6	22	769.68	769.68	0.0	119.5	1802.1	0.00
3l-cvrp08	6	22	845.50	845.50	0.0	140.6	1800.7	4.27
3l-cvrp09	8	25	630.13	630.13	0.0	112.4	1800.1	0.00
3l-cvrp10	8	29	826.66	827.25	0.1	1575.9	3600.5	0.84
3l-cvrp11	8	29	776.19	781.17	1.1	755.9	3600.2	-2.79
3l-cvrp12	9	30	612.25	612.25	0.0	46.8	3600.1	-0.38
3l-cvrp13	8	32	2665.33	2669.47	0.1	1373.3	3601.1	0.89
3l-cvrp14	9	32	1384.09	1406.81	0.8	2607.5	3600.8	2.81
3l-cvrp15	9	32	1341.73	1347.60	0.3	1636.4	3603.0	0.48
3l-cvrp16	11	35	698.61	698.61	0.0	318.6	3600.2	0.00
3l-cvrp17	14	40	866.40	867.10	0.2	60.9	3600.2	0.08
3l-cvrp18	11	44	1207.44	1213.32	0.6	2249.4	3604.8	0.46
3l-cvrp19	12	50	741.31	742.44	0.3	3756.2	7201.0	0.09
3l-cvrp20	18	71	580.38	583.19	0.4	4567.9	7213.0	-0.81
3l-cvrp21	17	75	1086.18	1095.10	0.5	4023.1	7202.1	0.45
3l-cvrp22	18	75	1147.93	1159.37	0.6	3321.0	7203.8	1.01
3l-cvrp23	17	75	1113.45	1121.88	0.5	4593.0	7202.3	-0.77
3l-cvrp24	16	75	1111.63	1122.62	0.8	3040.4	7200.4	0.58
3l-cvrp25	22	100	1382.84	1403.54	1.2	4204.4	7209.7	-0.27
3l-cvrp26	26	100	1566.11	1584.56	0.9	4809.2	7201.6	-0.99
3l-cvrp27	23	100	1494.29	1510.34	0.7	4849.6	7205.5	-1.28
AVG					0.4	1804.3	4202.0	0.16

Table A.6: Explicit results on 3L-CVRP using Dive & Generate with Limited Discrepancy Search, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_h = Time at which $UB = z_{min}$, sec_{tt} = average total execution time, $\%g^{bb}_{avg}$=average relative percentage deviation w.r.t. *bb-best* solution cost.

A.2.3 *Results for DING**

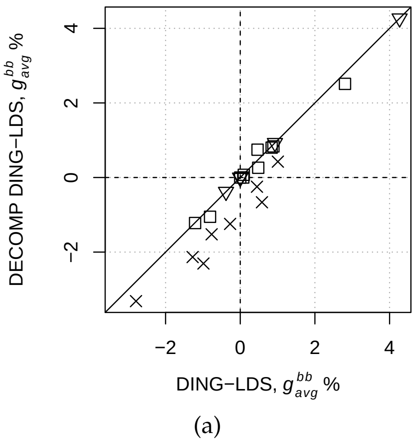
Instance			DING-DFS			DING*-DFS				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
3l-cvrp01	4	15	302.02	302.02	1800.1	302.02	302.02	0.1	0.00	0.00
3l-cvrp02	5	15	334.96	334.96	1800.1	334.96	334.96	0.0	0.00	0.00
3l-cvrp03	4	20	385.53	389.56	1800.2	385.53	387.58	1.5	0.00	-0.51
3l-cvrp04	6	20	437.19	437.19	1800.0	437.19	437.19	0.1	0.00	0.00
3l-cvrp05	6	21	447.73	447.73	1800.1	447.73	447.73	0.2	0.00	0.00
3l-cvrp06	6	21	498.16	498.16	1800.1	498.16	498.16	0.2	0.00	0.00
3l-cvrp07	6	22	769.68	769.68	1801.6	769.68	769.68	0.1	0.00	0.00
3l-cvrp08	6	22	845.50	845.50	1800.1	845.50	845.50	0.2	0.00	0.00
3l-cvrp09	8	25	630.13	631.38	1800.1	630.13	631.38	0.1	0.00	0.00
3l-cvrp10	8	29	826.66	833.59	3600.3	826.66	830.77	1.6	0.00	-0.34
3l-cvrp11	8	29	778.10	786.29	3600.2	778.10	783.07	0.2	0.00	-0.41
3l-cvrp12	9	30	612.25	612.25	3600.1	612.25	612.25	0.2	0.00	0.00
3l-cvrp13	8	32	2665.33	2671.16	3600.4	2665.33	2671.16	0.3	0.00	0.00
3l-cvrp14	9	32	1401.64	1417.16	3600.4	1401.64	1415.94	1.7	0.00	-0.09
3l-cvrp15	9	32	1342.32	1351.72	3600.8	1342.32	1348.59	4.6	0.00	-0.23
3l-cvrp16	11	35	698.61	700.40	3600.6	698.61	699.99	0.1	0.00	-0.06
3l-cvrp17	14	40	866.40	874.09	3600.6	866.40	870.90	0.1	0.00	-0.36
3l-cvrp18	11	44	1209.81	1233.29	3600.5	1209.80	1230.43	0.5	0.00	-0.23
3l-cvrp19	12	50	741.31	753.39	7200.5	741.31	752.02	0.3	0.00	-0.18
3l-cvrp20	18	71	583.29	603.94	7200.7	582.90	597.73	3.1	-0.07	-1.03
3l-cvrp21	17	75	1100.87	1125.64	7200.7	1098.82	1118.41	1.6	-0.19	-0.64
3l-cvrp22	18	75	1167.28	1202.89	7200.7	1167.28	1194.84	3.8	0.00	-0.67
3l-cvrp23	17	75	1119.59	1151.58	7200.7	1116.81	1144.84	2.4	-0.25	-0.59
3l-cvrp24	16	75	1131.51	1156.91	7200.6	1131.51	1149.60	3.1	0.00	-0.63
3l-cvrp25	22	100	1420.01	1437.07	7200.8	1408.37	1427.71	3.0	-0.82	-0.65
3l-cvrp26	26	100	1594.20	1637.32	7201.0	1592.79	1626.50	1.6	-0.09	-0.66
3l-cvrp27	23	100	1519.41	1554.80	7201.2	1519.41	1547.34	12.0	0.00	-0.48
AVG					4200.5			1.6	-0.05	-0.29

Table A.7: Comparison of Dive & Generate with Depth First Search (DING-DFS) and results obtained when solving problem $SCP(\mathcal{R}^*)$ over set of feasible routes $\mathcal{R}^*$ obtained in DING-DFS (DING*-DFS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, for DING*-DFS this corresponds to the time necessary to solve problem $SCP(\mathcal{R}^*)$. $g^{min/avg}$= relative percentage deviation of DING*-DFS w.r.t. DING-DFS , computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with DING*-DFS and $z^D_{min/avg}$ the results obtained with DING-DFS.

Instance			**DING-LDS**			**DING*-LDS**				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
3l-cvrp01	4	15	302.02	302.02	1800.1	302.02	302.02	0.1	0.00	0.00
3l-cvrp02	5	15	334.96	334.96	1800.0	334.96	334.96	0.0	0.00	0.00
3l-cvrp03	4	20	385.53	387.88	1800.2	385.53	387.32	1.2	0.00	-0.14
3l-cvrp04	6	20	437.19	437.19	1800.0	437.19	437.19	0.1	0.00	0.00
3l-cvrp05	6	21	447.73	447.73	1800.2	447.73	447.73	0.2	0.00	0.00
3l-cvrp06	6	21	498.16	498.16	1800.1	498.16	498.16	0.4	0.00	0.00
3l-cvrp07	6	22	769.68	769.68	1802.1	769.68	769.68	0.1	0.00	0.00
3l-cvrp08	6	22	845.50	845.50	1800.7	845.50	845.50	0.2	0.00	0.00
3l-cvrp09	8	25	630.13	630.13	1800.1	630.13	630.13	0.0	0.00	0.00
3l-cvrp10	8	29	826.66	827.25	3600.5	826.66	827.12	2.5	0.00	-0.02
3l-cvrp11	8	29	776.19	781.17	3600.2	776.19	778.60	0.5	0.00	-0.33
3l-cvrp12	9	30	612.25	612.25	3600.1	612.25	612.25	0.2	0.00	0.00
3l-cvrp13	8	32	2665.33	2669.47	3601.1	2665.33	2669.47	0.4	0.00	0.00
3l-cvrp14	9	32	1384.09	1406.81	3600.8	1384.09	1405.85	3.0	0.00	-0.07
3l-cvrp15	9	32	1341.73	1347.60	3603.0	1341.73	1343.94	16.1	0.00	-0.27
3l-cvrp16	11	35	698.61	698.61	3600.2	698.61	698.61	0.1	0.00	0.00
3l-cvrp17	14	40	866.40	867.10	3600.2	866.40	866.84	0.1	0.00	-0.03
3l-cvrp18	11	44	1207.44	1213.32	3604.8	1207.44	1210.78	0.8	0.00	-0.21
3l-cvrp19	12	50	741.31	742.44	7201.0	741.31	742.44	0.6	0.00	0.00
3l-cvrp20	18	71	580.38	583.19	7213.0	579.27	581.13	4.7	-0.19	-0.35
3l-cvrp21	17	75	1086.18	1095.10	7202.1	1081.17	1086.09	9.2	-0.46	-0.82
3l-cvrp22	18	75	1147.93	1159.37	7203.8	1145.18	1155.74	9.2	-0.24	-0.31
3l-cvrp23	17	75	1113.45	1121.88	7202.3	1109.61	1114.46	10.1	-0.34	-0.66
3l-cvrp24	16	75	1111.63	1122.62	7200.4	1111.63	1116.33	8.5	0.00	-0.56
3l-cvrp25	22	100	1382.84	1403.54	7209.7	1377.12	1387.93	55.6	-0.41	-1.11
3l-cvrp26	26	100	1566.11	1584.56	7201.6	1553.41	1566.65	38.9	-0.81	-1.13
3l-cvrp27	23	100	1494.29	1510.34	7205.5	1489.92	1492.64	69.5	-0.29	-1.17
AVG					4202.0			8.6	-0.10	-0.27

Table A.8: Comparison of Dive & Generate with Limited Discrepancy Search (DING-LDS) and results obtained when solving problem $SCP(\mathcal{R}^*)$ over set of feasible routes $\mathcal{R}^*$ obtained in DING-LDS (DING*-LDS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, for DING*-LDS this corresponds to the time necessary to solve problem $SCP(\mathcal{R}^*)$. $g^{min/avg}$= relative percentage deviation of DING*-LDS w.r.t. DING-LDS , computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with DING*-LDS and $z^D_{min/avg}$ the results obtained with DING-LDS.

A.2.4 *Comparison of DING-LDS and DECOMP DING-LDS*



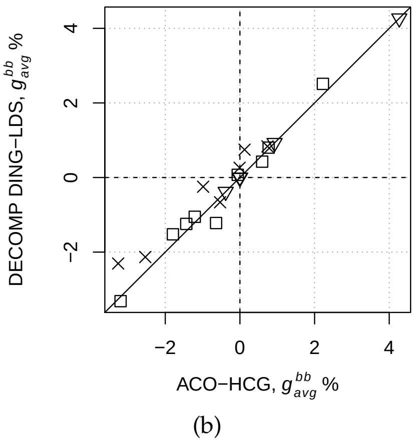


Figure A.3: Comparison of DECOMP DING-LDS with DING-LDS and with ACO-HCG Each point gives the average relative deviation from the *bb-best* solution over 10 independent runs. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

A.3 HEURISTIC BRANCH & PRICE

A.3.1 *Comparison with $\overline{\text{ACO-HCG}}$*

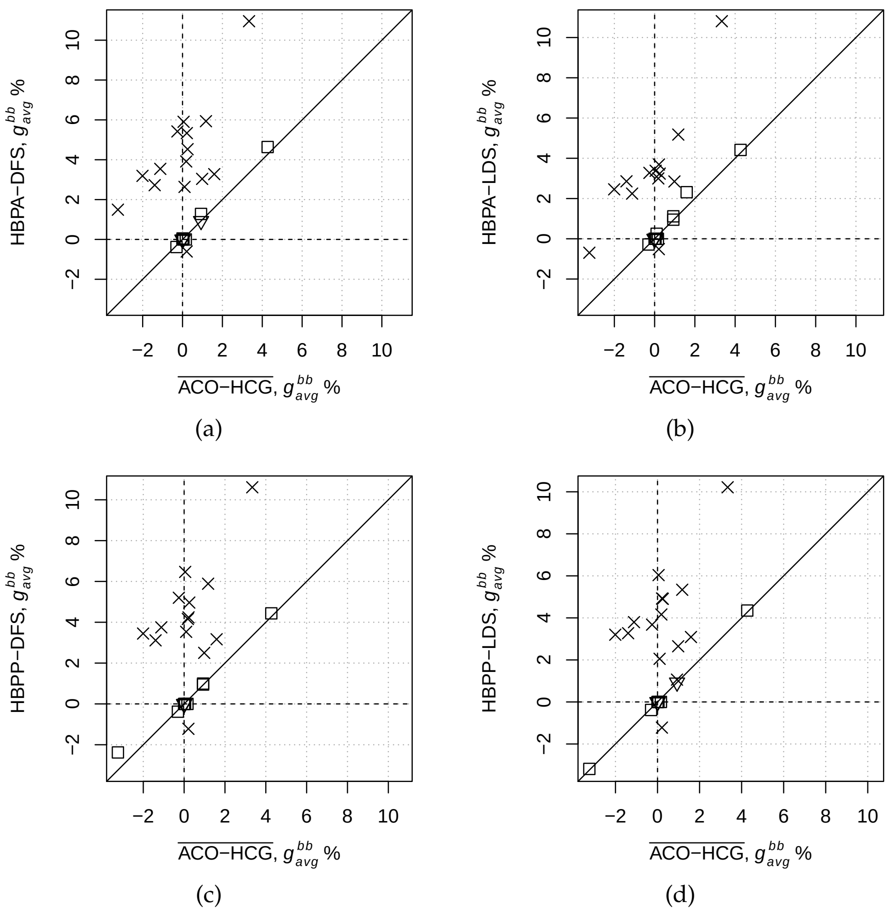


Figure A.4: Comparison of HBP and $\overline{\text{ACO-HCG}}$, the manual configuration of ACO-HCG. Each point gives the average relative deviation from the *bb-best* solution over 10 independent runs. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

A.3.2 *Explicit Results for HBP*

Instance			**HBPA-DFS**					
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_h	sec_{tt}	$\%g^{bb}_{avg}$
3l-cvrp01	4	15	302.02	302.02	0.0	49.3	1800.2	0.00
3l-cvrp02	5	15	334.96	334.96	0.0	0.8	1800.1	0.00
3l-cvrp03	4	20	388.09	390.25	0.6	451.6	1800.4	-0.61
3l-cvrp04	6	20	437.19	437.19	0.0	5.5	1800.0	0.00
3l-cvrp05	6	21	447.73	447.73	0.0	479.6	1800.8	0.93
3l-cvrp06	6	21	498.16	498.16	0.0	166.6	1800.1	0.00
3l-cvrp07	6	22	769.68	770.06	0.2	329.1	1805.3	0.05
3l-cvrp08	6	22	845.50	848.49	0.5	952.5	1800.7	4.64
3l-cvrp09	8	25	630.13	630.13	0.0	105.5	1800.2	0.00
3l-cvrp10	8	29	829.24	845.28	1.2	2133.5	3600.9	3.04
3l-cvrp11	8	29	787.04	815.61	1.7	1500.9	3603.4	1.49
3l-cvrp12	9	30	612.25	612.25	0.0	952.0	3600.1	-0.38
3l-cvrp13	8	32	2665.33	2679.80	0.5	1929.9	3607.5	1.28
3l-cvrp14	9	32	1497.31	1518.29	0.9	1333.1	3604.7	10.95
3l-cvrp15	9	32	1358.00	1412.71	2.6	2061.4	3604.2	5.34
3l-cvrp16	11	35	698.61	698.61	0.0	94.1	3600.1	0.00
3l-cvrp17	14	40	866.40	866.40	0.0	67.3	3600.2	0.00
3l-cvrp18	11	44	1225.62	1247.29	1.3	1878.6	3603.8	3.28
3l-cvrp19	12	50	741.31	761.29	2.7	5236.0	7204.0	2.64
3l-cvrp20	18	71	602.15	619.82	1.7	2882.7	7208.8	5.42
3l-cvrp21	17	75	1113.60	1139.60	1.3	3613.0	7204.7	4.53
3l-cvrp22	18	75	1190.92	1215.92	1.2	1246.8	7203.2	5.93
3l-cvrp23	17	75	1138.47	1170.57	1.1	1970.8	7207.9	3.54
3l-cvrp24	16	75	1164.36	1182.02	1.6	4810.5	7205.3	5.90
3l-cvrp25	22	100	1444.22	1462.44	0.9	2090.3	7217.0	3.91
3l-cvrp26	26	100	1635.49	1651.40	0.7	2126.7	7204.5	3.19
3l-cvrp27	23	100	1541.37	1571.54	1.3	2459.3	7212.7	2.72
AVG					0.8	1515.8	4203.7	2.51

Table A.9: Explicit results on 3L-CVRP using Heuristic Branch & Price, branching on arcs, with Depth First Search, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_h = Time of the first restart at which $UB = z_{min}$, sec_{tt} = average total execution time, $\%g^{bb}_{avg}$=average relative percentage deviation w.r.t. *bb-best* solution.

Instance			**HBPA-LDS**					
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_h	sec_{tt}	$\%g^{bb}_{avg}$
3l-cvrp01	4	15	302.02	302.02	0.0	60.2	1800.3	0.00
3l-cvrp02	5	15	334.96	334.96	0.0	1.2	1800.0	0.00
3l-cvrp03	4	20	385.53	390.61	0.7	737.6	1800.8	-0.51
3l-cvrp04	6	20	437.19	437.19	0.0	5.1	1800.0	0.00
3l-cvrp05	6	21	447.73	447.82	0.1	377.2	1800.4	0.95
3l-cvrp06	6	21	498.16	498.16	0.0	64.9	1800.1	0.00
3l-cvrp07	6	22	769.68	769.68	0.0	454.1	1803.2	0.00
3l-cvrp08	6	22	845.50	846.69	0.4	563.1	1801.0	4.41
3l-cvrp09	8	25	630.13	630.13	0.0	93.3	1800.1	0.00
3l-cvrp10	8	29	835.74	843.71	0.8	2516.4	3602.5	2.85
3l-cvrp11	8	29	778.24	798.03	1.8	2029.2	3602.1	-0.69
3l-cvrp12	9	30	612.25	612.84	0.2	654.6	3600.1	-0.28
3l-cvrp13	8	32	2670.50	2675.48	0.3	2245.4	3601.5	1.12
3l-cvrp14	9	32	1477.98	1516.36	1.1	1208.7	3606.6	10.81
3l-cvrp15	9	32	1349.73	1390.63	2.8	2107.9	3604.8	3.69
3l-cvrp16	11	35	698.61	698.61	0.0	88.2	3600.1	0.00
3l-cvrp17	14	40	866.40	866.40	0.0	63.3	3600.1	0.00
3l-cvrp18	11	44	1226.40	1235.68	0.8	2273.8	3608.4	2.31
3l-cvrp19	12	50	741.31	743.58	0.5	4732.0	7214.2	0.25
3l-cvrp20	18	71	594.50	607.20	1.7	3574.5	7205.7	3.27
3l-cvrp21	17	75	1103.02	1125.50	1.5	3759.9	7203.1	3.24
3l-cvrp22	18	75	1185.80	1207.22	1.1	1740.9	7203.5	5.18
3l-cvrp23	17	75	1128.80	1155.91	1.1	4429.9	7213.3	2.24
3l-cvrp24	16	75	1120.76	1153.84	1.5	4753.9	7204.1	3.38
3l-cvrp25	22	100	1421.81	1449.56	1.0	2927.2	7215.3	3.00
3l-cvrp26	26	100	1601.09	1639.66	1.1	3880.0	7207.7	2.46
3l-cvrp27	23	100	1541.37	1573.55	1.3	690.5	7207.5	2.86
AVG					0.7	1704.9	4203.9	1.87

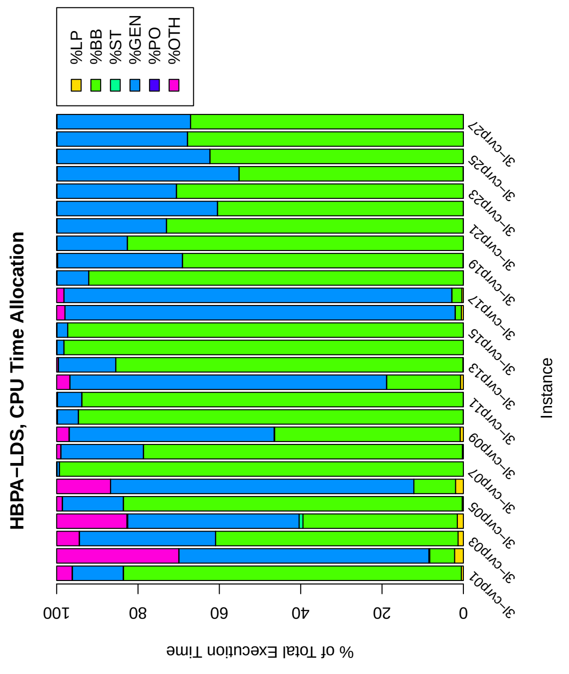
Table A.10: Explicit results on 3L-CVRP using Heuristic Branch & Price, branching on arcs, with Limited Discrepancy Search, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_h = Time at which $UB = z_{min}$, sec_{tt} = average total execution time, $\%g^{bb}_{avg}$=average relative percentage deviation w.r.t. *bb-best* solution.

Instance			HBPP-DFS					
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_h	sec_{tt}	$\%g^{bb}_{avg}$
3l-cvrp01	4	15	302.02	302.02	0.0	27.5	1800.1	0.00
3l-cvrp02	5	15	334.96	334.96	0.0	0.6	1800.0	0.00
3l-cvrp03	4	20	385.53	387.84	0.2	702.9	1800.4	-1.22
3l-cvrp04	6	20	437.19	437.19	0.0	4.4	1800.0	0.00
3l-cvrp05	6	21	447.73	447.82	0.1	557.1	1800.7	0.95
3l-cvrp06	6	21	498.16	498.16	0.0	35.8	1800.1	0.00
3l-cvrp07	6	22	769.68	769.68	0.0	258.5	1802.6	0.00
3l-cvrp08	6	22	845.50	846.86	0.3	975.7	1800.4	4.44
3l-cvrp09	8	25	630.13	630.13	0.0	62.0	1800.1	0.00
3l-cvrp10	8	29	828.75	840.87	1.1	1310.6	3601.5	2.50
3l-cvrp11	8	29	776.19	784.50	1.9	1665.2	3601.8	-2.38
3l-cvrp12	9	30	612.25	612.25	0.0	925.4	3600.1	-0.38
3l-cvrp13	8	32	2665.33	2672.30	0.2	1399.1	3606.3	1.00
3l-cvrp14	9	32	1437.55	1513.65	2.3	782.7	3604.8	10.61
3l-cvrp15	9	32	1356.08	1398.00	1.6	1994.5	3604.6	4.24
3l-cvrp16	11	35	698.61	698.61	0.0	55.3	3600.2	0.00
3l-cvrp17	14	40	866.40	866.40	0.0	195.4	3600.2	0.00
3l-cvrp18	11	44	1226.42	1245.96	0.7	1822.6	3616.8	3.17
3l-cvrp19	12	50	746.21	767.94	2.9	4741.1	7208.2	3.53
3l-cvrp20	18	71	601.55	618.53	1.9	3351.4	7205.8	5.20
3l-cvrp21	17	75	1108.16	1144.34	1.5	1428.7	7215.8	4.96
3l-cvrp22	18	75	1184.46	1215.38	1.5	2849.8	7206.1	5.89
3l-cvrp23	17	75	1158.35	1172.95	0.7	867.7	7208.6	3.75
3l-cvrp24	16	75	1168.77	1188.30	1.3	2014.8	7202.3	6.47
3l-cvrp25	22	100	1444.22	1465.93	0.9	647.1	7218.9	4.16
3l-cvrp26	26	100	1645.05	1655.50	0.4	2551.4	7204.1	3.45
3l-cvrp27	23	100	1548.00	1577.50	1.2	1796.4	7208.2	3.11
AVG					0.8	1223.1	4204.4	2.35

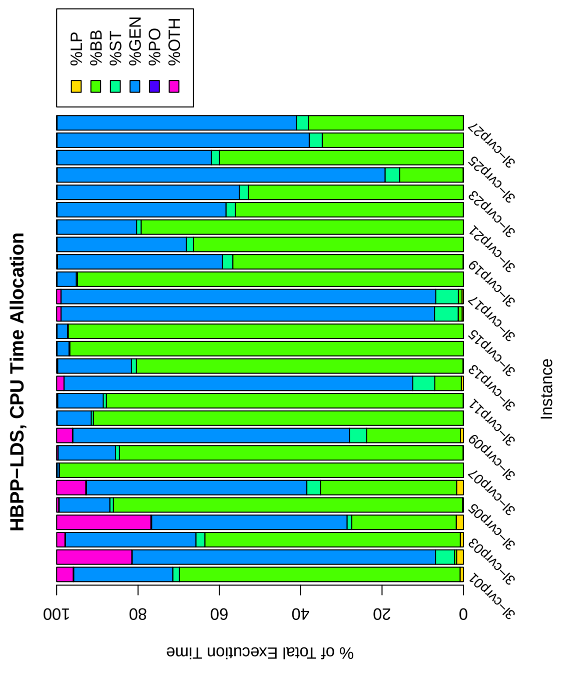
Table A.11: Explicit results on 3L-CVRP using Heuristic Branch & Price, branching on pairs, with Depth First Search, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_h = Time of the first restart at which $UB = z_{min}$, sec_{tt} = average total execution time, $\%g^{bb}_{avg}$=average relative percentage deviation w.r.t. *bb-best* solution.

Instance			**HBPP-LDS**					
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_h	sec_{tt}	$\%g^{bb}_{avg}$
3l-cvrp01	4	15	302.02	302.02	0.0	33.4	1800.1	0.00
3l-cvrp02	5	15	334.96	334.96	0.0	0.8	1800.0	0.00
3l-cvrp03	4	20	385.53	387.84	0.2	567.6	1800.3	-1.22
3l-cvrp04	6	20	437.19	437.19	0.0	3.7	1800.0	0.00
3l-cvrp05	6	21	447.73	447.73	0.0	643.0	1800.8	0.93
3l-cvrp06	6	21	498.16	498.16	0.0	29.7	1800.1	0.00
3l-cvrp07	6	22	769.68	769.68	0.0	298.9	1803.8	0.00
3l-cvrp08	6	22	845.50	846.16	0.2	353.5	1800.5	4.35
3l-cvrp09	8	25	630.13	630.13	0.0	51.1	1800.1	0.00
3l-cvrp10	8	29	828.75	842.09	1.4	2332.2	3601.3	2.65
3l-cvrp11	8	29	776.19	778.00	0.1	1224.3	3602.6	-3.19
3l-cvrp12	9	30	612.25	612.25	0.0	1344.9	3600.1	-0.38
3l-cvrp13	8	32	2670.50	2674.04	0.1	1534.0	3605.4	1.06
3l-cvrp14	9	32	1428.57	1508.20	2.4	658.3	3605.0	10.22
3l-cvrp15	9	32	1364.67	1407.16	1.9	1338.0	3604.3	4.92
3l-cvrp16	11	35	698.61	698.61	0.0	47.7	3600.1	0.00
3l-cvrp17	14	40	866.40	866.40	0.0	144.6	3600.2	0.00
3l-cvrp18	11	44	1225.46	1245.03	0.9	2046.5	3620.5	3.09
3l-cvrp19	12	50	743.62	756.96	2.2	5726.1	7200.7	2.05
3l-cvrp20	18	71	593.12	609.58	2.6	2784.2	7213.9	3.68
3l-cvrp21	17	75	1109.38	1143.71	1.4	2461.2	7209.8	4.91
3l-cvrp22	18	75	1184.05	1209.08	1.6	2828.4	7207.1	5.34
3l-cvrp23	17	75	1158.13	1173.47	0.7	1167.7	7203.6	3.80
3l-cvrp24	16	75	1154.37	1183.50	1.3	2365.9	7202.2	6.04
3l-cvrp25	22	100	1444.22	1465.93	0.9	595.9	7211.6	4.16
3l-cvrp26	26	100	1614.04	1651.56	1.1	2425.8	7208.0	3.20
3l-cvrp27	23	100	1548.00	1579.90	1.2	333.0	7209.8	3.27
AVG					0.7	1234.8	4204.1	2.18

Table A.12: Explicit results on 3L-CVRP using Heuristic Branch & Price, branching on pairs, with Limited Discrepancy Search, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_h = Time at which $UB = z_{min}$, sec_{tt} = average total execution time, $\%g^{bb}_{avg}$=average relative percentage deviation w.r.t. *bb-best* solution.



(a)



(b)

Figure A.5: HBPA-LDS and HBPP-LDS, CPU Time Allocation

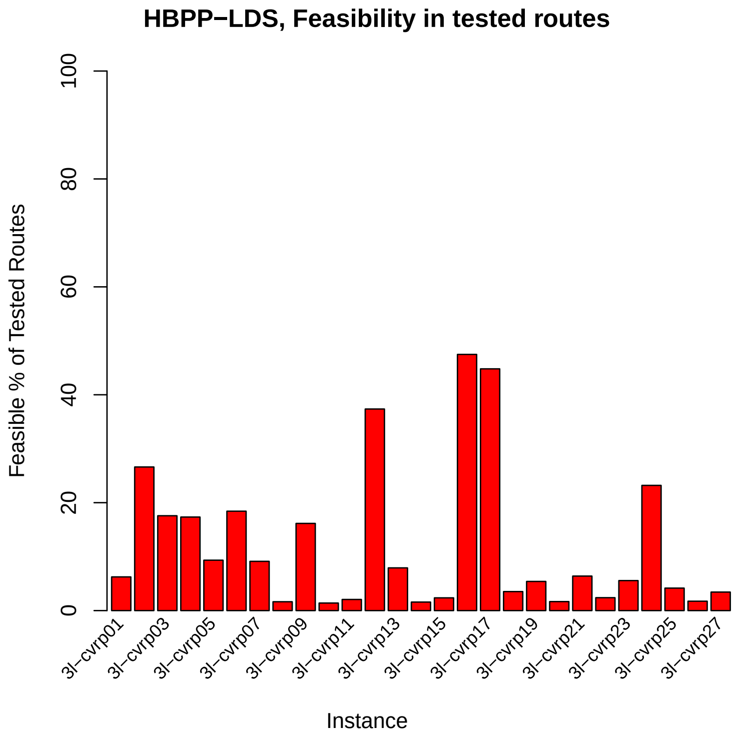


Figure A.6: HBPP-LDS, Percentage of feasible routes in tested routes

Instance	**% Total Execution Time**					**Routes in Ψ**	
	$\%LP$	$\%BB$	$\%ST$	$\%GEN$	$\%PO$	#Feasible routes	$\|\Psi\|$
3l-cvrp01	0.5	83.1	0.1	12.5	0.1	1874.3	14891.2
3l-cvrp02	2.2	6.1	0.2	61.5	0.0	788.1	2498.5
3l-cvrp03	1.3	59.6	0.0	33.5	0.1	2782.0	10434.2
3l-cvrp04	1.5	38.0	0.9	42.1	0.2	2753.6	17455.7
3l-cvrp05	0.3	83.3	0.0	15.0	0.0	2137.4	11371.1
3l-cvrp06	1.9	10.2	0.0	74.6	0.0	2325.3	5427.8
3l-cvrp07	0.0	99.3	0.0	0.6	0.1	1681.8	14715.9
3l-cvrp08	0.3	78.4	0.0	20.3	0.0	1297.3	8224.3
3l-cvrp09	0.8	45.6	0.2	50.4	0.1	2197.1	13447.7
3l-cvrp10	0.0	94.6	0.0	5.2	0.0	1532.6	5939.2
3l-cvrp11	0.0	93.8	0.0	6.0	0.0	1949.5	9737.9
3l-cvrp12	0.7	18.2	0.0	77.9	0.0	2076.2	5157.2
3l-cvrp13	0.1	85.4	0.0	14.1	0.0	2813.3	21002.1
3l-cvrp14	0.0	98.2	0.0	1.7	0.0	1515.9	6894.4
3l-cvrp15	0.0	97.3	0.0	2.6	0.0	1773.2	7773.6
3l-cvrp16	0.5	1.5	0.0	95.9	0.0	2224.7	5169.7
3l-cvrp17	0.4	2.4	0.1	95.3	0.0	2545.9	6153.5
3l-cvrp18	0.0	92.1	0.0	7.8	0.0	1602.5	11507.8
3l-cvrp19	0.1	69.0	0.0	30.7	0.0	4240.0	40640.8
3l-cvrp20	0.0	82.6	0.0	17.3	0.0	2036.5	15974.8
3l-cvrp21	0.0	73.0	0.0	26.9	0.0	2393.5	17160.7
3l-cvrp22	0.0	60.4	0.0	39.4	0.0	2487.1	17545.9
3l-cvrp23	0.0	70.5	0.0	29.4	0.0	2351.9	13466.1
3l-cvrp24	0.1	55.1	0.0	44.7	0.0	3185.6	8092.9
3l-cvrp25	0.0	62.3	0.0	37.6	0.0	2597.1	20158.8
3l-cvrp26	0.0	67.8	0.0	32.1	0.0	2212.6	22074.1
3l-cvrp27	0.0	67.1	0.0	32.9	0.0	2473.1	14892.7

Table A.13: HBPA-LDS : CPU Time Allocation and Ratio of Feasible routes / Routes in Feasibility Store ($|\Psi|$)

Instance	**% Total Execution Time**					**Routes in Ψ**	
	%LP	*%BB*	*%ST*	*%GEN*	*%PO*	#Feasible routes	|Ψ|
3l-cvrp01	0.8	69.0	1.6	24.4	0.1	1592.9	25545.3
3l-cvrp02	1.7	0.5	4.7	74.6	0.0	622.2	2338.0
3l-cvrp03	0.8	62.8	2.2	32.1	0.1	2440.9	13892.3
3l-cvrp04	1.8	25.7	1.2	48.0	0.2	2803.1	16174.4
3l-cvrp05	0.2	85.9	0.9	12.5	0.1	2051.2	22117.9
3l-cvrp06	1.7	33.5	3.4	54.2	0.2	2960.5	16107.2
3l-cvrp07	0.0	99.4	0.0	0.5	0.1	1573.6	17283.3
3l-cvrp08	0.1	84.5	0.9	14.1	0.0	893.0	54630.8
3l-cvrp09	0.7	23.1	4.2	68.0	0.1	2182.6	13513.2
3l-cvrp10	0.0	91.0	0.5	8.4	0.0	1198.9	87668.0
3l-cvrp11	0.1	87.7	0.8	11.2	0.0	2154.3	104977.8
3l-cvrp12	0.5	6.5	5.4	85.7	0.0	1951.6	5230.6
3l-cvrp13	0.1	80.3	1.2	18.2	0.1	2534.3	32199.4
3l-cvrp14	0.0	96.8	0.2	3.0	0.0	1146.3	73713.1
3l-cvrp15	0.0	97.1	0.2	2.6	0.0	1299.1	56551.5
3l-cvrp16	0.3	1.0	5.8	91.8	0.0	1977.0	4164.4
3l-cvrp17	0.3	0.9	5.6	92.1	0.0	2287.2	5106.1
3l-cvrp18	0.0	94.9	0.3	4.8	0.0	1205.3	34675.7
3l-cvrp19	0.1	56.6	2.5	40.6	0.0	3028.1	58186.3
3l-cvrp20	0.0	66.3	1.8	31.8	0.0	2017.1	135730.5
3l-cvrp21	0.0	79.2	1.1	19.6	0.0	2616.4	41800.7
3l-cvrp22	0.0	56.0	2.3	41.6	0.0	2345.8	100535.0
3l-cvrp23	0.0	52.9	2.2	44.8	0.0	2358.5	42981.7
3l-cvrp24	0.1	15.6	3.6	80.6	0.0	3230.3	13944.5
3l-cvrp25	0.0	59.9	2.0	38.0	0.0	2597.3	64209.3
3l-cvrp26	0.0	34.7	3.2	62.0	0.0	2105.1	121600.0
3l-cvrp27	0.0	38.1	2.9	58.9	0.0	2453.9	72544.9

Table A.14: HBPP-LDS : CPU Time Allocation and Ratio of Feasible routes / Routes in Feasibility Store (|Ψ|)

A.3.3 *Results for HBP**

Instance			HBPA-DFS			HBPA*-DFS				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
3l-cvrp01	4	15	302.02	302.02	1800.2	302.02	302.02	0.1	0.00	0.00
3l-cvrp02	5	15	334.96	334.96	1800.1	334.96	334.96	0.0	0.00	0.00
3l-cvrp03	4	20	388.09	390.25	1800.4	388.09	390.25	0.4	0.00	0.00
3l-cvrp04	6	20	437.19	437.19	1800.0	437.19	437.19	0.1	0.00	0.00
3l-cvrp05	6	21	447.73	447.73	1800.8	447.73	447.73	0.2	0.00	0.00
3l-cvrp06	6	21	498.16	498.16	1800.1	498.16	498.16	0.3	0.00	0.00
3l-cvrp07	6	22	769.68	770.06	1805.3	769.68	770.06	0.0	0.00	0.00
3l-cvrp08	6	22	845.50	848.49	1800.7	845.50	848.49	0.1	0.00	0.00
3l-cvrp09	8	25	630.13	630.13	1800.2	630.13	630.13	0.1	0.00	0.00
3l-cvrp10	8	29	829.24	845.28	3600.9	828.75	837.86	0.5	-0.06	-0.88
3l-cvrp11	8	29	787.04	815.61	3603.4	787.04	809.31	1.0	0.00	-0.77
3l-cvrp12	9	30	612.25	612.25	3600.1	612.25	612.25	0.3	0.00	0.00
3l-cvrp13	8	32	2665.33	2679.80	3607.5	2665.33	2677.72	0.3	0.00	-0.08
3l-cvrp14	9	32	1497.31	1518.29	3604.7	1469.56	1487.47	0.8	-1.85	-2.03
3l-cvrp15	9	32	1358.00	1412.71	3604.2	1352.41	1376.05	0.3	-0.41	-2.60
3l-cvrp16	11	35	698.61	698.61	3600.1	698.61	698.61	0.3	0.00	0.00
3l-cvrp17	14	40	866.40	866.40	3600.2	866.40	866.40	0.1	0.00	0.00
3l-cvrp18	11	44	1225.62	1247.29	3603.8	1225.62	1239.18	0.3	0.00	-0.65
3l-cvrp19	12	50	741.31	761.29	7204.0	741.31	755.78	0.7	0.00	-0.72
3l-cvrp20	18	71	602.15	619.82	7208.8	596.89	604.55	2.9	-0.87	-2.46
3l-cvrp21	17	75	1113.60	1139.60	7204.7	1108.56	1117.49	1.0	-0.45	-1.94
3l-cvrp22	18	75	1190.92	1215.92	7203.2	1184.17	1193.67	1.8	-0.57	-1.83
3l-cvrp23	17	75	1138.47	1170.57	7207.9	1133.19	1146.85	2.0	-0.46	-2.03
3l-cvrp24	16	75	1164.36	1182.02	7205.3	1145.15	1153.61	2.3	-1.65	-2.40
3l-cvrp25	22	100	1444.22	1462.44	7217.0	1424.37	1432.03	5.2	-1.37	-2.08
3l-cvrp26	26	100	1635.49	1651.40	7204.5	1606.10	1618.43	2.7	-1.80	-2.00
3l-cvrp27	23	100	1541.37	1571.54	7212.7	1524.21	1537.43	3.8	-1.11	-2.17
AVG					4203.7			1.0	-0.39	-0.91

Table A.15: Comparison of Heuristic Branch & Price, branching on arcs, with Depth First Search (HBPA-DFS) and results obtained when solving problem $SCP(\mathcal{R}^*)$ over set of feasible routes $\mathcal{R}^*$ obtained in HBPA-DFS (HBPA*-DFS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, for HBPA*-DFS this corresponds to the time necessary to solve problem $SCP(\mathcal{R}^*)$. $g^{min/avg}$= relative percentage deviation of HBPA*-DFS w.r.t. HBPA-DFS, computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with HBPA*-DFS and $z^D_{min/avg}$ the results obtained with HBPA-DFS.

Instance			**HBPA-LDS**			**HBPA*-LDS**				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
3l-cvrp01	4	15	302.02	302.02	1800.3	302.02	302.02	0.1	0.00	0.00
3l-cvrp02	5	15	334.96	334.96	1800.0	334.96	334.96	0.0	0.00	0.00
3l-cvrp03	4	20	385.53	390.61	1800.8	385.53	390.17	0.6	0.00	-0.11
3l-cvrp04	6	20	437.19	437.19	1800.0	437.19	437.19	0.0	0.00	0.00
3l-cvrp05	6	21	447.73	447.82	1800.4	447.73	447.82	0.2	0.00	0.00
3l-cvrp06	6	21	498.16	498.16	1800.1	498.16	498.16	0.2	0.00	0.00
3l-cvrp07	6	22	769.68	769.68	1803.2	769.68	769.68	0.0	0.00	0.00
3l-cvrp08	6	22	845.50	846.69	1801.0	845.50	846.63	0.1	0.00	-0.01
3l-cvrp09	8	25	630.13	630.13	1800.1	630.13	630.13	0.1	0.00	0.00
3l-cvrp10	8	29	835.74	843.71	3602.5	831.74	835.70	0.9	-0.48	-0.95
3l-cvrp11	8	29	778.24	798.03	3602.1	778.24	793.99	0.3	0.00	-0.51
3l-cvrp12	9	30	612.25	612.84	3600.1	612.25	612.84	0.4	0.00	0.00
3l-cvrp13	8	32	2670.50	2675.48	3601.5	2670.50	2675.48	0.3	0.00	0.00
3l-cvrp14	9	32	1477.98	1516.36	3606.6	1450.62	1475.24	2.0	-1.85	-2.71
3l-cvrp15	9	32	1349.73	1390.63	3604.8	1346.62	1368.15	0.9	-0.23	-1.62
3l-cvrp16	11	35	698.61	698.61	3600.1	698.61	698.61	0.3	0.00	0.00
3l-cvrp17	14	40	866.40	866.40	3600.1	866.40	866.40	0.1	0.00	0.00
3l-cvrp18	11	44	1226.40	1235.68	3608.4	1225.20	1230.92	0.3	-0.10	-0.39
3l-cvrp19	12	50	741.31	743.58	7214.2	741.31	743.58	0.5	0.00	0.00
3l-cvrp20	18	71	594.50	607.20	7205.7	590.13	596.81	2.2	-0.74	-1.71
3l-cvrp21	17	75	1103.02	1125.50	7203.1	1094.14	1104.63	1.5	-0.81	-1.85
3l-cvrp22	18	75	1185.80	1207.22	7203.5	1172.26	1177.95	3.2	-1.14	-2.42
3l-cvrp23	17	75	1128.80	1155.91	7213.3	1128.49	1140.62	7.1	-0.03	-1.32
3l-cvrp24	16	75	1120.76	1153.84	7204.1	1120.76	1137.39	4.3	0.00	-1.43
3l-cvrp25	22	100	1421.81	1449.56	7215.3	1411.21	1419.91	7.9	-0.75	-2.05
3l-cvrp26	26	100	1601.09	1639.66	7207.7	1588.60	1604.59	4.1	-0.78	-2.14
3l-cvrp27	23	100	1541.37	1573.55	7207.5	1521.98	1527.04	9.9	-1.26	-2.96
AVG					4203.9			1.8	-0.30	-0.82

Table A.16: Comparison of Heuristic Branch & Price, branching on arcs, with Limited Discrepancy Search (HBPA-LDS) and results obtained when solving problem $SCP(\mathcal{R}^*)$ over set of feasible routes $\mathcal{R}^*$ obtained in HBPA-LDS (HBPA*-LDS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, for HBPA*-LDS this corresponds to the time necessary to solve problem $SCP(\mathcal{R}^*)$. $g^{min/avg}$= relative percentage deviation of HBPA*-LDS w.r.t. HBPA-LDS, computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with HBPA*-LDS and $z^D_{min/avg}$ the results obtained with HBPA-LDS.

Instance			HBPP-DFS			HBPP*-DFS				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
3l-cvrp01	4	15	302.02	302.02	1800.1	302.02	302.02	0.1	0.00	0.00
3l-cvrp02	5	15	334.96	334.96	1800.0	334.96	334.96	0.0	0.00	0.00
3l-cvrp03	4	20	385.53	387.84	1800.4	385.53	387.84	0.5	0.00	0.00
3l-cvrp04	6	20	437.19	437.19	1800.0	437.19	437.19	0.1	0.00	0.00
3l-cvrp05	6	21	447.73	447.82	1800.7	447.73	447.82	0.2	0.00	0.00
3l-cvrp06	6	21	498.16	498.16	1800.1	498.16	498.16	0.3	0.00	0.00
3l-cvrp07	6	22	769.68	769.68	1802.6	769.68	769.68	0.0	0.00	0.00
3l-cvrp08	6	22	845.50	846.86	1800.4	845.50	846.86	0.1	0.00	0.00
3l-cvrp09	8	25	630.13	630.13	1800.1	630.13	630.13	0.1	0.00	0.00
3l-cvrp10	8	29	828.75	840.87	3601.5	826.66	833.06	0.3	-0.25	-0.93
3l-cvrp11	8	29	776.19	784.50	3601.8	776.19	783.48	0.3	0.00	-0.13
3l-cvrp12	9	30	612.25	612.25	3600.1	612.25	612.25	0.2	0.00	0.00
3l-cvrp13	8	32	2665.33	2672.30	3606.3	2665.33	2672.30	0.4	0.00	0.00
3l-cvrp14	9	32	1437.55	1513.65	3604.8	1436.04	1483.73	0.7	-0.11	-1.98
3l-cvrp15	9	32	1356.08	1398.00	3604.6	1353.88	1376.88	0.3	-0.16	-1.51
3l-cvrp16	11	35	698.61	698.61	3600.2	698.61	698.61	0.4	0.00	0.00
3l-cvrp17	14	40	866.40	866.40	3600.2	866.40	866.40	0.1	0.00	0.00
3l-cvrp18	11	44	1226.42	1245.96	3616.8	1225.62	1236.95	0.2	-0.07	-0.72
3l-cvrp19	12	50	746.21	767.94	7208.2	746.21	758.87	0.5	0.00	-1.18
3l-cvrp20	18	71	601.55	618.53	7205.8	593.12	601.81	0.9	-1.40	-2.70
3l-cvrp21	17	75	1108.16	1144.34	7215.8	1108.16	1121.75	1.8	0.00	-1.97
3l-cvrp22	18	75	1184.46	1215.38	7206.1	1178.48	1192.45	1.9	-0.50	-1.89
3l-cvrp23	17	75	1158.35	1172.95	7208.6	1135.73	1147.08	2.1	-1.95	-2.21
3l-cvrp24	16	75	1168.77	1188.30	7202.3	1147.91	1154.37	5.6	-1.78	-2.86
3l-cvrp25	22	100	1444.22	1465.93	7218.9	1425.87	1434.54	5.0	-1.27	-2.14
3l-cvrp26	26	100	1645.05	1655.50	7204.1	1602.77	1614.93	1.7	-2.57	-2.45
3l-cvrp27	23	100	1548.00	1577.50	7208.2	1527.45	1538.42	6.3	-1.33	-2.48
AVG					4204.4			1.1	-0.42	-0.93

Table A.17: Comparison of Heuristic Branch & Price, branching on customer pairs, with Depth First Search (HBPP-DFS) and results obtained when solving problem $SCP(\mathcal{R}^*)$ over set of feasible routes $\mathcal{R}^*$ obtained in HBPP-DFS (HBPP*-DFS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, for HBPP*-DFS this corresponds to the time necessary to solve problem $SCP(\mathcal{R}^*)$. $g^{min/avg}$= relative percentage deviation of HBPP*-DFS w.r.t. HBPP-DFS, computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with HBPP*-DFS and $z^D_{min/avg}$ the results obtained with HBPP-DFS.

Instance			**HBPP-LDS**			**HBPP*-LDS**				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
3l-cvrp01	4	15	302.02	302.02	1800.1	302.02	302.02	0.1	0.00	0.00
3l-cvrp02	5	15	334.96	334.96	1800.0	334.96	334.96	0.0	0.00	0.00
3l-cvrp03	4	20	385.53	387.84	1800.3	385.53	387.84	1.2	0.00	0.00
3l-cvrp04	6	20	437.19	437.19	1800.0	437.19	437.19	0.1	0.00	0.00
3l-cvrp05	6	21	447.73	447.73	1800.8	447.73	447.73	0.2	0.00	0.00
3l-cvrp06	6	21	498.16	498.16	1800.1	498.16	498.16	0.3	0.00	0.00
3l-cvrp07	6	22	769.68	769.68	1803.8	769.68	769.68	0.0	0.00	0.00
3l-cvrp08	6	22	845.50	846.16	1800.5	845.50	845.50	0.1	0.00	-0.08
3l-cvrp09	8	25	630.13	630.13	1800.1	630.13	630.13	0.1	0.00	0.00
3l-cvrp10	8	29	828.75	842.09	3601.3	826.66	832.70	1.2	-0.25	-1.12
3l-cvrp11	8	29	776.19	778.00	3602.6	776.19	778.00	0.3	0.00	0.00
3l-cvrp12	9	30	612.25	612.25	3600.1	612.25	612.25	0.3	0.00	0.00
3l-cvrp13	8	32	2670.50	2674.04	3605.4	2670.50	2673.36	0.3	0.00	-0.03
3l-cvrp14	9	32	1428.57	1508.20	3605.0	1428.57	1469.43	1.0	0.00	-2.57
3l-cvrp15	9	32	1364.67	1407.16	3604.3	1353.88	1364.35	0.5	-0.79	-3.04
3l-cvrp16	11	35	698.61	698.61	3600.1	698.61	698.61	0.3	0.00	0.00
3l-cvrp17	14	40	866.40	866.40	3600.2	866.40	866.40	0.1	0.00	0.00
3l-cvrp18	11	44	1225.46	1245.03	3620.5	1224.67	1233.63	0.3	-0.06	-0.92
3l-cvrp19	12	50	743.62	756.96	7200.7	741.31	750.70	0.5	-0.31	-0.83
3l-cvrp20	18	71	593.12	609.58	7213.9	587.52	595.34	3.5	-0.94	-2.34
3l-cvrp21	17	75	1109.38	1143.71	7209.8	1099.61	1108.31	2.3	-0.88	-3.10
3l-cvrp22	18	75	1184.05	1209.08	7207.1	1170.28	1179.01	7.1	-1.16	-2.49
3l-cvrp23	17	75	1158.13	1173.47	7203.6	1129.68	1138.51	6.1	-2.46	-2.98
3l-cvrp24	16	75	1154.37	1183.50	7202.2	1127.88	1135.78	5.7	-2.29	-4.03
3l-cvrp25	22	100	1444.22	1465.93	7211.6	1416.34	1424.03	12.9	-1.93	-2.86
3l-cvrp26	26	100	1614.04	1651.56	7208.0	1595.24	1604.41	3.8	-1.16	-2.85
3l-cvrp27	23	100	1548.00	1579.90	7209.8	1511.13	1525.70	11.6	-2.38	-3.43
AVG					4204.1			2.2	-0.54	-1.21

Table A.18: Comparison of Heuristic Branch & Price, branching on arcs, with Limited Discrepancy Search (HBPP-LDS) and results obtained when solving problem $SCP(\mathcal{R}^*)$ over set of feasible routes $\mathcal{R}^*$ obtained in HBPP-LDS (HBPP*-LDS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, for HBPP*-LDS this corresponds to the time necessary to solve problem $SCP(\mathcal{R}^*)$. $g^{min/avg}$= relative percentage deviation of HBPP*-LDS w.r.t. HBPP-LDS, computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with HBPP*-LDS and $z^D_{min/avg}$ the results obtained with HBPP-LDS.

B

MP-VRP: EXPERIMENTAL RESULTS - ADDITIONAL TABLES AND PLOTS

B.1 PHEROMONE-BASED HEURISTIC COLUMN GENERATION

b.1.1 *Comparison of architectures over 10 runs*

Majorana : architecture used in in chapter 11, AMD Opteron 6272 CPU (2.1 GHz, 16 MB cache size), gcc 4.4.6, CPLEX 12.4 and Boost Libraries v. 1.41

Claus : architecture used in in chapter 13: AMD Opteron 6284 SE CPU (2.7 Ghz, 16MB cache size), gcc 4.4.7, CPLEX 12.4 and Boost Libraries v. 1.53

Instance			*Majorana*			*Claus*				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g^{min}	g^{avg}
CMT01-1	1	50	590.24	590.83	1833.90	587.81	592.06	1823.6	-0.41	-0.12
CMT01-2	2	50	615.11	615.69	1813.20	615.11	623.51	1804.7	0.00	0.46
CMT01-3	3	50	623.44	624.55	1805.50	623.44	626.73	1804.4	0.00	0.03
CMT02-1	1	75	975.56	975.57	1848.40	975.56	976.63	1855.3	0.00	0.06
CMT02-2	2	75	897.51	899.91	1823.56	898.38	907.33	1837.9	0.10	0.34
CMT02-3	3	75	889.26	890.82	1815.70	889.26	894.53	1817.8	0.00	0.00
CMT03-1	1	100	1180.21	1184.01	1825.20	1183.76	1189.33	1809.1	0.30	0.22
CMT03-2	2	100	1219.13	1221.16	1951.70	1219.15	1225.52	1893.8	0.00	0.11
CMT03-3	3	100	1157.22	1159.23	1812.70	1157.22	1168.67	1818.6	0.00	0.17
CMT04-1	1	150	1607.63	1615.17	1907.40	1617.12	1646.22	1934.3	0.59	0.92
CMT04-2	2	150	1544.61	1549.13	1941.10	1547.30	1560.32	1984.4	0.17	0.30
CMT04-3	3	150	1541.35	1545.35	1886.30	1541.41	1558.26	1883.7	0.00	0.17
CMT05-1	1	199	2019.80	2027.71	2032.80	2024.91	2052.36	2037.5	0.25	0.69
CMT05-2	2	199	1832.18	1837.98	2010.00	1849.76	1882.64	2038.8	0.96	1.39
CMT05-3	3	199	1948.15	1952.12	1919.20	1952.32	1978.17	1985.1	0.21	0.49
CMT06-1	1	120	2244.32	2249.43	2137.10	2250.76	2273.50	1914.9	0.29	0.41
CMT06-2	2	120	2089.17	2094.35	1985.00	2085.16	2116.88	1863.9	-0.19	0.45
CMT06-3	3	120	2153.45	2163.95	1839.60	2152.84	2179.06	1848.3	-0.03	0.23
CMT07-1	1	100	1140.29	1144.35	1832.10	1139.85	1156.57	1845.5	-0.04	0.42
CMT07-2	2	100	1214.95	1216.10	1858.10	1214.95	1219.77	1843.4	0.00	0.09
CMT07-3	3	100	1152.16	1163.92	1843.80	1153.81	1182.56	1836.4	0.14	0.33
AVG					1891.54			1880.1	0.11	0.34

Table B.1: Comparison of different architectures, $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, $g^{min/avg}$= relative percentage deviation w.r.t. *Majorana*, computed as $100 \cdot \frac{z_{min/avg} - z^{Maj}_{min/avg}}{z^{Maj}_{min/avg}}$ where $z_{min/avg}$ are the results obtained on *Claus* and $z^{Maj}_{min/avg}$ the results obtained on *Majorana*

B.1.2 *Comparison of ACO-HCG and ACO-HCG-NOPO*

Instance			**ACO-HCG**			**ACO-HCG-NOPO**				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
CMT01-1	1	50	587.81	590.10	1823.60	587.29	591.20	1817.7	-0.09	0.19
CMT01-2	2	50	615.11	618.50	1804.70	615.11	618.44	1809.5	0.00	-0.01
CMT01-3	3	50	623.44	624.76	1804.40	623.44	625.08	1813.8	0.00	0.05
CMT02-1	1	75	975.56	976.16	1855.30	975.63	976.17	1880.6	0.01	0.00
CMT02-2	2	75	898.38	902.93	1837.90	898.80	903.04	1851.8	0.05	0.01
CMT02-3	3	75	889.26	890.85	1817.80	889.26	895.09	1825.1	0.00	0.48
CMT03-1	1	100	1183.76	1186.61	1809.10	1185.20	1188.01	1813.8	0.12	0.12
CMT03-2	2	100	1219.15	1222.52	1893.80	1222.04	1224.33	1905.9	0.24	0.15
CMT03-3	3	100	1157.22	1161.15	1818.60	1157.54	1163.46	1821.9	0.03	0.20
CMT04-1	1	150	1617.12	1630.01	1934.30	1615.80	1631.04	1927.6	-0.08	0.06
CMT04-2	2	150	1547.30	1553.76	1984.40	1553.29	1559.78	1941.4	0.39	0.39
CMT04-3	3	150	1541.41	1548.00	1883.70	1550.46	1554.69	1906.0	0.59	0.43
CMT05-1	1	199	2024.91	2041.67	2037.50	2042.46	2053.66	2096.4	0.87	0.59
CMT05-2	2	199	1849.76	1863.53	2038.80	1867.68	1880.19	2048.6	0.97	0.89
CMT05-3	3	199	1952.32	1961.75	1985.10	1960.35	1976.87	2006.2	0.41	0.77
CMT06-1	1	120	2250.76	2258.60	1914.90	2271.79	2281.34	1994.3	0.93	1.01
CMT06-2	2	120	2085.16	2103.69	1863.90	2119.27	2128.38	1997.8	1.64	1.17
CMT06-3	3	120	2152.84	2168.93	1848.30	2170.27	2194.78	1854.4	0.81	1.19
CMT07-1	1	100	1139.85	1149.18	1845.50	1145.40	1153.74	1913.8	0.49	0.40
CMT07-2	2	100	1214.95	1217.20	1843.40	1218.31	1222.82	1939.5	0.28	0.46
CMT07-3	3	100	1153.81	1167.71	1836.40	1159.78	1171.35	1832.8	0.52	0.31
AVG					1880.07			1904.7	0.39	0.42

Table B.2: Comparison of ACO-HCG in automatic configuration (ACO-HCG) with ACO-HCG in automatic configuration without Post-optimization of routes (ACO-HCG-NOPO). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time. $g^{min/avg}$= relative percentage deviation of ACO-HCG-NOPO w.r.t. ACO-HCG , computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with ACO-HCG-NOPO and $z^D_{min/avg}$ the results obtained with ACO-HCG.

Instance	**% Total Execution Time**						**Routes in Ψ**	
	$\%IP$	$\%LP$	$\%BB$	$\%ST$	$\%GEN$	$\%PO$	#Feasible routes	$\|\Psi\|$
CMT01-1	0.4	0	98.4	0.0	1.1	0	4131.2	16108.5
CMT01-2	0.6	0	96.7	0.0	2.5	0	6874.8	35290.9
CMT01-3	3.2	0	89.9	0.1	6.6	0	10221.3	77316.7
CMT02-1	7.1	0	85.5	0.1	7.2	0	7842.4	75237.7
CMT02-2	5.6	0	88.0	0.1	6.2	0	7507.4	55324.9
CMT02-3	2.1	0	90.9	0.1	6.8	0	8768.7	61307.7
CMT03-1	0.3	0	91.3	0.1	8.3	0	6537.8	54999.6
CMT03-2	9.5	0	76.6	0.1	13.6	0	8895.7	90504.0
CMT03-3	1.6	0	87.3	0.1	10.8	0	7890.0	66238.7
CMT04-1	4.6	0	84.2	0.1	11.1	0	5136.6	47030.4
CMT04-2	6.3	0	75.3	0.1	18.2	0	6844.1	69327.8
CMT04-3	4.3	0	78.3	0.1	17.1	0	6794.0	63787.1
CMT05-1	11.3	0	74.3	0.1	14.3	0	4840.7	47215.2
CMT05-2	10.4	0	74.7	0.1	14.7	0	5200.8	40980.9
CMT05-3	8.7	0	68.3	0.1	22.8	0	6088.4	66215.3
CMT06-1	7.3	0	89.9	0.0	2.7	0	5325.7	33638.2
CMT06-2	7.8	0	88.9	0.0	3.2	0	5940.4	37121.9
CMT06-3	1.7	0	91.4	0.1	6.6	0	9054.8	70531.8
CMT07-1	9.9	0	79.9	0.1	10.0	0	9064.4	56164.5
CMT07-2	8.6	0	78.9	0.1	12.1	0	9849.2	71783.6
CMT07-3	0.8	0	94.9	0.0	4.2	0	5126.2	26837.2

Table B.3: ACO-HCG-NOPO : CPU Time Allocation, # feasible routes, # routes in Feasibility Store ($|\Psi|$)

b.1.3 *Comparison of ACO-HCG and DECOMP ACO-HCG*

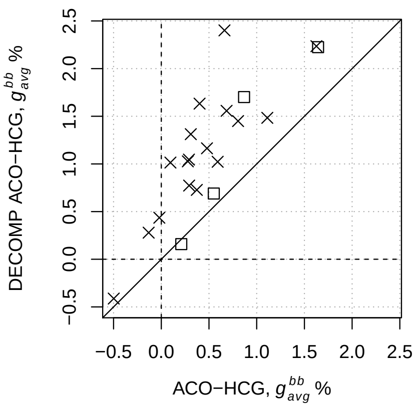
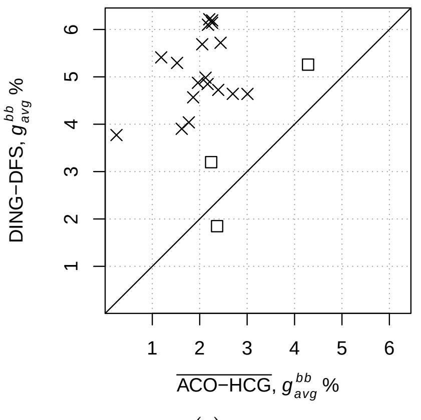


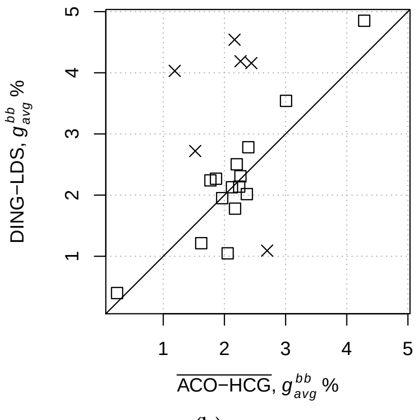
Figure B.1: Comparison of DECOMP ACO-HCG and ACO-HCG. Each point gives the average relative deviation from the *bb-best* solution over 10 independent runs. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

B.2 DIVE & GENERATE

b.2.1 *Comparison with $\overline{\text{ACO-HCG}}$*



(a)



(b)

Figure B.2: Comparison of DING and $\overline{\text{ACO-HCG}}$, ACO-HCG in the manual configuration. Each point gives the average relative deviation from the *bb-best* solution over 10 independent runs. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

B.2.2 *Explicit results for DING*

Instance			Dive & Generate-DFS					
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_h	sec_{tt}	$\%g^{bb}_{avg}$
CMT01-1	1	50	590.08	598.14	0.9	1189.3	1800.6	1.85
CMT01-2	2	50	631.19	643.67	2.3	848.8	1800.3	4.64
CMT01-3	3	50	628.94	647.79	2.2	458.4	1800.3	3.90
CMT02-1	1	75	983.07	1018.05	2.0	664.5	1800.3	3.77
CMT02-2	2	75	919.97	941.38	1.7	563.2	1800.3	4.87
CMT02-3	3	75	903.61	942.57	3.4	347.0	1800.4	6.10
CMT03-1	1	100	1212.75	1255.76	2.7	678.5	1800.4	5.69
CMT03-2	2	100	1251.43	1268.20	1.2	468.7	1800.4	4.04
CMT03-3	3	100	1190.55	1228.74	2.3	554.0	1800.3	6.22
CMT04-1	1	150	1672.16	1705.93	1.1	390.7	1800.6	4.98
CMT04-2	2	150	1586.18	1623.18	1.5	392.3	1801.0	4.57
CMT04-3	3	150	1599.27	1636.48	1.7	586.3	1800.5	6.14
CMT05-1	1	199	2065.69	2100.89	1.0	989.3	1801.2	3.20
CMT05-2	2	199	1900.76	1918.47	0.8	623.9	1803.4	4.64
CMT05-3	3	199	2008.70	2040.91	0.8	732.3	1803.4	4.72
CMT06-1	1	120	2312.36	2368.72	1.6	477.9	1800.4	5.72
CMT06-2	2	120	2130.06	2178.89	1.4	843.2	1804.7	5.26
CMT06-3	3	120	2213.72	2268.24	2.3	645.2	1800.5	5.29
CMT07-1	1	100	1171.28	1191.66	1.5	510.9	1800.6	4.85
CMT07-2	2	100	1253.61	1283.33	1.8	408.8	1800.4	5.41
CMT07-3	3	100	1192.92	1229.35	2.0	459.2	1800.4	6.19
AVG					1.7	611.1	1801.0	4.86

Table B.4: Explicit results on MP-VRP using Dive & Generate with Depth First Search, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_h = Time at which $UB = z_{min}$, sec_{tt} = average total execution time, $\%g^{bb}_{avg}$=average relative percentage deviation w.r.t. *bb-best* solution cost.

Instance			**Dive & Generate-LDS**					
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_h	sec_{tt}	$\%g^{bb}_{avg}$
CMT01-1	1	50	594.54	599.13	0.6	1550.8	1857.3	2.02
CMT01-2	2	50	616.74	621.84	0.8	961.7	1800.5	1.09
CMT01-3	3	50	623.91	631.02	0.6	906.1	1800.1	1.21
CMT02-1	1	75	976.63	984.97	0.4	788.9	1800.1	0.40
CMT02-2	2	75	907.50	915.13	0.5	917.6	1800.7	1.95
CMT02-3	3	75	889.88	904.18	0.9	1145.7	1800.7	1.78
CMT03-1	1	100	1189.45	1200.64	0.9	1118.8	1800.7	1.05
CMT03-2	2	100	1234.40	1246.28	0.8	831.9	1800.2	2.24
CMT03-3	3	100	1174.49	1185.81	0.8	708.4	1800.2	2.50
CMT04-1	1	150	1629.61	1659.57	1.2	1351.5	1815.0	2.13
CMT04-2	2	150	1573.16	1587.47	0.6	1026.5	1800.9	2.27
CMT04-3	3	150	1568.01	1577.40	0.8	1340.5	1800.3	2.31
CMT05-1	1	199	2052.35	2079.31	0.8	1075.7	1809.6	2.14
CMT05-2	2	199	1875.58	1898.33	0.5	1061.8	1814.0	3.54
CMT05-3	3	199	1992.71	2003.07	0.5	1191.2	1801.6	2.78
CMT06-1	1	120	2298.87	2333.72	1.4	1240.3	1825.3	4.16
CMT06-2	2	120	2130.10	2170.45	1.1	723.3	1832.7	4.85
CMT06-3	3	120	2190.97	2212.80	0.4	1040.3	1803.3	2.72
CMT07-1	1	100	1177.31	1188.15	0.8	673.1	1800.1	4.54
CMT07-2	2	100	1236.68	1266.53	1.9	375.8	1801.1	4.03
CMT07-3	3	100	1190.39	1206.16	1.0	804.3	1803.4	4.19
AVG					0.8	992.1	1808.0	2.57

Table B.5: Explicit results on MP-VRP using Dive & Generate with Limited Discrepancy Search, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_h = Time at which $UB = z_{min}$, sec_{tt} = average total execution time, $\%g^{bb}_{avg}$=average relative percentage deviation w.r.t. *bb-best* solution cost.

B.2.3 *Results for DING**

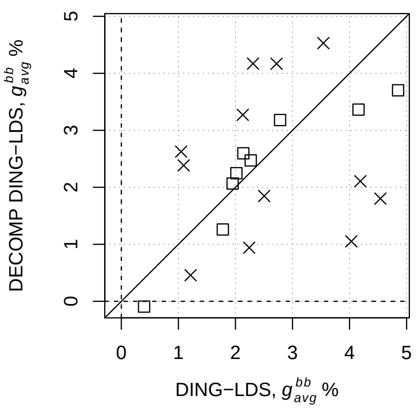
Instance			DING-DFS			DING*-DFS				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g^{min}	g^{avg}
CMT01-1	1	50	590.08	598.14	1800.60	590.08	597.88	0.6	0.00	-0.04
CMT01-2	2	50	631.19	643.67	1800.30	631.19	643.01	0.7	0.00	-0.10
CMT01-3	3	50	628.94	647.79	1800.30	628.94	647.66	0.3	0.00	-0.02
CMT02-1	1	75	983.07	1018.05	1800.30	983.07	1015.87	0.8	0.00	-0.21
CMT02-2	2	75	919.97	941.38	1800.30	914.79	940.12	0.6	-0.56	-0.13
CMT02-3	3	75	903.61	942.57	1800.40	903.61	942.38	0.6	0.00	-0.02
CMT03-1	1	100	1212.75	1255.76	1800.40	1212.75	1253.48	0.5	0.00	-0.18
CMT03-2	2	100	1251.43	1268.20	1800.40	1246.42	1264.51	5.0	-0.40	-0.29
CMT03-3	3	100	1190.55	1228.74	1800.30	1190.55	1227.69	1.8	0.00	-0.09
CMT04-1	1	150	1672.16	1705.93	1800.60	1668.89	1703.70	2.4	-0.20	-0.13
CMT04-2	2	150	1586.18	1623.18	1801.00	1584.85	1621.24	4.3	-0.08	-0.12
CMT04-3	3	150	1599.27	1636.48	1800.50	1599.27	1633.58	0.6	0.00	-0.18
CMT05-1	1	199	2065.69	2100.89	1801.20	2065.69	2097.54	2.7	0.00	-0.16
CMT05-2	2	199	1900.76	1918.47	1803.40	1880.58	1913.34	9.2	-1.06	-0.27
CMT05-3	3	199	2008.70	2040.91	1803.40	2008.70	2037.58	6.8	0.00	-0.16
CMT06-1	1	120	2312.36	2368.72	1800.40	2298.10	2354.57	54.9	-0.62	-0.60
CMT06-2	2	120	2130.06	2178.89	1804.70	2128.31	2172.23	90.6	-0.08	-0.31
CMT06-3	3	120	2213.72	2268.24	1800.50	2204.00	2262.76	10.8	-0.44	-0.24
CMT07-1	1	100	1171.28	1191.66	1800.60	1171.28	1182.94	7.4	0.00	-0.73
CMT07-2	2	100	1253.61	1283.33	1800.40	1241.91	1274.77	2.6	-0.93	-0.67
CMT07-3	3	100	1192.92	1229.35	1800.40	1188.63	1222.13	2.6	-0.36	-0.59
AVG					1800.97			9.8	-0.23	-0.25

Table B.6: Comparison of Dive & Generate with Depth First Search (DING-DFS) and results obtained when solving problem $SCP(\mathcal{R}^*)$ over set of feasible routes $\mathcal{R}^*$ obtained in DING-DFS (DING*-DFS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, for DING*-DFS this corresponds to the time necessary to solve problem $SCP(\mathcal{R}^*)$. $g^{min/avg}$= relative percentage deviation of DING*-DFS w.r.t. DING-DFS , computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with DING*-DFS and $z^D_{min/avg}$ the results obtained with DING-DFS.

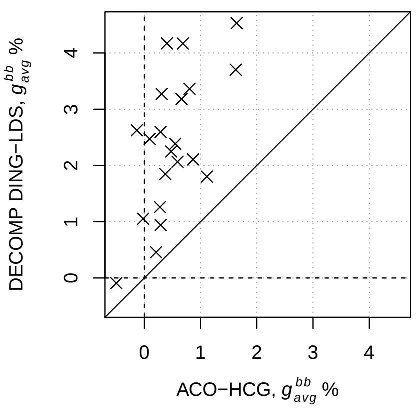
Instance			DING-LDS			DING*-LDS				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
CMT01-1	1	50	594.54	599.13	1857.30	592.69	598.52	1.0	-0.31	-0.10
CMT01-2	2	50	616.74	621.84	1800.50	616.74	621.84	0.9	0.00	0.00
CMT01-3	3	50	623.91	631.02	1800.10	623.91	630.58	2.8	0.00	-0.07
CMT02-1	1	75	976.63	984.97	1800.10	976.63	984.00	7.1	0.00	-0.10
CMT02-2	2	75	907.50	915.13	1800.70	907.50	913.24	10.7	0.00	-0.21
CMT02-3	3	75	889.88	904.18	1800.70	889.88	902.56	6.1	0.00	-0.18
CMT03-1	1	100	1189.45	1200.64	1800.70	1189.45	1197.59	2.0	0.00	-0.25
CMT03-2	2	100	1234.40	1246.28	1800.20	1229.52	1236.93	29.4	-0.40	-0.75
CMT03-3	3	100	1174.49	1185.81	1800.20	1174.49	1181.56	8.9	0.00	-0.36
CMT04-1	1	150	1629.61	1659.57	1815.00	1624.25	1642.24	139.7	-0.33	-1.04
CMT04-2	2	150	1573.16	1587.47	1800.90	1558.12	1569.50	127.9	-0.96	-1.13
CMT04-3	3	150	1568.01	1577.40	1800.30	1554.68	1567.49	85.2	-0.85	-0.63
CMT05-1	1	199	2052.35	2079.31	1809.60	2039.67	2055.77	269.3	-0.62	-1.13
CMT05-2	2	199	1875.58	1898.33	1814.00	1854.16	1867.87	276.7	-1.14	-1.60
CMT05-3	3	199	1992.71	2003.07	1801.60	1971.53	1984.11	224.3	-1.06	-0.95
CMT06-1	1	120	2298.87	2333.72	1825.30	2264.34	2276.72	76.4	-1.50	-2.44
CMT06-2	2	120	2130.10	2170.45	1832.70	2100.80	2127.79	79.9	-1.38	-1.97
CMT06-3	3	120	2190.97	2212.80	1803.30	2183.62	2190.82	25.2	-0.34	-0.99
CMT07-1	1	100	1177.31	1188.15	1800.10	1152.50	1156.68	29.7	-2.11	-2.65
CMT07-2	2	100	1236.68	1266.53	1801.10	1219.31	1228.23	14.0	-1.40	-3.02
CMT07-3	3	100	1190.39	1206.16	1803.40	1156.83	1170.96	12.5	-2.82	-2.92
AVG					1807.99			68.1	-0.72	-1.07

Table B.7: Comparison of Dive & Generate with Limited Discrepancy Search (DING-LDS) and results obtained when solving problem $SCP(\mathcal{R}^*)$ over set of feasible routes $\mathcal{R}^*$ obtained in DING-LDS (DING*-LDS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, for DING*-LDS this corresponds to the time necessary to solve problem $SCP(\mathcal{R}^*)$. $g^{min/avg}$= relative percentage deviation of DING*-LDS w.r.t. DING-LDS , computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with DING*-LDS and $z^D_{min/avg}$ the results obtained with DING-LDS.

b.2.4 *Comparison of DING-LDS and DECOMP DING-LDS*



(a)



(b)

Figure B.3: Comparison of DECOMP DING-LDS with DING-LDS and with ACO-HCG Each point gives the average relative deviation from the *bb-best* solution over 10 independent runs. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

B.3 HEURISTIC BRANCH & PRICE

b.3.1 *Comparison with $\overline{\text{ACO-HCG}}$*

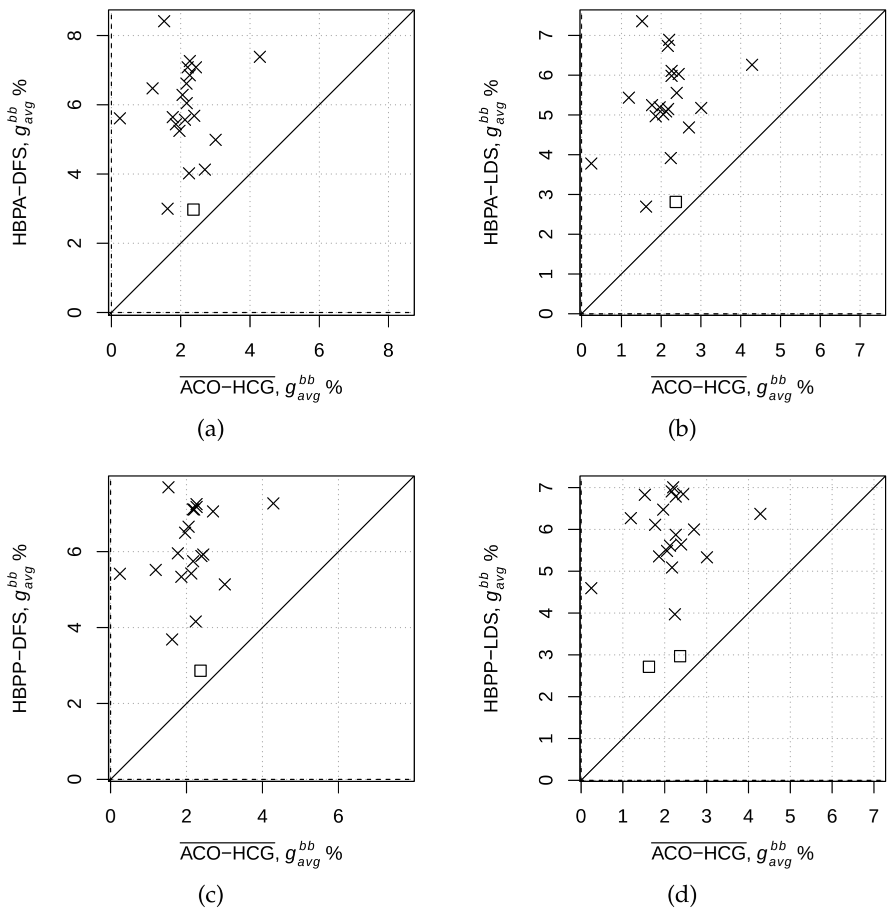


Figure B.4: Comparison of HBP and $\overline{\text{ACO-HCG}}$, the manual configuration of ACO-HCG. Each point gives the average relative deviation from the *bb-best* solution over 10 independent runs. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

b.3.2 *Explicit results for HBP*

Instance			HBPA-DFS					
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_h	sec_{tt}	$\%g^{bb}_{avg}$
CMT01-1	1	50	590.24	604.74	1.2	1395.0	1822.8	2.97
CMT01-2	2	50	630.37	640.50	1.1	1250.7	1800.3	4.13
CMT01-3	3	50	633.24	642.14	1.0	1038.3	1800.2	3.00
CMT02-1	1	75	1008.81	1036.08	1.7	1106.3	1801.4	5.61
CMT02-2	2	75	921.24	944.71	1.9	1263.8	1803.5	5.25
CMT02-3	3	75	916.07	942.10	1.3	1078.2	1802.3	6.05
CMT03-1	1	100	1230.50	1262.85	1.3	1126.8	1802.6	6.28
CMT03-2	2	100	1273.55	1287.75	0.8	1023.0	1802.9	5.64
CMT03-3	3	100	1208.34	1238.78	1.5	851.5	1802.9	7.08
CMT04-1	1	150	1690.84	1715.42	0.9	630.2	1814.7	5.57
CMT04-2	2	150	1610.61	1636.66	1.2	490.8	1813.2	5.44
CMT04-3	3	150	1608.01	1647.48	1.2	832.9	1813.0	6.85
CMT05-1	1	199	2097.42	2117.61	0.9	752.5	1827.1	4.02
CMT05-2	2	199	1876.02	1924.84	1.3	845.6	1828.9	4.99
CMT05-3	3	199	2030.21	2059.56	1.0	587.9	1833.7	5.68
CMT06-1	1	120	2335.93	2399.26	2.0	702.7	1806.3	7.08
CMT06-2	2	120	2159.63	2222.91	1.8	704.2	1826.7	7.38
CMT06-3	3	120	2287.59	2335.43	1.7	1122.2	1805.7	8.41
CMT07-1	1	100	1185.63	1211.69	1.6	772.5	1803.7	6.61
CMT07-2	2	100	1248.98	1296.27	2.5	487.9	1803.5	6.47
CMT07-3	3	100	1223.04	1241.75	1.2	565.1	1802.6	7.26
AVG					1.4	887.0	1810.4	5.80

Table B.8: Explicit results on MP-VRP using Heuristic Branch & Price, branching on arcs, with Depth First Search, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_h = Time of the first restart at which $UB = z_{min}$, sec_{tt} = average total execution time, $\%g^{bb}_{avg}$=average relative percentage deviation w.r.t. *bb-best* solution cost.

Instance			**HBPA-LDS**					
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_h	sec_{tt}	$\%g^{bb}_{avg}$
CMT01-1	1	50	598.24	603.81	0.9	1335.3	1838.7	2.81
CMT01-2	2	50	634.93	643.94	0.8	1042.4	1800.3	4.69
CMT01-3	3	50	631.68	640.25	0.8	1236.3	1800.4	2.69
CMT02-1	1	75	993.97	1018.12	1.1	1333.8	1802.4	3.78
CMT02-2	2	75	933.69	944.22	0.9	1398.2	1803.4	5.19
CMT02-3	3	75	926.99	934.13	0.5	846.6	1801.1	5.15
CMT03-1	1	100	1232.23	1247.77	0.7	1151.1	1803.0	5.01
CMT03-2	2	100	1261.79	1282.92	1.1	338.7	1801.9	5.25
CMT03-3	3	100	1215.68	1236.53	1.0	551.2	1803.2	6.89
CMT04-1	1	150	1682.22	1707.66	0.8	808.1	1808.7	5.09
CMT04-2	2	150	1612.69	1629.38	0.7	477.1	1806.2	4.97
CMT04-3	3	150	1610.17	1634.05	0.9	721.4	1810.2	5.98
CMT05-1	1	199	2085.40	2115.48	1.0	624.6	1835.4	3.92
CMT05-2	2	199	1899.32	1928.29	1.1	772.3	1823.9	5.18
CMT05-3	3	199	2030.56	2057.08	0.9	649.9	1823.9	5.55
CMT06-1	1	120	2337.28	2375.67	1.3	948.5	1805.6	6.03
CMT06-2	2	120	2147.80	2199.64	1.6	887.8	1837.6	6.26
CMT06-3	3	120	2231.28	2312.68	2.7	846.8	1807.1	7.36
CMT07-1	1	100	1186.52	1213.12	1.5	419.7	1802.4	6.74
CMT07-2	2	100	1236.68	1283.61	2.3	414.5	1802.1	5.43
CMT07-3	3	100	1199.89	1228.38	1.5	566.9	1802.2	6.11
AVG					1.1	827.2	1810.5	5.24

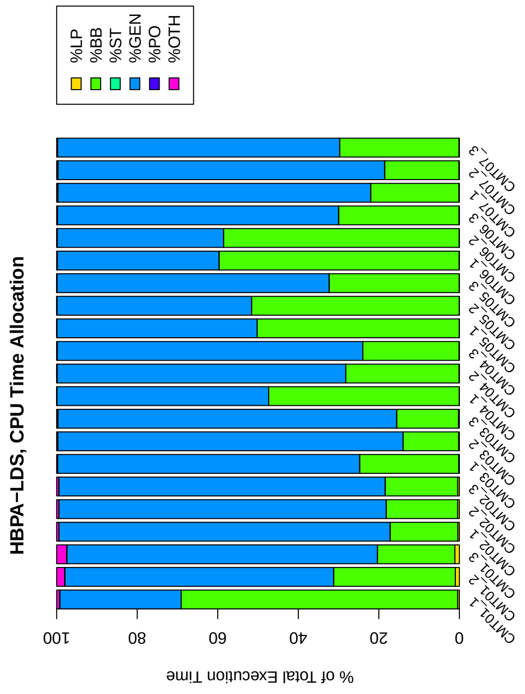
Table B.9: Explicit results on MP-VRP using Heuristic Branch & Price, branching on arcs, with Limited Discrepancy Search, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_h = Time of the first restart at which $UB = z_{min}$, sec_{tt} = average total execution time, $\%g^{bb}_{avg}$=average relative percentage deviation w.r.t. *bb-best* solution cost.

Instance			HBPP-DFS					
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_h	sec_{tt}	$\%g^{bb}_{avg}$
CMT01-1	1	50	594.88	604.10	0.9	1398.0	1802.2	2.86
CMT01-2	2	50	627.26	658.54	2.8	831.4	1804.6	7.06
CMT01-3	3	50	633.03	646.43	1.6	1046.3	1800.9	3.69
CMT02-1	1	75	1000.02	1034.18	1.8	714.9	1805.3	5.42
CMT02-2	2	75	927.61	955.95	1.5	850.3	1803.8	6.50
CMT02-3	3	75	921.57	939.43	1.4	1110.2	1810.6	5.75
CMT03-1	1	100	1231.14	1267.26	1.2	755.9	1806.7	6.66
CMT03-2	2	100	1268.38	1291.56	1.1	588.6	1810.9	5.96
CMT03-3	3	100	1202.59	1239.06	1.7	692.7	1806.3	7.11
CMT04-1	1	150	1689.00	1713.11	1.0	448.7	1819.4	5.42
CMT04-2	2	150	1612.69	1635.11	1.0	446.3	1816.0	5.34
CMT04-3	3	150	1633.95	1653.62	0.8	254.5	1826.1	7.25
CMT05-1	1	199	2091.13	2120.45	0.9	377.9	1844.3	4.16
CMT05-2	2	199	1876.02	1927.59	1.5	682.8	1834.4	5.14
CMT05-3	3	199	2038.80	2063.51	0.9	369.4	1849.7	5.88
CMT06-1	1	120	2328.45	2373.29	1.4	896.6	1819.3	5.92
CMT06-2	2	120	2145.30	2220.54	1.8	451.0	1827.1	7.27
CMT06-3	3	120	2216.30	2319.96	2.9	779.1	1816.1	7.70
CMT07-1	1	100	1185.63	1217.40	1.9	496.1	1810.1	7.11
CMT07-2	2	100	1239.03	1284.62	2.3	501.4	1806.8	5.52
CMT07-3	3	100	1196.21	1240.75	2.1	541.7	1809.6	7.18
AVG					1.5	677.8	1815.7	5.95

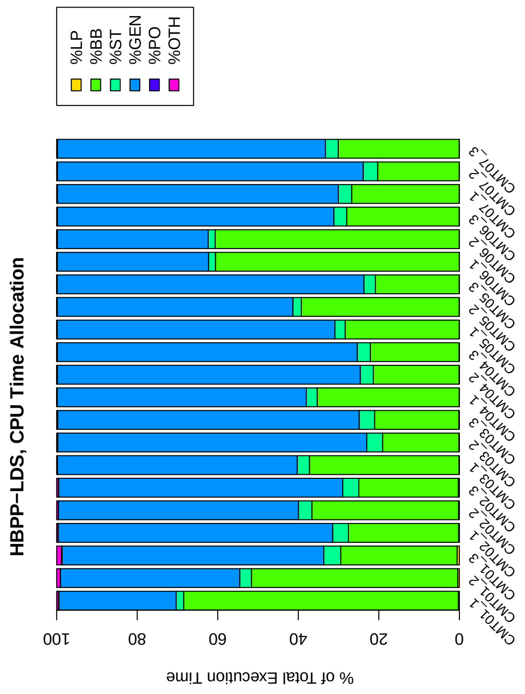
Table B.10: Explicit results on MP-VRP using Heuristic Branch & Price, branching on pairs, with Depth First Search, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_h = Time of the first restart at which $UB = z_{min}$, sec_{tt} = average total execution time, $\%g^{bb}_{avg}$=average relative percentage deviation w.r.t. *bb-best* solution cost.

Instance			HBPP-LDS					
	K	n	z_{min}	z_{avg}	$\%RSD$	sec_h	sec_{tt}	$\%g^{bb}_{avg}$
CMT01-1	1	50	598.14	604.72	0.5	1240.4	1809.7	2.97
CMT01-2	2	50	630.13	652.02	2.7	968.3	1800.7	6.00
CMT01-3	3	50	631.68	640.38	1.2	1042.1	1800.6	2.71
CMT02-1	1	75	991.88	1026.13	2.4	811.4	1804.2	4.59
CMT02-2	2	75	943.53	955.71	1.2	985.9	1802.7	6.47
CMT02-3	3	75	914.48	933.59	1.3	706.7	1804.1	5.09
CMT03-1	1	100	1210.47	1253.36	1.9	800.2	1811.8	5.49
CMT03-2	2	100	1268.12	1293.43	1.3	41.5	1807.2	6.11
CMT03-3	3	100	1206.35	1237.90	1.5	535.7	1808.5	7.01
CMT04-1	1	150	1690.84	1716.13	1.0	194.2	1822.3	5.61
CMT04-2	2	150	1612.69	1635.40	1.0	219.2	1818.4	5.36
CMT04-3	3	150	1617.00	1646.42	1.1	329.0	1822.1	6.79
CMT05-1	1	199	2098.10	2116.59	0.8	496.3	1843.2	3.97
CMT05-2	2	199	1898.15	1931.19	1.3	261.0	1843.7	5.33
CMT05-3	3	199	2030.21	2058.76	1.0	235.8	1829.4	5.64
CMT06-1	1	120	2318.86	2393.91	2.7	616.9	1818.5	6.84
CMT06-2	2	120	2147.80	2201.92	1.7	327.4	1814.3	6.37
CMT06-3	3	120	2234.47	2301.26	2.8	682.8	1814.7	6.83
CMT07-1	1	100	1175.08	1215.07	2.2	394.9	1808.2	6.91
CMT07-2	2	100	1248.98	1293.77	2.2	302.3	1807.8	6.27
CMT07-3	3	100	1199.47	1225.63	1.3	712.0	1807.9	5.87
AVG					1.6	566.9	1814.3	5.63

Table B.11: Explicit results on MP-VRP using Heuristic Branch & Price, branching on pairs, with Limited Discrepancy Search, $z_{min/avg}$ = best/average solution cost over 10 runs, $\%RSD$ = relative percentage standard deviation over 10 runs, sec_h = Time of the first restart at which $UB = z_{min}$, sec_{tt} = average total execution time, $\%g^{bb}_{avg}$=average relative percentage deviation w.r.t. *bb-best* solution cost.



(a)



(b)

Figure B.5: CPU Time Allocation for HBPA-LDS and HBPP-LDS

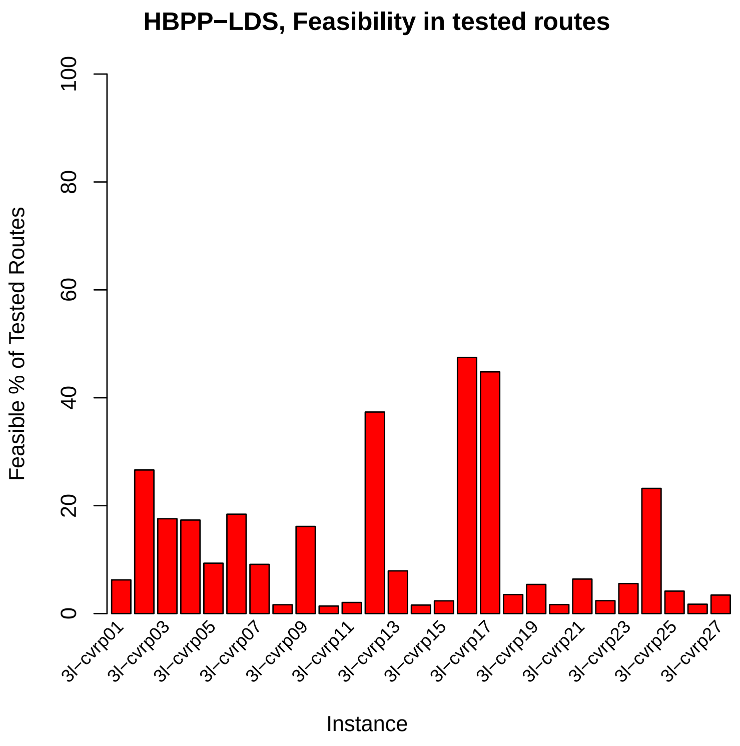


Figure B.6: HBPP-LDS, Percentage of feasible routes in tested routes

Instance	% Total Execution Time					Routes in Ψ	
	$\%LP$	$\%BB$	$\%ST$	$\%GEN$	$\%PO$	#Feasible routes	$\lvert\Psi\rvert$
CMT01-1	0.4	68.6	0	30.1	0.1	2398.6	8940.9
CMT01-2	1.0	30.2	0	66.7	0.1	2590.0	8763.8
CMT01-3	1.1	19.2	0	77.1	0.1	2898.6	12314.2
CMT02-1	0.4	16.8	0	82.2	0.0	2456.6	7829.5
CMT02-2	0.4	17.8	0	81.2	0.0	2513.4	8110.3
CMT02-3	0.4	18.0	0	81.0	0.0	2421.2	9031.0
CMT03-1	0.2	24.6	0	75.0	0.0	2352.6	8825.2
CMT03-2	0.2	13.8	0	85.7	0.0	2227.8	8744.4
CMT03-3	0.2	15.4	0	84.1	0.0	2491.5	9039.4
CMT04-1	0.1	47.3	0	52.6	0.0	2058.8	9597.7
CMT04-2	0.1	28.1	0	71.7	0.0	2395.7	11405.9
CMT04-3	0.1	23.9	0	75.9	0.0	2503.6	11828.0
CMT05-1	0.0	50.2	0	49.7	0.0	2431.7	12700.7
CMT05-2	0.1	51.5	0	48.3	0.1	2583.7	13152.6
CMT05-3	0.1	32.3	0	67.5	0.1	2538.3	14662.5
CMT06-1	0.1	59.6	0	40.2	0.1	2646.5	11534.1
CMT06-2	0.1	58.5	0	41.3	0.1	2506.0	11160.1
CMT06-3	0.1	29.9	0	69.8	0.1	2850.9	14385.6
CMT07-1	0.1	21.9	0	77.7	0.0	2825.8	8360.8
CMT07-2	0.1	18.4	0	81.1	0.0	2637.8	8617.9
CMT07-3	0.1	29.6	0	70.1	0.0	2552.5	8167.9

Table B.12: HBPA-LDS : CPU Time Allocation and Ratio of Feasible routes / Routes in Feasibility Store ($\lvert\Psi\rvert$)

Instance	**% Total Execution Time**					**Routes in Ψ**	
	%*LP*	%*BB*	%*ST*	%*GEN*	%*PO*	#Feasible routes	$\|\Psi\|$
CMT01-1	0.3	68.2	1.9	29.1	0.1	2908.4	10534.5
CMT01-2	0.4	51.2	2.9	44.5	0.1	3585.0	29536.2
CMT01-3	0.6	28.9	4.2	65.0	0.1	3540.0	22950.9
CMT02-1	0.2	27.4	3.9	68.2	0.1	2674.0	29900.0
CMT02-2	0.2	36.4	3.4	59.5	0.1	3700.0	22408.3
CMT02-3	0.3	24.7	4.0	70.5	0.1	3765.8	17903.6
CMT03-1	0.1	37.1	3.0	59.5	0.1	2955.9	13875.6
CMT03-2	0.1	19.0	3.9	76.7	0.1	3126.9	20497.9
CMT03-3	0.1	20.9	3.8	74.9	0.1	3153.2	18006.0
CMT04-1	0.1	35.3	2.7	61.9	0.1	2656.1	17845.2
CMT04-2	0.1	21.3	3.3	75.2	0.1	3138.1	18515.1
CMT04-3	0.1	22.0	3.3	74.5	0.1	3137.0	17542.8
CMT05-1	0.1	28.3	2.5	69.0	0.1	3082.0	24357.9
CMT05-2	0.1	39.2	2.1	58.5	0.1	2981.4	20973.5
CMT05-3	0.1	20.8	2.8	76.2	0.1	2840.5	21426.9
CMT06-1	0.0	60.6	1.7	37.6	0.1	2717.3	35039.2
CMT06-2	0.0	60.6	1.7	37.5	0.1	3186.1	24643.8
CMT06-3	0.1	27.9	3.2	68.7	0.1	3246.8	31300.2
CMT07-1	0.0	26.7	3.3	69.8	0.1	2293.1	27081.1
CMT07-2	0.0	20.2	3.6	75.9	0.1	2199.0	29395.9
CMT07-3	0.0	30.0	3.2	66.5	0.1	2329.3	24009.3

Table B.13: HBPP-LDS : CPU Time Allocation and Ratio of Feasible routes / Routes in Feasibility Store ($|\Psi|$)

b.3.3 *Results for HBP**

Instance			HBPA-DFS			HBPA*-DFS				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
CMT01-1	1	50	590.24	604.74	1822.8	590.24	604.74	0.5	0.00	0.00
CMT01-2	2	50	630.37	640.50	1800.3	630.37	639.28	1.0	0.00	-0.19
CMT01-3	3	50	633.24	642.14	1800.2	633.24	641.36	1.2	0.00	-0.12
CMT02-1	1	75	1008.81	1036.08	1801.4	992.66	1027.44	0.9	-1.60	-0.83
CMT02-2	2	75	921.24	944.71	1803.5	921.24	943.06	1.0	0.00	-0.17
CMT02-3	3	75	916.07	942.10	1802.3	916.07	938.99	1.6	0.00	-0.33
CMT03-1	1	100	1230.50	1262.85	1802.6	1225.60	1259.31	0.9	-0.40	-0.28
CMT03-2	2	100	1273.55	1287.75	1802.9	1273.55	1285.40	1.6	0.00	-0.18
CMT03-3	3	100	1208.34	1238.78	1802.9	1208.34	1232.12	1.2	0.00	-0.54
CMT04-1	1	150	1690.84	1715.42	1814.7	1690.84	1709.88	0.6	0.00	-0.32
CMT04-2	2	150	1610.61	1636.66	1813.2	1607.01	1630.84	1.1	-0.22	-0.36
CMT04-3	3	150	1608.01	1647.48	1813.0	1608.01	1639.60	0.9	0.00	-0.48
CMT05-1	1	199	2097.42	2117.61	1827.1	2071.30	2109.95	1.0	-1.25	-0.36
CMT05-2	2	199	1876.02	1924.84	1828.9	1875.50	1916.43	0.7	-0.03	-0.44
CMT05-3	3	199	2030.21	2059.56	1833.7	2030.21	2053.39	1.7	0.00	-0.30
CMT06-1	1	120	2335.93	2399.26	1806.3	2322.91	2341.24	2.6	-0.56	-2.42
CMT06-2	2	120	2159.63	2222.91	1826.7	2152.36	2198.62	1.6	-0.34	-1.09
CMT06-3	3	120	2287.59	2335.43	1805.7	2264.09	2318.10	1.3	-1.03	-0.74
CMT07-1	1	100	1185.63	1211.69	1803.7	1182.93	1202.36	1.1	-0.23	-0.77
CMT07-2	2	100	1248.98	1296.27	1803.5	1248.98	1282.31	1.6	0.00	-1.08
CMT07-3	3	100	1223.04	1241.75	1802.6	1208.23	1228.20	0.8	-1.21	-1.09
AVG					1810.4			1.2	-0.33	-0.58

Table B.14: Comparison of Heuristic Branch & Price, branching on arcs, with Depth First Search (HBPA-DFS) and results obtained when solving problem $SCP(\mathcal{R}^*)$ over set of feasible routes $\mathcal{R}^*$ obtained in HBPA-DFS (HBPA*-DFS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, for HBPA*-DFS this corresponds to the time necessary to solve problem $SCP(\mathcal{R}^*)$. $g^{min/avg}$= relative percentage deviation of HBPA*-DFS w.r.t. HBPA-DFS, computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with HBPA*-DFS and $z^D_{min/avg}$ the results obtained with HBPA-DFS.

Instance			**HBPA-LDS**			**HBPA*-LDS**				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
CMT01-1	1	50	598.24	603.81	1838.7	598.24	603.81	0.7	0.00	0.00
CMT01-2	2	50	634.93	643.94	1800.3	624.72	641.77	1.3	-1.61	-0.34
CMT01-3	3	50	631.68	640.25	1800.4	631.68	640.25	1.1	0.00	0.00
CMT02-1	1	75	993.97	1018.12	1802.4	993.97	1011.24	3.3	0.00	-0.68
CMT02-2	2	75	933.69	944.22	1803.4	925.13	939.55	1.3	-0.92	-0.49
CMT02-3	3	75	926.99	934.13	1801.1	926.05	932.14	1.4	-0.10	-0.21
CMT03-1	1	100	1232.23	1247.77	1803.0	1229.64	1242.39	1.5	-0.21	-0.43
CMT03-2	2	100	1261.79	1282.92	1801.9	1260.28	1276.73	3.3	-0.12	-0.48
CMT03-3	3	100	1215.68	1236.53	1803.2	1213.92	1234.23	2.7	-0.14	-0.19
CMT04-1	1	150	1682.22	1707.66	1808.7	1681.34	1694.74	2.0	-0.05	-0.76
CMT04-2	2	150	1612.69	1629.38	1806.2	1607.01	1623.21	2.7	-0.35	-0.38
CMT04-3	3	150	1610.17	1634.05	1810.2	1610.17	1627.74	2.2	0.00	-0.39
CMT05-1	1	199	2085.40	2115.48	1835.4	2081.75	2103.98	1.9	-0.18	-0.54
CMT05-2	2	199	1899.32	1928.29	1823.9	1898.23	1917.92	2.6	-0.06	-0.54
CMT05-3	3	199	2030.56	2057.08	1823.9	2030.21	2050.66	4.4	-0.02	-0.31
CMT06-1	1	120	2337.28	2375.67	1805.6	2292.75	2329.22	9.3	-1.91	-1.96
CMT06-2	2	120	2147.80	2199.64	1837.6	2138.72	2172.53	3.4	-0.42	-1.23
CMT06-3	3	120	2231.28	2312.68	1807.1	2227.06	2289.97	8.2	-0.19	-0.98
CMT07-1	1	100	1186.52	1213.12	1802.4	1174.59	1191.84	3.7	-1.01	-1.75
CMT07-2	2	100	1236.68	1283.61	1802.1	1234.86	1262.32	4.9	-0.15	-1.66
CMT07-3	3	100	1199.89	1228.38	1802.2	1189.52	1210.14	1.7	-0.86	-1.48
AVG					1810.5			3.0	-0.39	-0.70

Table B.15: Comparison of Heuristic Branch & Price, branching on arcs, with Limited Discrepancy Search (HBPA-LDS) and results obtained when solving problem $SCP(\mathcal{R}^*)$ over set of feasible routes $\mathcal{R}^*$ obtained in HBPA-LDS (HBPA*-LDS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, for HBPA*-LDS this corresponds to the time necessary to solve problem $SCP(\mathcal{R}^*)$. $g^{min/avg}$= relative percentage deviation of HBPA*-LDS w.r.t. HBPA-LDS, computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with HBPA*-LDS and $z^D_{min/avg}$ the results with HBPA-LDS.

Instance			HBPP-DFS			HBPP*-DFS				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
CMT01-1	1	50	594.88	604.10	1802.2	594.88	602.15	0.6	0.00	-0.32
CMT01-2	2	50	627.26	658.54	1804.6	625.82	651.93	0.9	-0.23	-1.00
CMT01-3	3	50	633.03	646.43	1800.9	633.03	645.77	1.3	0.00	-0.10
CMT02-1	1	75	1000.02	1034.18	1805.3	1000.02	1028.74	1.1	0.00	-0.53
CMT02-2	2	75	927.61	955.95	1803.8	924.43	950.94	1.7	-0.34	-0.52
CMT02-3	3	75	921.57	939.43	1810.6	921.57	936.61	1.1	0.00	-0.30
CMT03-1	1	100	1231.14	1267.26	1806.7	1231.13	1258.93	1.2	0.00	-0.66
CMT03-2	2	100	1268.38	1291.56	1810.9	1266.68	1284.60	1.4	-0.13	-0.54
CMT03-3	3	100	1202.59	1239.06	1806.3	1202.59	1232.89	1.0	0.00	-0.50
CMT04-1	1	150	1689.00	1713.11	1819.4	1687.26	1705.37	1.0	-0.10	-0.45
CMT04-2	2	150	1612.69	1635.11	1816.0	1607.01	1629.02	1.4	-0.35	-0.37
CMT04-3	3	150	1633.95	1653.62	1826.1	1631.14	1642.08	1.5	-0.17	-0.70
CMT05-1	1	199	2091.13	2120.45	1844.3	2083.93	2112.91	1.3	-0.34	-0.36
CMT05-2	2	199	1876.02	1927.59	1834.4	1875.50	1917.54	1.8	-0.03	-0.52
CMT05-3	3	199	2038.80	2063.51	1849.7	2030.21	2055.26	0.9	-0.42	-0.40
CMT06-1	1	120	2328.45	2373.29	1819.3	2310.02	2335.63	3.4	-0.79	-1.59
CMT06-2	2	120	2145.30	2220.54	1827.1	2143.60	2195.97	4.4	-0.08	-1.11
CMT06-3	3	120	2216.30	2319.96	1816.1	2216.30	2291.16	5.3	0.00	-1.24
CMT07-1	1	100	1185.63	1217.40	1810.1	1180.12	1198.42	0.7	-0.46	-1.56
CMT07-2	2	100	1239.03	1284.62	1806.8	1236.36	1274.14	1.5	-0.22	-0.82
CMT07-3	3	100	1196.21	1240.75	1809.6	1196.11	1227.72	0.7	-0.01	-1.05
AVG					1815.7			1.7	-0.18	-0.70

Table B.16: Comparison of Heuristic Branch & Price, branching on customer pairs, with Depth First Search (HBPP-DFS) and results obtained when solving problem $SCP(\mathcal{R}^*)$ over set of feasible routes $\mathcal{R}^*$ obtained in HBPP-DFS (HBPP*-DFS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, for HBPP*-DFS this corresponds to the time necessary to solve problem $SCP(\mathcal{R}^*)$. $g^{min/avg}$= relative percentage deviation of HBPP*-DFS w.r.t. HBPP-DFS, computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with HBPP*-DFS and $z^D_{min/avg}$ the results obtained with HBPP-DFS.

Instance			**HBPP-LDS**			**HBPP*-LDS**				
	K	n	z_{min}	z_{avg}	sec_{tt}	z_{min}	z_{avg}	sec_{tt}	g_{min}	g_{avg}
CMT01-1	1	50	598.14	604.72	1809.7	598.14	604.47	0.7	0.00	-0.04
CMT01-2	2	50	630.13	652.02	1800.7	630.13	644.99	1.4	0.00	-1.08
CMT01-3	3	50	631.68	640.38	1800.6	631.68	640.33	1.6	0.00	-0.01
CMT02-1	1	75	991.88	1026.13	1804.2	986.73	1013.52	1.9	-0.52	-1.23
CMT02-2	2	75	943.53	955.71	1802.7	932.45	944.66	2.6	-1.17	-1.16
CMT02-3	3	75	914.48	933.59	1804.1	914.48	928.22	1.8	0.00	-0.58
CMT03-1	1	100	1210.47	1253.36	1811.8	1210.47	1241.12	1.5	0.00	-0.98
CMT03-2	2	100	1268.12	1293.43	1807.2	1258.36	1277.22	3.4	-0.77	-1.25
CMT03-3	3	100	1206.35	1237.90	1808.5	1196.75	1220.56	2.1	-0.80	-1.40
CMT04-1	1	150	1690.84	1716.13	1822.3	1682.98	1700.29	4.5	-0.46	-0.92
CMT04-2	2	150	1612.69	1635.40	1818.4	1594.00	1616.81	2.9	-1.16	-1.14
CMT04-3	3	150	1617.00	1646.42	1822.1	1607.24	1630.78	5.5	-0.60	-0.95
CMT05-1	1	199	2098.10	2116.59	1843.2	2083.93	2104.01	4.1	-0.68	-0.59
CMT05-2	2	199	1898.15	1931.19	1843.7	1892.48	1918.40	5.2	-0.30	-0.66
CMT05-3	3	199	2030.21	2058.76	1829.4	2023.63	2046.94	5.4	-0.32	-0.57
CMT06-1	1	120	2318.86	2393.91	1818.5	2305.92	2329.25	4.7	-0.56	-2.70
CMT06-2	2	120	2147.80	2201.92	1814.3	2147.30	2168.29	5.8	-0.02	-1.53
CMT06-3	3	120	2234.47	2301.26	1814.7	2223.18	2271.34	3.3	-0.51	-1.30
CMT07-1	1	100	1175.08	1215.07	1808.2	1174.27	1187.85	1.4	-0.07	-2.24
CMT07-2	2	100	1248.98	1293.77	1807.8	1237.95	1264.92	2.0	-0.88	-2.23
CMT07-3	3	100	1199.47	1225.63	1807.9	1195.24	1210.52	2.1	-0.35	-1.23
AVG					1814.3			3.0	-0.44	-1.13

Table B.17: Comparison of Heuristic Branch & Price, branching on arcs, with Limited Discrepancy Search (HBPP-LDS) and results obtained when solving problem $SCP(\mathcal{R}^*)$ over set of feasible routes $\mathcal{R}^*$ obtained in HBPP-LDS (HBPP*-LDS). $z_{min/avg}$ = best/average solution cost over 10 runs, sec_{tt} = average total execution time, for HBPP*-LDS this corresponds to the time necessary to solve problem $SCP(\mathcal{R}^*)$. $g^{min/avg}$= relative percentage deviation of HBPP*-LDS w.r.t. HBPP-LDS, computed as $100 \cdot \frac{z_{min/avg} - z^D_{min/avg}}{z^D_{min/avg}}$ where $z_{min/avg}$ are the results obtained with HBPP*-LDS and $z^D_{min/avg}$ the results obtained with HBPP-LDS.

BIBLIOGRAPHY

[ABM98] Gerald Y. Agbegha, Ronald H. Ballou, and Kamlesh Mathur, *Optimizing auto-carrier loading*, Transportation science **32** (1998), no. 2, 174–188.

[Ach05] T Achterberg, *Branching rules revisited*, Operations Research Letters **33** (2005), 42–54.

[ASH06] C. Archetti, M. G. Speranza, and A. Hertz, *A tabu search algorithm for the split delivery vehicle routing problem*, Transportation Science **40** (2006), no. 1, 64–73.

[ASV13] C. Archetti, M. G. Speranza, and D. Vigo, *Vehicle routing problems with profits*, Tech. Report WPDEM2013/3, University of Brescia, 2013.

[Ba09] José Brandão, *A deterministic tabu search algorithm for the fleet size and mix vehicle routing problem*, European Journal of Operational Research **195** (2009), no. 3, 716–728.

[BBMR10] Roberto Baldacci, Enrico Bartolini, Aristide Mingozzi, and Roberto Roberti, *An exact solution framework for a broad class of vehicle routing problems*, Computational Management Science **7** (2010), 229–268.

[BJN$^+$94] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin WP Savelsbergh, and Pamela H. Vance, *Branch-and-price: Column generation for solving huge integer programs*, Mathematical Programming: State of the Art (1994), 186–207.

[BLR$^+$12] R. Borndörfer, A. Löbel, M. Reuther, T. Schlechte, and S. Weider, *Rapid branching*, Tech. Report 12-10, ZIB - Konrad-Zuse-Zentrum für Informationstechnik, Berlin, 2012.

[Bon08] Boris Bontoux, *Techniques hybrides de recherche exacte et approchée : application à des problèmes de transport*, Ph.D. thesis, Université d'Avignon et des Pays de Vaucluse, December 2008.

[Bor12] Andreas Bortfeldt, *A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints*, Computers & Operations Research **39** (2012), no. 9, 2248–2257.

[BVH04] Russell Bent and Pascal Van Hentenryck, *A two-stage hybrid local search for the vehicle routing problem with time windows*, Transportation Science **38** (2004), no. 4, 515–530 (en).

[BVH07] R. Bent and P. Van Hentenryck, *Randomized adaptive spatial decoupling for large-scale vehicle routing with time windows*, Proceedings of the national conference on artificial intelligence, vol. 22, 2007, p. 173.

[BYBS10] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle, *F-race and iterated f-race: An overview*, Experimental methods for the analysis of optimization algorithms, Springer, 2010, p. 311–336.

[CGL97] Jean-François Cordeau, Michel Gendreau, and Gilbert Laporte, *A tabu search heuristic for periodic and multi-depot vehicle routing problems*, Networks **30** (1997), no. 2, 105–119 (en).

[Cha06] Alain Chabrier, *Vehicle routing problem with elementary shortest path based column generation*, Computers & Operations Research **33** (2006), no. 10, 2972–2990.

[CLM01] J-F Cordeau, G. Laporte, and A. Mercier, *A unified tabu search heuristic for vehicle routing problems with time windows*, The Journal of the Operational Research Society **52** (2001), no. 8, 928–936.

[Coo11] William J Cook, *In pursuit of the traveling salesman: Mathematics at the limits of computation*, Princeton University Press, 2011.

[CRS09] Alberto Ceselli, Giovanni Righini, and Matteo Salani, *A column generation algorithm for a rich vehicle-routing problem*, Transportation Science **43** (2009), no. 1, 56–69.

[CVH08] Ann Melissa Campbell, Dieter Vandenbussche, and William Hermann, *Routing for relief efforts*, Transportation Science **42** (2008), no. 2, 127–145.

[CW64] G. Clarke and J. W. Wright, *Scheduling of vehicles from a central depot to a number of delivery points*, Operations Research **12** (1964), no. 4, 568–581 (en).

[Dan98] George Dantzig, *Linear programming and extensions*, Princeton university press, 1998.

[DBS06] M. Dorigo, M. Birattari, and T. Stützle, *Ant colony optimization*, Computational Intelligence Magazine, IEEE **1** (2006), no. 4, 28–39.

[DD08] Ulrich Derigs and Thomas Döhmer, *Indirect search for the vehicle routing problem with pickup and delivery and time windows*, OR Spectrum **30** (2008), no. 1, 149–165.

[DDD05] Grégoire Dooms, Yves Deville, and Pierre Dupont, *Cp (graph): Introducing a graph computation domain in constraint programming*, Principles and Practice of Constraint Programming-CP 2005 (2005), 211–225.

[DFH$^+$07] K.F. Doerner, G. Fuellerer, R.F. Hartl, M. Gronalt, and M. Iori, *Metaheuristics for the vehicle routing problem with loading constraints*, Networks **49** (2007), no. 4, 294–307.

[Dom09] Michael Dom, *Algorithmic aspects of the consecutive-ones property*, Bulletin of the European Association for Theoretical Computer Science (2009), 27–59.

[DR59] G.b. Dantzig and J.h. Ramser, *The truck dispatching problem*, Management Science **6** (1959), no. 1, 80–91.

[FDHI10] G. Fuellerer, K. F Doerner, R. F Hartl, and M. Iori, *Metaheuristics for vehicle routing problems with three-dimensional loading constraints*, European Journal of Operational Research **201** (2010), no. 3, 751–759.

[Fei10] D. Feillet, *A tutorial on column generation and branch-and-price for vehicle routing problems*, 4OR: A Quarterly Journal of Operations Research **8** (2010), no. 4, 407–424.

[GBLJ03] M. Grötschel, R. Borndörfer, A. Löbel, and W. Jäger, *Duty scheduling in public transit*, Mathematics—Key Technologies for the Future, Springer, Berlin (2003), 653–674.

[GGW10] Chris Groër, Bruce Golden, and Edward Wasil, *A library of local search heuristics for the vehicle routing problem*, Mathematical Programming Computation **2** (2010), no. 2, 79–101.

[GHL94] Michel Gendreau, Alain Hertz, and Gilbert Laporte, *A tabu search heuristic for the vehicle routing problem*, Management Science **40** (1994), no. 10, 1276–1290.

[GILM06] M. Gendreau, M. Iori, G. Laporte, and S. Martello, *A tabu search algorithm for a routing and container loading problem*, Transportation Science **40** (2006), no. 3, 342–350.

[GILM08] Michel Gendreau, Manuel Iori, Gilbert Laporte, and Silvaro Martello, *A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints*, Networks **51** (2008), no. 1, 4–18.

[GKL$^+$06] O. Günlük, T. Kimbrel, L. Ladanyi, B. Schieber, and G.B. Sorkin, *Vehicle routing and staffing for sedan service*, Transportation science **40** (2006), no. 3, 313–326.

[GL98] Fred Glover and Manuel Laguna, *Tabu search*, vol. 1, Springer, 1998.

[GLT97] Bruce L. Golden, Gilbert Laporte, and Éric D. Taillard, *An adaptive memory heuristic for a class of vehicle routing problems with minmax objective*, Computers & Operations Research **24** (1997), no. 5, 445–452.

[Goe09] A. Goel, *Vehicle scheduling and routing with drivers' working hours*, Transportation Science **43** (2009), 17–26.

[GP10] Dr Leo J. Grady and Dr Jonathan R. Polimeni, *Appendix discrete calculus*, Discrete Calculus, Springer London, January 2010, pp. 199–242 (en).

[GRW08] Bruce L Golden, Subramanian Raghavan, and Edward A Wasil, *The vehicle routing problem: latest advances and new challenges*, vol. 43, Springer, 2008.

[GTA99] Luca Maria Gambardella, Éric Taillard, and Giovanni Agazzi, *MACS-VRPTW: a multiple colony system for vehicle routing problems with time windows*, New Ideas in Optimization, McGraw-Hill, 1999, pp. 63–76.

[Hoo11] John N Hooker, *Integrated methods for optimization*, vol. 170, Springer, 2011.

[HS04] Holger H. Hoos and Thomas Stützle, *Stochastic local search: Foundations & applications*, Morgan Kaufmann, 2004.

[ID05] Stefan Irnich and Guy Desaulniers, *Column generation*, ch. Shortest Path Problems With Resource Constraints, Springer, 2005.

[IM10] Manuel Iori and Silvano Martello, *Routing problems with loading constraints*, TOP **18** (2010), no. 1, 4–27.

[Irn08] Stefan Irnich, *A unified modeling and solution framework for vehicle routing and local search-based metaheuristics*, INFORMS Journal on Computing **20** (2008), no. 2, 270–287.

[ISGV07] Manuel Iori, Juan-José Salazar-González, and Daniele Vigo, *An exact approach for the vehicle routing problem with two-dimensional loading constraints*, Transportation Science **41** (2007), no. 2, 253–264.

[JMS$^+$10] C. Joncour, S. Michel, R. Sadykov, D. Sverdlov, and F. Vanderbeck, *Column generation based primal heuristics*, Electronic Notes in Discrete Mathematics **36** (2010), 695–702.

[KNBG07] Jari Kytöjoki, Teemu Nuortio, Olli Bräysy, and Michel Gendreau, *An efficient variable neighborhood search heuristic for very large scale vehicle routing problems*, Computers & Operations Research **34** (2007), no. 9, 2743–2757.

[Lau78] Jena-Lonis Lauriere, *A language and a program for stating and solving combinatorial problems*, Artificial intelligence **10** (1978), no. 1, 29–127.

[LIDLSB11] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari, *The irace package, iterated race for automatic algorithm configuration*, Tech. Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

[MDVH12] F. Massen, Y. Deville, and P. Van Hentenryck, *Pheromone-based heuristic column generation for vehicle routing problems with black box feasibility*, Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems (2012), 260–274.

[MH97] Nenad Mladenović and Pierre Hansen, *Variable neighborhood search*, Computers & Operations Research **24** (1997), no. 11, 1097–1100.

[MJ76] R. H. Mole and S. R. Jameson, *A sequential route-building algorithm employing a generalised savings criterion*, Operational Research Quarterly (1970-1977) **27** (1976), no. 2, 503–511.

[MKKS11] C.M. Meyer, H. Kopfer, A.L. Kok, and M. Schutten, *Distributed decision making in combined vehicle routing and break scheduling*, Dynamics in Logistics (2011), 125–133.

[MLISD13] Florence Massen, Manuel López-Ibáñez, Thomas Stützle, and Yves Deville, *Experimental analysis of pheromone-based heuristic column generation using irace*, Hybrid Metaheuristics (2013), 92–106.

[MO11] Ball Michael O., *Heuristics based on mathematical programming*, Surveys in Operations Research and Management Science **16** (2011), no. 1, 21–38.

[PDH08] Sophie N. Parragh, Karl F. Doerner, and Richard F. Hartl, *A survey on pickup and delivery problems*, Journal für Betriebswirtschaft **58** (2008), no. 1, 21–51.

[PDH10] ______, *Variable neighborhood search for the dial-a-ride problem*, Computers & Operations Research **37** (2010), no. 6, 1129–1138.

[PGDDR10] Eric Prescott-Gagnon, Guy Desaulniers, Michael Drexl, and Louis-Martin Rousseau, *European driver rules in vehicle routing with time windows*, Transportation Science **44** (2010), no. 4, 455–473.

[PR09] Sandro Pirkwieser and Günther R. Raidl, *A column generation approach for the periodic vehicle routing problem with time windows*, Proceedings of the International Network Optimization Conference, vol. 2009, 2009.

[PU11] Patrick Prosser and Chris Unsworth, *Limited discrepancy search revisited*, Journal of Experimental Algorithmics (JEA) **16** (2011), 1–6.

[RCL13] Marie-Eve Rancourt, Jean-François Cordeau, and Gilbert Laporte, *Long-haul vehicle routing and scheduling with working hour rules*, Transportation Science **47** (2013), no. 1, 81–107.

[RGP02] Louis-Martin Rousseau, Michel Gendreau, and Gilles Pesant, *Solving small VRPTWs with constraint programming based column generation*, Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02), 2002, pp. 333–344.

[RSD02] Marc Reimann, Michael Stummer, and Karl Doerner, *A savings based ant system for the vehicle routing problem*, Proceedings of the genetic and evolutionary computation conference, 2002, p. 1317–1326.

[RTS11] Jidong Ren, Yajie Tian, and T. Sawaragi, *A relaxation method for the three-dimensional loading capacitated vehicle routing problem*, 2011 IEEE/SICE International Symposium on System Integration (SII), December 2011, pp. 750 – 755.

[RVBW06] Francesca Rossi, Peter Van Beek, and Toby Walsh, *Handbook of constraint programming*, vol. 2, Elsevier Science, 2006.

[RZMS13] Qingfang Ruan, Zhengqian Zhang, Lixin Miao, and Haitao Shen, *A hybrid approach for the vehicle routing problem with three-dimensional loading constraints*, Computers & Operations Research **40** (2013), no. 6, 1579–1589.

[Sal05] Matteo Salani, *Branch-and-price algorithms for vehicle routing problems*, Ph.D. thesis, Univeristà degli studi di Milano, 2005.

[Sha98] Paul Shaw, *Using constraint programming and local search methods to solve vehicle routing problems*, Principles and Practice of Constraint Programming—CP98 (1998), 417–431.

[Sha04] ______, *A constraint for bin packing*, Principles and Practice of Constraint Programming–CP 2004 (2004), 648–662.

[Sol87] Marius M. Solomon, *Algorithms for the vehicle routing and scheduling problems with time window constraints*, Operations Research **35** (1987), no. 2, 254–265 (en).

[Sol10] Christine Solnon, *Ant colony optimization and constraint programming*, Wiley-IEEE Press, 2010.

[Tai93] Éric Taillard, *Parallel iterative search methods for vehicle routing problems*, Networks **23** (1993), no. 8, 661–673.

[TDHI09] Fabien Tricoire, Karl F. Doerner, Richard F. Hartl, and Manuel Iori, *Heuristic and exact algorithms for the multi-pile vehicle routing problem*, OR Spectrum (2009), 931–f959.

[Tho01] Erlendur Thorsteinsson, *Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming*, Principles and Practice of Constraint Programming—CP 2001, 2001, p. 16–30.

[TLG96] Éric D. Taillard, Gilbert Laporte, and Michel Gendreau, *Vehicle routeing with multiple use of vehicles*, The Journal of the Operational Research Society **47** (1996), no. 8, 1065–1070.

[TV02] Paolo Toth and Daniele Vigo (eds.), *The vehicle routing problem, siam monographs on discrete mathematics and applications*, Society for Industrial and Applied Mathematics, 2002.

[TZK09] C.D. Tarantilis, E.E. Zachariadis, and C.T. Kiranoudis, *A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem*, IEEE Transactions on Intelligent Transportation Systems **10** (2009), no. 2, 255–271.

[VBJN94] Pamela H. Vance, Cynthia Barnhart, Ellis L. Johnson, and George L. Nemhauser, *Solving binary cutting stock problems by column generation and branch-and-bound*, Computational optimization and applications **3** (1994), no. 2, 111–130.

[VMdCM11] Cristiano Arbex Valle, Leonardo Conegundes Martinez, Alexandre Salles da Cunha, and Geraldo R. Mateus, *Heuristic and exact algorithms for a min–max selective vehicle routing problem*, Computers & Operations Research **38** (2011), no. 7, 1054–1065.

[Vog12] Ulrich Vogel, *A flexible metaheuristic framework for solving rich vehicle routing problems*, vol. 17, Shaker Verlag, 2012.

[Wol98] LA Wolsey, *Integer programming*, New York [etc.]: Wiley, 1998.

[XCRA03] H. Xu, Z.L. Chen, S. Rajagopal, and S. Arunapuram, *Solving a practical pickup and delivery problem*, Transportation Science **37** (2003), no. 3, 347–364.

[ZQLW12] Wenbin Zhu, Hu Qin, Andrew Lim, and Lei Wang, *A two-stage tabu search algorithm with enhanced packing heuristics for the 3L-CVRP and M3L-CVRP*, Computers & Operations Research **39** (2012), no. 9, 2178–2195.

[ZTK12] E. E. Zachariadis, C. D. Tarantilis, and C. T. Kiranoudis, *The pallet-packing vehicle routing problem*, Transportation Science **46** (2012), no. 3, 341–358.