UNIVERSITY OF THE AEGEAN

SCHOOL OF BUSINESS

DEPARTMENT OF FINANCIAL AND MANAGEMENT ENGINEERING

“A Toolkit for the Optimal Solution of the Vehicle Routing Problem with Time Windows and Capacity Constraints”

Georgios Dikas

Supervisor: Ioannis Minis

Chios, 2008

to my parents

ACKNOWLEDGMENTS

Firstly, I would like to thank my Professor Ioannis Minis for giving me the opportunity to undertake the present diploma thesis, for his valuable supervision and guidance. I gratefully thank Athanasopoulos Theodore and Ninikas Georgios, PhD Candidates of the University of the Aegean, for encouragement, countless discussions and for their help with writing the thesis. Their support was really significant for me. Furthermore, I am grateful to Vasilis Koutras PhD Candidate of the University of the Aegean, for his advises and support throughout the project. I would also like to acknowledge the support and the recourses made available to me through the DeOPSys Lab of the Financial and Management Engineering (FME) Department of the University of the Aegean, and especially to Taxiarxis Kourounis PhD Candidate of the University of the Aegean, for encouragement and for making the average work day more fun and interesting. Finally, I would like to thank my family and friends for their support during all these years of my educational period.

Thank you

ΠΕΡΊΛΗΨΗ (IN GREEK)

Σε παρούσα την διπλωματική εργασία παρουσιάζεται η δημιουργία προγραμματιστικών εργαλείων για την επίλυση του προβλήματος δρομολόγησης οχημάτων με χρονικά παράθυρα (Vehicle Routing Problem with Time Windows - VRPTW). Τα εργαλεία αυτά βασίστηκαν στην μέθοδο γραμμικού προγραμματισμού Column Generation για την εύρεση του κατώτερου ορίου (lower bound) του χαλαρωμένου (relaxed) προβλήματος και στον αλγόριθμο Branch and Bound για την επίτευξη της βέλτιστης ακεραία λύσης του προβλήματος. Η ανάπτυξη της μεθόδου έχει βασιστεί στην διδακτορική εργασία του Larsen (2001) και τις εργασίες άλλων ερευνητών οι οποίοι έχουν ασχοληθεί με το VRPTW. Το συγκεκριμένο εργαλείο θα χρησιμοποιηθεί από το εργαστήριο Συστημάτων Σχεδιασμού Παραγωγής και Λειτουργιών του τμήματος Μηχανικών Οικονομίας και Διοίκησης (ΤΜΟΔ), του Πανεπιστημίου Αιγαίου, στα πλαίσια του ερευνητικού του αντικειμένου που σχετίζεται με διάφορες παραλλαγές των προβλημάτων δρομολόγησης οχημάτων. Η μέθοδος Column Generation αποτελεί μια ευρέως χρησιμοποιούμενη μέθοδο τα τελευταία χρόνια και θα μπορούσε να εφαρμοστεί σε πολλά και διαφορετικά προβλήματα του πεδίου της Επιχειρησιακής Έρευνας που μελετούνται στον χώρο του ΤΜΟΔ και του Πανεπιστημίου Αιγαίου.

Το πρόβλημα VRPTW αφόρα την εύρεση των δρομολογίων ελάχιστου κόστους για ένα στόλο οχημάτων που ξεκινούν και επιστρέφουν σε μία κοινή αποθήκη, με σκοπό να εξυπηρετήσουν ακριβώς μία φορά κάθε πελάτη από ένα σύνολο πελατών. Οι πελάτες χαρακτηρίζονται από την ζήτηση, τον χρόνο εξυπηρέτησης και το χρονικό παράθυρο εξυπηρέτησης τους (time windows). Το κάθε όχημα έχει συγκεκριμένη χωρητικότητα (περιορίζοντας τον αριθμό των πελατών οι οποίοι μπορούν να εξυπηρετηθούν από ένα όχημα) και χαρακτηρίζεται από ένα μέγιστο χρόνο διαδρομής. Τα χρονικά παράθυρα και η ζήτηση του κάθε πελάτη θεωρούνται γνωστά εκ των προτέρων.

Η μέθοδος Column Generation εκμεταλλεύεται την ιδιαίτερη δομή των περιορισμών του προβλήματος (η οποία μπορεί να αποσυντεθεί σε μικρότερα προβλήματα) και είναι ιδιαιτέρα αποτελεσματική σε προβλήματα, όπου ο αριθμός των μεταβλητών ξεπερνάει κατά πολύ τον αριθμό τον περιορισμών (όπως στην περίπτωση του VRPTW). Βασικό χαρακτηριστικό της μεθόδου είναι ότι, σε αντίθεση με τις κοινές μεθόδους γραμμικού προγραμματισμού, αρχικά μόνο μια μικρή ομάδα μεταβλητών συμμετέχει στο πρόβλημα και σταδιακά προστίθενται νέες μεταβλητές. Η μέθοδος διαχωρίζεται σε δύο προβλήματα τα οποία συνδέονται μεταξύ τους, το Κυρίως Πρόβλημα (Master Problem) και το Υπό-Πρόβλημα (Sub-Problem).

Στην περίπτωση μας, το Κυρίως Πρόβλημα μοντελοποιήθηκε ως ένα πρόβλημα Κατάτμησης Συνόλου (Set Partitioning Problem) του οποίου έχουν χαλαρωθεί (relaxed) οι περιορισμοί ακεραιότητας των μεταβλητών. Η επίλυση του διεξάχθηκε με την Revised Simplex μέθοδο, η οποία αποτελεί παραλλαγή του κλασσικού αλγορίθμου Simplex, επιτυγχάνοντας μείωση του υπολογιστικού χρόνου και της

απαιτούμενης υπολογιστικής μνήμης. Ως μεταβλητές του συγκεκριμένου προβλήματος ορίστηκαν ολόκληρα μονοπάτια (δρομολόγια), σε αντίθεση με την κλασσική/ πλήρη μοντελοποίηση του VRPTW όπου οι μεταβλητές του προβλήματος αντιστοιχούν στις ακμές οι οποίες συνδέουν τους πελάτες. Οι μεταβλητές που συμμετέχουν σε αυτό το μοντέλο είναι όλα τα πιθανά εφικτά δρομολόγια. Οι περιορισμοί του μοντέλου αφορούν μόνο το πλήθος εξυπηρέτησης του κάθε πελάτη (κάθε πελάτης επιτρέπεται να μόνο μία φόρα και από ένα δρομολόγιο). Η διαδικασία επίλυσης αρχικοποιείται με μία «γρήγορη» λύση όπου κάθε δρομολόγιο συντίθεται από ένα πελάτη (αριθμός δρομολογίων ίσος με αριθμό πελατών). Η εύρεση των πιθανών εφικτών δρομολογίων παρέχεται από την επίλυση του Υπό-Προβλήματος, μέσω του οποίου σε κάθε επανάληψη προστίθενται καινούργια δρομολόγια.

Το Υπό-Πρόβλημα μοντελοποιήθηκε ως ένα πρόβλημα ελάχιστης διαδρομής (Shortest Path Problem) με επιπλέον περιορισμούς, οι οποίοι αφορούν τα χρονικά παράθυρα και την χωρητικότητα του οχήματος (Elementary Shortest Path with Time Windows ad Capacity Constraints). Ο όρος *elementary* αφορά την απαγόρευση δημιουργίας κύκλων στα μονοπάτια του προβλήματος. Ως κόστη κάθε ακμής του προβλήματος έχουν οριστεί τροποποιημένα κόστη (modified costs) με βάση της σκιώδης τιμές (shadow prices) που παράγονται από την λύση του Κυρίως Προβλήματος. Με τον τρόπο αυτό αναπαράγονται τα reduced cost coefficients της μεθόδου Simplex αν η επίλυση του προβλήματος διενεργούταν με τον κλασσικό τρόπο. Η επίλυση του Υπό-Προβλήματος διεξάχθηκε μέσω χρήσης δυναμικού προγραμματισμού, όπως περιγράφεται στην εργασία του Larsen (2001). Στην συγκεκριμένη εργασία αντιμετωπίζεται το *non-elementary* πρόβλημα, όπου επιτρέπονται κύκλοι στα δρομολόγια. Σε άλλες αντιμετωπίσεις του προβλήματος, Chabrier (2005), επιλύεται το πρόβλημα με την μορφή που παρουσιάζεται στην παρούσα εργασία, δηλαδή δεν επιτρέπονται κύκλοι. Ο αλγόριθμος βασίστηκε στον αλγόριθμο του Dijkstra (1959) όπου για κάθε πελάτη δημιουργείται μία ταμπέλα (label), η οποία επισημαίνει το κόστος της συντομότερης διαδρομής για τον κάθε πελάτη. Οι επιπλέον περιορισμοί, καθώς και η ύπαρξη αρνητικού κόστους στις ακμές του προβλήματος, απαιτούν την δημιουργία σύνθετων *labels* τα οποία χαρακτηρίζονται από το κόστος της συντομότερης διαδρομής, την αθροισμένη ζήτηση, τον αθροισμένο χρόνο και το δρομολόγιο μέχρι τον πελάτη τον οποίο αφορά το κάθε *label*. Επίσης απαιτείται να διατηρούνται περισσότερες από μία ταμπέλες για κάθε πελάτη καθώς τα επιπρόσθετα αυτά χαρακτηριστικά καθιστούν δύσκολη την αναγνώριση και την απόρριψη των *labels* «χαμηλής ποιότητας». Η απόρριψη των *labels* «χαμηλής ποιότητας» αποτελεί σημαντικό παράγοντα ο οποίος επηρεάζει τόσο τον υπολογιστικό χρόνο επίλυσης του προβλήματος, όσο και την επίτευξη της βέλτιστης λύσης. Η διαδικασία αυτή διενεργείται μέσω κριτηρίων κυριαρχίας (Dominance Rules) ενός *label* σε ένα άλλο. Τα κριτήρια κυριαρχίας τα οποία χρησιμοποιήθηκαν στην παρούσα εργασία έχουν παρουσιαστεί από τους Larsen (2001) και Chabrier (2005).

Τα βήματα της συνολικής μεθόδου παρουσιάζονται συνοπτικά κατωτέρω:

1. Εύρεση της αρχικής λύσης για το Κυρίως Πρόβλημα.
2. Επίλυση του Υπό-Προβλήματος με την μέθοδο Revised Simplex και παράγωγη των σκιωδών τιμών της καλύτερης λύσης (lower bound).
3. Υπολογισμός των τροποποιημένων κοστών (modified costs).
4. Επίλυση του Υπό-Προβλήματος
5. Εάν υπάρχουν δρομολόγια με αρνητικό συνολικό κόστος (reduced cost), εισαγωγή των δρομολογίων αυτών στο Κυρίως Πρόβλημα και επιστροφή στο βήμα 2. Αλλιώς, αν δεν υπάρχει κανένα δρομολόγιο με αρνητικό κόστος συνέχεια στο Βήμα 6.
6. Τερματισμός της διαδικασίας. Η λύση του τελευταίου προβλήματος είναι και η καλύτερη.

Η λύση που παράγεται από την παραπάνω μέθοδο αποτελεί το κατώτερο όριο (lower bound) του προβλήματος μιας και έχουν χαλαρωθεί (relaxed) οι περιορισμοί ακεραιότητας των μεταβλητών. Στην συνέχεια, η εύρεση της ακέραιας λύσης διενεργείται μέσω του αλγορίθμου Branch and Bound, μέσα στον οποίο έχει ενσωματωθεί η μέθοδος Column Generation. Μέσω του αλγορίθμου αυτού και εάν η Column Generation καταλήξει σε μη ακεραία λύση δημιουργεί δύο νέα Υπό-Προβλήματα με επιπλέον περιορισμούς, οι οποίοι αφορούν την επιλεγμένη μεταβλητή διακλάδωσης (branching). Στην συνεχεία επιλύονται τα νέα Υπό-Προβλήματα και η διαδικασία αυτή επαναλαμβάνεται μέχρι να βρεθεί η κατάλληλη ακέραια λύση.

Η πειραματική διερεύνηση διεξήχθη μέσω των πειραμάτων του Solomon (Solomon, 1987) τα οποία έχουν δημιουργηθεί ειδικά για το πρόβλημα VRPTW. Σκοπός των πειραμάτων ήταν η μελέτη των χαρακτηριστικών της προτεινόμενης μεθόδου, του μεγέθους των προβλημάτων που μπορούν να επιλυθούν, καθώς και η αποτελεσματικότητα της. Τα πειράματα επικεντρώθηκαν στην μελέτη των διαφορετικών κανόνων κυριαρχίας μιας και επηρεάζουν σημαντικά τόσο τον υπολογιστικό χρόνο όσο και την επίτευξη βέλτιστων λύσεων. Για την διεξαγωγή προβλημάτων με μικρό αριθμό πελατών, για τα οποία δεν υπάρχουν δημοσιοποιημένες οι βέλτιστες λύσεις, και τον έλεγχο της αποτελεσματικότητας της προτεινόμενης μεθόδου, χρησιμοποιήθηκε ένας εξαντλητικός αλγόριθμος (Athanasopoulos, 2008b).

Ως βασικά συμπεράσματα, σχετικά με την προτεινομένη μέθοδο, αναφέρονται τα εξής:

- Η μέθοδος χωρίς την χρησιμοποίηση κριτηρίων κυριαρχίας στο Υπό-Πρόβλημα παρουσιάζει μικρότερο χρόνο επίλυσης συγκριτικά με τον εξαντλητικό αλγόριθμο και επιτυγχάνει βέλτιστες λύσεις.
- Τα κριτήρια κυριαρχίας που προτάθηκαν από τον Chabrier (2005) βοηθούν στην βελτίωση της ταχύτητας επίλυσης των προβλημάτων, επιτυγχάνοντας πάλι την βέλτιστη λύση.
- Τα κριτήρια κυριαρχίας που προτάθηκαν από τον Larsen (2001) βελτιώνουν σημαντικά της ταχύτητας επίλυσης των προβλημάτων, αλλά

σε ορισμένες περιπτώσεις η βέλτιστη λύση δεν επιτυγχάνετε σε συνδυασμό με την μέθοδο επίλυσης του *elementary* Υπό-Προβλήματος.

- Μέσω του εξαντλητικού αλγορίθμου και της Column Generation χωρίς την χρησιμοποίηση κριτηρίων κυριαρχίας, επιλύθηκαν προβλήματα έως 9 πελάτες. Μέσω της χρήσης των κριτηρίων κυριαρχίας του Chabrier (2005) επιλύθηκαν προβλήματα έως 25 πελάτες. Προβλήματα περισσοτέρων πελατών αύξαναν δραματικά τον επιθυμητό υπολογιστικό χρόνο. Αντίθετα, μέσω των κριτηρίων κυριαρχίας του Larsen (2001) επιτεύχθηκε επίλυση προβλημάτων μέχρι και 100 πελατών.
- Από τα 44 προβλήματα με 25, 50 και 100 πελάτες, που επιλύθηκαν με την με την χρήση των κριτηρίων κυριαρχίας του Larsen (2001) στα 31 βρέθηκε η βέλτιστη λύση.
- Αντίστοιχα, η διαδικασία εισαγωγής πολλαπλών στηλών (δρομολογίων) από κάθε Υπό-Πρόβλημα στο Κυρίως Πρόβλημα παρουσίασε σημαντική μείωση του υπολογιστικού χρόνου επίλυσης (περισσότερο από 600%).

Γενικότερα, παρότι η μέθοδος αποτελεί μία σημαντική μέθοδο για την ακριβή επίλυση του VRPTW, περεταίρω έρευνα για την δημιουργία πιο αποτελεσματικών κριτηρίων κυριαρχίας και τεχνικών που θα μειώσουν τον απαιτούμενο υπολογιστικό χρόνο και θα διατηρήσουν την ποιότητα της παρεχόμενης λύσης, θα πρέπει να μελετηθούν και να αναπτυχθούν.

ABSTRACT

In this thesis we develop a toolkit to obtain efficient solutions for the Vehicle Routing Problem with Time Windows and Capacity Constraints (**VRPTW**) This toolkit has been based on the work of Larsen (2001) and Chabrier (2005) and has implemented the following methods / algorithms proposed in these references:

- A Linear Programming Algorithm that uses the revised simplex method
- A Column Generation Technique for linear problems
- A Dynamic Labeling Algorithm for the Shortest Path Problem with additional constraints.
- A Branch and Bound Algorithm to obtain integer solutions

These techniques are used in the following framework originally proposed by Larsen (2001): To obtain a good lower bound, the integrality constraints are relaxed, and the resulting general linear problem is divided in two separate problems; the Master Problem and the Sub-problem. The first one is formulated as a Set Partitioning Problem and it is solved through the revised simplex method. The dual (shadow) prices produced by this problem are sent to the sub-problem. The latter is formulated as an Elementary Shortest Path Problem with Time Windows and Capacity Constraints (**ESPPTWCC**). It is solved through a dynamic labeling algorithm. The sub-problem provides the necessary new columns (routes) to be inserted to the master problem, which is then, solved again. The final solution obtained by this iterative procedure is a good lower bound of the original integer problem. In order to obtain integer solutions, the aforementioned methods have been incorporated in a branch and bound scheme, which calls them iteratively to obtain the optimal (or a near optimal) solution.

The unified Branch and Bound and Column Generation framework used is called "Branch and Price via Column Generation". The shadow prices produced by the solution of the linear problem are considered to be a "weighting" factor of the network arcs, and they affect the selection of the proposed routes (columns) by the sub-problem. In the Column Generation technique not all the feasible routes have to be known in advance since routes (columns) will be created from the solution procedure. This is a strong advantage of the proposed method, which achieves an efficient solution to large-scale problems within reasonable computational time.

The methods proposed by Larsen (2001) guarantee optimality for problems of appropriate complexity. In large scale problems, however, the computational complexity may be prohibitive. In these cases, key in obtaining the optimum is the "intelligent" guidance of the column generation method in selecting the next appropriate column to be inserted in the linear program. This aspect is further analyzed in the present thesis, by testing two labeling techniques that “accelerate” the solution of the **ESPPTWCC** problem.

All the related experiments were performed using the Solomon Benchmark problems and compared against optimal solutions provided in the literature, and / or against an exhaustive search algorithm. Analytical results for the behavior of the proposed method, solution quality and computational complexity, are presented and discussed.

Keywords: Vehicle Routing; Column Generation; Branch and Price; Elementary Shortest Path Problem with Time Window and Capacity Constraints; Revised Simplex

CONTENTS

CHAPTER 1 INTRODUCTION

Transportation and distribution of goods are important areas of the supply chain since they affect the total cost of the product and the quality of customer service. Transportation is also a main contributor to global pollution. Therefore, any improvement on the time or distance covered by transportation vehicles will not only reduce the cost of services provided, but it will also help decreasing CO_2 emissions leading to more environmental friendly options and operations in transportation services.

For many decades, optimization of transportation operations has been studied and many techniques have been proposed. Currently the need for faster, more accurate and scalable techniques easily applied to real world problems has been increased and has gained the attention of the research community. Many problems in the field of transportation have been studied with operations research methods, including the so called vehicle routing problem (VRP) that attracted considerable attention. The VRP includes the design of a set of minimum cost routes starting and ending at a depot for a fleet of vehicles serving exactly once a set of customers with known demands and service costs. These routes should be designed subject to several constraints, such as the total time of travel (route length), or the maximum capacity of each vehicle. Many variations on this classical problem exist using different restrictions constraints, depending on problem under investigation.

The use of classical methods adopted from operations research in order to produce the optimal solution, seems to be inefficient in terms of computational time, especially when applied to medium-to-large-scale problems. Since many real world problems involve several hundred customers served by a large fleet, exact methods are not practical and, thus heuristics are used to obtain a good feasible solution in a timely manner. Vehicle routing problems belong to class of NP-hard optimization problems, in which computational time increases exponentially with the problems size. Although heuristics can handle complex problems, there are no guarantees that they will solve the problem optimally. Recently, several metaheuristics have also been put forward to solve the VRP. In contrast to heuristic that terminate when they reach to a local optimum, metaheuristics may search larger subsets of the solution space to find better solutions (even the optimal one) within a reasonable and acceptable time framework.

In the last twenty years, advanced exact optimization methods have been used extensively to these problems, aiming at decomposing the main problem into many smaller problems. Decomposition techniques like Lagrangian Relaxation or Column Generation within a Branch and Bound framework deal with vehicle routing problems and find the optimal solution in reasonable computation time by strengthening the conventional OR tools in terms of searching the feasible space and, thus, decrease considerably execution time.

In this work we are developing a toolkit that solves vehicle routing problems using Column Generation algorithm to derive a lower bound for the VRP and a Branch and Bound algorithm to obtain the optimal integer solution. The toolkit has been inspired by the doctoral work of Larsen (2001) as well as resent related work of other researchers and will be used by the Design, Operations & Production Systems Lab (DeOPSys) of the Financial and Management Engineering (FME) Department of the University of the Aegean, where several algorithms for problems of the VRP class have been developed and studied. Additionally, the implemented algorithms were based on the work of Athanasopoulos (2008b), which is conducted as part of his post-graduate studies. This toolkit will be used to produce benchmark solutions and will validate the quality of the algorithms under investigation. Furthermore, Column Generation is a sophisticated method, which could also be applied to other operations research (OR) fields being studied in FME to yield optimal solutions within a variety of constraints.

This thesis focuses on the Vehicle Routing Problem with Time Windows and Capacity Constraints, in which the service at each location can take place only during a given interval, called *time window* and total pickup or delivery loads must not exceed the vehicles capacity. Vehicles are not allowed to arrive to a customer after the end of the time window, while in case they arrive before the time window opens, the service cannot start until the time window begins. The fleet of vehicles is homogenous, meaning that all vehicles have the same capacity and all customers must be served exactly once. The scope of our methodology is to produce the optimal solution from the entire search space by examining the most fruitful regions only.

The methodology employed in this toolkit implement Larsen's method and comprises two parts: i) initially the lower bound (linear solution) is obtained through a Column Generation technique, and ii) the optimal integer solution of the problem is obtained using a Branch and Bound scheme (B&B) with embedded the column generation technique. This scheme is denoted as Branch and Price Column Generation. In the first part, a linear solution is initially obtained from a portion of all possible routes. The column generation technique determines if there are other routes to be included in the solution that could further reduce the objective value. Using the dual variables of the existing linear solution, a shortest path problem (with additional capacity and time-window constraints) is solved in order to identify if there is any route that could be included in the formulation. This step generates potential column(s) (routes) to be inserted, and the resulting and extended linear problem is solved. This procedure is continued until there are no additional routes that could reduce the objective function value, and, therefore, the lower bound has been reached. The second step (in cases where the lower bound is not integer) uses the linear solution produced the first step as a lower bound, and through a branch and bound tree the optimal integer solution is reached. Note that in every branch all steps of the first part (Column Generation) are repeated.

The remainder of the thesis is structured as follows: Chapter 2 describes the basic theoretical background and linear programming methods used, such as the Revised Simplex Method and Column Generation techniques. This is followed by the description of the Branch and Bound method and the most popular implementation strategies. Several variations of the VRP are also been described regarding modeling approaches, constraints and solution algorithms. Chapter 2 ends with the description of the basic theory of the Shortest Path Problem (SPP), which is critical for the implementation of the algorithm.

Chapter 3 presents the solution method of the VRP via Column Generation. It consists of two parts, firstly, the description of the master problem and, secondly, the description of the sub-problem, i.e. the Elementary Shortest Path Problem with Time Windows and Capacity Constraints (ESPPTWCC) is presented. The method used for the ESPPTWCC also incorporates key concepts of the dynamic labeling algorithm proposed by Kohl (1995), Larsen (2001) and Chabrier (2005).

Chapter 4 describes the policies used, regarding the branch and bound framework (that the column generation has been incorporated in), in order to find the optimal integer solution for the VRPTW. Also, the unified structure of the column generation within a branch and bound framework (branch and price) is presented.

Chapter 5 presents the description of the Solomon benchmark problems, which used to test the efficiency of the proposed algorithm. Additionally, the results obtained along with basic conclusions and findings are presented.

Finally, Chapter 6 presents the conclusions of the overall thesis and an evaluation of the performance of the methods used, along with several directions for future research.

CHAPTER 2 THEORETICAL BACKGROUND

The present chapter overviews the theoretical background for the methods employed in the VRPTW toolkit, that is, Column Generation algorithm and Branch and Bound technique. The first part of the chapter is dedicated to the linear programming theory that is related to Column Generation, including the Revised Simplex Method and the basic concept of Column Generation. The second part presents the well-known Branch and Bound method including its policies and characteristics. Finally, the Vehicle Routing problem is presented, along its variants, characteristics and solution methodologies.

2.1 LINEAR PROGRAMMING THEORY

2.1.1 THE REVISED SIMPLEX METHOD PROCEDURE

Revised Simplex Method (RSM) is a modification of the well-known simplex method. RSM is based on calculations, which are directly based on the inverse of the basis matrix and the main characteristic of it, is that calculations are performed to a part of the tableau. For example, if the number of constraints is much lower than the number of variables (the A matrix has fewer rows than columns) only a small number of columns will participate in the calculations to find the optimal solution. By this method, several calculations are avoided resulting in a considerable decrease of computation time and memory requirements.

As stated in Luenberger (1989) and Bradley (1977), Simplex method is expected to solve linear problems (in optimality) in about m or 3/2m pivot operations, where m is the number of constraints and n is the number of columns Thus, if the numbers of rows is considerably smaller that the number of columns ($m << n$) then, the pivoting operations needed to reach an optimal solution will address a small number of columns. In contrast, the traditional Simplex Method considers all elements of the involved matrices in the relevant calculations, thus increasing complexity and time. In many real-life problems, linear programming matrices have more columns than rows and many coefficients are likely to be zero (sparse matrices). According to the characteristics of the LP problems and the way the simplex method works, it can be easily shown that a significant amount of redundant information is generated at each step. Some references on RSM include the following [Luenberger, (1989); Bradley, (1977)]. The following are based on these references.

Consider a minimization problem in the standard form:

$$\begin{aligned} &\min Z = C\quad X \\ &s.t. \qquad AX = b \\ &\qquad\qquad X \geq 0 \end{aligned} \tag{2.1}$$

where $C\quad (1 \times n)$ is the cost vector, $X(n \times 1)$is the variable vector, $A(m \times m)$is the coefficient table and $b(m \times 1)$ is the right hand side coefficients. In order to present the revised simplex method, the above formulation should be rewritten in terms of the basic and non-basic variables, where B represents a sub-matrix resulting from matrix A, which corresponds to the basic variables (note that matrix B is defined always based on the initial A matrix) and is of size $(m \times m)$. At this point we have $A = [B, D], X = (X_B, X_D),\ C^T = (C_B^T, C_D^T)$ and the minimization problem becomes:

$$\begin{aligned} &\min \quad C_B\, X_B + C_D\, X_D \\ &s.t.\ BX_B + DX_D = b \\ &\qquad X_B, X_D \geq 0 \end{aligned} \tag{2.2}$$

The method of revised simplex can be described as follows:

Step 1

Starting by a dummy initial feasible solution $X_B = X_D = 0$, where X_B are m slack variables (each one for every constraint). The inverse of the current basis B is given by B^{-1} and the basic variables' values can be calculated as:

$$X_B = B^{-1}b \tag{2.3}$$

Step 2

Next, the non basic variable (X_{Dj}) to enter the basis should be calculated. Every variable is associated with a column vector D_j from the initial D matrix. As in the simplex procedure, the reduced cost coefficients r for each non basic variable are computed from the following formula:

$$r_j = \pi D_j - C_{D\ j} \tag{2.4}$$

Where π is a row vector containing all the shadow (or dual prices) of the current solution and π_j is the cost coefficient of the variable X_{D_j}The shadow prices produce a vector of $1 \times m$ elements (equal to the numbers of rows) and define the change in optimal objective function value per unit increase of a corresponding right hand side (RHS) coefficient and are given by:

$$\pi = C_B\ B^{-1} \tag{2.5}$$

If all reduced cost coefficients are positive or equal to zero ($r_j \geq 0\ \forall\ j$), then the optimal solution has been obtained. Otherwise, the most negative element (r_j) should be selected and the corresponding variable enters the basis.

Step 3

If variable X_{D_e} is selected to enter the basis with negative reduced cost coefficient r_e, then the representation of the column D_e (coefficients' column for variable X_{D_e} in the initial matrix D), that will enter the basis (as it will appear after all the pivoting operated up to now) is given by:

$$y_e = B^{-1}D_e \tag{2.6}$$

Step 5

In this step, the variable to exit the basis must be defined. Firstly, the minimum ratio rule is applied. The variable to exit the basis is selected by the following formula, where y_{ie} indicates the i^{st} element of column e, and b_i the i^{st} element of RHS column vector.

$$\min \{\frac{b_i}{y_{ie}} \;/\; y_{ie} > 0\} \tag{2.7}$$

The variable to exit from the basis is selected (variableX_{B_e}). In case there is no solution, then the problem is unbounded and no feasible solution can be found.

Step 6

In the final step, variable X_{B_e} is exchanged with variable X_{D_e} (and the respective columns in B are exchanged). The new basis B^{-1}, the RHS $b' = B^{-1}b$ and the current solution $z^* = C_B\, b'$ are calculated.

The iterations of the algorithm described above continue until an optimum is found or until the problem comes to an infeasible end.

An analytical example of the revised simplex method is given in appendix A. Figure 2.1 below provides an illustrative flowchart of the revised simplex method.

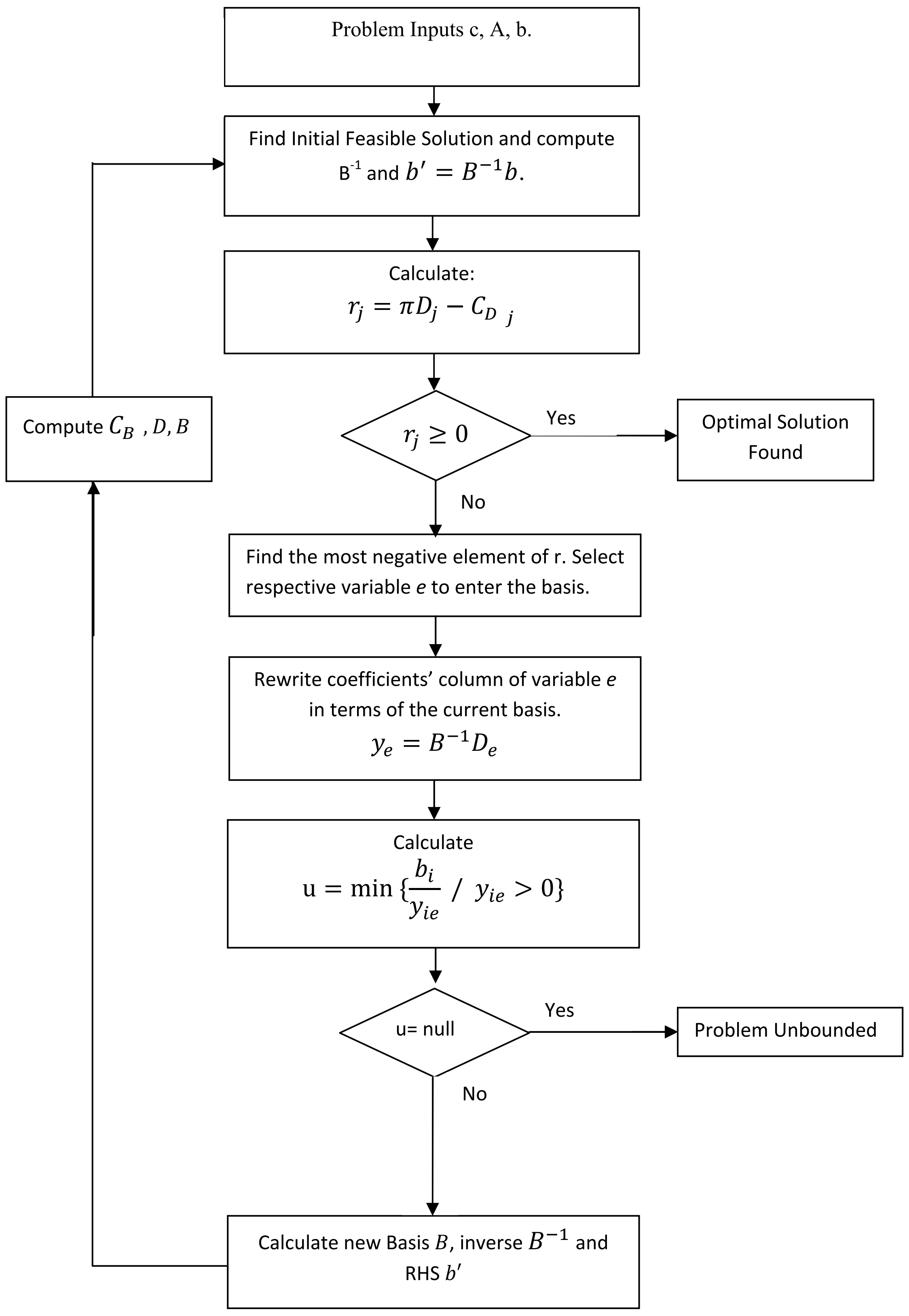

Problem Inputs c, A, b.
Find Initial Feasible Solution and compute B^{-1} and $b' = B^{-1}b$.
Calculate: $r_j = \pi D_j - C_{D\ j}$
$r_j \geq 0$
Yes
Optimal Solution Found
No
Compute C_B , D, B
Find the most negative element of r. Select respective variable *e* to enter the basis.
Rewrite coefficients' column of variable *e* in terms of the current basis. $y_e = B^{-1}D_e$
Calculate $u = \min\{\frac{b_i}{y_{ie}} \ / \ y_{ie} > 0\}$
u= null
Yes
Problem Unbounded
No
Calculate new Basis B, inverse B^{-1} and RHS b'

Figure 2.1: Flowchart of the Revised Simplex Method

2.1.2 Decomposition Method

Decomposition methods have proved a powerful tool for linear problems, where the coefficients structure (constraints) presents a certain pattern. Figure 2.2 presents several indicative patterns for which decomposition methods may be applied.

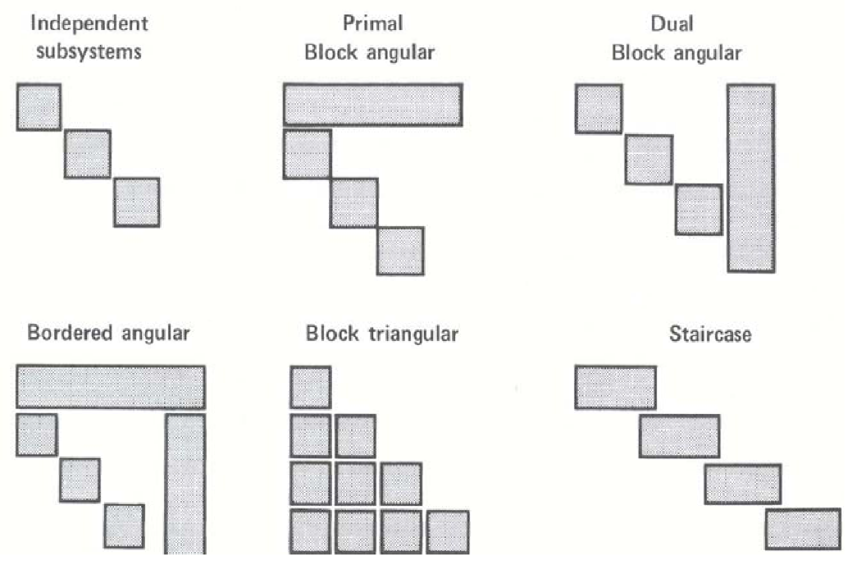


Figure 2.2: Several Constraints Structure (Bradley, 1977)

As stated in Bradley (1977), if a problem can be divided in x separate sub-problems then the solution of each sub-problem will require$(\frac{m}{x})^3$ computations, and the full problem will require $x(\frac{m}{x})^3 = \frac{m^3}{x^2}$ computations which are considerably smaller than the computations of a full problem (m^3).Additionally, the sub-problems can be solved in parallel achieving better computational savings and the ability to solve smaller problems has made this method as a very useful tool for large scale problems. Decomposition method initially introduced by Dantzig and Wolfe (1960).

The primal block angular structure will be presented. In this structure, there are several sub-problems, which are independent and can be solved separately, but there exists a set of constraints, which "connects" all the sub-problems together. These constraints are known as "coupling" constraints. The following are based on Luenberger, (1989) and Bradley, (1977).

The standard linear programming structure of a linear problem is:

$$\begin{aligned} \min Z &= C \quad X \\ s.t. \qquad AX &= b \\ X &\geq 0 \end{aligned} \tag{2.8}$$

where $C \quad (1 \times n)$, $X(n \times 1)$, $A(m \times m)$ and $b(m \times 1)$.

If the above problem presents the block angular structure, then it can be expanded as below:

$$
\begin{array}{llllll}
\min & C_1\ X_1 + C_2\ X_2 + \cdots + C_N\ X_N & & & & \\
& L_1\ X_1 + L_2\ X_2 + \cdots + C_N\ X_N & & & = & b_0 \\
& A_1\ X_1 & & & = & b_1 \\
& & A_2\ X_2 & & = & b_2 \\
& & \ddots & & & \vdots \\
& & & A_N X_N & = & b_N
\end{array}
\qquad (2.9)
$$

As it can be seen, the sub-problems can be regarded as N independent linear programming problems, which can be solved separately. Each sub-problem has the standard linear form:

$$
\begin{array}{ll}
\min & C_i\ \ X_i \\
s.t. & A_i X_i = b \\
& X_i \geq 0
\end{array}
\qquad (2.10)
$$

The solution for each sub-problem will lie upon a specific extreme point of the convex hull (of the sub-problem's constraints). For that, a transformation of the above formulation is proposed in order to reflect the extreme points of each sub-problem. The master problem constraint coefficients are now the extreme points of the sub-problems and its variables are the weight factors between the extreme points of each sub-problem. Note that a solution to a sub-problem is not always feasible for the master problem due to the linking constraints. This deficiency will be overcome by weighting the extreme points (by the selected variables) and the solution may not lie upon an extreme point of the sub-problems. The solution will be a combination of the extreme points, which optimize the master problem and respecting all constraints. Further references on the decomposition method and its principles are given in Bradley (1997) and Luenberger (1984).

2.1.3 Column Generation Procedure

As described previously, decomposition methods exploit the special structure of the constraints of some problem cases in order to provide a better solution in terms of computational time. Nevertheless, if the number of variables (therefore the size of the coefficient matrix A) is too large, it can still be prohibited to find the optimal solution. Column generation, in association with decomposition methods, can overpass this obstacle by simply generate coefficients columns only when needed by the optimization procedure. Analytical review on column generation is given in Bradley (1977), Desaulniers *et al.* (2005). In Figure 2.3 the grey part represents the variables that are not necessary to find the optimal solution and CG will not.

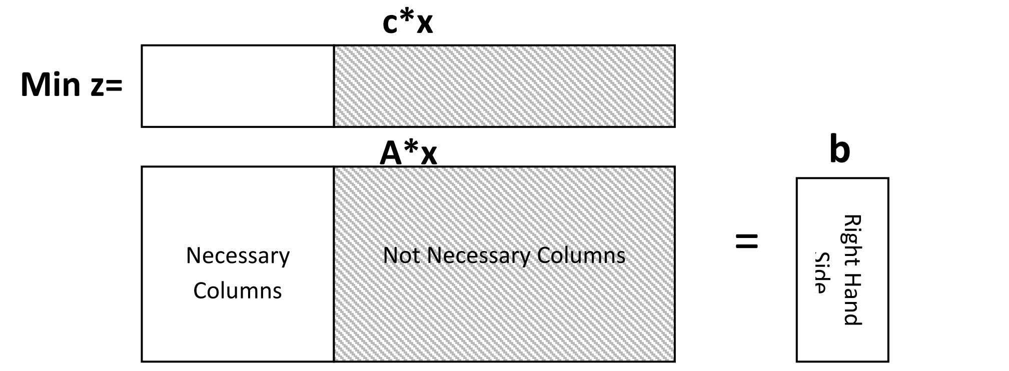


Figure 2.3: This figure represents a linear programming problem. The grey part contains those variables that are not necessary to find the optimum and they could be discarded.

Lubbecke and Desrosiers (2005) provide a historical review of the column generation along with many applications and research directions. Ford and Fulkerson (1958) are being referenced as the initiators of the column generation idea. Dantzing and Wolfe (1960) provided a strategy to deal with a linear program by splitting it into a master and several sub-problems, where columns to the master problem will be added continuously. Gilmore and Gomory (1961; 1963) presented the first practical implementation by solving a cutting stock problem by column generation. This problem, nowadays, consist of the most used example for column generation. In Desaulniers *et al.* (2005) a theoretical background, as well as many applications on column generation are presented.

Column generation deals with large scale linear programming problems. One of the main characteristics of the method is that it succeeds in determining an optimal solution without having to enumerate all variables of the problem. It is very efficient with problems that contain a large number of variables (columns) and a relatively small number of constraints ($n >> m$).

Generally, in Column Generation, the initial problem is called the Master Problem (MP). From the MP the Restricted Master Problem (RMP) is produced by including only a subset of the j variables, where $1 \leq j \leq n$, and all other variables are reduced to zero. So, initially, the RMP contains only those variables, as well as all the constraints of the MP (related to these variables). Note that the variables included in the RMP should result in a feasible solution. Finally, the next column (variable) to enter the RMP is selected by solving a special optimization problem called the sub-problem.

The steps of the method are presented below

Step 1

Initially, we define a feasible RMP that includes only $J \leq n$ variables. All other variables will be reduced to zero. RMP is usually produced from a heuristic algorithm, or even, starting with dummy variables.

Step 2

RMP is being solved (by simplex or by revised simplex) and generates an optimal solution regarding variables $x_1\ to\ x_j$. This solution is, also, feasible for the MP since all constraints have been maintained. In this step, the shadow (dual) prices are generated. Shadow prices ($\pi_i^j = c_B B^{-1}$) represent the rate of change of the Objective Function, by increasing the specific variable. In RSM, shadow prices are used to define the next entering variable and to test whether the Optimal Solution has been reached.

Step 3

The Sub-problem is being solved $z_{sub} = \min\{\sum_{i=1}^{m} \pi_i^j a_{ij} - c_j\}$. Where c_j are the cost coefficients, π_i^j are the shadow prices and a_{ij} are the coefficients of A and the variables of the sub-problem. In order to obtain feasible solutions from the sub-problem, several constraints have to be included. These constraints represent the region of feasible solutions and are based on the structural information of each sub-problem

If $z_{sub} \leq 0$, then the current solution is optimal (without having to enumerate all a_{ij}).

Else if $z_{sub} > 0$, then the current solution is not optimal and the sub-problem will provide all a_{ij} for the specific variable j to enter the basis of the RMP.

Step 4

Add a new column to RMP. The new column to be added has been provided by Step 3, by calculating the current representation of the a_{ij} provided based on the current basis ($y_j = B^{-1}a_j$). Increase the variables of the RMP by 1 (J + 1).

Column generation and decomposition method can be regarded as similar methods. A difference of column generation with decomposition methods is that in column generation the sub-problem to be solved can be of any form (dynamic programming, non linear, etc.) and not only linear programming. This characteristic makes column generation a strong tool, which can be applied to several different operations research problems. An analytical example of the column generation method is given in Appendix A.

2.2 Branch and Bound

Branch and Bound (B&B) technique has been used in order to provide integer solution to linear programming problems. It is the most common technique for integer programming. Over the years different aspects of dealing with B&B have been proposed making the B&B technique more effective and problem specific.

The basic idea behind B&B is to divide the feasible solution space into subdivisions. For each subdivision a new linear program can be solved by adding one additional constraint. This constraint marks the subdivision's space. The decision on the subdivisions is based on the non integer variables of the solution. Each non integer variable partitions the solution space into two subdivisions. This procedure is repeated until an integer solution is obtained. In general, there exist several alternatives in dividing the feasible region, and several different B&B tactics are proposed in the literature. For analytical information on the B&B techniques and methodologies see Lawler and Wood (1966), Lee and Mitchell (2001) and Chinneck (2003). The following review is based on Lee and Mitchell (2001).

2.2.1 Branch and Bound terminology and general description

In order to describe the B&B method the following terminology is being used, in general:

- A *Node* represents a solution obtained by the linear programming and it can be either integer, linear or infeasible; a *Bud* (or bud node) represents a solution obtained, which is linear and may be further expanded; and a *Leaf* (or leaf node) is a solution obtained, either feasible or infeasible which cannot be further expanded to other nodes. Nodes set comprise a superset of leaf and bud nodes sets.
- *Bounding function* is the method used for finding the optimal objective value in every node.
- *Branching* refers to the selection of the variable to branch (variable selection) and to the creation of the successor nodes of a bud node (partitioning).
- *Incumbent* is the best integer solution found so far.

In order to formulate a complete B&B technique, the following policies are required:

- *Variable selection policy* refers to the selection of the variable to further elaborate in the successor nodes of a bud node.
- *Partitioning policy* refers to the strategy the next branches will be created, upon the selection of the variable policy.
- *Node selection policy* refers to the next node that will select and therefore examined by the bounding function.

- Finally, terminating rules of the process should be determined.
    - *Fathoming rule* is the rule that stops the growing of a node (or branch), thus, making it a leaf node.
    - *Terminate rule* is the rule that terminates the whole process, regardless whether a final solution has been obtained or not.

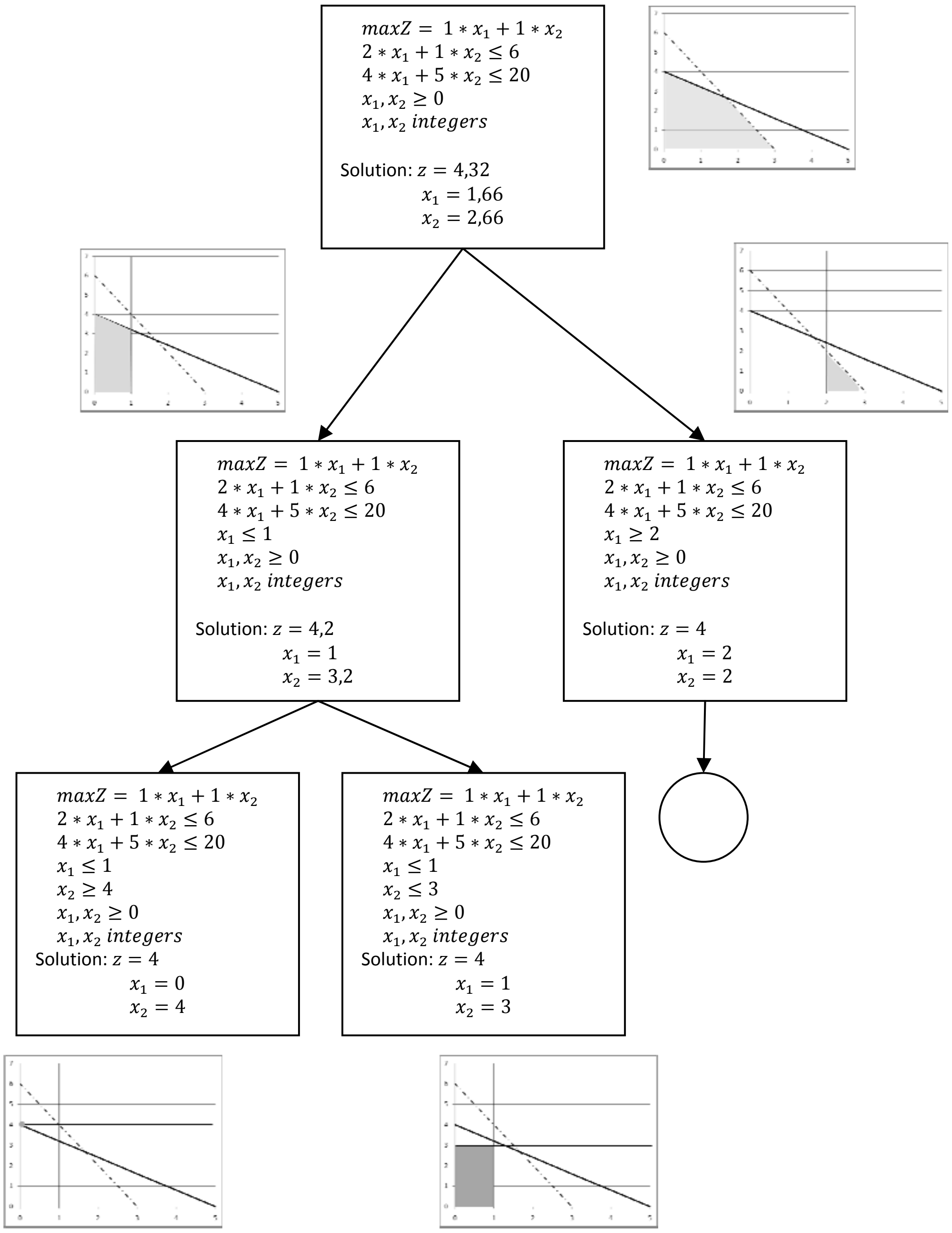
Figure 2.4 presents an example for the process of a B&B algorithm in an integer-programming problem.

2.2.2 VARIABLE SELECTION POLICIES

Variable selection is one of the critical aspects of B&B, since it affects the running time of the algorithm. Several approaches have been proposed by the researchers. The most common variable selection policy (which is addressed it the current thesis) is the *Most/Least Infeasible Integer Variable* (Danna, 2005). This policy selects the variable, which is most fractional and is farthest/closest from/to an integral value, as the branching variable. The selection of the farthest or the closest one is user-defined. Other indicative policies are mentioned:

- Driebeck-Tomlin Penalties (Driebeck, 1966; Tomlin, 1971)
- Strong Branching (Applegate *et al.*, 1995)
- Pseudo-Cost Estimate (Benichou *et al.,* 1971)
- Pseudo -Shadow Prices (Land, 1979)

Solving VRPs with the B&B technique in order to obtain integer solutions, branching policies (both partitioning and variable selection) are problem specific. Several general B&B branching operations reported in the literature can be applied, depending on the features of the problem.



$maxZ = 1 * x_1 + 1 * x_2$
$2 * x_1 + 1 * x_2 \leq 6$
$4 * x_1 + 5 * x_2 \leq 20$
$x_1, x_2 \geq 0$
x_1, x_2 $integers$

Solution: $z = 4{,}32$
$x_1 = 1{,}66$
$x_2 = 2{,}66$

$maxZ = 1 * x_1 + 1 * x_2$
$2 * x_1 + 1 * x_2 \leq 6$
$4 * x_1 + 5 * x_2 \leq 20$
$x_1 \leq 1$
$x_1, x_2 \geq 0$
x_1, x_2 $integers$

Solution: $z = 4{,}2$
$x_1 = 1$
$x_2 = 3{,}2$

$maxZ = 1 * x_1 + 1 * x_2$
$2 * x_1 + 1 * x_2 \leq 6$
$4 * x_1 + 5 * x_2 \leq 20$
$x_1 \geq 2$
$x_1, x_2 \geq 0$
x_1, x_2 $integers$

Solution: $z = 4$
$x_1 = 2$
$x_2 = 2$

$maxZ = 1 * x_1 + 1 * x_2$
$2 * x_1 + 1 * x_2 \leq 6$
$4 * x_1 + 5 * x_2 \leq 20$
$x_1 \leq 1$
$x_2 \geq 4$
$x_1, x_2 \geq 0$
x_1, x_2 $integers$
Solution: $z = 4$
$x_1 = 0$
$x_2 = 4$

$maxZ = 1 * x_1 + 1 * x_2$
$2 * x_1 + 1 * x_2 \leq 6$
$4 * x_1 + 5 * x_2 \leq 20$
$x_1 \leq 1$
$x_2 \leq 3$
$x_1, x_2 \geq 0$
x_1, x_2 $integers$
Solution: $z = 4$
$x_1 = 1$
$x_2 = 3$

Optimal solution $z = 4$ for:
x_1=2 and $x_2 = 2$
x'_1=0 and $x'_2 = 4$
x''_1=1 and $''x_2 = 3$

Figure 2.4: Example of the solution of an integer programming problem obtained with B&B

2.2.3 Partitioning Policies

Partitioning is typically realized with the addition of linear constraints, that is, the formation of new nodes on each division. Suppose x^R is the optimal solution to the relaxation of a branch and bound node. Common partitioning policies as overviewed in Lee and Mitchell (2001) include:

Variable Dichotomy: If x_j^R is fractional, then two new nodes are created, one with the simple bound $x_j \leq \lfloor x_j^R \rfloor$ and the other one with the bound $x_j \geq \lceil x_j^R \rceil$, where $\lceil \cdot \rceil$ denotes the ceiling value of a real number and $\lfloor \cdot \rfloor$ the floor value. If x_j is restricted to be binary, the branching reduces to fixing $x_j = 1$ and $x_j = 0$, respectively (Dakin 1965).

Generalized Upper Bound Dichotomy (GUB): If the constraint $\sum_{i \in Q} x_j = 1$ is present in the original integer program and $x_i^R, i \in Q$ are fractional the Q can be partitioned $Q = Q_1 \cup Q_2$ such that $\sum_{i \in Q_1} x_j = 0$ and $\sum_{i \in Q_2} x_j = 1$ respectively (Beale and Tomlin 1970).

Multiple branches for bounded integer variable: If x_j^R is fractional and $x_j \in \{0, \ldots, l\}$, then $l + 1$ new nodes can be created, with $x_j = k$ for node $k, k = 0, \ldots, l$. This idea was proposed in the first B&B algorithm by Land and Doig (1960).

2.2.4 Node Selection policies

Node selection policy refers to the selection of the next node of the B&B tree to be solved. It consists of a critical policy, which strongly affects the computational time of the algorithm. Many general and problem specific policies have been proposed in the literature; the most common as described in Lee and Mitchell (2001) are presented below.

Dept-First Search with Backtracking: This policy is as follows: Choose a successor (child) node from the previous node as the next node to examine; if it is already examined, choose another child node; if there is no child nodes unexamined, then, choose the most recent unexplored node. This method examines in depth a certain branch of the B&B tree and returns back to the closest unexplored node.

Best-Bound: This policy is as follows: Choose among all nodes the one that has the best linear objective value; If a node has linear objective value less (greater) than the incumbent do not expand further it. This policy minimizes the number of nodes explored since a node with larger linear objective value than the incumbent will not be examined because the corresponding integer value could never be less (higher) than the linear value in case of a minimization (maximization) problem.

Sum of Integer Infeasibilities: This policy chooses the node with either maximum or minimum sum of infeasibilities. The sum of infeasibilities (minimum case) at a node is calculated as

$$s = \sum_j \min(f_j, 1 - f_j) \tag{2.11}$$

Other known policies are *Best-Estimate using Pseudo-Cost*, *Best-Estimate using Pseudo -Shadow Prices* and *Best Projection.*

2.2.5 Branch And Bound with Column Generation

The column generation method embedded in a branch and bound framework is called branch-and-price. Initially, Desrosiers, Soumis, and Desrochers (1984) were the first to implement a branch and price method for solving the VRPTW problem (see Chapter 3). An analytical presentation of the method is given in Barnhart et al. (1998). Other references on column generation with integer programming include Desrosiers et al. (1995), Soumis (1997), and Wilhelm (2001). Although it seems that it is a straight forward implementation of the column generation technique into a B&B framework several researchers have mentioned the difficulties that have been raised (Johnson 1989, Barnhart et al. 1998).

2.3 LINEAR PROGRAMMING METHODS FOR VRP

2.3.1 THE VEHICLE ROUTING PROBLEM

The Vehicle Routing Problem (VRP) is related to many distribution and transportation problems. The supply chain costs until it reaches the end customer may reach 20% of its total cost (Reimann et al., 2003). This significant cost consuming service, led both academics and industry to optimize the operations in this area. The VRP is a generalization of the classic Traveling Salesman Problem (TSP) (Christofidis, 1979; Cornuejols and Nemhauser, 1978; Gendreau et al, 1997) and consists of finding a set of routes to serve a number of geographically dispersed customers at minimum cost. The VRP was introduced by Dantzing and Ramser (1959), in which the authors proposed a mathematical programming formulation and an algorithmic approach for a real-life problem for the delivery of gasoline to service stations. At the present time, VRP is one of the most studied problems of Operations Research, with many extensions and solution methods (Tatarakis 2007).

The objective of the VRP is to find a set of minimum cost routes for an available fleet of vehicles starting and ending at a depot, in order to deliver goods to a set of customers with known demand (Clarke and Wright, 1964; Golden & Assad, 1998; Laporte and Osman, 1995). A very useful survey of significant research results in this content is given by Toth & Vigo (2002).

According to Stewart and Golden (1983), a compact and convenient formulation for the VRP can be written as follows:

$$Minimize \sum_{k}\sum_{i,j} c_{ij}x_{ijk} \tag{2.12}$$

Subject to

$$\sum_{i,j} \mu_i x_{ijk} \leq Q \quad k = 1,2,\dots,m \tag{2.13}$$

$$x = \lfloor x_{ijk} \rfloor \in S_m \tag{2.14}$$

where:

c_{ij} = the cost of traveling from i to j

x_{ijk} = 1 if the vehicle k travels from i to j and x_{ijk} = 0 otherwise

m = the number of available vehicles

S_m = the set of all feasible solutions in m-traveling salesman problem (m-TSP)

μ_ι = the demand at location i

Q = the vehicle capacity

From the above formulation it is clear that the VRP is an integer - programming problem. It is also an NP-hard problem, and therefore, practical problem instances cannot be optimally solved within reasonable time (see Toth and Vigo, 2002). Figure 2.5 present as network of customers along with the feasible routing solution.

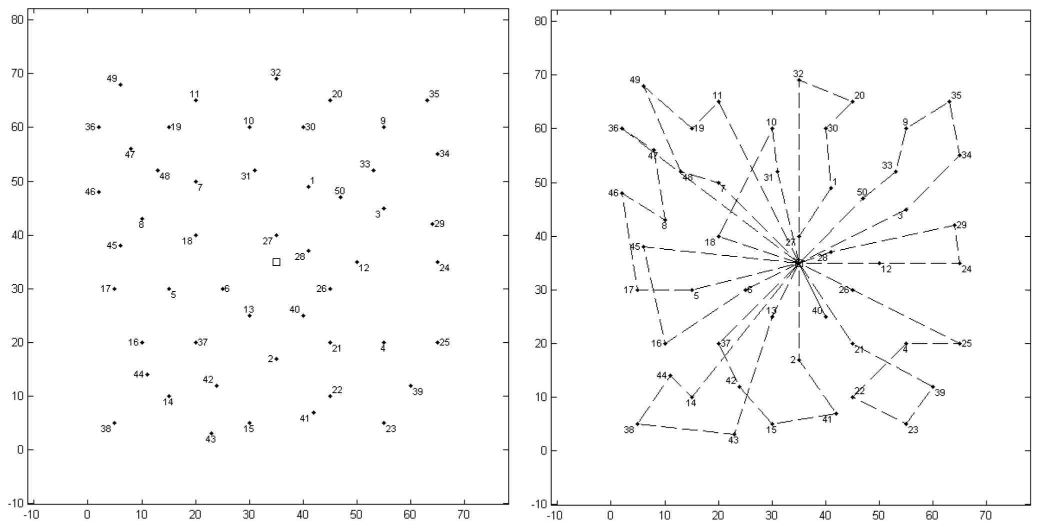


Figure 2.5: A solution example of the VRP

2.3.2 Modeling Approaches for the VRP

According to Toth and Vigo (2002): "*Three basic modeling approaches have been proposed in the literature for the VRP. The models of the first type are known as vehicle flow formulations and they use integer variables associated with each arc or edge of the graph, which count the number of times the arc or edge is traversed by a vehicle. These are the most frequently used models for the basic versions of the VRP; they are particularly suited for cases in which a) the cost of the solution can be expressed as the sum of the costs associated with the arcs, and b) the most relevant constraints concern the direct transition between the customers within the route, so they can be effectively modeled through an appropriate definition of the arc set and the arc costs. On the other hand, vehicle flow models cannot be used to handle some particular issues, such as in cases in which the cost of a solution depends on the overall vertex sequence, or on type of the vehicle assigned to a particular route (Toth and Vigo 2002). The second family of models is based on the so-called commodity flow formulation. In this type of model, additional integer variables are associated with the arcs or edges and represent the flow of commodities along the paths traveled by the vehicles. Only recently these types have been used as the basis for the exact solution of Capacitated VRP (CVRP).*

The models of the third family have an exponential number of binary variables, each associated with a different feasible circuit. The VRP is then formulated as a Set-Partitioning Problem (SPP) seeking a collection of circuits that minimize cost, serving each customer once and possibly satisfying additional constrains. A main *advantage of this model is that allows for extremely general route costs (for modeling cost that depend on the sequence of the arcs and/or the vehicle type). Moreover, the additional side constraints do not need to take into account restrictions concerning the feasibility of a single route. As a result, the constraints can often be replaced with a compact set of inequalities. This produces a formulation, the linear relaxation of which is typically much tighter than previous model types (Toth and Vigo 2002).”*

2.3.3 Basic Problems of the VRP Class

In this section, we will describe the most common VRP variants which respect to the different types of constraints. We focus on three main variants of the VRP, the Capacitated VRP, the VRP with Time Windows and the VRP with Pickup and Delivery.

Capacitated VRP

The Capacitated VRP (CVRP) is almost similar to the simple VRP, since most of the VRP models introduced to the literature contain capacity restrictions for the vehicles. In the CVRP all demands from the customers are deterministic, known in advance and cannot be split. All vehicles are identical and start from a main depot. The objective is to minimize the total routing cost by serving all customers exactly once without exceeding capacity constraints (Toth and Vigo 2002).

VRP with Time Windows

The VRP with Time Windows (VRPTW) is an extension of the CVRP, where in addition to the capacity constraints, every customer i should be served in a specific, predefined time interval (time window). The time window contains an early arrival time a_i and a late arrival time b_i. The time interval $[a_i, b_i\ ]$ is the time window, in which the customer i should start being served. Travel time from customer i to customer j, t_{ij}, and service time for each customer, s_i, are given (Toth and Vigo, 2002; Ahn and Shin, 1991; Atkinson, 1994). This class of vehicle routing problems is studied in the present thesis and it will be further described in the following sections.

Other VRP variations are:

- VRP with Pickup and Delivery (Toth and Vigo, 2002; Daganzo and Hall, 1993)
- Distance Constrained VRP(Toth and Vigo, 2002)
- Multi-Depot VRP (Bianco *et al.*, 1994; Carpaneto *et al.*, 1989)
- Heterogeneous Capacitated VRP (Taillard, 1996)
- VRP with Backhauls (Toth and Vigo, 2002; Golden *et al.*1988)
- Multi-Period VRP (Tan and Beasley, 1984; Christofides and Beasley, 1984)

2.3. 4 Solution Algorithms for the VRP with Time Windows

In this section we will describe solution methods for the VRP with Time Windows (VRPTW) as described by Toth and Vigo (2002).

2.3.4.1 Heuristic Approaches

Heuristic approaches are typically used in practice (real-life problems) and focus on finding a feasible solution until all customers are served and all constraints are satisfied. Typically, heuristics approaches are problem-oriented and usually end to a sub-optimal solution (Toth and Vigo, 2002). In this section we examine three different classes of heuristics – route construction procedures, two phase algorithms and tour improvement procedures. Interested readers should see Assad (1998), Christofides (1979). Each one is described below:

Route Construction

Route construction algorithms gradually build a feasible solution while keeping an eye on solution cost, but they do not contain an improvement phase. They are typically divided into sequential and parallel methods. Sequential methods construct one route at a time until all customers are included while parallel procedures are characterized by the construction of a number of routes simultaneously. The routes are either allowed to form freely or their number is fixed a priori (Clarke and Wright, 1964).

Two – phase algorithms

Two-phase heuristics are mainly divided in two classes: i) Cluster-first, route-second methods, where vertices (customers) are first organized into feasible clusters and then a vehicle route is constructed for each of them and ii) Route-first, cluster-second methods, where a tour is firstly built including all vertices and is then segmented into feasible vehicle routes.

Route Improvement

Route improvement methods perform local searches for better neighborhood solutions in order to improve a given initial one. This is usually achieved by edge or vertex exchanges within or between the vehicle routes. The process terminates when the current solution cannot be further improved (Toth and Vigo 2002).

2.3.4.2 Metaheuristics

Over the last few years, many authors have proposed new heuristic approaches, called metaheuristics, for tackling the VRP. These perform a thorough exploration of the solution space, the exploration of deteriorating, or, even, infeasible solutions during the procedure. Their main advantage is that they do not terminate when a local optimum is reached and they explore a larger sub-set of the solution space in order to find a solution closer to the optimal one. Generally, metaheuristics could be classified in three main categories: a) local search (simulated annealing, tabu search), b) population search (adaptive memory procedures, genetic search) and, c) learning mechanisms (neural networks, ant colony systems) (Toth and Vigo 2002).

2.3.4.3 Decomposition Approaches

The main idea of the decomposition approaches is to exploit the special structure of the VRPTW problems, which allows the problem to split to several sub-problems, or to relax the constraints of the problem resulting in tighter lower bounds. This section describes the two more popular decomposition techniques: Lagrangian Relaxation and Column Generation. The interested reader could find more information in Huisman *et al.* (2005). For column generation references see Chapter 3.

Lagrangian Relaxation

In Lagrange relaxation, several constraints are selected and relaxed. That is, the selected constraints are added to the objective function followed by a penalty factor (langrangian multiplier λ). The master problem consists of finding the values of the langrangian multipliers, and the subproblem is a network flow problem with additional constraints (the ones that have not been removed). See Kohl (1997), Geoffrion (1974) and Fisher (1985).

Column Generation

As described in 2.1.2, the decomposition is based on two structures: a) the master problem, which in VRPTW usually is formulated as set partitioning problem, and b) the sub-problem, which in VRPTW is an elementary shortest path problem with time windows and capacity constraints (ESPPTWCC); the two problems interact and the required information is sent to each problem. ESPPTWCC uses modified costs based on the real costs and the dual prices obtained from the master problem and the Master Problem uses the new columns information to be added to the problem. The general column generation method is presented in Chapter 2.1.3 and the VRPTW with Column Generation is presented in Chapter 3.

Since the current work is based on column generation, the set partitioning problem and the shortest path problem are overviewed next.

2.3.5 Set Partitioning Model

The set partitioning model was first proposed by Balinski and Quandt (1964). A direct formulation of the master problem, used in column generation, for the vehicle routing problem with time windows can be given through a set partitioning model. To describe this model, let G be a directed graph, C be the set of customers and V a set of identical vehicles (Desrochers 1992; Toth and Vigo 2002). The variable x_r is defined as:

$$x_r = \begin{cases} 1, \text{if the route } r \text{ is used in the solution} \\ 0, \text{otherwise} \end{cases} \tag{2.15}$$

and c_r denotes the total cost of the route. As a result, the model has the following form:

$$min \sum_{r \in R} c_r \, x_r \tag{2.16}$$

$$\sum_{r \in R} \delta_{ir} \, x_r = 1 \tag{2.17}$$

$$x_r \ \in \{0,1\} \tag{2.18}$$

where R is the set of all feasible routes, and δ_{ir} is 1 if the customer r is serviced by route r and 0 otherwise.

Since enumerating all feasible routes is considered as a NP-hard problem, the column generation approach starts from an initial solution that contains a small number of feasible routes. Additional routes are added to the above formulation only when needed. This is achieved by solving a sub-problem, in compliance with the above master problem (see Section 2.4.3). Note that all the vehicle, time window and capacity constraints are included in the sub-problem, and, therefore, the routes added to the set partitioning formulation are all feasible.

2.4 THE SHORTEST PATH PROBLEM

The **Shortest Path Problem** (SPP) refers to the problem of finding a path between a start node (vertex) and an end node, in order to minimize the sum of the costs (weights) of the traversed edges. In many methods, SPP is used as a sub-problem and it is one of the most studied problems in graph theory. Cormen *et al.* (2003) present an analytical review of several variants of the shortest path problems. This section describes the single-source, single-destination shortest path problem and is based on the aforementioned reference.

Consider a directed graph $G\ (V, E)$, where V is the set of vertices and E the set of edges in the graph. Each arc (i, j) is associated with a weight w. Each vertex (v) is associated with a label $\lambda[v]$ (the sum of weights for the current shortest path to node v from the initial node. It represents the upper bound of the shortest path). The shortest path from a start vertex (s) to an end vertex (e) is denoted as $\delta\ [s, e]$. In Section 2.4.1 some core properties of the shortest path problem are given and in Section 2.4.2 the most known solutions algorithms are provided.

2.4.1 PROPERTIES OF THE SHORTEST PATH

Optimal substructure: Considering a shortest path $\delta[u, v]$ from node u to node v, then every path included in $\delta[u, v]$ is also a shortest path. For example in Figure 2.5, if the shortest path from node 1 to node 5 is[1,3,2,4,5] with cost 20, then the shortest way to travel from node 3 to node 4 is [3,2,4] with cost 10.

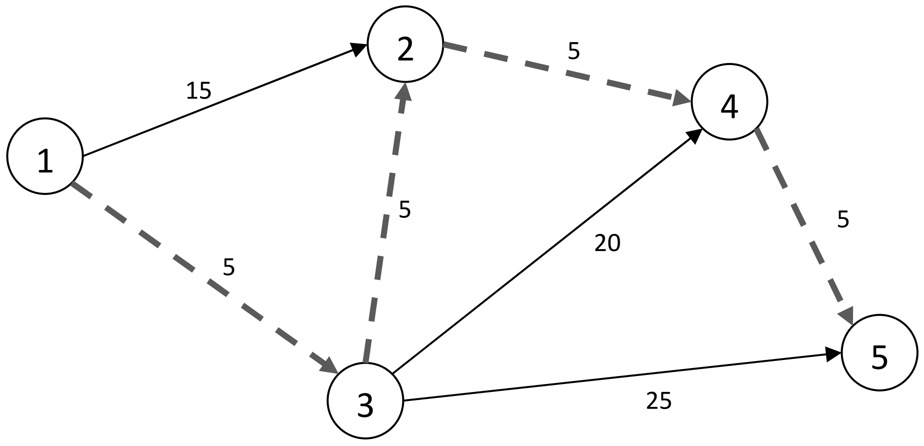


Figure 2.5: Shortest path example

Triangle inequality in Shortest Paths: For every set of arcs $(u, v, s) \in E$, where s is the starting vertex, then the weight of the shortest path $\delta[s, v]$ from s to v is less than the weight of the shortest path $\delta[s, u]$ plus the weight of the arc (u, v). That is, $\delta[s, v] \leq \delta[s, u] + w(u, v)$.

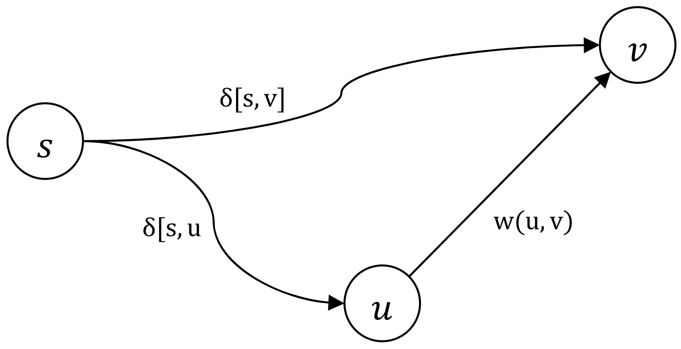


Figure 2.6: Representation of the triangle inequality property on Shortest Paths

Upper Bound Property: For each vertex $v \epsilon V,$ the value of $\lambda(v)$ cannot be less than the total sum of weights of the shortest path $\delta[s, v]$. That is, $\lambda(v) \geq \delta[s, v]$. If the label of a vertex is assigned with the value $\lambda(v) =$ [s, v] then it will remain until the solution process ends.

Convergence Property: Assume that a shortest path exists, which traverses vertices u and v sequentially, that is $s \rightsquigarrow u \rightarrow v$. If $\lambda(u)$ has been assigned to value δ[s, u] (meaning the shortest path from s to u has already found) prior to the update of label $\lambda(v)$, then $\lambda(v)$ will be equal to δ[s, v] (shortest) after the update.

No-path Property: If a path from $s\ to\ v$ does not exist then $\lambda(v) = \delta$[s, v] = ∞ .

2.4.2 Solution Algorithms

In this chapter, the two most known shortest path algorithms are presented. That is, the Dijkstra's algorithm (Dijkstra, 1959) and the Bellman-Ford algorithm (Bellman, 1958; Ford and Fulkerson, 1962)

Dijkstra's algorithm:

Using Dijkstra's algorithm the directed single-source shortest-path problem with non-negative edge path weights is solved. Although it is a greedy algorithm that resembles the breadth-first search it provides the shortest paths on the graph. The algorithm starts at a source vertex s. A list T contains all the vertices h, for which the shortest path from the starting vertex $\delta[s, h]$ has been found. Vertices are added to T in order of distance, i.e. first s, then the vertex closest to s, then the next closest, and so on. The value $\lambda(g)$ of every vertex g not in T, which is connected to the vertices in T, is updated based on the minimum cumulative weights. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined.

Bellman-Ford algorithm

Bellman – Ford algorithm solves the directed single-source shortest-path problem. In difference with Dijkstra's algorithm it also finds shortest paths in graphs where negative edge weights exist. Initially, the algorithm sets the label of the source vertex to 0 and all other vertices to ∞. Then, for $h-1$ iterations (h is the number of vertices) it traverses the edges updating the labels $l(u)$. Finally, negative weight cycles are checked by the following equation:

$$l[v] > l[u] + w(u,v) \quad ,\forall (u,v) \in E \tag{2.19}$$

If such a negative cycle is present solution does not exist.

2.4.3 Shortest Path with Time Windows and Capacity Constraints

When vehicle routing problems are solved with column generation techniques, the SPP with resource constraints (or other variants) can be used as the sub-problem. A resource corresponds to a quantity, such as the time and the demand. In our approach, these resources correspond to time windows and capacity constraints. The following are based on Larsen (2001).

The mathematical formulation of the problem is:

$$(\text{ESPPTWCC}) \quad \min \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij} \tag{2.20}$$

$$s.t. \quad \sum_{i \in C} d_i \sum_{j \in N} x_{ij} \leq q \tag{2.21}$$

$$\sum_{j \in N} x_{0j} = 1 \tag{2.22}$$

$$\sum_{i \in N} x_{ih} - \sum_{j \in N} x_{hj} = 0 \qquad \forall h \in C \tag{2.23}$$

$$\sum_{i \in N} x_{i,n+1} = 1 \tag{2.24}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \qquad \forall S \subseteq N \ , |S| \geq 2 \tag{2.25}$$

$$s_i + t_{ij} - K(1 - x_{ij}) \leq s_i \qquad \forall i,j \in N \tag{2.26}$$

$$a_i \leq s_i \leq b_i \qquad \forall i \in N \tag{2.27}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in N \tag{2.28}$$

Constraint (2.21) is the capacity constraint of the route (the demand of the customers assigned to each route should be less or equal to q); constraints (2.22), (2.24) are the constraints ensuring that each route will start and finish to the depot; Constraint (2.23) is the flow conservation constraints. Constraint (2.25) is the well known sub-tour elimination constraint. Constraints (2.26) and (2.27) ensure that every customer will be served in its time window and, finally, constraint (2.28) forces variables x_{ij} to be binary.

The above problem is known as the Elementary Shortest Path with Time Windows and Capacity Constraints (ESPPTWCC) since every customer can exist in a route only one time (no repetitions are allowed). In order to overcome the computational complexity of the aforementioned problem, the customers are allowed to be serviced more than once in each route, by creating cycles. This type of problem is known as the non-elementary SPPTWCC. In this formulation, a node may be visited more than once, leading to cycles, for example a cycle of the form $(i) - (j) - (i)$. Note that cycling will terminate at a point where the time window or capacity constraint will be met. Figure 2.7 presents an illustrative cycling example in a directed graph of five nodes. In this example, a "cycle-path" 1-2-3-4-2-3-5 may occur, while in path 1-3-5 there is no cycle. Eliminating those cycles has been addressed in the literature by several researchers, specifically the two-cycle elimination in the context of shortest path algorithms was first presented by Houck et al. (1980) and Christofides, Mingozzi and Toth (1981). Larsen (2001) uses the SPPTWCC problem as a sub-problem with a two-cycle elimination technique.

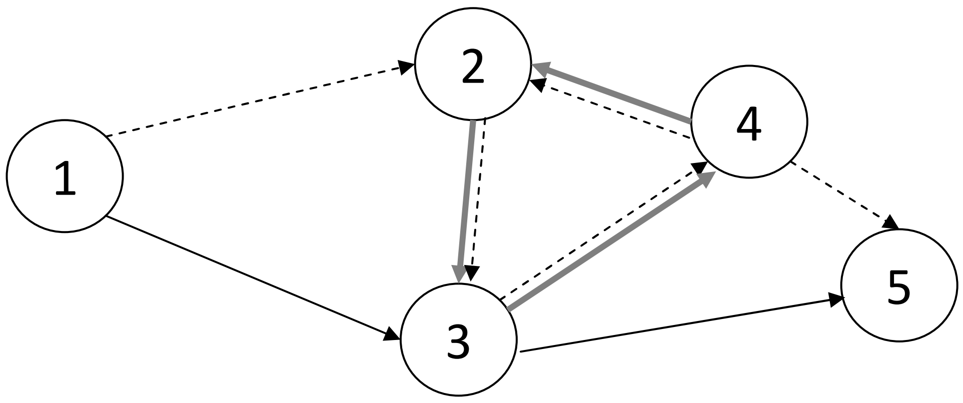


Figure 2.7: Example of generated "cycle-paths"

Since the solution of the elementary problem was considered very difficult for practical reasons, several researchers proposed different relaxations for the elementary problem. Recently, Irnich and Villeneuve (2003) showed that by eliminating larger cycles, the lower bound obtained by column generation process was drastically improved. Furthermore, Feillet (2004) and Chabrier (2005) proposed the use of the elementary shortest path problem with time windows and capacity constraints as the sub-problem of the column generation process.

CHAPTER 3 A GOOD LOWER BOUND FOR THE VRPTW USING COLUMN GENERATION

In this chapter the implementation of the column generation method for the Vehicle Routing Problem with Time Windows and Capacity Constraints will be described. The mathematical programming framework used to produce the lower bound relies on two basic concepts. The first concept is decomposition that transforms the original model into a model that contains many columns and fewer rows, called master problem, and the second concept is column generation. In order to solve the linear relaxation of the extensive formulation one does not generate the entire model since the latter is typically is very large - the number of variables often grows exponentially with the size of the original problem. Instead, columns are generated dynamically using a technique known as column generation. In this chapter the set partitioning model, used as the master problem and the Elementary Shortest Path with Time Windows and Capacity Constraints (ESPPTWC) used as the sub-problem are analytically described. Finally, the dynamic programming algorithm for the ESPPTWC is also described. The mathematical formulations and notations presented in this chapter are based on Larsen (2001).

The vehicle routing problem with time windows is defined by a directed graph G, a set of customers C and a fleet of vehicles V. The fleet is assumed to be homogeneous, meaning that all vehicles have the same capacity and cover the same distance in the same time. The graph contains $|C| + 2$ vertices, where there are n customers plus the two starting and ending positions of the vehicle. Let vertex 0 denote the starting position of the vehicle and $n + 1$ be the ending position. Both starting and ending positions correspond to the depot. The whole set of vertices $0, 1, \dots, n + 1$ is denoted as $N = |C| + 2$. The set of arcs A represents the direct connections between all vertices and the depot. Each arc(i, j), where $i \neq j$, has an associated cost c_{ij} and a time t_{ij}. All vehicles have the same capacity q. Each customer i must be serviced within a time window $[a_i, b_i]$. In case a vehicle arrives before the opening of that time interval, it must wait until a_i to start serving that customer (Kallehauge *et al*, 2005). The variable s_{ik} for each vertex i and each vehicle k denotes the time vehicle k starts to service customer i. Two sets of decision variables participate in this model x and s. For every arc (i, j), where $i \neq j, i \neq n + 1, j \neq 0$ and each vehicle is defined as k. x_{ijk} defined as:

$$x_{ijk} = \begin{cases} 1, \text{if vehicle } k \text{ drives from vertex } i \text{ to } j \\ 0, \text{otherwise} \end{cases}$$

A less compact formulation than the one in Section 2.3.1 is presented below:

$$min \quad \sum_{k \in V} \sum_{i \in N} \sum_{j \in N} c_{ij}\, x_{ijk} \qquad (3.1)$$

$$\text{s.t.} \quad \sum_{i \in N} \sum_{j \in N} x_{ijk} = 1 \qquad \forall i \in C \qquad (3.2)$$

$$\sum_{i \in C} d_i \sum_{j \in N} x_{ijk} \leq q \qquad \forall k \in V \qquad (3.3)$$

$$\sum_{j \in N} x_{0jk} = 1 \qquad \forall k \in V \qquad (3.4)$$

$$\sum_{i \in N} x_{ihk} - \sum_{j \in N} x_{hjk} = 0 \qquad \forall h \in C, \forall k \in V \qquad (3.5)$$

$$\sum_{i \in N} x_{i,n+1} = 1 \qquad \forall k \in V \qquad (3.6)$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - 1 \qquad \forall S \subseteq N \ , |S| \geq 2, \forall k \in V \qquad (3.7)$$

$$s_{ik} + t_{ij} - K(1 - x_{ijk}) \leq s_{jk} \qquad \forall i, j \in N, \forall k \in V \qquad (3.8)$$

$$a_i \leq s_{ik} \leq b_i \qquad \forall i \in N, \forall k \in V \qquad (3.9)$$

$$x_{ijk} \in \{0,1\} \qquad \forall i, j \in N, \forall k \in V \qquad (3.10)$$

Constraint (3.2) ensures that each customer is visited exactly once. Equation (3.3) represents the capacity constraints and Eqs. (3.4) and (3.6) state that each vehicle will start and end at the depot. Constraint (3.5) is the flow conservation constraint. Constraint (3.7) is the well known sub-tour elimination constraints. Constraint (3.8) ensures that customer j cannot be serviced immediately after customer i if the $s_{ik} + t_{ij}$ is more than s_{jk} (K represents a large scalar). Equation (3.9) represents time window constraints and (3.10) represents the binary conditions for the variables.

As described in Section 2.1.3, Column Generation deals with linear programming problems. Specifically, the general idea of column generation is to use only a relevant small number of columns in order to find the optimal solution out of a large set of possible columns. This is achieved by dividing the process into two parts: the Master-Problem (MP) and the Sub-Problem (SP). In the Set-Partitioning formulation of the Master-Problem, as we will further describe in Section 3.2, each column represents one feasible route and each constraint (row) represent one customer. The goal of the Master-Problem is to find the minimal cost, as common methods do, given specific feasible routes and to produce the shadow prices (one for every row) of the temporary optimal solution to be used as input in the Sub-Problem. In our method, the SP is modeled as the Elementary Shortest Path Problem with Time Windows and Capacity Constraints (ESPPTWCC), where elementary means that each customer can appear at most once in the shortest path. The traveling cost of every pair of customers and the depot is a function of the actual cost and the shadow price of each customer. This cost will be called the modified cost.

On the other hand, the Sub-Problem produces one or more routes to be added in the Master-Problem. The Sub-Problem deals with all constraints of the problem; In the case of VRPTWCC, time window and capacity constraints. The routes to be added in the MP must satisfy all VRPTWCC constraints. The modified cost of each route to be entered is selected to be the most negative one. If there is no route with negative modified cost, the process terminates.

3.1 THE MASTER PROBLEM

A direct formulation of the Master problem of the VRPTW, under a column generation framework, is usually the set partitioning problem. In this type of problem each column corresponds to one route and each constraint corresponds to a customer participation in the available routes. If the customer participates in the route, then the coefficient of its row in the constraint matrix is 1 and 0 otherwise. Given variable x_r, which is defined as below:

$$x_r = \begin{cases} 1, \text{if the route } r \text{ is used in the solution} \\ 0, \text{otherwise} \end{cases} \tag{3.11}$$

and c_r, which denotes the total cost route r , then the Master Problem is of the following form:

$$min \sum_{r \in R} c_r \, x_r \tag{3.12}$$

$$\sum_{r \in R} \delta_{ir} \, x_r = 1 \tag{3.13}$$

$$x_r \in \{0,1\} \tag{3.14}$$

where R is the set of all feasible routes, and δ_{ir} is 1 if the customer i is serviced by route r and 0 otherwise.

Since the lower bound of the set partitioning problem is considered, constraint (3.14) is eliminated and the linear relaxation of the above formulation is solved.

The above model, in the column generation approach, contains only the routes that have been generated by the ESPPTWCC solution plus the initially given routes. Each time the ESPPTWCC is being solved, new columns (routes) are being generated and inserted in the set partitioning formulation.

The outcome will be a primal and a dual solution, which may or may not be integer. In this case, the dual prices of the simplex multipliers are denoted as: $\pi = \pi_1, \pi_2, \dots, \pi_{|C|}$.

3.2 The Sub-Problem

The sub-problem is of significant importance due to its relation with the accuracy and the speed of the provided solution and since it provides the next column(s) to enter the basis of the master problem. As presented above, the set partitioning formulation of the master problem does not consider the vehicle, capacity and time window constraints. All these constraints are included in the sub-problem, and the generated columns (routes) are all feasible routes. The sub-problem returns the column(s) with the most negative reduced cost.

The sub-problem has been defined as an elementary shortest path problem with time windows and capacity constraints (ESPPTWCC). Note that in Larsen (2001), the sub-problem had been defined as a SPPTWCC where cycles (multiple visits to a single node in a route were allowed). ESPPTWCC allows no cycles in the generated routes. The cost of the arcs for the ESPPTWCC (denoted as modified cost) has been defined from the actual cost (c_{ij}) of the arcs and the shadow prices (π_i) generated by the master problem. The following type gives the modified costs:

$$\hat{c}_{ij} = c_{ij} - \pi_i , \forall \, i \neq 0 \tag{3.15}$$

For $i = 0, \; \hat{c}_{ij} = c_{ij}$. The total cost ($\hat{C}_r$) of the generated column (route) r is defined as the sum of the modified costs of the arcs participating in route r, that is:

$$\hat{C}_r = \sum_{(i,j) \in r} \hat{c}_{ij} \tag{3.16}$$

Note that this cost is equal to the reduced cost that would have resulted from the master problem if the route r was already present in the master problem, that is:

$$\hat{C}_r = \sum_{(i,j)\in r} \hat{c}_{ij} = \sum_{(i,j)\in r} c_{ij} - \sum_{i\in r} \pi_i = c_r - \pi\delta_r \tag{3.17}$$

Where c_r is the actual cost of route r, π is the shadow (dual) prices of the simplex multipliers, and δ_r is the route vector as denoted in Chapter 3.1.

Consequently, the result from the solution of the sub-problem will be one or more paths (routes) with negative modified cost. These columns will be added to the master problem and the master problem will be solved again. If there are no paths with negative cost, generated from the sub-problem, the method terminates and the results from the last master problem solution are kept. This solution is considered as a lower bound (if it is not integer) for the VRPTW, and further investigation through a branch and bound framework is required. In case the solution is integer the algorithm terminates.

3.2.1 THE ALGORITHM

Dijkstra's algorithm consists of the most used method to deal with the Shortest Path Problem (SPP). In this method, a labeling technique is used for each node of the network. Assume that there is a network of arcs and nodes, with non-negative edge costs c_{ij} and several geographically dispersed nodes (customers). Dijkstra algorithm finds the path with the minimum total distance (based on the c_{ij}'s) from a starting node (s) to an ending node (f). The steps of the method are as follows: Initially, only the starting node s is considered visited and all other nodes are considered unvisited. All nodes (i) are associated with a label (L_i), which is equal to 0 for $i = s$ and equal to ∞ for all other nodes. L_i denotes the length of the shortest path (found so far) from the starting node s to node i. At each iteration of the method the all the out-going arcs (i, j) for every node i, which has been considered as a visited customer are checked. For each j (reached from each visited node i) the value of the $L_i + c_{ij}$ is calculated. If this is less than the existing label L_i at node j, then the latter is updated to $L_i = L_i + c_{ij}$. Among all the unvisited nodes considered, the node with the minimum-length label is selected and characterized as visited. This procedure is repeated until there are no more unvisited nodes.

Dijkstra's algorithm finds the optimal solution in networks with non-negative costs only. Also, it cannot guarantee optimality to the addition of more constraints. In order to solve the shortest path problem with additional constraints such as time window and capacity constraints and use a network defined based on the modified costs, which may be negative, an altered method, based on the labeling algorithm for the SPPTWCC has been proposed by Larsen (2001).

The labeling algorithm is an extension of Dijkstra's algorithm. In this method each label is defined as $c(i, t, d)$ where i is the vertex, t and d is the time and demand, respectively, based on the partial path up to node$i$. c denotes the total cost of each label. The labeling algorithm selects the label with the minimum t (note that t is always increasing over time) and expands it further to the nodes (say t) connected to it (in the VRPTW case to all other nodes), taking under consideration the feasibility constraints (time window, capacity), that is, the label $c(i,t,d)$ can reach node j and create a successor label $c(j,t',d')$ if the following holds:

$$t + t_{ij} \leq b_j \tag{3.18}$$
$$d + d_j \leq q \tag{3.19}$$

Note that the maximum route length has been set equal to the latest time window of the depot, and therefore, equation (3.18) satisfies this constraint.

Initially, the only label present is the starting node (depot$=0$) label. For each node j a new label $c(j,t',d')$ is generated. At this point, we cannot discard a previous assigned label to node j as described in Dijkstra algorithm and, therefore, more than one label may exist for each node. Including the new created labels, again the minimum t label is selected and the procedure is repeated. Note that, each time a label is selected to be expanded to the nodes connected to it, it is discarded (deleted) from the labels list, unless the label is associated with node $n+1$. In the latter case these labels are deleted and stored separately and represent the feasible paths. Note that discarding a label does not mean that the associated node will not be checked again, since this node can be reached from other nodes of the network and new labels associated to it, will be created. The procedure ends when there are no other labels to be examined. The labeling algorithm can be described by the dynamic program:

$$c(0,0,0) = 0 \tag{3.20}$$
$$c(j,t,d) = \min_i \{\hat{c}_{ij} + c(i,T',D') \mid T' + t_{ij} = T \wedge D' + d_i = D\} \tag{3.21}$$

Cycle Elimination

In the formulation presented by Larsen (2001) and Kohl (1995), the SPPTWCC problem was solved as a sub-problem where cycles (i.e. visiting the same customer more than once in the same path) were allowed in the generated paths. In order to tackle this, a *two-cycle elimination* was proposed, which by keeps an extra component ($pred$) in the label $c(i,t,d,pred)$. $pred$ is the predecessor of node i. The phenomenon of cycles in the paths generated from the shortest path problem solution has been analyzed and described in Section 2.4.

In ESPPTWCC, which is the sub-problem solved in our formulation, cycles are not allowed in the generated paths. This is tackled by keeping an extra component ($path$) in the label $c(i,t,d,path)$. $path$ stores all the path information (visited nodes and sequence) from the depot to node i. If label $c(i,t,d,path)$ examines the creation of a new label for node j [$c(j,t',d',path)$] and j exists in the $path$, then $c(j,t',d',path)$ label is not created.

Dominance Rules

In large scale problems the number of labels can be extremely increased, constituting the solution procedure computationally and memory prohibitive. Many researchers have tried to overcome this by recognizing and discarding labels (and therefore partial paths), which can be considered as 'low quality' labels, before they reach the end node. If a label (partial path) is discarded, then all its possible extensions will not be examined. As "low quality" labels are considered the labels, which are expected to not participate in any feasible route (up to node $n + 1$). An algorithm that recognizes and discards these labels, will not affect the optimality of the ESPPTWCC solution, but as it will be shown, it still remains computationally expensive. This procedure is denoted as "Dominance Rules".

Over the years different rules have been proposed, see Dumas (1986), Kohl (1995), Larsen (2001) and Chabrier (2005). However, many of the proposed dominance criteria do not ensure optimality because several "good" labels are discarded that may lead to the optimal solutions. On the other hand, tighter rules speed up the solution procedure because of the elimination of the generated labels.

Two different sets of dominance rules will be presented. The first one is a simplified version of the dominance rules presented in Larsen (2001) aiming to the decrease of the computational complexity and to the solution of large scale problems. Note that these rules do not ensure optimality to all test-cases; the second set is based on the dominance rules introduced in Chabrier (2005). Although these rules ensure optimality, the computational complexity is higher than the aforementioned rules.

Simplified Dominance Rules (Larsen)

Assuming that there are two labels for the same node $c(i, t1, d1, path1)$ and $c(i, t2, d2, path2)$ with costs c_1 and c_2, respectively. The first label dominates the second one if and only if the following hold:

$$c_1 \leq c_2 \quad (3.22)$$
$$t_1 \leq t_2 \quad (3.23)$$
$$d_1 \leq d_2 \quad (3.24)$$

The second label can be discarded since, as it appears, any path from node i to the depot will be always better (or equal) considering the $path1$. Although this is true, there are several other aspects that are not been taken under consideration and can lead to suboptimal solutions.

The implementation in Larsen (2001) with the above dominance rules in parallel with the relaxed shortest path problem (SPPTWCC) seems to perform better than with the ESPPTWCC. This can be supported by the fact that the allowance of cycles in the generated routes allows more labels to be created and therefore more paths to be extended to nodes that otherwise would have been eliminated.

Exact dominance rules

The simplified dominance rules in relation to the elementary-SPPTWCC do not ensure optimality in all cases. As first introduced by Dumas (1986) and further extended by Chambier (2005), extra rules should be included to ensure optimality. Assuming, again $path1$ and $path2$, the following rule is included to the aforementioned rules:

$$path1 \supseteq path2 \tag{3.25}$$

This rule ensure that the $c(i, t1, d1, path1)$ label will dominate $c(i, t2, d2, path2)$ label only if $path1$ contains at least the same customers with $path2$. Adding this rule more labels will be allowed to be created, but the algorithm is expected to converge to an optimal solution. Several improvements, regarding the acceleration of these dominance rules, introduced in Chambier (2005). In our formulation only the aforementioned rule was considered without the acceleration techniques proposed.

3.3 Solution Method

In this chapter the unified method, including all the aforementioned procedures will be described. As mentioned previously, the method begins with a feasible initial solution. This solution can be generated using a heuristic algorithm or can be a dummy feasible solution. In our case, the single-customer routes $D-1-D, D-2-D, \dots, D-C-D$ were selected, where D is the depot. By doing so, the constraints of the master problem are immediately equal to the total constraints of the full problem (if we knew all feasible routes). This procedure helps in avoiding several degeneracy problems, and no redundant rows in the constraint matrix of the master problem will be created.

The master problem (set partitioning) is solved using the Revised Simplex Method (presented in Chapter 2.1.1). The Sub-Problem (ESPPTWCC) is solved using the dynamic algorithm described in Section 3.2.2. This algorithm returns the feasible route(s) with the most negative reduced cost for the current problem, if one exists. The generated route(s) are added to the routes already in the master problem, and the master problem is solved again. In our implementation we allow the ESSPPTWCC to serve the master problem with multiple columns (specifically, with all generated columns with negative reduced cost). The procedure continues until the ESPPTWCC cannot produce any other columns (routes), with negative reduced cost, to enter the master problem. The last solution of the master problem obtained is considered as the final solution (lower bound obtained). The development of the algorithm is based on Athanasopoulos (2008b). The basic steps of the proposed method are described below:

1. Find an initial solution for the master problem (set partitioning problem).
2. Solve the master problem using the Revised Simplex Method and obtain the shadow prices of the optimal solution.
3. Produce the modified costs, as described in Section 3.2 using equation (3.15).
4. Solve the sub-problem.
5. If there are generated route(s) with negative reduced cost, add these route(s) to existing routes of the master problem and go to Step 2; Else, if there no routes with negative reduced cost, go to Step 6.
6. Terminate procedure. The solution obtained from the last master problem is the lower bound obtained.

Figure 3.1 presents the column generation method flowchart.

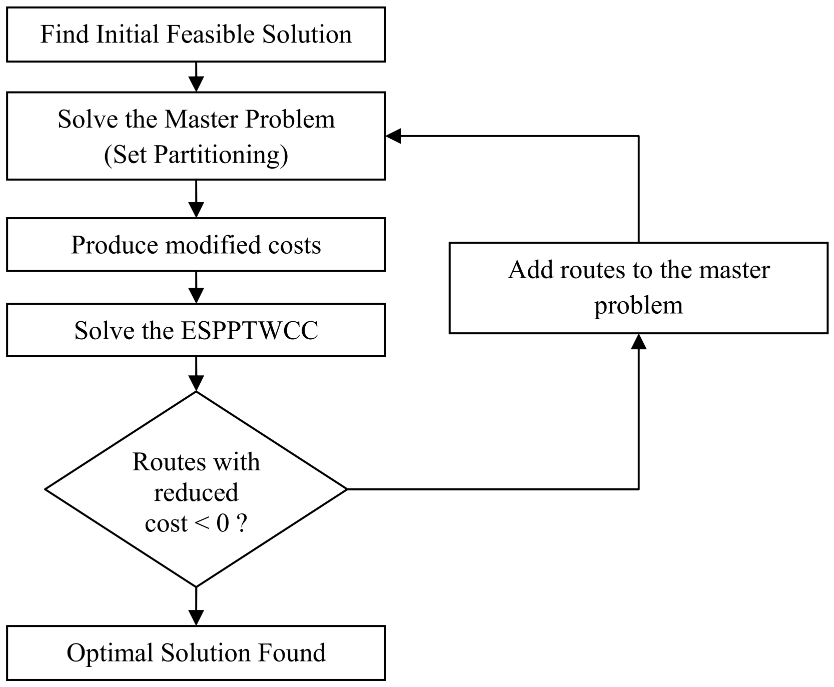

Find Initial Feasible Solution
Solve the Master Problem (Set Partitioning)
Produce modified costs
Solve the ESPPTWCC
Routes with reduced cost < 0 ?
Add routes to the master problem
Optimal Solution Found

Figure 3.1: Flowchart of column generation method for the VRP

CHAPTER 4 THE OPTIMAL SOLUTION FOR THE VRPTW USING BRANCH AND BOUND

This chapter presents the IP formulation applied to the Vehicle routing Problem with Time windows and Capacity Constraints that consists of the Column Generation (CG) algorithm presented in the previous chapter embedded within a Branch and Bound (B&B) framework. Firstly, the methods and policies used for the VRPTW class problems are presented and then, the policies applied on our algorithm will be emphasized.

4.1 Setting up the Branch and Bound Algorithm

As described in Chapter 2.2, Branch and Bound is a case-sensitive algorithm and should be adapted to the characteristics of the problem. In our case Column Generation (CG) produces solutions, which is non-integer in general. B&B produces extra restrictions dividing the search space into two parts in order to progressively achieve integrality. A variety of policies and strategies can be applied to achieve the best integer solution. If CG returns a fractional (non-integer) solution, two new branches are created with extra conditions at each one and the linear solution returned at that point is used as a lower bound, meaning that branch cannot return an integer solution better than the lower bound. Each child is solved with Column Generation; if an integer solution is obtained, then the current branch stops growing. The lowest integer solution is set to be the General Upper Bound (GUB). If there is a node with lower bound worse than GUB, this node is discarded.

4.1.1 Terminology

This Section provides the terminology related to the branch and bound features and characteristics, as described in Section 2.2, and defines the selected policies used in our problem for each step of the algorithm.

Bounding function: The bounding function includes the mathematical programming formulation used for the solution of the problem at each node. Obviously, in our case, we use the Column Generation framework, as described in Chapter 3.

Incumbent: As incumbent, we will consider the best integer solution found so far at a certain step. This is the General Upper Bound (GUB).

Fathoming Rule: A bud node will be forced to stop expanding, if i) the solution is worse than the GUB, ii) the solution is infeasible or iii) an integer solution has been reached.

Terminate Rule: If there is no bud node left to examine, the whole process will be terminated. This termination criterion has been chosen because it is directly linked with the fathoming rule (i.e. the process will inevitably reach a state from which there will be no more bud nodes left to be examined).

4.1.2 Implementation Issues

Following the guidelines presented in Chapter 2, our implementation of the branch-and-bound algorithm consists of the following steps:

Step 1 (*Lower bound*): Solve the LP relaxation of the VRPTW

Step 2 (*Optimality check*): If the solution provided from the above step is integer, compare it with the current GUB (if exists) and go to Step 5, otherwise continue to Step 3

Step 3 (*Variable Selection Policy*): Choose the arc of the head node with the most fractional part (see below) to be the basis for the newly created constraints.

Step 4 (*Partitioning Policy*): The head node is being divided in two children and an additional constraint is added to each one of them, based on the arc selected in the previous step.

Step 5 (*Node Selection Policy*): Choose the bud node with the minimum lower bound from the entire tree. If there is no bud node left to be examined, or the lower bound of the selected bud node is higher than GUB, then the optimal integer solution has been achieved (current GUB). Otherwise, return to Step 1.

The basic policies used in the above methodology are further described below:

Variable Selection Policy

The *Branching on Arcs* policy is used for the selection of the variable, which will determine the two sub-spaces to be created. In the proposed policy when a non-integer solution is obtained by the set partitioning model (master problem), the most fractional arc is selected as the branching variable. Since the variables of the master problem are the routes and not the arcs connecting costumers, the following must be performed: From the routes participating in the solution and their customer visiting sequence, sum of the vehicle flows (defined as the values of the route variables) in each one of the arcs participating in the solution. The arc (i, j) with the fractional value closest to 0.5 in the linear solution obtained is chosen for branching (Danna, 2005).

Partitioning Policy

The policy used in this step dichotomizes the "father" node in the children. Suppose that arc (i, j) is selected as branching variable. Then for the first branch (child node), the vehicle can reach customer j only from customer i and from customer i can travel only to customer j or back to the depot. For the second branch, movements from customer i to j are strictly forbidden in any case. Note that these restrictions affect only the sub-problem (ESPPTWCC) because the constraints are added only to this formulation. In the set partitioning formulation of the VRPTW where variables represent only routes and not arcs, routes that contain this arc are discarded.

Node Selection Policy

We have chosen to implement a best-bound node selection policy, where the node with the best LP objective value among the entire tree is selected to be explored. This policy provides a fast lower bound in terms of computational time and ensures that the optimal solution will be found, compared to other popular policies.

4.2 The Branch and Bound Procedure

This section provides a short overview of the Branch-and-Bound method applied on the solution of the linear relaxation solution provided by the column generation process. The root of the B&B tree is the initial column generation solution. And each node (leaf) is a separate optimization problem based on the initial problem with the addition of a specific constraint for the fractional variable.

A pseudo-code of the main branch-and-bound algorithm is shown in Figure 4.1. Since the B&B procedure follows the linear relaxation of the main integer problem, the main inputs for the B&B algorithm are the solution obtained by solving the linear relaxation problem, i.e. the lower bound and the variables value (denoted as lb and X respectively) and the original problem's objective and constraints (denoted as *parentnode*). The initial state of the procedure is to generate the branch-and-bound tree. This is achieved by selecting the variable to be branched and partitioning the parent node (original problem) into two new nodes. The functions that execute these operations are italicized in Figure 4.1 due to their importance and are further described in the remainder of this chapter.

```
Algorithm Branch&Bound(lb, X, parentnode)
// $lb$ := lower bound obtained by linear relaxation
Initialize tree_list = ∅                 // branch-and-bound tree
GUB = $inf$                               // General upper bound
($arc_{ij}$) = VariableSelection(X, parentnode)
(newnodes) = NodePartition($arc_{ij}$, parentnode)
tree_list = tree_list ∪ newnodes
while (tree_list ≠ ∅) do
      node = NodeSelection(tree_list)
      (lb′, X′, node) = lpsolver(node)
      if (solution ≔ infeasible) or (lb′ > GUB then)
            node = ∅; // Delete node
      else
            if X′ ≔ integer then
                  GUB = lb′
                  node = ∅
            else
                  ($arc_{ij}$) = VariableSelection(X′, node)
                  (newnodes) = NodePartition($arc_{ij}$, node)
                  tree_list = tree_list ∪ newnodes
                  node = ∅
            end if
      end if
return
```

Figure 4.1. Branch-and-Bound procedure

In the main loop of the procedure, at each iteration the algorithm selects a leaf node from the current tree (remaining nodes) and solves the corresponding linear problem.

The node selection function chooses the node with the smaller lower bound so far to be the next node to be examined (and solved). The selected node is then solved by the previously well-explained column generation technique as a relaxed linear problem. The result of this function has three main alternatives.

i. If the resulting solution is infeasible, or the current value of the sub-problem (lb') is larger than the best current objective value of a known integer solution (GUB), then no further examination on this node or its possible descendents will give a better integer solution, and this node is safely dropped from the tree (fathoming).
ii. If the resulting solution of the current sub-problem satisfies the integrality constraints and its objective value (lb') is lower than the best known integer solution (GUB), then this value is set to be the current best known integer solution of the entire tree.
iii. The third alternative takes under consideration the alternative where none of the above cases are met and this node needs further examination. In this case, two new nodes are created according to certain criteria that will be described in detail later, and added to the branch-and-bound tree as the children of the current node. Then, this node is deleted from the remaining nodes of the tree to be examined.

With the end of the above functions, the algorithm chooses the next node to solve and the same procedure is repeated until there are no more nodes to be examined in the branch-and-bound tree.

The two functions mentioned above, that concern the branching strategy and consist of the **variable selection strategy** ($VariableSelection$) and the **partitioning policy** ($NodePartition$) are of major importance, since they identify the variable to be branched and partition the parent node into two nodes. This selection affects strongly the speed of the branch and bound solution.

The variable selection strategy adopted in our case is based on the branching on arcs. This means that the arc (i,j) that will be branched has to be identified. The arc that participates in the current solution with the most fractional part (i.e. the one closest to 0.5) is detected. Every arc $(i,j)_k$ of the routes k that participate in the solution X, takes the corresponding value of the variable x_k. The values of the same arcs are summed up and the arc with the most fractional part is then selected to be branched. For example, in case of two routes $\{[0-1-2-0],[0-1-0]\}$ with $x_1 = 0.5$ and $x_2 = 0.5$, the arc $(0,1)$ takes the value 1 and the rest of the arcs the value 0.5. Consequently, the selected variable for branching is one of the arcs with value 0.5.

```
Algorithm VariableSelection(X, parentnode)
Initialize $nodelist_{nxn} = \emptyset$
for k = 1 to all routes in parentnode do
      for i = 1 to n do
            for j = 1 to n do
                  if $(i,j)$ exists in route $k$ then
                        $nodelist$(i,j) = $nodelist(\mathrm{i},\mathrm{j}) + X_k$
                  end
            end
      end
end
$nodelist$ = $nodelist$ - 0.5
$nodelist$ = abs($nodelist$)
$(arc_{ij})$ = min($nodelist$)
```

Figure 4.2: Variable Selection Policy

Figure 4.2 presents a pseudo-code for the variable selection function. After the selection of the variable, the parent node is divided in two new nodes. This is operated by the partitioning policy function, where an additional constraint is added to each node. Given the selected arc from the variable selection policy, supposing (i,j), the first node contains the inclusion of the selected arc in the solution, while in the second node the same arc is excluded in any case. For the implementation of this policy in the B&B procedure, these restrictions are incorporated to the cost matrix of the sub-problem. The cost matrix of the parent node is transformed according to the restrictions of each node. For the first case, all elements of row i and column j are set to infinite, except the element j of row i and the element i of column j. For the second case, only the element corresponding to (i,j) of the cost matrix is set to infinite. This technique allows us to easily incorporate the new changes in the two nodes, since any movement to j from any other node except i will be infeasible for the first case, and obviously movements from i to j will be infeasible for the second case. The Branch and bound method is summarized in Figure 4.3 below.

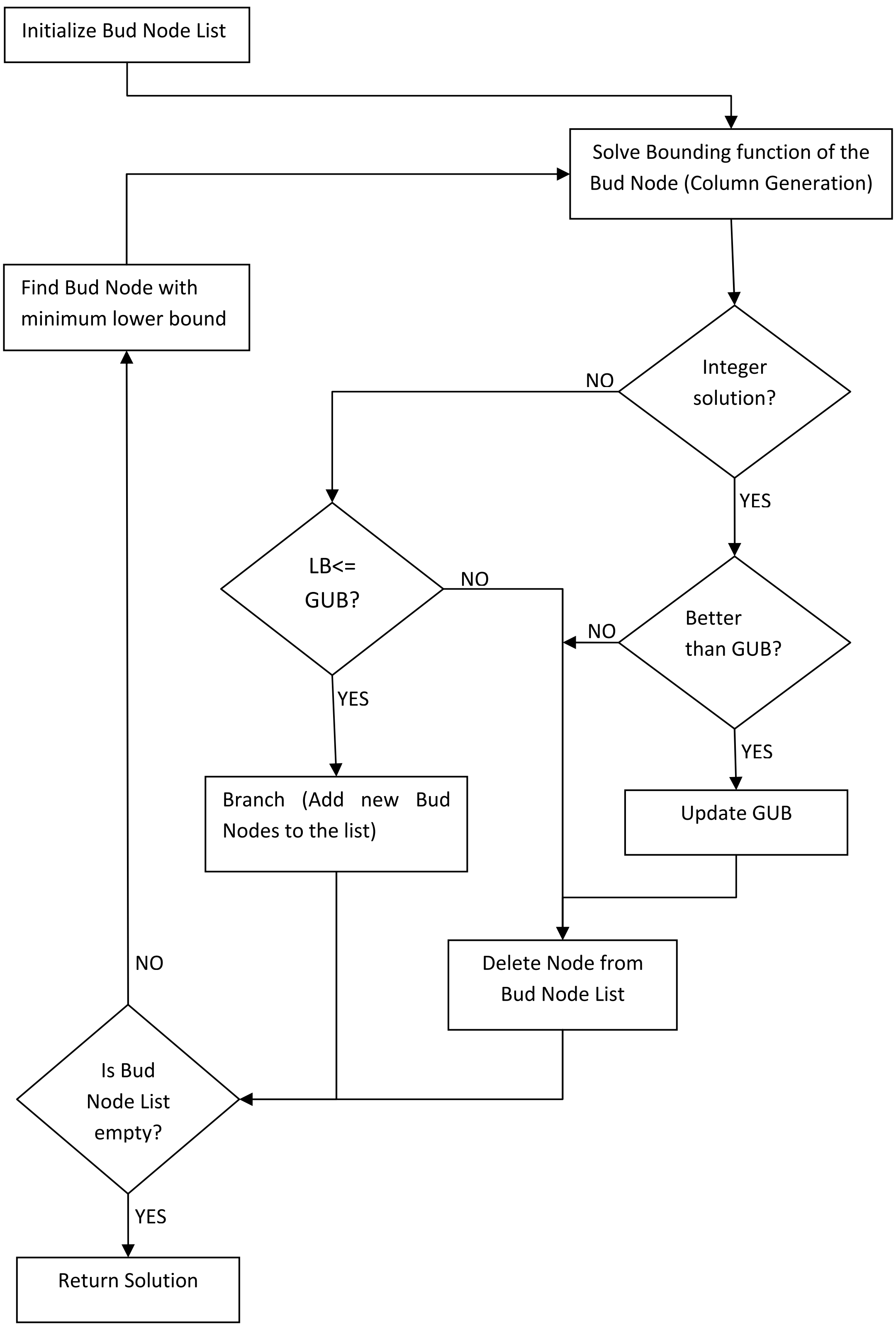

Initialize Bud Node List
Solve Bounding function of the Bud Node (Column Generation)
Find Bud Node with minimum lower bound
Integer solution?
NO
YES
LB<= GUB?
NO
Better than GUB?
NO
YES
YES
Branch (Add new Bud Nodes to the list)
Update GUB
Delete Node from Bud Node List
NO
Is Bud Node List empty?
YES
Return Solution

Figure 4.3: Diagram of the Branch and price algorithm

CHAPTER 5 TEST CASES FOR THE VRPTW

In this chapter the method presented in Chapters 3 and 4 is tested using the well-known Solomon benchmarks. Firstly the Solomon benchmark test cases are presented. Subsequently we examine the limits of the method without using the dominance criteria. In this case the optimal is always determined; however, computational times are expensive preventing the solution of large scale problems. By adding a simplified version of the dominance criteria of Larsen (2001) to the algorithm, it is shown that computational complexity is reduced dramatically, but in many cases, especially in particular node configuration the optimal in not reached. Finally, by adopting the dominance criteria of Chabrier (2005) it is shown that an increased number of cases are solved to optimality, but computational complexity (and times) are increased with respect of the simplified Larsen criteria case.

5.1THE SOLOMON'S BENCHMARK PROBLEMS

The Solomon's problems (Solomon, 1987) are the most commonly used benchmark data for testing heuristics and exact methods in vehicle routing problems. These problems are based on a set of VRPTW problems by Christofides *et al.* (1979), and have been further extended by adding time window and altered capacity constraints. The Solomon are identified as [**LNXX**], where L refers to letters R, C and RC, N refers to numbers 1 and 2, and XX refers to different test instances of the LN problems.

The R, C and RC naming conventions refer to the three following types of customer (nodes) allocation on the $[0,100]^2$ space.

R problems: Problems where customers are uniform and randomly dispersed. An example is presented in Figure 5.1.

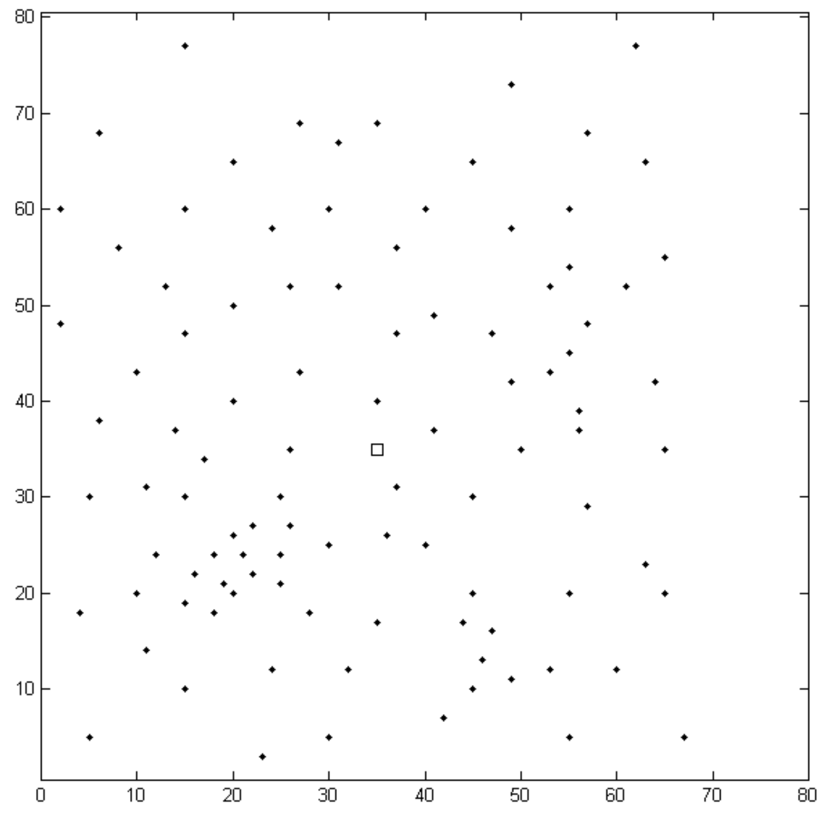


Figure 5.1: The R101 problem with 100 customers. The depot is marked as a square.

C problems: In these problems all customers (nodes) appear strongly clustered, and are, again, dispersed. An example is presented in Figure 5.2.

RC problems: The RC case comprises a hybrid dispersion of customers, with some customers uniformly distributed, and others clustered. An example is presented in Figure 5.3.

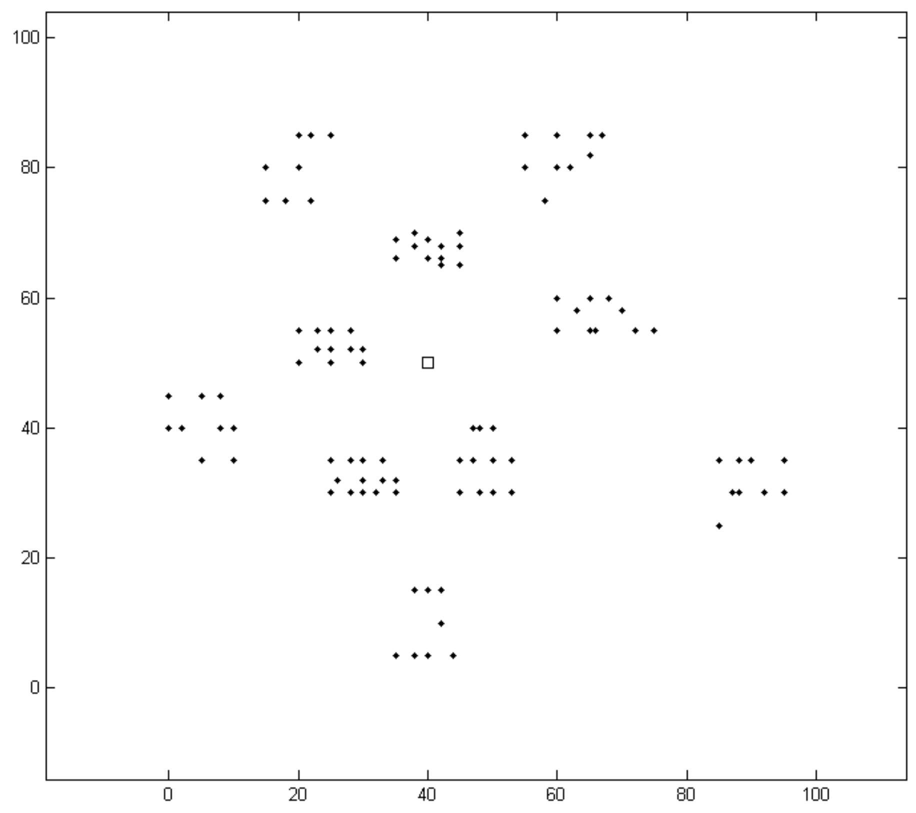


Figure 5.2: The C101 problem with 100 customers. The depot is marked as a square.

The (N = 1 or 2) naming conventions are as follows:

N=1: The capacity of maximum route length constraints allow for up to 5 to 10 customers per route.

N=2: The capacity and maximum route length constraints allow for 30 or more customers to e served by a single route.

Each one of the LN test sets (as described) above incudes 8 to 12 different problem instances, with a total number of customers up to 100 customers. The problems can be found in:

- http://web.cba.neu.edu/~msolomon/problems.htm
- http://www.idsia.ch/~luca/macs-vrptw/problems/welcome.htm

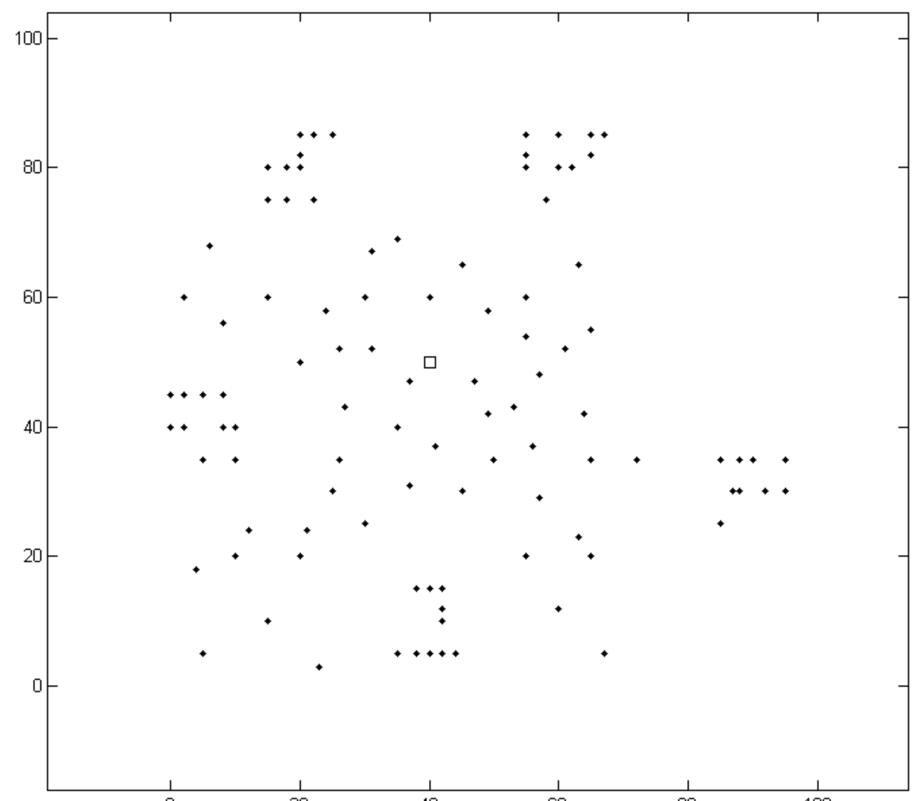


Figure 5.3: The RC101 problem with 100 customers, where the depot is marked as a square.

Distance and Travel Time Matrix Calculations

Since there is not a common approach for calculating the Distance and Travel Time matrices among the academic community, the approach proposed by Kohl (1995), in order to be consistent with the cost matrix calculations and able to compare our results with the results obtained from Larsen (2001) and Kohl (1995) has been used. In this approach, the distance between two nodes, (x_i, y_i) and (x_j, y_j), is calculated as follows:

- Initially, the actual distance is derived from the well-known equation:

$$c_{ij} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} \tag{5.1}$$

- Additionaly, the value 0.1 is being added to each c_{ij}, $i \neq 0, j \neq n+1$, in order to preserve the triangular inequality (see Kohl, 1995)

- Then, by using the following equation c_{ij} is rounded down to 1 demical digit.

$$c_{ij} = \frac{\lfloor 10c_{ij} \rfloor}{10} \tag{5.2}$$

where $\lfloor\cdot\rfloor$ denotes the floor value of the c_{ij}.

- Finally, c_{ij} is multiplied by 10 in order to be represented as integer in the final distance matrix.

Travel times are calculated by adding he service time, st_i of each customer i to the distance c_{ij} of each arc (i, j), and the final travel time of each arc is derived as:

$$t_{ij} = c_{ij} + st_i \tag{5.3}$$

5.2 Description of the Test Bed

The method described in chapter 3 and 4 has been implemented in Matlab® 7 software, which allows multithreading computations. All tests were performed on a 2.00 GHz 8-core PC System running Windows XP®, Note that the parallel use of multiple processors was performed only at the linear programming part of the algorithm (Master Problem) and not on the sub-problem. Some features of the Matlog Toolbox (Kay, 2008) have been used for the implementation of the algorithms. The initial solution given to all test cases was the "unit matrix" solution, where each route contains only one customer (Depot – Customer – Depot). Note that our implementation considers multiple column (with negative reduced cost) insertions in the master problem, in every iteration (each time the sub-problem is called, it generates more than one column which can improve the existing cost). Several tests were performed in order to study the effectiveness and accuracy of the proposed methods.

In section 5.3, the proposed algorithm without using dominance criteria (ND) is compared to an exhaustive search algorithm (ESA) created for this specific purpose (Athanasopoulos, 2008b). The ND method solves the ESPPTWCC without discarding labels that are created in the nodes of the network. By this, all possible paths starting and ending in the depot are explored and the labels and, therefore, paths with non-negative modified cost are discarded only after the finalization of the ESPPTWCC. The computational time and the final cost obtained by the two different methods are compared.

In section 5.4, two versions of the proposed methods are compared: The ND method (see above) and the column generation method with the simplified dominance criteria (SD) [See Chapter 3.2]. The SD method appears to be very fast due to the elimination of several labels. This elimination leaves a significant smaller number of labels to be further explored, but in some cases by sacrificing optimality. Again, the computational time and the final integer cost obtained are compared.

As stated above, the dominance criteria operated in the SD method do not always reach the optimal solution. Section 5.5 presents results for these cases and compares the SD method with a different dominance criteria method (CD). This method is based in a simplified version of the dominance rules presented in Chambier (2005), in which more intelligent but time-consuming dominance criteria are proposed (See Chapter 3.2). Again, the computational time and the final integer cost obtained are compared.

In Section 5.6, the SD algorithm is tested in large integer problems (25 to 100 customers) from the Solomon Benchmarks, and analytical results for each test-case are presented. The results obtained from the SD method are compared to the optimal results from Larsen (2001) and Kohl (1995).

Finally, Section 5.7 summarizes the results obtained from the previous sections using the column generation method with different dominance criteria, and the results obtained from solving large scale (up to 100 customers) instances.

5.3 Results of Method Without Dominance Criteria (ND)

In this section the results of column generation method without dominance criteria (ND) against the results of the exhaustive search algorithm are presented. The exhaustive search procedure is analyzed below.

Description of the Exhaustive algorithm

Scope of the exhaustive search algorithm is to produce optimal solutions to the test-cases of the Solomon benchmarks. Since, the computational complexity of the ESA method is high; it can only provide solutions to small instances. Thus, problems up to 9 customers have been solved. ESA, initially, produces all combinations of nodes that form a route. The infeasible routes are then discarded, leaving all the feasible routes. Then, the remaining feasible routes are transformed in their set partitioning formulation. Finally the problem using LINPROG (the classical LP solver routine of Matlab) is being solved. A brief overview of ESA is as follows:

1. Initially all possible routes from depot 0 to depot $n + 1$ are identified.
2. Feasibility criteria (time window, capacity, max route length) are applied to each route and the infeasible routes are discarded.
3. The cost of all remaining feasible routes is being calculated.
4. The calculated costs and the set partitioning form of the feasible routes construct a linear problem, which is solved to optimality.

Note that due to the size of the problems (up to 9 customers) and the respective time window, capacity and maximum route length constraints of the Solomon test-cases, all problems result in a single route.

Results Obtained

Table 5.1 presents indicative results of computational time and total cost for each algorithm. (For the analytical results, see Appendix B)

- Columns "Cost" present the total cost (achieved without performing B&B operations) for each method

- Columns "Computational Time" present the computational time spend in finding the lower bound

Table 5.1: ESA vs. ND method. Indicative Results.

		ESA method		ND method	
Problem	**Customers**	**Cost**	**Time(sec)**	**Cost**	**Time(sec)**
R101	5,00	115,60	0,02	115,60	0,02
	6,00	156,70	0,10	156,70	0,01
	7,00	157,50	0,53	157,50	0,02
	8,00	195,90	3,56	195,90	0,02
	9,00	217,30	26,78	217,30	0,04
C101	5,00	42,70	0,04	42,70	0,01
	6,00	42,80	0,12	42,80	0,03
	7,00	46,70	0,56	46,70	0,09
	8,00	48,20	3,77	48,20	0,10
	9,00	50,30	26,98	50,30	0,25
RC101	5,00	87,20	0,03	87,20	0,01
	6,00	89,40	0,12	89,40	0,04
	7,00	108,30	0,55	108,30	0,26
	8,00	112,40	3,69	112,40	0,22
	9,00	121,60	27,03	121,60	0,52

In all cases (Appendix B and Table 5.1) the optimal solution is obtained by the Column Generation without dominance criteria (ND). Therefore, the column generation method along with the algorithm for solving the ESPPTWCC without dominance criteria leads to optimal solutions for the test-cases solved. Although the ND method performs considerably better than the ESA method regarding the computational time, it is still very slow and cannot provide solutions to problems of more customers. Figure 5.4 shows the significant difference of the computational time of the column generation against the ESA method for an indicative test-case.

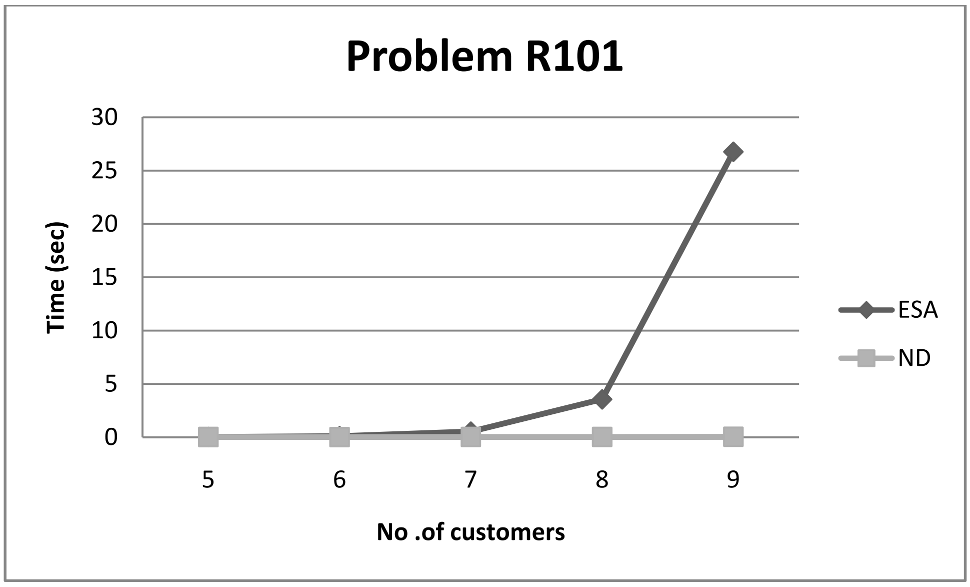


Figure 5.4 Computational time of problem R101, using the methods ESA and ND

5.4 Results of Method with Simplified Dominance Criteria (SD)

This section compares the column generation algorithm using the simplified Larsen dominance rules (SD), described in Section 3.2, against the ND method. Indicative results can be seen in Table 5.2 (for the analytical results, see Appendix B).

Table 5.2: ND vs. SD method. Indicative Results.

		ND Method		SD Method	
Problem	**Customers**	**Cost**	**Time(sec)**	**Cost**	**Time(sec)**
R105	5,00	115,60	0,02	115,60	0,02
	6,00	142,10	0,03	142,10	0,02
	7,00	157,50	0,03	157,50	0,02
	8,00	188,80	0,04	188,80	0,03
	9,00	214,95	0,11	214,95	0,09
C105	5,00	42,70	0,02	42,70	0,01
	6,00	42,80	0,03	42,80	0,03
	7,00	46,70	0,05	46,70	0,03
	8,00	48,20	0,20	48,20	0,07
	9,00	50,30	0,25	50,30	0,06
RC105	5,00	82,80	0,03	82,80	0,03
	6,00	87,70	0,08	87,70	0,06
	7,00	93,50	0,36	93,50	0,15
	8,00	**99,60**	**2,53**	**100,90**	**0,41**
	9,00	109,60	10,33	109,60	0,69

Based on Table 5.2 and the analytical results (Appendix B), several instances solved did not reach the optimal solution (as presented in the ND method results). After 120 tests (see Appendix B) in problems of five to nine customers of all three Solomon test cases, eleven non-optimal solutions identified. As discussed in Chapter 3.2, this behavior relates to the dominance criteria used in SD method, in which several node labels are discarded as dominated. Some of these dominated labels are the labels, which would have led the algorithm to the optimal solution.

On the contrary, the computational time obtained from the SD method in comparison to the ND method is significantly lower. Figure 5.5 presents an indicative example (test-case RC105) where the computational time versus the customers of each instance is presented.

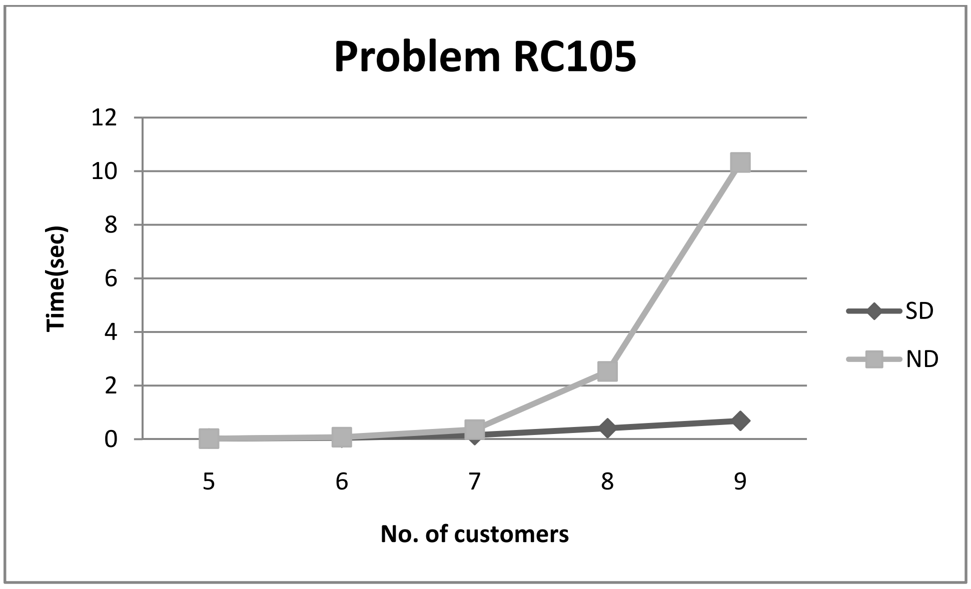


Figure 5.5 Computational time of problem RC105, using the methods SD and ND

5.5 Results of Method With Chabrier Dominance Criteria (CD)

Section 5.5 presents the results obtained by the column generation using the dominance criteria proposed in Chabrier (2005). Note that only the test-cases in which the SD method could not converge to the optimal solution are presented in order to show the improvement of the CD method (for analytical results see Appendix C). Although, CD method leads to optimal solutions, it consumes a considerable amount of computational time.

Table 5.3: CD vs. SD method. Indicative Results.

		SD Method		CD Method	
Problem	**Customers**	**Cost**	**Time(sec)**	**Cost**	**Time(sec)**
R102	**6**	134,6	0,039	131,0	0,092
R103	**6**	134,6	0,039	131,0	0,091
R104	**6**	134,6	0,040	131,0	0,0912
C104	**9**	50,3	0,829	49,3	0,875
RC102	**7**	92,1	0,124	88,7	1,171

		SD Method		CD Method	
Problem	Customers	Cost	Time(sec)	Cost	Time(sec)
RC102	8	94,8	0,333	93,5	0,734
RC103	7	92,1	0,124	88,7	1,157
RC103	8	94,8	0,333	93,5	0,734
RC104	9	100,2	0,509	96,6	1,68

The exact dominance rules proposed by Chabrier (2005) solve this problem optimally but the solution procedure is much slower than the SD method. Chabrier (2005) proposes two methods to accelerate the solution procedure, i) by an exact approach, and ii) a heuristic approach. Several results are presented, in which some instances are solved for the first time.

Since the CD method leads to optimal solutions, the computational time of the method is compared against the computational time of the exhaustive method (ESA). Figure 5.6 presents the computational time of the two methods for different customer instances for an indicative problem (R101). Also, the computational time of ESA method is presented to provide useful comparisons of the three different methods. Analytical results regarding the comparison of the computational time of the SD and CD methods for problems up to 20 customers are presented in Appendix C.

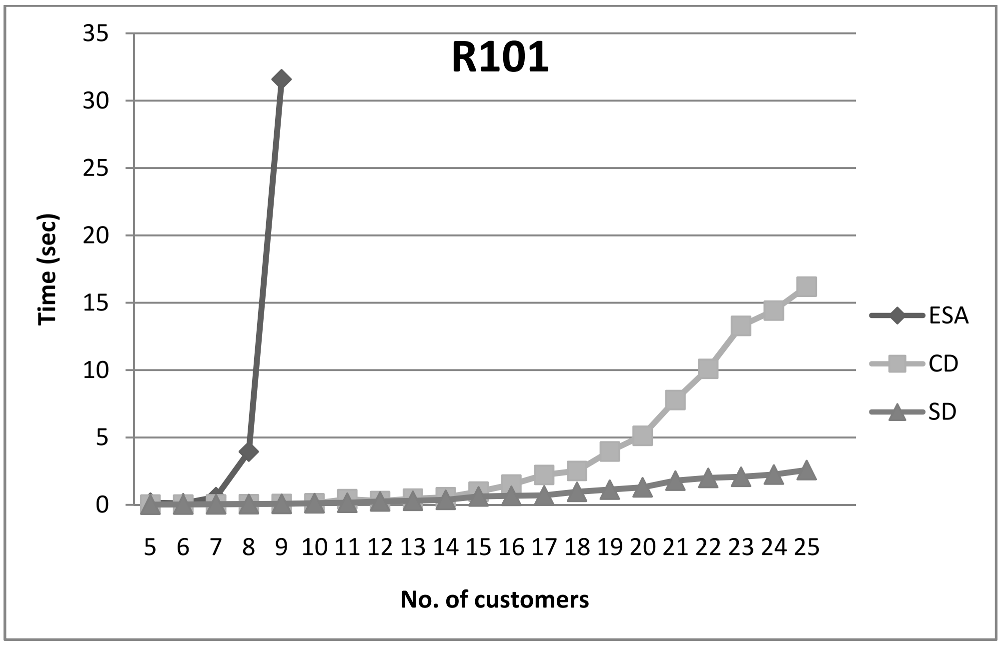


Figure 5.6 Computational Time using methods ESA, CD and SD (R101 instances)

5.6 Testing Large problems

Obtaining solutions to larger problems (more than 10 customers) was feasible only with the SD method. All other methods implemented were unable to handle larger instances due to computational time. In this section, the results obtained using the SD method, for problems up to 100 customers from the Solomon test-cases, are presented. Table 5.4 presents the results of a subset of the aforementioned test-cases. Table 5.4 is structured as follows

- Initially the instances and the number of customers used are presented.
- Columns "Cost" present the lower bound cost obtained (without performing B&B operations) and the final integer cost (with B&B operations). If these two costs are equal, then the integer solution was found without performing B&B operations. Signs (*) indicate cases in which our method (SD) was unable to reach the best integer optimal solution reported in the literature (Larsen, 2001). Note that the (+) sign indicates a different lower bound from the results presented in Larsen (2001).
- Column "No. of Routes" presents the numbers of routes participating in the final integer solution.
- Columns "B&B Tree" shows the total nodes (from the B&B tree) created and the nodes explored by the algorithm. Note that these two columns differ, since based on the *Node Selection Policy,* not all nodes have to be examined.
- Columns "Generated Columns" present the total columns created and transferred to the master problem by the sub-problem at the initial node, resulting in the lower bound (prior entering the B&B operation), and the average number of columns generated in each B&B node explored.
- Columns "Computational Time" present the computational time spent in finding the lower bound, the average time spend per B&B node and the total cumulative time in finding the integer solution, respectively.

Table 5.4: Results of the SD method for 25, 50 and 100 customers

		Cost			B&B Tree		Generated Columns		Computational Time			
Problem	**Customers**	**Lower Bound**	**Integer Solution**	**# of Routes**	**Total Nodes**	**Nodes Explored**	**First Node**	**Average per B&B Node**	**First Node**	**Average per B&B Node**	**Total**	**Optimal Solution**
R101	**25**	617,1	617,1	8	0	0	400	0	0,47	0	0,47	617,1
	50	1044^{+}	1044	12	0	0	1764	0	4,73	0	4,73	1044
	100	1631,15	1637,7	20	24	16	7648	203	170,21	34,93	729,33	1637,7
R102	**25**	546,33	547,1	7	8	4	1906	55	2,94	0,65	5,56	547,1
	50	909,00	909	11	0	0	7586	0	61,58	0	61,58	909
	100	1467,7^{+}	1467,7*	18	0	0	13698		4003,58		4003,87	1466,6
R103	**25**	454,60	454,6	5	0	0	3656	0	11,00	0	10,80	454,6
	50	769,29	772,9	9	18	10	15014	435	178,36	72,03	898,74	772,9
R104	**25**	416,90	416,9	4	0	0	5918	0	25,13	0	25,13	416,9
R105	**25**	530,5	530,5	6	0	0	952	0	1,45	0	1,49	530,5
	50	898,47^{+}	911,8*	9	76	40	3562	334	12,88	4,02	173,81	899,3
R106	**25**	457,3	465,4	5	12	8	2742	125	7,89	1,53	19,62	465,4
R107	**25**	424,3^{+}	424,3	4	0	0	4636	0	19,87	0	19,87	424,3
	50	707,61^{+}	711,6*	7	28	16	15306	188	218,54	75,09	1420,19	711,1
R108	**25**	397.27^{+}	397,3	4	4	2	6810	130	48,14	6,3	55,90	397,3
R109	**25**	441,3	441,3	5	0	0	1792	0	3,41	0	3,40	441,3
	50	779,2^{+}	790,7*	8	818	426	7524	280	112,79	0,44	3709,45	786,8
R110	**25**	440,45^{+}	444,1	5	22	12	2994	95	9,50	1,6	28,62	444,1
R111	**25**	427,36^{+}	428,8	4	8	4	3282	130	12,20	2,5	21,18	428,8
R112	**25**	389,45^{+}	393	4	24	12	4660	95	24,40	3,1	66,75	393

Problem	Customers	Cost		# of Routes	B&B Tree		Generated Columns		Computational Time			Optimal Solution
		Lower Bound	Integer Solution		Total Nodes	Nodes Explored	First Node	Average per B&B Node	First Node	Average per B&B Node	Total	
RC101	**25**	413,6^{+}	466,5*	4	434	272	962	23	1,83	0,37	102,77	461,1
RC102	**25**	351,8	351,8	3	0	0	2164	0	6,30	0	6,30	351,8
RC103	**25**	334,2^{+}	334,2*	3	0	0	3554	0	8,36	0	8,36	333,1
RC104	**25**	311,7^{+}	311,7*	3	0	0	4196	0	18,98	0	18,99	306,6
RC105	**25**	419^{+}	419*	4	0	0	1418	0	4,88	0	4,88	411,3
RC106	**25**	345,5^{+}	345,5	3	0	0	1934	0	5,87	0	5,88	345,5
RC107	**25**	302,7^{+}	302,7*	3	0	0	3354	0	18,72	0	18,72	298,3
RC108	**25**	304,6^{+}	304,6*	3	0	0	5010	0	18,46	0	18,45	294,4
C101	**25**	191,3	191,3	3	0	0	2580	0	4,93	0	4,94	191,3z
	50	362,4	362,4	5	0	0	6154	0	31,26	0	31,25	362,4
C102	**25**	191,1	191,1*	3	0	0	10382	0	41,48	0	41,48	190,3
	50	362,4^{+}	362,4*	5	0	0	23570	0	198,99	0	199,00	361,4
C103	**25**	190,3	190,3	3	0	0	13968	0	85,56	0	85,56	190,3
	50	361,4	361,4	5	0	0	42144	0	764,69	0	764,75	361,4
C104	**25**	186,9	186,9	3	0	0	17530	0	134,43	0	134,45	186,9
	50	360,8^{+}	360,8*	5	0	0	61212	0	2879,15	0	2879,36	358,0
C105	**25**	191,3	191,3	3	0	0	2806	0	5,94	0	5,94	191,3
	50	362,4	362,4	5	0	0	8576	0	49,17	0	49,19	362,4
C106	**25**	191,3	191,3	3	0	0	2456	0	6,43	0	6,42	191,3
	50	362,4	362,4	5	0	0	8058	0	50,34	0	50,36	362,4
C107	**25**	191,3	191,3	3	0	0	4114	0	11,51	0	11,52	191,3
	50	362,4	362,4	5	0	0	11604	0	117,84	0	117,86	362,4
C108	**25**	191,3	191,3	3	0	0	6018	0	21,40	0	21,41	191,3
	50	362,4	362,4	5	0	0	16572	0	151,71	0	151,72	362,4

The majority of the computational time is being spent in the first node (obtaining the lower bound). This is shown in Table 5.5, where indicative analytical results for the R106 - obtaining the lower bound are shown, presenting the total number of generated columns, and the computational time spend for the master problem and the sub-problem, respectively. Finally the aforementioned information for each B&B node is presented. Note that, in the B&B nodes, the results are cumulative and are not presented per iteration. The best integer solution obtained is highlighted in the table.

As it is observed, the initial given solution, which is the worst integer solution, forces the algorithm to create a large number of columns, which is reduced drastically in the next iterations. Using a heurist algorithm to obtain a good initial solution can eliminate this phenomenon. Figure 5.7 presents an illustration of the cost obtained after each iteration for an indicative case (R106 with 25 customers).

Table 5.5: Computational Time for R106 with 25 Customers.

		Computational Time (sec)			**Columns**	
	Iteration	**Sub-problem**	**Master Problem**	**Total**	**Generated**	**Total Cost**
Initial Solution	0	-	0,01	0,01	0	1244,60
First Node. Lower Bound Iterations	1	30.10	0,72	**30,10**	2248	485,15
	2	0,57	0,06	0,63	262	468,90
	3	0,52	0,06	0,57	88	457,30
	4	0,43	0,01	0,44	8	457,30
	5	0,42	0,01	0,43	4	457,30
	6	0,45	0,01	0,45	4	457,30
	7	0,43	0,01	0,44	2	457,30
	8	0,44	0,01	0,45	2	457,30
	9	0,40	-	0,40	0	457,30
B&B Nodes	1	1,46	0,12	1,58	48	464,85
	2	1,73	0,16	1,89	48	459,95
	3	1,05	0,13	1,18	66	469,00
	4	1,89	0,14	2,03	22	464,10
	5	0,86	0,12	0,98	28	465,60
	6	**2,03**	**0,14**	**2,17**	**20**	**465,40**
	7	0,84	0,10	0,94	64	468,00
	8	0,73	0,11	0,84	12	476,40
	Total	**44,29**	**1,87**	**45,52**	**2926**	

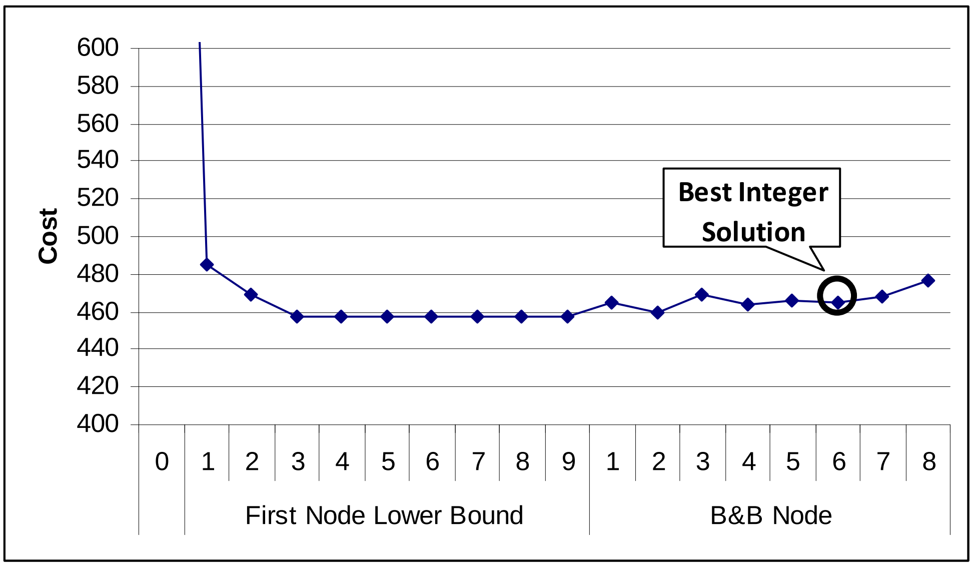


Figure 5.7: Cost of R106 (25 customers)

Computational complexity increases with the number of customers. Figure 5.8 presents an indicative test-case. For this test-case, the computational time spent in obtaining the lower bound by solving the linear program and the ESPPTWCC, along with the total cumulative time is presented. It is clear that the increase is exponential.

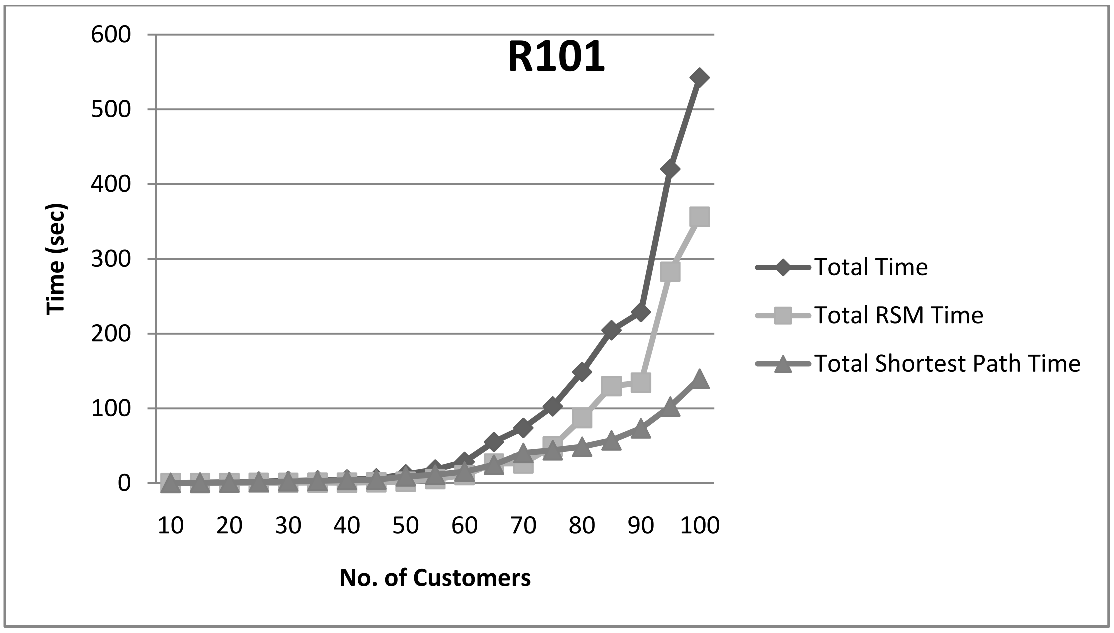


Figure 5.8: Computational time for problem R101 for customers 10 to 100 (SD method)

Multiple Columns Generation vs. Single Column Generation

In order to strengthen the use of multiple columns generation from the sub-problem against the generation of just a single column the following table presents indicative results for the R101 to R107 test-cases by generating just a single column per sub-problem iteration. The results obtained along with the difference in computational time with the results of Table 5.4 are presented. The increase of the computational time is tremendous in this case, since the single column generation forces the algorithm to perform a large number of iterations (by solving the master problem and the sub-problem repeatedly) in order to obtain the lower bound and the integer solution.

Table 5.6: Results with single column generation and difference in computational times against multiple column generation.

Problem	Customers	B&B Tree		Generated Columns		Computational Time			
		Total Nodes	Nodes Explored	Initial Node	Average/ B&B node	First Node	Average per B&B Node	Total	Difference (%)
R101	**25**	0	0	45	-	3,91	-	3,91	731,28
R102	**25**	4	4	75	7,25	35,80	2,65	46,41	734,64
R103	**25**	4	2	81	15,50	99,29	10,78	120,84	1018,93
R104	**25**	0	0	98	-	261,36	-	261,38	940,09
R105	**25**	0	0	65	-	14,04	-	14,05	842,75
R106	**25**	32	18	86	11,39	75,90	5,04	166,63	749,26
R107	**25**	0	0	82	-	143,13	-	143,14	620,38

5.7 CONCLUSIONS

Three different methods for dominance criteria were presented in Chapter 5. All methods use column Generation as the main solution framework. From the results obtained in the previous sections the following can be concluded:

- Although, the SD method is a fast method, with the ability to solve large problem instances (up to 100 customers), it cannot always converge to the optimal solution reported in the literature. In a total of 44 problems with 25, 50 and 100 customers solved, only 13 did not converged to an optimal solution. This is mainly based in the dominance criteria structure which eliminate several labels (therefore, possible routes) from the solution of the ESSPPTW.

- The CD method performs better than the SD method since it achieves optimal solutions to instances, for which the SD method was unable to converge. On the other hand, the computational time consumed, even for small instances, was high and larger instances (more than 25 customers) could not be solved.

- Both methods were able to solve larger instances from the exhaustive search method (ESA) created.
- In most cases, the larger portion of the computational time was consumed in the parent node in order to find the lower bound of the problem. This observation can be strengthened by the use of a dummy initial solution used by our methods. The use of a better initial solution (through a heuristic algorithm) could have minimized the computational time of the parent node.
- Multiple columns against single column insertion dramatically reduced the computational time of the process (more than 500%).
- Another observation made is that in problems with a large number of B&B nodes created (and explored), the cost difference between the lower bound and the integer solution is usually larger than in the instances with less B&B nodes, and the computational time higher, due to the number of B&B nodes that should be explored.

CHAPTER 6 Conclusions

In the present thesis, a Branch and Price via Column Generation toolkit for solving the Vehicle Routing Problem with Time Windows (VRPTW) was developed. The methods employed in this toolkit are able to provide optimal or efficient solutions to the aforementioned problem. Specifically, the toolkit employs advanced linear and integer programming (such as problem decomposition, column generation and branch and bound) and advanced network flow (shortest path problem with additional constraints) techniques. The overall method is based on Larsen (2001), where a similar algorithm based on Shortest Path Problem with Time Windows and Capacity Constraints (SPPTWCC), was implemented, and Chabrier (2005), where the solution is provided through the elementary SPPTWCC. Column Generation is considered as one of the most promising methods to provide exact solutions to VRPTW problems in reasonable computational time.

Method Description

The problem is decomposed into two (2) sub-problems: (a) The Master Problem deals with the solution of a set partitioning problem where the routes to be included in the final solution are selected. The only constraints that the Master Problem considers are the allowance of each customer to be routed in only one route; (b) the sub-problem works on the decomposed problem structure by providing the additional routes to be included in the Master Problem. The two problems can be considered as two different parts, which have to come in a common decision. The sub-problem generates feasible routes and the Master problem decides which routes will be included in the final solution and additionally provides the shadow prices to the sub-problem. The shadow prices can be considered as the weights of the current solution upon all the other undefined solutions to be provided by the sub-problem.

The sub-problem structure and solution implementation can be of any kind (linear programming, dynamic programming, heuristics) and restrictions of all kinds can be included making the whole method easy to be adjusted in the variations of the VRP and problems of other fields. This characteristic provides a strong advantage of the column generation method; that is, Column Generation easily adapts to several different problems.

Analysis and Results

Since the VRPTW is an NP-hard problem, solution for large instances cannot be obtained in rational computational time. This difficulty was tackled through two different label dominance techniques, which applied to the sub-problem (Elementary Shortest Path Problem with Time Windows and Capacity Constraints). The first one is a simplified version of the dominance rules presented in Larsen (2001) for the non-elementary SPPTWCC, where several “bad” labels are discarded based on the dominance criteria before they are further extended. This technique reduces effectively the number of total generated labels and therefore the complexity and computational time. The second one is an extension of the latter based on Chabrier (2005), where an additional rule is introduced compatible with the elementary SPPTWCC. By this, several labels previously regarded as “bad” and discarded are now examined. The latter leads to exact solutions but with considerable computational time increase. The aforementioned techniques are compared against the same column generation method without dominance criteria and an exhaustive search algorithm.

The related experiments were tested using the well-known Solomon test-cases, which consist of the primal benchmarks for the academic researchers in the routing area. In all test-cases solved, the results obtained were compared against optimal solutions provided in the academic literature, or against the results from an exhaustive search algorithm (of course for small scale problems). The analysis of dominance rules showed that using no dominance (ND) or the CD dominance method may result in the optimal solution but only for small test instances. On the other hand, the SD method was able to solve problems up to 100 customers but it could not always converge to the optimal solution provided in the literature. Specifically:

- Using the SD method, 9 out of 120 tests (up to 9 customers) conducted were not able to find the optimal solution; on the other hand the computational time was 60% less than the CD method, and 79% less than the ND method (for the 8 and 9 customers test-cases).
- The ND and CD methods were able to converge to the optimal solutions for small test-cases up to 10 and 25 customers, respectively.
- In large scale instances, the SD method converged to the optimal integer solution for 31 out of 44 tests (up to 100 customers).
- The use of multiple columns generation from the sub-problem achieved a reduction of more than 600% of the computational time.

From the experiments conducted, it is clear that the dominance criteria used affect strongly both the computational time of the algorithm, as well as the solution quality. The dominance criteria are strongly related to the sub-problem structure, that is, the elementary or the non-elementary sub-problem can perform differently using different dominance criteria.

Future Research

The main scope of developing a column generation toolkit was to utilize it in the academic research conducted by the Design, Operations & Production Systems Lab (DeOPSys) of the Financial and Management Engineering (FME) Department of the University of the Aegean, where over the last years many heuristic algorithms for problems of the VRP class have been developed.

This toolkit will be used to produce benchmark solutions for practical VRP class problems that will be studied in DeOPSys and will validate the quality of the algorithms to be produced. Two classes of such problems currently under investigation concern multi-period problems, (Athanasopoulos and Minis 2008), and dynamic vehicle routing problems (Ninikas et al., 2007),

Future work on the toolkit includes further strengthening by adding features, such as acceleration techniques and different branching and bounding strategies. Further investigation regarding the dominance criteria should be also conducted due to their importance regarding the computational time and solution quality in different sub-problem structures.

REFERENCES

Ahn, B.H., Shin, L., (1991), “Vehicle routing with time windows and time varying congestions”, *Journal of the Operation Research Society,* 42(5), 393-400.

Applegate, D., Bixby, R.E., Cook, W., Cox, A., Lee, E. K., (1995), “Parallel Mixed Integer Programming”, Technical Report CRPC-TR95554, Center for Research on Parallel Computation, Rice University, Houston, Texas.

Assad A.A., (1998), “Modelling and implementation issues”, In: Golden, B.L, Assad, A.A. (eds.), “Vehicles Routing: Methods and Studies”. North-Holland, Amsterdam.

Athanasopoulos, T., Minis, I., (2008) “Dealing with Uncertainty and Dynamics in Hybrid Courier Operations through Rolling Horizon Planning”, International Journal of Production Economics (submitted to journal).

Athanasopoulos, T., (2008b), “Column Generation Method for a Multi-Period Routing Problem”, working paper.

Atkinson L.B., (1994), “A greedy, look-ahead heuristic for combinatorial optimization: An application to vehicle scheduling with time windows”, *Journal of the Operation Research Society,* 45(6), 673-684.

Balinski, M., Quandt R., (1964), “On an integer program for a delivery problem”, *Operations Research*, 12, 300-304.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., Vance, P. H. (1998), “Branch-and-Price: Column Generation for Solving Huge Integer Programs”, *Operations Research*, 46(3), 316-329.

Beale, E.M.L., Tomlin, J.A., (1970), “Special Facilities in a General Mathematical Programming System for Non-convex Problems Using Ordered Sets of Variables”, In: Lawerence, J. (ed.), *“Proceedings of the Fifth International Conference on Operations Research”, Tavistock Publications*, 447-454.

Benichou, M., Gauthier, J.M., Girodet, P., Hehntges, G., Ribiere, G., Vincent, O., (1971), “Experiments in Mixed Integer Linear Programming”, *Mathematical Programming,* 1, 76-94.

Bianco, L., Mingozzi, A., Ricciardelli, S., (1994), “A set partitioning approach to the multiple depot vehicle scheduling problem”, *Optimization Methods and Software*, 3, 163–194.

Belman, R., (1958),”On a routing problem”, *Quarterly Applied Mathematics,* 16(1), 87-90.

Bradley, S.P., Hax, A.C., Magnanti, T.L., (1977), Applied Mathematical Programming, Addison-Wesley Publishing Company, Reading, MA.

Carpaneto, G., Ammico, M. D., Fischetti, M., Toth, P., (1989), “A branch and bound algorithm for the multiple depot vehicle problem”, *Networks,* 19(5), 531 – 548.

Chabrier, A., (2005), "Vehicle Routing Problem with elementary shortest path based column generation", *Computers & Operations Research,* 33, 2972–2990, 2006.

Chinneck, J.W., (2003), "Practical Optimization: A Gentle Introduction", Department Systems and Computer Engineering, Carleton University. Available from < http://www.sce.carleton.ca/faculty/chinneck/po.html>.

Christofides, N., (1979), "Traveling salesman problem", In; Christofides, N., Mingozzi, R., Toth, P., Sandi, C.., (eds.), Combinatorial Optimization, Wiley, New York.

Christofides, N., Beasley, J.E., (1984), "The period routing problem", *Networks,* 14, 237-256.

Christofides, N., Mingozzi, R., Toth, P., (1981), " Exact algorithms for the vehicle routing problem, based on spanning trees and shortest path relaxation", *Mathematical Programming*, 20(3), 255-282.

Clarke, G., Write, J., (1964), "Scheduling of vehicles from a central depot to a number of delivery points", *Operations Research,* 12(4), 568-581.

Clausen, J., (1999), "Branch and Bound Algorithms- Principles and Examples", Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK- 2100 Copenhagen, Denmark.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., (1990*), Introduction to algorithms*, MIT Press.

Cornuejols, G., Nemhauser, G.L., (1978), "Tight bounds for Christofides traveling salesman heuristic", *Mathematical Programming*, 14(1), 116-121.

Daganzo, C.E., Hall, R.W., (1993), "A routing problem with pickups and deliveries: No capacity restrictions on the secondary items", *Transportation Science,* 27, 315-329.

Dakin, R.J. (1965), "A Tree Search Algorithm for Mixed Integer Programming Problems", *Computer Journal*, 8, 250-255.

Danna, E., Pape, C. (2005), "Branch-and-Price heuristics: A case study on the vehicle routing problem with time windows", In: Desaulniers, G., Desrosiers, J., Solomon, M., "*Column Generation"*, Springer Science and Business Media, Inc.

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

Dantzig, G.B, Wolfe, P., (1960), "Decomposition principle for linear programs", *Operations research*, 8 (1), 101-111.

Desaulniers, G., Desrosiers, J., Solomon, M., (1991), "A new optimization algorithm for the vehicle routing problem with time windows", *Operations Research,* 40(2), 342-354.

Desaulniers, G., Desrosiers, J., Solomon, M., (1992), “A new optimization algorithm for the vehicle routing problem with time windows”. *Operations Research,* 40(2), 342-354.

Desaulniers, G., Desrosiers, J., Solomon, M., (2005), *Column Generation*, by Springer Science and Business Media, Inc..

Desrosiers, J., Dumas, Y., Solomon, M., Soumis, F., (1995), “Time constrained routing and scheduling”, In: Ball, M. O., Magnanti, T. L., Monma, C. L., Nemhauser, G. L., (eds.), “*Network Routing*”*,* Handbooks in Operations Research and Management Science*,* 8, North-Holland, Amsterdam, The Netherlands, 35–139.

Dijkstra, E.W, (1959): “ A note on two problems in Connexion with Graphs”, Numerische Mathematik, 1, 269-271.

Driebeck, N.J., (1966), “An Algorithm for the Solution of Mixed Integer Programming Problems”, *Management Science,* 21, 576-587.

Dumas, Y., Desrosiers, J., (1986), “A shortest path problem for vehicle routing with pick-up, delivery, and time windows”, *Les Cahiers du GERAD*, no. G-86-09.

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

Fisher, M.L., (1985), “An Application Oriented Guide to Lagrangean Relaxation”, *Interfaces,* 15, 10-21.

Ford, L.R., Fulkerson D.R. (1962), Flows in Networks, Princeton University Press.

Gelinas, S., Desrochers, M., Desrosiers, J., Solomon, M., (1995), “A new branching strategy for time constrained routing problem with applications to backhauling”, *Annals of operations Research*, 61, 91-109.

Gendreau, M., Laporte, G., Hertz, A., (1997), “An Approximation Algorithm for the Traveling Salesman Problem with Backhauls”, *Operation Research,* 45(4), 639-641.

Geofrion, A.M., (1974), “Lagrangean Relaxation and Its Uses in Linear Programming”, *Mathematical Programming”*, 2, 329-344.

Gilmore, P.C., Gomory, R.E., (1961), “A Linear Programming Approach to the Cutting Stock Problem”, *Operations Research*, 9(6), 849-859.

Gilmore, P.C., Gomory, R.E., (1963), “A Linear Programming Approach to the Cutting Stock Problem-Part II”, *Operations Research*, 11(6), 863-888.

Houck, Jr. D.J., Piscard, J.C, Queyranne, M., Vemuganti, R.R., (1980), “The Traveling Salesman Problem as a Constraint Shortest Path Problem: theory and computational Experience”, *Opsearch*, 17, 93-109.

Huisman , D., Jans, R., Peeters, M. Wagelmans A., (2005), “Combining Column Generation and Lagrangian Relaxation”, In: Desaulniers G., Desrosiers J., Solomon M. (eds.), "*Column Generation*", Springer Science and Business Media, Inc.

Irnich, S., Villeneuve, D., (2003), "The shortest path problem with resource constraints and *k*-cycles elimination for $k \geq 3$", Technical Report G-2003-55, GERAD, Montreal, Canada.

Johnson, E.L., (1989), "Modeling and strong linear programs for mixed integer programming", In: Wallace, S.W. (eds.), "*Algorithms and Model Formulations in Mathematical Programming*", NATO ASI Series 51, 1-41.

Kallehauge, B., Larsen, J., Madsen, B.G., Solomon M., (2005), "Vehicle Routing Problem with Time Windows", In: Desaulniers G., Desrosiers J., Solomon M. (eds.), "*Column Generation*", Springer Science and Business Media, Inc.

Kay, M.G., (2008), "Matlog: Logistics Engineering Matlab Toolbox" Version 10, 17-Oct-2008, Department of Industrial and Systems Engineering, North Carolina University. Available from: <http://www.ise.ncsu.edu/kay/matlog>

Kohl, N., (1995), "Exact Methods for the Time Constrained Routing and Related Scheduling Problems", PhD thesis (IMM-PHD-1995-16), Department of Mathematical Modeling, Technical University of Denmark.

Land, A.H., Doig, A.G., (1960), "An Automatic Method for Solving Discrete Programming Problems", *Econometrica,* 8, 497-520.

Land, A.H., Powell, S., (1979), "Computer codes for problems of integer programming", *Annals of Discrete Mathematics,* 5, 221-269.

Laporte G. and Osman I.H., (1996), "Routing problems: A bibliography", *Annals of Operation Research,* 61(1), 227-262.

Larsen, J. (2001), "Parallelization of the Vehicle Routing Problem with Time Windows", PhD thesis (IMM-PHD-2001-62), Department of Mathematical Modeling, Technical University of Denmark.

Lawler E. L, Wood, D. E., (1966), "Branch-And-Bound Methods: A Survey" *Operations Research*, 14(4), 699-719.

Lee, E. K., Mitchell, J. E. (2001), "Integer Programming: Branch-and-Bound Methods", in: *Encyclopedia of Optimization*, Volume II, 509-519, Kluwer Academic Publishers.

Lubbecke M.E., Desrosiers, J., (2005), "Selected Topics in Column Generation", *Operations Research,* 53(6), 1007–1023.

Luenberger D.G., (1989), Linear and Nonlinear Programming. , Addison-Wesley, Reading, MA.

Morgan S., (1997), "A Comparison of Simplex Method Algorithms", Thesis, Department of Computer and Information Science and Engineering, University of Florida. Available from:<http://www.cise.ufl.edu/~davis/Morgan/abstract.htm>

Ninikas, G., Athanasopoulos, T., Marentakis, H., Zeimpekis, V., Minis, I., (2008) "Development of an Integrated Fleet Management System for Real-Time Courier

Services: Design and Implementation", In: *Proceedings of the 20th National Conference of the Hellenic Society of Operational Research,* Spetses, Greece, June 19-21, 1111-1123

Reimann, M., Doerner, K., Hartl, R.F., (2003), "D-Ants: Savings Based Ants divided and conquer the vehicle problem", *Computers & Operations Research*, 31(4), 563-591.

Ropke, S., (2005), "Heuristics and exact algorithms for the vehicle routing problems" PhD thesis, Department of Computer Science at the University of Copenhagen (DIKU), Technical University of Denmark.

Soumis, F., (1997), "Decomposition and column generation", In: Amico, M.D., Maffioli, F., Martello, S. (eds.), "*Annotated Bibliographies in Combinatorial Optimization",* John Wiley and Sons, Chichester, UK, 115–126.

Stewart, W.R, Golden, B.L. (1983), "Stochastic Vehicle Routing: A comprehensive approach", *European Journal of Operation Research,* 14, 371-385.

Tailard, E.D., (1996), "A Heuristic Column Generation Method for the Heterogeneous Fleet VRP", *RAIRO,* 33, 1-14.

Tan, C.C.R., Beasly, J.E., (1984), "A heuristic algorithm for the period vehicle routing problem", *Omega*, 12(5), 497-504.

Tatarakis, A., (2007), "A Class of Single Routing Problems with Predefined Customer Sequence and Depot Returns", PhD Dissertation, Department of Financial and Management Engineering, School of Business, University of the Aegean.

Tomlin, J.A., (1971), "An Improved Branch and Bound Method for Integer Programming", *Operations Research,* 19, 1070-1075.

Toth, P.,Vigo, D., (2002), "The vehicle routing problem", SIAM, Philadelphia.

Wilhelm, W. E., (2001), "A technical review of column generation in integer programming", *Optim. and Engrg.,* 2, 159–200.

APPENDICES

APPENDIX A: EXAMPLES

REVISED SIMPLEX EXAMPLE

$$\begin{array}{l} \max Z = 2x_1 + 5x_2 \\ 1x_1 \quad + 3x_2 \quad \leq \quad 12 \\ 1x_1 \quad + 1x_2 \quad \leq \quad 6 \\ 2x_1 \quad + 1x_2 \quad \leq \quad 10 \end{array}$$

Initially we have:

$$A = \begin{bmatrix} 1 & 3 \\ 1 & 1 \\ 2 & 1 \end{bmatrix} \; b = \begin{bmatrix} 12 \\ 6 \\ 10 \end{bmatrix} \; c = [2 \quad 5]$$

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \; c_B = [0\;0\;0]$$

Solving the problem with Revised Simplex Method
Iteration 1st
Step 1st – search for the variable to enter the basis

$$\Upsilon = c_\beta B^{-1} = [0 \quad 0 \quad 0] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [0 \quad 0 \quad 0]$$

$$z_j - c_j = YP_j - c_j$$

$$x_1 \rightarrow \; z_1 - c_1 = YP_1 - c_1 = [0 \quad 0 \quad 0] \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} - 2 = -2$$

$$x_2 \rightarrow \; z_2 - c_2 = YP_2 - c_2 = [0 \quad 0 \quad 0] \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} - 5 = -5$$

$$r = \{-2, -5\}$$

We choose the most negative element of r, so X_2 will enter the basis.

Step 2nd – search for the variable to exit the basis

$$\chi_B = B^{-1}b = \begin{bmatrix} 12 \\ 6 \\ 10 \end{bmatrix}$$

$$a^j = B^{-1}P_j = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix}$$

$$U = \min \left\{ \frac{\chi_B}{a_k^j}, a_k^j > 0 \right\} = \min\{4, 6, 10\} = 4$$

So the variable to exit the basis isS_1.

Step 3rd – calculate the new basis

$$B = \begin{bmatrix} 3 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, so\ B^{-1} = \begin{bmatrix} 1/3 & 0 & 0 \\ -1/3 & 1 & 0 \\ -1/3 & 0 & 1 \end{bmatrix}$$

$$\chi_B = \begin{array}{c} \chi_2 \rightarrow \\ S_2 \rightarrow \\ S_3 \rightarrow \end{array} \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} and\ c_B = [5 \quad 0 \quad 0]$$

End of first iteration, go back to step 1st.

Iteration 2nd
Step 1st – search for the variable to enter the basis

$$\Upsilon = c_\beta B^{-1} = [5 \quad 0 \quad 0] \begin{bmatrix} 1/3 & 0 & 0 \\ -1/3 & 1 & 0 \\ -1/3 & 0 & 1 \end{bmatrix} = [5/3 \quad 0 \quad 0]$$

$$z_j - c_j = YP_j - c_j$$

$$x_1 \rightarrow z_1 - c_1 = YP_1 - c_1 = [05/3 \quad 0 \quad 0] \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} - 2 = -1/3$$

$$x_3 \rightarrow z_3 - c_3 = YP_3 - c_3 = [5/3 \quad 0 \quad 0] \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} - 0 = 5/3$$

$$r = \{-\frac{1}{3}, \quad \frac{5}{3}\}$$

We choose the most negative element of r, so X1 will enter the basis.

Step 2nd – search for the variable to exit the basis

$$\chi_B = B^{-1}b = \begin{bmatrix} 4 \\ 2 \\ 6 \end{bmatrix}$$

$$a^j = B^{-1}P_j = \begin{bmatrix} 1/3 & 0 & 0 \\ -1/3 & 1 & 0 \\ -1/3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1/3 \\ 2/3 \\ 5/3 \end{bmatrix}$$

$$U = \min \left\{ \frac{\chi_B}{a_k^j}, a_k^j > 0 \right\} = \min\{12, 3, 3.6\} = 3$$

So the variable to exit the basis isS_2.

Step 3rd – calculate the new basis

$$B = \begin{bmatrix} 3 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 2 & 1 \end{bmatrix}, so\ B^{-1} = \begin{bmatrix} 1/2 & -1/2 & 0 \\ -1/2 & 3/2 & 0 \\ -1/2 & 5/2 & 1 \end{bmatrix}$$

$$\chi_B = \begin{array}{r} \chi_2 \rightarrow \\ x_1 \rightarrow \\ S_3 \rightarrow \end{array} \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix}\ and\ c_B = [5 \quad 2 \quad 0]$$

End of second iteration, go back to step 1st.

Iteration 3rd

Step 1st – search for the variable to enter the basis

$$\Upsilon = c_\beta B^{-1} = [5 \quad 2 \quad 0] \begin{bmatrix} 1/2 & -1/2 & 0 \\ -1/2 & 3/2 & 0 \\ -1/2 & 5/2 & 1 \end{bmatrix} = [3/2 \quad 1/2 \quad 0]$$

$$z_j - c_j = YP_j - c_j$$

$$S_2 \rightarrow z_4 - c_4 = YP_4 - c_4 = [3/2 \quad 1/2 \quad 0] \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} - 0 = 1/2$$

$$S_1 \rightarrow z_3 - c_3 = YP_3 - c_3 = [3/2 \quad 1/2 \quad 0] \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} - 0 = 3/2$$

$$r = \{\frac{1}{2}, \frac{3}{2}\}$$

All elements of r are positive or equal to zero ($r_j \geq 0$) then we have already found the optimal solution.

$$z = c_B x_b = [5 \quad 2 \quad 0] \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} = 21$$

$$x_1 = 3 \ \ x_2 = 3$$

End.

COLUMN GENERATION EXAMPLE

$$\max Z = 2x_1 + 4x_2 + 3x_3$$

st ,

$$x_1 + x_2 + x_3 \le 12$$

$$x_1 + 3x_2 + 3x_3 \le 24$$

$$3x_1 + 6x_2 + 4x_3 \le 90$$

$$x_1, x_2, x_3 \ge 0$$

Initially,

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 3 \\ 3 & 6 & 4 \end{bmatrix} \quad c = [2 \quad 4 \quad 3] \quad b = \begin{bmatrix} 12 \\ 24 \\ 90 \end{bmatrix}$$

Iteration 1

Generate RMP

Starting with J=1 All other variables will be reduced to zero, initially:

$$A = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix} \quad c = [2] \quad b = \begin{bmatrix} 12 \\ 24 \\ 90 \end{bmatrix}$$

Hence, RMP is,

$$z^1 = max \sum_{j=1}^{1} c_j x_j$$

st., $\sum_{j=1}^{1} a_{ij} x_j = b \;\; (j = 1,2, \dots, m)$

$$x_1 \ge 0$$

Solving RMP

We are solving the **RMP** using the Revised Simplex Method and the optimal shadow prices are being generated.

Hence,

$$x_1 = 12\ so, zopt = 2 * x_1 = 24$$

At the optimal solution we get,

$$\pi^1 = [2 \quad 0 \quad 0],\ B = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix}, A = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, c_b = [2\ 0\ 0] and\ b = \begin{bmatrix} 12 \\ 12 \\ 54 \end{bmatrix}$$

Solving the Sub-problem

Solving the Sub-problem we find the entering column.

$$z_{sub} = \max_{1 \le j \le n} \{c_j - \sum_{j=1}^{m} \pi_i^j a_{ij}\}$$

So,

$$z_{sub} = \max_{1 \le j \le n} \{c_j - [2 \quad 0 \quad 0] a_{ij}\}$$

If j=1, $z_1 = 2 - [2 \quad 0 \quad 0] \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix} = 0$

If j=2, $z_1 = 4 - [2 \quad 0 \quad 0] \begin{bmatrix} 1 \\ 3 \\ 6 \end{bmatrix} = 2$

If j=3 , $z_1 = 3 - [2 \quad 0 \quad 0] \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix} = 1$

$$z_{sub} = \max\{0,2,1\} = 2$$

Hence, column entering RMP is for j=2.

Add new column to RMP

We should calculate the representation of this column, at the current iteration ($y_j = B^{-1} a_j$) and increase the variables of the RMP by 1($J + 1$).

Hence,

$$\pi^1 = [2 \quad 0 \quad 0],\ B = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix}, A = \begin{bmatrix} 1 & 1 \\ 0 & 3 \\ 0 & 6 \end{bmatrix}, c_b = [2\ 0\ 0]\ ,$$

$b = \begin{bmatrix} 12 \\ 12 \\ 54 \end{bmatrix}$ $and\ c = [0\ \ 4]$

Iteration 2

Solving RMP

We are solving the **RMP** using the Revised Simplex Method and the optimal shadow prices are being generated.

Hence,

$$x_1 = 6\ x_2 = 6\ \kappa\alpha\iota\ s_3 = 36$$

So,

$$zopt = 2 * x_1 + 4 * x_2 + \ 0 * s_3 = 36$$

$\pi^2 = [1 \quad 1 \quad 0]$, $B = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 3 & 0 \\ 3 & 6 & 1 \end{bmatrix}, A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, c_b = [2\ 4\ 0]\ , b = \begin{bmatrix} 6 \\ 6 \\ 36 \end{bmatrix}$ $adn\ c =$ $[0\ \ 0]$

Solving the Sub-problem

Solving the Sub-problem we find the entering column.

$$z_{sub} = \max_{1 \le j \le n} \{c_j - \sum_{i=1}^{m} \pi_i^j a_{ij} \}$$

Hence,

$$z_{sub} = \max_{1 \le j \le n} \{c_j - [1 \quad 1 \quad 0][a_{ij}]\}$$

If j=1, $z_1 = 2 - [1 \quad 1 \quad 0] \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix} = 0$

If j=2, $z_1 = 4 - [1 \quad 1 \quad 0] \begin{bmatrix} 1 \\ 3 \\ 6 \end{bmatrix} = 0$

If j=3, $z_1 = 3 - [1 \quad 1 \quad 0] \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix} = -1$

Arose to $z_{sub} \le 0$

End

Optimal solution for the MP

$$x_1 = 6\ x_2 = 6\ and\ s_3 = 36$$

And,

$$zopt = 2 * x_1 + 4 * x_2 + \ 0 * s_3 = 36$$

$$zotp = 36$$

APPENDIX B: ANALYTICAL RESULTS FOR SMALL INSTANCES

Table B.1 presents results on selected test-cases from the Solomon benchmark problems for five to nine customers. The computational time and the lower bound obtained are shown for the methods ESA, ND, SD, CD.

Table B.1: Selected results for 5 to 9 customers.

		ESA		SD		ND		CD	
Pro.	**Cust.**	**Cost**	**Time**	**Cost**	**Time**	**Cost**	**Time**	**Cost**	**Time**
R101	**5**	115,60	0,02	115,60	0,02	115,60	0,02	115,6	0,031
	6	156,70	0,10	156,70	0,01	156,70	0,02	156,7	0,032
	7	157,50	0,53	157,50	0,02	157,50	0,03	157,5	0,031
	8	195,90	3,56	195,90	0,02	195,90	0,02	195,9	0,032
	9	217,30	26,78	217,30	0,04	217,30	0,03	217,3	0,032
R102	**5**	93,30	0,05	93,30	0,02	93,30	0,02	93,3	0,031
	6	131,00	0,13	134,60	0,04	131,00	0,07	131	0,078
	7	135,20	0,58	135,20	0,07	135,20	0,15	135,2	0,079
	8	151,40	3,70	151,40	0,15	151,40	0,56	151,4	0,344
	9	178,00	27,17	178,00	0,13	178,00	0,87	178	0,546
R103	**5**	93,30	0,04	93,30	0,02	93,30	0,04	93,3	0,016
	6	131,00	0,12	134,60	0,04	131,00	0,07	131	0,078
	7	135,20	0,56	135,20	0,07	135,20	0,15	135,2	0,078
	8	151,40	3,77	151,40	0,14	151,40	0,56	151,4	0,36
	9	178,00	27,17	178,00	0,14	178,00	0,86	178	0,532
R104	**5**	93,30	0,04	93,30	0,03	93,30	0,02	93,3	0,016
	6	131,00	0,15	134,60	0,04	131,00	0,07	131	0,078
	7	131,60	0,64	131,60	0,09	131,60	0,32	131,6	0,141
	8	141,10	3,99	141,10	0,13	141,10	1,63	141,1	0,188
	9	162,10	27,89	162,10	0,24	162,10	3,32	162,1	0,875
R105	**5**	115,60	0,03	115,60	0,02	115,60	0,02	115,6	0,016
	6	93,30	0,03	93,30	0,03	93,30	0,02	142,1	0,016
	7	119,80	0,15	119,80	0,06	119,80	0,08	157,5	0,485
	8	135,20	0,61	135,20	0,09	135,20	0,22	188,8	0,063
	9	151,40	3,81	151,40	0,17	151,40	0,86	65535	0,093
R106	**5**	93,30	0,05	93,30	0,02	93,30	0,02	93,3	0,031
	6	131,00	0,13	134,60	0,04	131,00	0,07	119,8	0,062
	7	135,20	0,58	135,20	0,07	135,20	0,15	135,2	0,109
	8	151,40	3,70	151,40	0,15	151,40	0,56	151,4	0,515
	9	178,00	27,17	178,00	0,13	178,00	0,87	177,5	1,109
R107	**5**	93,30	0,03	93,30	0,03	93,30	0,02	93,3	0,015
	6	119,80	0,14	119,80	0,05	119,80	0,08	119,8	0,078
	7	135,20	0,60	135,20	0,09	135,20	0,21	135,2	0,109
	8	151,40	3,89	151,40	0,17	151,40	0,87	151,4	0,515
	9	177,50	28,22	177,50	0,22	177,50	1,59	177,5	1,109
R108	**5**	93,30	0,03	93,30	0,02	93,30	0,02	93,3	0,016
	6	119,80	0,13	119,80	0,05	119,80	0,07	119,8	0,078
	7	120,40	0,71	120,40	0,12	120,40	0,38	120,4	0,141
	8	141,10	4,26	141,10	0,18	141,10	2,01	141,1	0,219

		ESA		SD		ND		CD	
Pro.	**Cust.**	**Cost**	**Time**	**Cost**	**Time**	**Cost**	**Time**	**Cost**	**Time**
	9	162,10	30,36	162,10	0,36	162,10	5,95	162,1	1,422
C101	**5**	42,70	0,04	42,70	0,01	42,70	0,01	42,7	0,015
	6	42,80	0,12	42,80	0,02	42,80	0,03	42,8	0,015
	7	46,70	0,56	46,70	0,03	46,70	0,09	46,7	0,047
	8	48,20	3,77	48,20	0,05	48,20	0,10	48,2	0,093
	9	50,30	26,98	50,30	0,06	50,30	0,25	50,3	0,172
C102	**5**	42,70	0,03	42,70	0,02	42,70	0,02	42,7	0,031
	6	42,80	0,16	42,80	0,06	42,80	0,13	42,8	0,078
	7	46,50	0,71	46,50	0,14	46,50	0,45	46,5	0,297
	8	47,20	4,97	47,20	0,30	47,20	5,73	47,2	0,625
	9	49,30	48,18	50,30	0,60	49,30	139,12	49,3	1,86
C103	**5**	42,70	0,04	42,70	0,02	42,70	0,03	42,7	0,032
	6	42,80	0,17	42,80	0,06	42,80	0,13	42,8	0,094
	7	46,50	0,72	46,50	0,14	46,50	0,52	46,5	0,281
	8	47,20	5,05	47,20	0,30	47,20	5,89	47,2	0,64
	9	49,30	50,41	50,30	0,58	49,30	5,89	49,3	1,859
C104	**5**	42,70	0,04	42,70	0,02	42,70	0,03	42,7	0,031
	6	42,80	0,18	42,80	0,07	42,80	0,14	42,8	0,078
	7	46,50	0,97	46,50	0,31	46,50	1,41	46,5	0,5
	8	47,20	11,40	47,20	0,37	47,20	46,35	47,2	0,875
	9	49,30	190,19	50,30	0,79	49,30	1093,32	49,3	3,406
C105	**5**	42,70	0,03	42,70	0,01	42,70	0,02	42,7	0
	6	42,80	0,11	42,80	0,03	42,80	0,03	42,8	0,031
	7	46,70	0,53	46,70	0,03	46,70	0,05	46,7	0,047
	8	48,20	4,24	48,20	0,07	48,20	0,20	48,2	0,11
	9	50,30	29,70	50,30	0,06	50,30	0,25	50,3	0,172
C106	**5**	42,70	0,03	42,70	0,02	42,70	0,02	42,7	0,015
	6	42,80	0,10	42,80	0,03	42,80	0,03	42,8	0,016
	7	46,70	0,53	46,70	0,03	46,70	0,04	46,7	0,031
	8	48,20	3,77	48,20	0,05	48,20	0,11	48,2	0,094
	9	50,30	27,81	50,30	0,07	50,30	0,23	50,3	0,172
C107	**5**	42,70	0,03	42,70	0,03	42,70	0,02	42,7	0,016
	6	42,80	0,11	42,80	0,02	42,80	0,03	42,8	0,032
	7	46,70	0,71	46,70	0,04	46,70	0,07	46,7	0,063
	8	48,20	3,83	48,20	0,07	48,20	0,14	48,2	0,11
	9	50,30	29,19	50,30	0,09	50,30	0,32	50,3	0,157
C108	**5**	42,70	0,03	42,70	0,04	42,70	0,03	42,7	0,016
	6	42,80	0,12	42,80	0,03	42,80	0,03	42,8	0,031
	7	46,50	0,59	46,50	0,04	46,50	0,07	46,5	0,079
	8	48,20	3,50	48,20	0,06	48,20	0,28	48,2	0,141
	9	50,30	27,42	50,30	0,09	50,30	0,89	50,3	0,235
RC101	**5**	5,00	87,20	0,03	87,20	0,02	87,20	87,2	0,016
	6	6,00	89,40	0,12	89,40	0,03	89,40	89,4	0,016
	7	7,00	108,30	0,55	108,30	0,04	108,30	108,3	0,062
	8	8,00	112,40	3,69	112,40	0,06	112,40	112,4	0,14
	9	9,00	121,60	27,03	121,60	0,19	121,60	121,6	0,547
RC102	**5**	5,00	82,80	0,04	82,80	0,03	82,80	82,8	0,031
	6	6,00	84,60	0,16	84,60	0,06	84,60	84,6	0,063
	7	7,00	88,70	0,74	92,10	0,11	88,70	88,7	0,265

		ESA		SD		ND		CD	
Pro.	**Cust.**	**Cost**	**Time**	**Cost**	**Time**	**Cost**	**Time**	**Cost**	**Time**
	8	8,00	93,50	5,44	94,80	0,29	93,50	94,8	0,734
	9	9,00	100,00	50,56	100,00	0,46	100,00	100,2	1,281
RC103	**5**	5,00	82,80	0,04	82,80	0,02	82,80	82,8	0,031
	6	6,00	84,60	0,16	84,60	0,06	84,60	84,6	0,062
	7	7,00	88,70	0,74	92,10	0,12	88,70	88,7	0,265
	8	8,00	93,50	5,39	94,80	0,30	93,50	94,8	0,734
	9	9,00	100,00	50,20	100,00	0,43	100,00	100,2	1,281
RC104	**5**	5,00	82,80	0,05	82,80	0,03	82,80	82,8	0,031
	6	6,00	84,60	0,16	84,60	0,07	84,60	84,6	0,062
	7	7,00	88,70	0,81	88,70	0,20	88,70	88,7	0,156
	8	8,00	93,50	6,48	93,50	0,24	93,50	93,5	0,5
	9	9,00	96,60	117,10	100,20	0,51	96,60	96,6	1,688
RC105	**5**	5,00	82,80	0,04	82,80	0,03	82,80	82,8	0,032
	6	6,00	87,70	0,18	87,70	0,06	87,70	89,4	0,11
	7	7,00	93,50	0,67	93,50	0,15	93,50	93,5	0,313
	8	8,00	99,60	4,35	100,90	0,41	99,60	104,8	0,86
	9	9,00	109,60	29,61	109,60	0,69	109,60	109,6	1,235
RC106	**5**	5,00	82,80	0,03	82,80	0,03	82,80	82,8	0,032
	6	6,00	89,40	0,12	89,40	0,03	89,40	89,4	0,032
	7	7,00	103,20	0,59	103,20	0,12	103,20	103,2	0,172
	8	8,00	107,30	3,89	107,30	0,15	107,30	107,3	0,328
	9	9,00	108,30	27,66	108,30	0,40	108,30	108,3	1,188
RC107	**5**	5,00	82,80	0,03	82,80	0,03	82,80	82,8	0,031
	6	6,00	84,60	0,12	84,60	0,05	84,60	89,4	0,032
	7	7,00	88,70	0,62	88,70	0,10	88,70	88,7	0,156
	8	8,00	93,50	3,99	93,50	0,15	93,50	93,5	0,25
	9	9,00	98,60	31,05	98,60	0,27	98,60	98,6	0,891
RC108	**5**	5,00	82,80	0,04	82,80	0,03	82,80	82,8	0,016
	6	6,00	84,60	0,15	84,60	0,07	84,60	86,9	0,156
	7	7,00	88,70	0,96	91,00	0,11	88,70	88,7	0,391
	8	8,00	93,50	7,33	95,80	0,48	93,50	93,5	1,078
	9	9,00	96,60	124,47	100,40	0,90	96,60	96,6	3,36

APPENDIX C: COMPARISON OF THE SD AND CD METHODS

Table C.1 presents results on selected test-cases from the Solomon benchmark for 5 to 25 customers. The testes where conducted using the SD and CD methods presented in Chapter 5. The results for the SD methods are presented and the total computational time is compared against the CD method.

Table C.1: Selected results of the SD and CD methods for 5 to 19 customers.

		Cost			Computational Time				
Pro.	Cust.	Lower Bound	Integer Solution	# of Routes	Generated Columns	First Node	Total	SD Total	Difference %
R101	5	156,2	156,2	2	20	0,033	0,031	0,016	106%
	7	195,2	195,2	3	28	0,030	0,031	0,031	-3%
	9	241,2	241,2	4	52	0,053	0,047	0,031	72%
	11	297,4	297,4	4	70	0,087	0,094	0,047	86%
	13	312,8	312,8	4	96	0,138	0,14	0,078	77%
	15	383,1	383,1	5	118	0,207	0,219	0,094	120%
	17	431,9	431,9	6	162	0,283	0,281	0,141	100%
	19	470,2	470,2	6	176	0,380	0,375	0,172	121%
R102	5	130,5	130,5	1	78	0,078	0,078	0,063	23%
	7	150,7	150,7	1	288	0,326	0,328	0,11	196%
	9	201,3	201,3	3	376	0,601	0,609	0,203	196%
	11	260,2	260,5	3	504	1,311	1,609	0,594	121%
	13	270,6	270,6	3	822	3,305	3,297	0,735	350%
	15	326,1	326,1	4	810	3,728	3,735	0,86	333%
	17	365,9	365,9	4	948	4,526	4,532	1,047	332%
	19	421,3	424	6	1046	7,344	13,234	4,078	80%
R103	5	130,5	130,5	1	78	0,079	0,078	0,047	69%
	7	150,7	150,7	1	288	0,327	0,328	0,11	197%
	9	201,3	201,3	3	376	0,601	0,61	0,188	219%
	11	260,2	260,5	3	504	1,313	1,61	0,578	127%
	13	270,6	270,6	3	822	3,302	3,297	0,734	350%
	15	326,1	326,1	4	914	4,761	4,765	0,984	384%
	17	351,1	356,4	4	1384	12,832	21,297	5,453	135%
	19	354,2	360,8	5	2130	24,682	28,953	4,89	405%
R104	5	130,5	130,5	1	78	0,080	0,078	0,047	70%
	7	140,4	140,4	1	210	0,180	0,172	0,094	91%
	9	179,6	179,6	2	450	0,978	0,969	0,328	198%
	11	216,7	216,7	2	838	4,473	4,485	0,906	394%
	13	232,6	232,6	2	1348	8,378	8,375	1,563	436%
	15	288,4	288,4	3	1550	13,049	13,047	2,468	429%
	17	308,9	308,9	3	2446	38,939	38,953	5,359	627%
	19	327,1444	329,1	3	3736	116,988	205,562	28,937	304%
R105	5	141,6	141,6	2	30	0,025	0,031	0,015	67%
	7	188,1	188,1	3	76	0,063	0,062	0,047	34%
	9	238,85	241,2	4	98	0,106	0,14	0,079	35%
	11	285,8	286,5	3	150	0,203	0,391	0,203	0%
	13	299,975	301,9	3	208	0,406	0,672	0,328	24%
	15	353,8	354,4	4	228	0,675	0,89	0,375	80%
	17	390,7	390,7	4	318	0,889	0,891	0,281	216%

		Cost			Computational Time				
Pro.	**Cust.**	**Lower Bound**	**Integer Solution**	**# of Routes**	**Generated Columns**	**First Node**	**Total**	**SD Total**	**Difference %**
	19	415	415	5	484	2,200	2,203	0,703	213%
R106	**5**	119,3	119,3	1	86	0,075	0,078	0,046	64%
	7	150,7	150,7	1	296	0,479	0,469	0,156	207%
	9	195	195	2	438	1,023	1,015	0,25	309%
	11	249,3	249,3	3	642	2,360	2,359	0,547	331%
	13	258,2	258,2	3	928	5,088	5,094	0,937	443%
	15	316,2	319,3	4	970	5,848	14,204	4,219	39%
	17	356,05	357,7	4	1368	12,902	17,719	3,797	240%
	19	368,475	380,7	5	1506	18,096	52,796	15,016	21%
R107	**5**	119,3	119,3	1	86	0,069	0,078	0,047	47%
	7	150,7	150,7	1	296	0,481	0,484	0,156	208%
	9	195	195	2	438	1,022	1,015	0,265	286%
	11	249,3	249,3	3	642	2,363	2,359	0,547	332%
	13	258,2	258,2	3	928	5,088	5,094	0,922	452%
	15	311,3667	315,2	3	1100	7,528	11,594	2,938	156%
	17	336,3	336,3	4	1654	17,894	17,89	2,328	669%
	19	339,4	339,4	4	2634	45,879	45,875	5,703	704%
R108	**5**	119,3	119,3	1	86	0,068	0,078	0,047	45%
	7	140,4	140,4	1	236	0,205	0,203	0,093	121%
	9	179,6	179,6	2	500	1,210	1,219	0,344	252%
	11	209,9	209,9	2	1010	6,518	6,516	1,047	523%
	13	218,8	218,8	2	1552	12,726	12,735	1,891	573%
	15	277,7	277,7	3	1562	15,797	15,797	2,469	540%
	17	296,9	296,9	3	2434	47,780	47,781	5,594	754%