UNIVERSITY OF THE AEGEAN
SCHOOL OF BUSINESS

DEPARTMENT OF FINANCIAL AND MANAGEMENT ENGINEERING

POSTGRADUATE STUDIES PROGRAMME IN
"FINANCIAL MANAGEMENT ENGINEERING"

FREQUENCY OF SERVICE IN RETAIL DISTRIBUTION NETWORKS: ENHANCEMENT AND IMPLEMENTATION OF AN OPTIMAL APPROACH

AUTHOR:

LAZAROS AMANATIDIS

SUPERVISING COMITEE:

I. MINIS

A. PLATIS

V.ZEIMPEKIS

CHIOS, 2010

To my beloved wife

Acknowledgments

Firstly, I would like to thank my Professor Ioannis Minis for giving me the opportunity to undertake the present master thesis, for his valuable supervision and guidance.

I gratefully thank Ninikas Georgios, PhD Candidate of the University of the Aegean, for encouragement, countless discussions and for his help with writing the thesis. Their support was really significant for me.

I would also like to acknowledge the support and there courses made available to me through the DeOPSys Lab of the Financial and Management Engineering (FME) Department of the University of the Aegean

Finally, I would like to thank my family and friends for their support during all this process.

Thank you

Περίληψη (In Greek)

Η προτεινόμενη διπλωματική εργασία εστιάζει στο πρόβλημα Καθορισμού της Επισκεψιμότητας σε Δίκτυο Διανομής με στόχο την μεγιστοποίηση του κέρδους σε συγκεκριμένο χρονικό ορίζοντα. Το πρόβλημα περιγράφεται από μοντέλο ακέραιου προγραμματισμού και επιλύεται με τον αλγόριθμο διαδοχικών ορίων (Branch and Bound) για την εύρεση των βέλτιστων λύσεων και με προτεινόμενο ευρετικό αλγόριθμο για την επίλυση προβλημάτων πρακτικού μεγέθους.

Το συγκεκριμένο πρόβλημα έχει επιλυθεί προηγουμένως από προπτυχιακή διπλωματική εργασία που αναπτύχθηκε στο εργαστήριο ΣυΣΠαΛ του Πανεπιστημίου Αιγαίου. Στη προσπάθεια αυτή, το πρόβλημα επιλύθηκε για προβλήματα πρακτικού μεγέθους με ευρετικό αλγόριθμο. Η προσέγγιση βέλτιστης επίλυσης του προβλήματος αυτού με τη μέθοδο των διαδοχικών ορίων περιορίστηκε μόνο σε ένα μικρό πλήθος πελατών λόγω της μεγάλης πολυπλοκότητας του προβλήματος (μεγάλο πλήθος μεταβλητών και περιορισμών).

Συνεπώς, στο πλαίσιο της παρούσας Διπλωματικής εργασίας υλοποιήσαμε το πρόγραμμα με εργαλεία C++ και συνεισφέραμε στα εξής:

- Επίλυση υφισταμένων προβλημάτων σε ταχύτερους υπολογιστικούς χρόνους
- Επίλυση προβλημάτων μεγαλύτερης κλίμακας συγκριτικά με τη δυνατότητα του προηγούμενου αλγορίθμου.

Επιπρόσθετα, στα πλαίσια της παρούσας διατριβής αναπτύχθηκε κατάλληλη εφαρμογή με δυνατότητα γραφικού περιβάλλοντος χρήστη, η οποία πέρα από την υλοποίηση του αλγόριθμου, παρέχει γραφικό περιβάλλον με πραγματικούς γεωγραφικούς χάρτες για τον καθορισμό των δεδομένων εισόδου αλλά και την οπτικοποίηση των παραγόμενων αποτελεσμάτων (διαδρομή και κόστος αυτής).

Αναλυτικότερα, η κεντρική ιδέα του όλου εγχειρήματος είναι ο καθορισμός της συχνότητας με την οποία πρέπει να γίνονται οι επισκέψεις στους πελάτες δοθέντος συγκεκριμένου δικτύου διανομής και στόλου οχημάτων ώστε να μεγιστοποιείται το συνολικό κέρδος.

Ο καθορισμός της συχνότητας επίσκεψης στον κάθε πελάτη προϋποθέτει να γνωρίζουμε τι ανάγκες έχει ο κάθε πελάτης του δικτύου, δηλαδή τη ζήτηση. Στο υπό μελέτη πρόβλημα η ζήτηση δεν είναι ένας σταθερός αριθμός αλλά μεταβάλλεται με βάση το πλήθος των επισκέψεων που πραγματοποιούνται. Για την ακρίβεια, για κάθε πελάτη υπάρχει μια ελάχιστη ζήτηση και η δυνατότητα πραγματοποίησης πρόσθετων πωλήσεων σε κάθε επίσκεψη μετά την πρώτη. Επιπλέον, υπάρχει και ένα μέγιστο πλήθος δυνατών επισκέψεων, πέραν του οποίου δεν θα πραγματοποιείται καμία πώληση. Η ζήτηση, όπως και το μέγιστο και ελάχιστο

πλήθος επισκέψεων για κάθε πελάτη που ανήκει στο δίκτυο είναι στοιχεία που ανήκουν στα δεδομένα του υπό μελέτη προβλήματος.

Στη μελέτη του προβλήματος εύρεσης ρυθμού επισκεψιμότητας σε δίκτυο λιανικής για τη μεγιστοποίηση του συνολικού κέρδους έγιναν οι ακόλουθες παραδοχές:

- Εάν το πλήθος των επισκέψεων στον πελάτη i ισούται με το ελάχιστο δυνατό $n_{i_{min}}$ τότε η ζήτηση θα είναι D_i^0 και ισοδυναμεί με το ελάχιστο πλήθος προιόντων που πρέπει να παραδοθούν στον πελάτη για να καλυφθούν οι ανάγκες του στο υπό εξέταση χρονικό διάστημα.
- Κάθε επόμενη επίσκεψη θα οδηγήσει σε αύξηση της ζήτησης D_i με σταθερό ρυθμό k_i.
- Όταν ο πελάτης δεχθεί το μέγιστο αποδεκτό πλήθος επισκέψεων, $n_{i_{max}}$, τότε η αντίστοιχη ζήτηση $D_{i_{max}}$ θα έχει φτάσει στο άνω όριο της για το συγκεκριμένο χρονικό διάστημα.
- Κάθε επόμενη επίσκεψη στον πελάτη i μετά την $n_{i_{max}}$ δεν θα επιφέρει καμία αλλαγή στη ζήτηση ούτε και θα οδηγεί σε νέες πωλήσεις προιόντων.

Τελικό ζητούμενο είναι ο προσδιορισμός του πλήθους επισκέψεων σε κάθε πελάτη ώστε να μεγιστοποιείται το κέρδος εντός συγκεκριμένου χρονικού ορίζοντα (π.χ. 5 ημέρες).

Το παραπάνω πρόβλημα αποτελεί ένα δυαδικό πρόβλημα γραμμικού προγραμματισμού. Στόχος μας είναι η επίλυση του χρησιμοποιώντας ένα αλγόριθμο με μικρή πολυπλοκότητα και υψηλή υπολογιστική ταχύτητα, ώστε να είναι εφικτή η επίλυση σε ικανοποιητικό χρόνο στιγμιότυπων που συναντώνται στη καθημερινότητα. Έχοντας τα παραπάνω ως στόχο, η προτεινόμενη λύση συνίσταται από το συνδυασμό της μεθόδου διακλάδωσης και περιορισμού (branch and bound) με τη μέθοδο της revised simplex και τεχνικές χαλάρωσης περιορισμών (relaxation). Συγκεκριμένα, τα βήματα που ακολουθούνται για την επίλυση του προβλήματος που περιγράφηκε παραπάνω, έχουν ως εξής:

1. Λήψη των δεδομένων του προβλήματος:
    a. D_i^0: αρχική ζήτηση για κάθε πελάτη, η οποία αντιστοιχεί στον ελάχιστο αποδεκτό αριθμό επισκέψεων $n_{i_{min}}$
    b. $n_{i_{min}}$ και $n_{i_{max}}$: ο ελάχιστος και μέγιστος επιτρεπτός αριθμός επισκέψεων για κάθε πελάτη που ανήκει στο υπό εξέταση δίκτυο διανομής
    c. k_i: ρυθμός αύξησης της ζήτησης ανά πελάτη μετά τις πρώτες $n_{i_{min}}$ επισκέψεις
    d. c_{ij}: Το γινόμενο της απόστασης σε km μεταξύ κάθε ζεύγους πελατών που ανήκουν στο δίκτυο διανομής ή μεταξύ αποθήκης και πελάτη πολλαπλασιασμένο επί το δοθέν κόστος ανά km. Συνήθως οι

συντεταγμένες που καθορίζουν τη θέση του κάθε πελάτη δίνονται και η απόσταση υπολογίζεται με χρήση της εξίσωσης: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

e. p: Το κέρδος ανά μονάδα πωληθέντος προϊόντος.

2. Εκφράζουμε το πρόβλημα με μαθηματικές εξισώσεις, ως εξής:

$$max\left[\sum_{i\in V_u}\left[D_i^0 + \left(\sum_{d=1}^{n_d} y_i^d - n_{i_{min}}\right)k_i\right]p - \sum_{d=1}^{n_d}\sum_{i\in V}\sum_{\substack{j\in V\\ i\neq j}} c_{ij}x_{ij}^d\right]$$

$$under\ constraints:$$

$$\sum_{j\in V, i\neq j} x_{ji}^d - y_i^d = 0$$

$$\sum_{j\in V, i\neq j} x_{ij}^d - y_i^d = 0$$

$$\sum_{j\in V} x_{0j}^d = 1$$

$$\sum_{j\in V} x_{j0}^d = 1$$

$$\sum_{i\in S} y_i^d - \sum_{i\in S}\sum_{j\in S} x_{ij}^d \geq 1$$

$$\sum_{i\in V}\sum_{j\in V, i\neq j} c_{ij}x_{ij}^d \leq C$$

$$\sum_{d=1}^{n_d} y_i^d \leq n_{i_{max}}$$

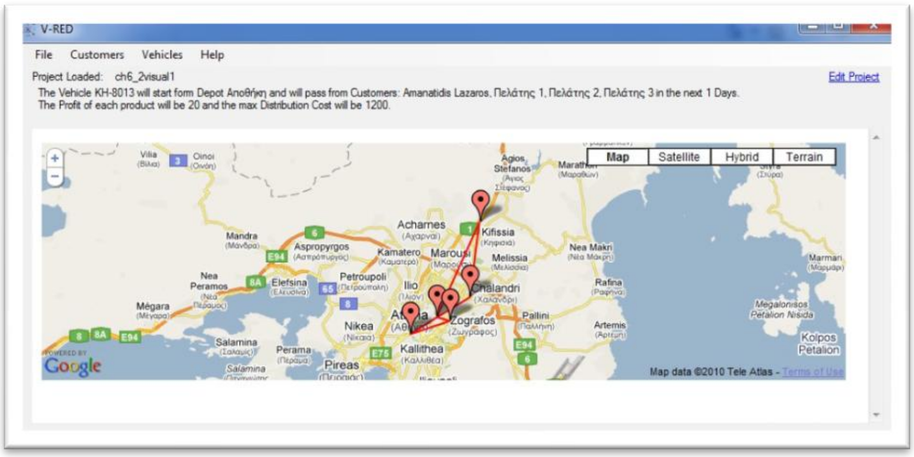
$$\sum_{d=1}^{n_d} y_i^d \geq n_{i_{min}}$$

3. Μετατροπή του προβλήματος σε κανονική μορφή με τη προσθήκη τεχνητών μεταβλητών, πλεονάσματος και περιθωρίου όπου απαιτείται. Συγκεκριμένα, οι περιορισμοί 1, 2, 3 και 4 είναι ισότητες οπότε απαιτείται η προσθήκη μιας τεχνητής μεταβλητής σε καθέναν από αυτούς, ο περιορισμός 6 και το δεύτερο μέρος του περιορισμού 7 είναι ανισότητες με μικρότερο ή ίσο οπότε απαιτείται η προσθήκη μεταβλητής περιθωρίου σε κάθε έναν από αυτούς ενώ στους λοιπούς περιορισμούς που είναι ανισότητες με μεγαλύτερο ή ίσο, χρήζει αφαίρεση μεταβλητής πλεονάσματος και πρόσθεση τεχνητής μεταβλητής. Σε αυτό το σημείο πρέπει να σημειωθεί ότι σε κάθε περίπτωση που αρνητική τιμή

εμφανίζεται στο δεξί σκέλος ανισότητας, πρώτα πολλαπλασιάζουμε με -1 και στη συνέχεια ακολουθούμε την παραπάνω διαδικασία.

4. Χαλάρωση του αρχικού προβλήματος με την αφαίρεση των περιορισμών που αφορούν την απαίτηση οι λύσεις να λαμβάνουν μόνο τις τιμές 0 ή 1, οπότε αρκεί οι λύσεις να είναι ακέραιες μεταβλητές και όχι δυαδικές όπως ήταν στην αρχική έκφραση του προβλήματος.
5. Επίλυση του προβλήματος χρησιμοποιώντας τη Revised Simplex μέθοδο. Η διαφορά της μεθόδου αυτής από την κλασσική simplex έγκειται στο ότι η μεν αρχική μορφή υπολογίζει και αποθηκεύει όλους τους αριθμούς που αποτελούν το ταμπλό άσχετα με το αν απαιτούνται ενώ η Revised Simplex χρησιμοποιεί τη μέθοδο Gauss – Jordan για τον υπολογισμό του αντίστροφου του πίνακα και από εκεί και πέρα οι μόνες πράξεις που απαιτούνται είναι πολλαπλασιασμοί πινάκων που γίνονται με χρήση πολύ περιορισμένων υπολογιστικών πόρων.
6. Επιπλέον, στην υπό μελέτη περίπτωση το πρόβλημα πάντοτε περιέχει τεχνητές μεταβλητές, συνεπώς απαιτείται η χρήση είτε της μεθόδου του μεγάλου Μ (Big M) είτε των δύο φάσεων (Two Phase) για την επίλυση του. Επειδή η μέθοδος των δύο φάσεων είναι καταλληλότερη για υλοποίηση σε υπολογιστικά συστήματα και οδηγεί σε πιο ακριβή συστήματα, επιλέχθηκε η χρήση της στην συγκεκριμένη περίπτωση. Με βάση τη μέθοδο αυτή, αρχικά σχηματίζουμε ένα πρόβλημα ελαχιστοποίησης χρησιμοποιώντας μόνο τις τεχνητές μεταβλητές και το επιλύουμε. Εάν το προκύπτον πρόβλημα και όλες οι τεχνητές μεταβλητές λαμβάνουν μηδενική τιμή τότε το αρχικό πρόβλημα έχει εφικτή λύση και όλες οι στήλες που αντιστοιχούν σε τεχνητές μεταβλητές μπορούν να αγνοηθούν ειδάλλως δεν έχει εφικτές λύσεις.
7. Χρήση της λύσης που βρέθηκε στο προηγούμενο βήμα και του προβλήματος στο οποίο αφαιρέθηκαν οι περιορισμοί των μεταβλητών από δυαδικές σε ακέραιες για την εκκίνηση του αλγόριθμου branch και bound.

Με βάση τα παραπάνω αναπτύχθηκε η εφαρμογή V-RED. Η αρχική της μορφή ήταν μια απλή παραθυρική εφαρμογή που έκανε χρήση της οθόνης τερματικού για την πληκτρολόγηση των δεδομένων και την εμφάνιση των αποτελεσμάτων. Στην συνέχεια αναπτύχθηκε ένα εξελιγμένο παραθυρικό περιβάλλον για την διεπαφή με το χρήστη όπου η εισαγωγή των δεδομένων γίνεται με χρήση γεωγραφικών συντεταγμένων και η απεικόνιση τόσο των δεδομένων εισόδου όσο και των αποτελεσμάτων πραγματοποιείται σε διαδραστικούς γεωγραφικούς χάρτες.



Project Loaded: ch6_2visual1

The Vehicle KH-8013 will start form Depot Αποθήκη and will pass from Customers: Amanatidis Lazaros, Πελάτης 1, Πελάτης 2, Πελάτης 3 in the next 1 Days.
The Profit of each product will be 20 and the max Distribution Cost will be 1200.

Εικόνα 1 V-RED Interface

Η εφαρμογή ελέγχθηκε για την ορθότητα των αποτελεσμάτων που παράγει και την καλή της λειτουργία μέσω συγκρίσεων των παραγόμενων αποτελεσμάτων που αφορούσαν στιγμιότυπα του προβλήματος που ήταν ήδη γνωστά τα αναμενόμενα αποτελέσματα.

Στην συνέχεια η εφαρμογή χρησιμοποιήθηκε για τον υπολογισμό των αποτελεσμάτων στιγμιότυπων που είχαν εξεταστεί σε προηγούμενη διπλωματική εργασία (Ασημακόπουλος, 2001), όπου χρονομετρήθηκε ο χρόνος (CPU time) που απαιτείται για την παραγωγή των αποτελεσμάτων με την νέα υλοποίηση του αλγορίθμου και έγινε σύγκριση των χρόνων αυτών με τους αντίστοιχους της προηγούμενης υλοποίησης. Με βάση τα αποτελέσματα η νέα εφαρμογή έχει πολύ καλύτερους υπολογιστικούς χρόνους.

Επιπλέον όπως φαίνεται στον πίνακα 1, δοκιμάστηκε η επέκταση της χρήσης της εφαρμογής και σε μεγαλύτερου μεγέθους προβλήματα, τόσο όσο αφορά το πλήθος των πελατών όσο και το χρονικό ορίζοντα. Περαιτέρω έρευνα απαιτείται για τον σαφή προσδιορισμό των άνω ορίων για τη συγκεκριμένη υλοποίηση του αλγορίθμου και μελέτη της συμπεριφοράς του σε πιο πολύπλοκα στιγμιότυπα του προβλήματος.

Πίνακας 1 9 - 12 Πελάτες Αποτελέσματα

	9 ΠΕΛΑΤΕΣ		10 ΠΕΛΑΤΕΣ		11 ΠΕΛΑΤΕΣ		12 ΠΕΛΑΤΕΣ	
ΗΜΕΡΕΣ	ΧΡΟΝΟΣ (Sec)	ΚΕΡΔΟΣ (€)	ΧΡΟΝΟΣ (Sec)	ΚΕΡΔΟΣ (€)	ΧΡΟΝΟΣ (Sec)	ΚΕΡΔΟΣ (€)	ΧΡΟΝΟΣ (Sec)	ΚΕΡΔΟΣ (€)
1	4,40	3920,33	10,20	4220,33	30,80	4795,14	98,50	5487,10
2	12,20	6716,66	35,20	7316,66	120,60	8270,27	444,10	14914,20
3	30,60	9512,99	93,40	10412,99	245,90	11745,41		
4	54,90	12309,32	158,40	13509,32	406,60	14845,74		
5	93,80	15105,65	235,00	16487,65				

Abstract

The focus of the current master thesis is set on the definition of the frequency of service in a distribution network by maximizing the profit in a specific time window. The problem is modeled using integer mathematical programming and is solved using the Brach and Bound algorithm in order to find the optimal solution.

This problem has already been solved within an Undergraduate thesis developed by the DeOpSys team of the University of the Aegean by Asimakopoulos (2006). In that effort the focus was given in solving practical sized problems using a specific heuristic algorithm. The actual solution of the problem using the Branch and Bound algorithm was narrowed down to a small number of clients due to the high complexity of the problem (number of variables and constraints).

Therefore, primary goal of this thesis is to enhanced solution method in order to have better computational times than the previous approach and be able to solve larger problems.

More specifically, we developed a software tool in C++ that implements the above mentioned algorithm and provides:

- Better solution times for problems being solved by Asimakopoulos (2006) application
- Solution of bigger problems while keeping the same algorithm

Finally, a windows application as interface of the algorithm was implemented. This application provides interface for entering the project's input data and representation of coordinates in geographical maps (Google maps) as well as generation of the results (route and cost) on the same maps.

Keywords

Retail distribution network, branch and bound, tsp

Figures

Tables

1 Introduction

Over the past decade, the traditional purchasing and logistics functions have evolved into a broader strategic approach to materials and distribution management known as supply chain management (Choon Tan, 2001). Furthermore, transportation and distribution of goods are key issues faced by the supply chain management sector, since they affect the total cost of the product and the quality of customer service (Eskigun et al., 2005). This is the reason that all companies aiming at being competitive on the market give special attention to the analysis of the supply chain in order to improve the customer service level without an uncontrolled growth of costs (Ambrosino and Scutellà, 2005).

Based on the needs of the supply chain sector, many researchers the last 40 years have widely developed distribution network models in supply chain systems usually having as objective to find capacities, location of new facilities and the best flow of material in the network (Ghezavati, Jabal-Ameli and Makui, 2009).A common objective in designing such a distribution network is to determine the least cost system design such that the demands of all customers are satisfied and frequency of service is maximized without exceeding the predefined cost level (Amiri, 2006).This kind of optimization problems have high degree of complexity and belong to the class of NP-hard optimization problems, in which computational time increases exponentially with the problem's size (Goel and Gruhn, 2008).

One of the most interesting problems in this sector is the Vehicle Routing Problem (VRP), which is a transportation problem where goods are delivered from a central depot to a set of customers, as shown in Figure 1. Several constraints, such as vehicle capacity, allowed working period (e.g. driver's shift), time windows imposed by customers where the service can be performed, etc., should be satisfied. The aim is to design a set of minimum cost routes starting and ending to a depot of a fleet of vehicles serving a set of customers with known demands and service costs (Laporte, 1992).

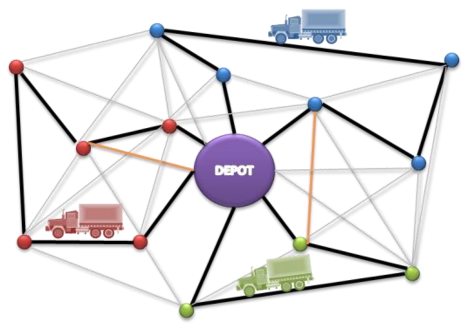


Figure 2 Vehicle Routing Problem

However, many variations on this classical problem exist; usually created by having different constraints, like the one studied in this thesis, where it is added an extra parameter that is the frequency of service to each customer. More specifically, we are solving a problem, related to VRP, which concerns the frequency of service in retail distribution networks. The problem herein presented is about the design of a set of minimum cost routes starting and ending at a depot for a vehicle serving a variable number of times a group of variable customers and where minimum demands and service costs are given. The algorithm's output is the optimal route based on the set of constraints that include but are not limited to the total time of travel (route length), the maximum capacity of each vehicle and the optimal number of visits to each customer.

The herein discussed model has been developed by the DeOpSys team of the University of the Aegean; where in Asimakopoulos (2006) thesis was presented a MATLAB tool for maximizing the total profit earned by delivering goods to the retail network's customers during a specified time window and within the capacity of the used depot. In that implementation, no results were returned when handling more than eight customers and the overall execution time seemed to grow exponentially.

The idea in this second approach of the problem is, to develop for the given optimization method a new algorithm that keeps the same mathematical model but can be executed for more clients and days giving results faster than the previous implementation. In other words the goal is to minimize the execution time by lowering the algorithm's complexity. If this can be achieved then a fully functional software application for vehicles routing could be developed.

Therefore, primary goal of this thesis is to optimally solve the problem for a greater number of clients. More specifically, research will be performed on the restrictions and parameters that raise the problem's complexity and computational time in order to find ways to minimize it.

Finally, a set of methods and techniques will be implemented to get the optimal solution faster and for a larger set of clients. The implementation will be a software application that takes all the delivery network's parameters as input data and returns the optimal routing solution. Emphasis is given on the reduction of the computational time on order to be applicable on a software application.

Thesis Structure

The remainder of the thesis is structured as follows:

In **Chapter 2** the theoretical background of the presented method is analysed, starting from general notions like Delivery Networks and Linear Programming Theory to the crucial for this method, Branch and Bound algorithm and the optimization methods for computing the rate of visits in retail distribution networks developed by other researchers.

In **Chapter 3** the proposed solution for computing the frequency of service in retail distribution networks is presented by describing the problem and formalizing it through mathematical equations that describe the objective function and the restrictions that are set.

In **Chapter 4** the usage of V-RED is described along with experimental results of its application in comparison with those provided from the previous implementation by Asimakopoulos (2005).

Chapter 5 contains information about the requirements analysis and software architectural design of the software tool – V-RED, the implementation process followed for developing V-RED, its main components, the algorithm used to implement each step of the method and the modeled behavior of the system under normal and exceptional circumstances.

Finally, in **Chapter 6** are presented the conclusions of the overall thesis and an evaluation of the performance of the newly developed tool, along with directions for future research.

In the **Appendix** are presented the processes to certify the correctness and the reliability of the developed software along with test scenarios that prove that the provided results are always the same given specific inputs and exactly those that were expected to be based on the analytical solution of the problem.

2 Theoretical Background

2.1 Introduction

This chapter deals with the theoretical background needed to analyze the problem of the definition of optimum visit rates in distribution retail networks and develop a software tool that will be able to solve the problem in a timely manner. For this purpose basic notions of distribution networks and linear programming theory are studied. Special focus is given to the Branch and Bound method including its policies and characteristics and the Revised Simplex Method. Finally, are presented the most common optimization methods for computing the frequency of service in retail distribution networks.

2.2 Distribution Networks

Companies need to be competitive in order to be sustainable; a common way to accomplish that is by analyzing the supply chain in order to improve the customer service level without an uncontrolled growth of costs. This is achieved through the optimization of the flows of goods through the producer network, also called in the literature ***distribution network***, from the supply points to the demand points, essentially the customers of the retailer when we refer to retail distribution networks. More precisely, the goal is to select the optimum numbers, locations and capacities of plants and warehouses to open so that all customer demand is satisfied at minimum total costs of the distribution network (Amiri, 2006).

Distribution network design problems are core problems for every company because they involve strategic decisions which influence tactical and operational decisions. In particular, they involve facility location, transportation and inventory decisions, which affect the cost of the distribution system and the quality of the customer service level (Crainic and Laporte, 1997).

A distribution network analysis has two main axes: *the optimization of the flows of goods*: where we consider an existing distribution network, and we aim to optimize the flows of goods through the network; and *the improvement of the existing network*: where the goal is to choose the best configuration of the facilities that consist the network in order to minimize the overall cost while the company's goals are satisfied. Usually, distribution network problems involve both kinds of analysis (Ambrosino and Scutellà, 2005).

In our case, given a specific retail distribution network, comprising of a depot, a vehicle and a variable number of customers residing in different locations, we aim to find the optimum route and number of visits in order to have the maximum sales within a specific time window and with minimum overall cost. Thus, it is a problem of optimization of the flows of goods through the network. It has its bases in the well-known Travelling Salesman Problem (TSP) that belongs to the combinatorial optimization domains and has been in depth studied in operations research and theoretical computer science. In the TSP given a list of cities and their pairwise distances, the task is to find a shortest possible tour that visits each city exactly once. In our case we have a few twists on this basis. First of all we can and should make multiple visits to our customers\cities; secondly, we want to know not only the shortest path but also the optimal number of visits to maximize profits.

2.3 Optimization Techniques

2.3.1 Linear Programming Theory

The general linear programming problem in standard form can be stated as follows (Kolman and Beck, 1980):

Find values $x_1, x_2,\ldots x_n$ which will

Maximize $z=c_1x_1+c_2x_2+\ldots+c_nx_n$

Subject to the constraints

$$a_{11}x_1+a_{12}x_2+..a_{1n}x_n\leq b_1$$
$$a_{21}x_1+a_{22}x_2+..a_{2n}x_n\leq b_2$$
$$\ldots\ldots$$
$$a_{m1}x_1+a_{m2}x_2+..a_{mn}x_n\leq b_m$$
$$x_j\geq 0, j=1,2,\ldots,n$$

Where the inequalities represent the constraints and the function z is the objective function.

A linear programming problem in standard or canonical form can be compactly described by matrix notation. Let $A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{\mathrm{m}1} & \cdots & a_{\mathrm{mn}} \end{bmatrix}$, **x**=$\begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$ and **c**=$\begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$ then the linear programming problem can be written:

Find vector$\boldsymbol{x} \in \mathbb{R}^n$

Maximize z=$\mathbf{c}^T \boldsymbol{x}$

Subject to the constraints

$$\mathbf{A}\,\boldsymbol{x} \le \mathbf{b}$$

$$\boldsymbol{x} \ge \mathbf{0}$$

An inequality constraint (≤ form)can be converted into an equality constraint using **slack** variables, that are positive numbers that are added to the left side of an inequality so that they effectively 'take up the slack' between the left side and the right side of the inequality.

When we have constraints with equalities "=" the artificial variable technique is used to transform the problem in standard form. This technique introduces a dummy variable, called **artificial** variable into each constraint that is in equality form and assigns an overwhelming penalty to having added this positive artificial variable by changing the objective function.

In case that we have constraints with inequalities in "≥" form, we add an **artificial** variable as in the equalities case and subtract a **surplus** variable from its left side.

The Simplex Method

The Simplex Method is "a systematic procedure for generating and testing candidate vertex solutions to a linear program" (Gill, Murray and Wright, 1982). The algorithm to solve the general linear programming problem in its standard form, using the Simplex Method is briefly depicted below (Hilier and Lieberman, 2005):

- **1st Phase-Initialization**:
    a. Introduce slack variables.
    b. Select the decision variables to be nonbasic variables
    c. Select the slack variables to be the initial basic variables
- **2nd Phase-Optimality Test:**
    a. Are all the coefficients of the row corresponding to the objective function nonnegative?
    b. If yes, then the current solution is optimal and the process stops.
    c. If not, then go to next iteration (see 3rd phase).
- **3rd Phase-Iteration:**
    a. Determine the entering basic variable by selecting the variable with the negative coefficient having the largest absolute value.
    b. Determine the leaving basic variable by applying the minimum ratio test (minimum of ratio side divided by corresponding positive element of pivot column).

The Two Phase Method

There are two methods to solve linear programming problems that are not in the standard form, namely the big-M method and the two-phase method. These methods are used in cases that the linear programming problem contains constraints that are in “equality” or “greater than” form. The only deference between the big-M method and the two-phase method is in the formulation of the objective function and the ease of use in computer based calculations. The algorithm of this method is described below (Hilier and Lieberman, 2005):

- **1st Phase-Initialization**: Revise the constraints of the original problem by introducing slack, surplus and artificial variables as needed to obtain an obvious initial solution for the artificial problem.
- **2nd Phase:** Find an optimal solution for the artificial problem.

  min z= $\sum \boldsymbol{artificial\, variables}$ ***,*** *subject to revised constraints*

  If phase 1 ends with all artificial variables driven to zero, then the second phase can be launched.
- **3rd Phase:** Find an optimal solution for the real problem, drop the artificial variables and starting from the optimal solution obtained at the end of phase 2, use the simplex method to solve the real problem.

The Revised Simplex Method Procedure

Revised Simplex Method is a modification of the well-known simplex method. The Revised Simplex Method describes linear programs as matrix entities and presents the Simplex Method as a series of linear algebra computations. Instead of spending time updating dictionaries at the end of each iteration, the Revised Simplex Method does its heavy calculation at the beginning of each iteration, resulting in much less at the end. Based on the nature of data of the initial problem, iterations of Revised Simplex Method can but not necessarily are, faster than the standard simplex’s iterations.

The general rule is that large and sparse linear programming problems are solved faster and with the Revised Simplex Method, because it is based on calculations made directly on the inverse of the basis matrix. This is the feature that gives consistent advantage in cases where the number of constraints is much lower than the number of variables and results in minor time and memory requirements**Invalid source specified.**.

If we represent the m non zero values in a basic solution, in other words the basic variables, as a vector $\mathbf{x_B}$ and the corresponding columns of **A** *(see section 3.3.1)* are used to create an m x m matrix B and the result of the objective function is the $\mathbf{c_B}$ $= \begin{bmatrix} c_{i1} \\ \vdots \\ c_{im} \end{bmatrix}$ vertex then it follows that:

$$\mathbf{b} = \mathbf{B}\mathbf{x_b}$$
$$\mathbf{x_B} = \mathbf{B^{-1}b}$$

and

$$\mathbf{z} = \mathbf{c_B^T x_B}$$

Using the above notations the Revised Simplex algorithm consists of the following steps, as visualized in Figure 2:

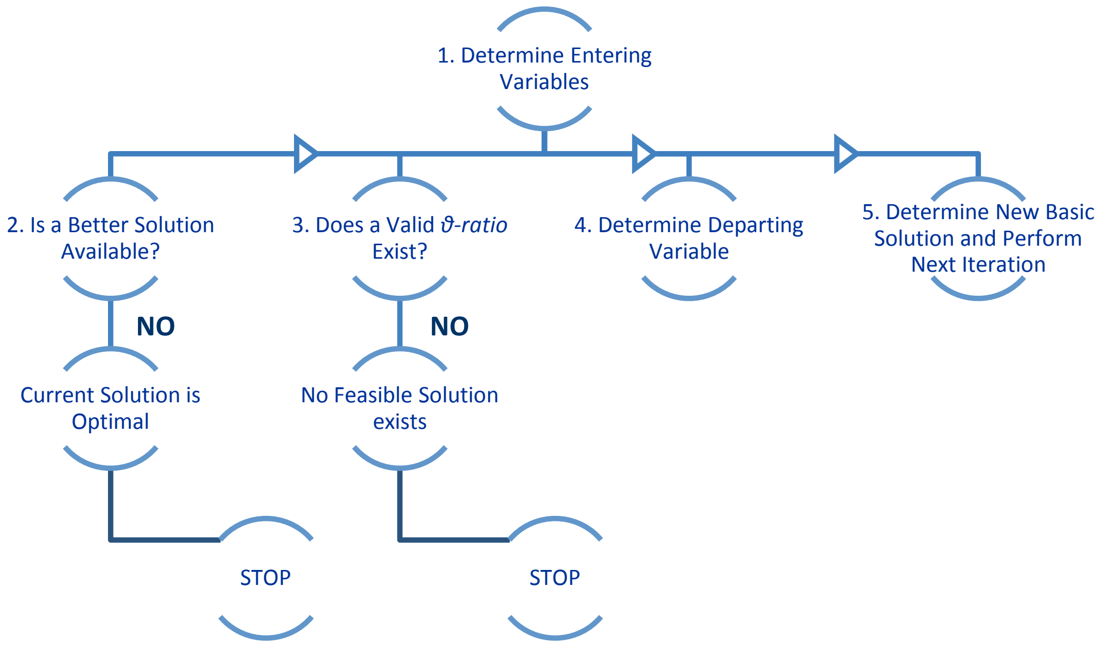

1. Determine Entering Variables
2. Is a Better Solution Available?
3. Does a Valid ϑ-ratio Exist?
4. Determine Departing Variable
5. Determine New Basic Solution and Perform Next Iteration
NO
NO
Current Solution is Optimal
No Feasible Solution exists
STOP
STOP

Figure 3 Revised Simplex Method

1. Define the entering variable by choosing the variable that will cause the greatest increase in the objective function.

$$\boxed{\textbf{FIND } j \textbf{ THAT MINIMIZES } c_B^T B^{-1} A_j - c_j}$$

2. If all the parameters of the objective function are non-negative then an optimal solution has been found, so stop execution.

$$\boxed{\textbf{IF } \forall\, j\ \left(c_B^T B^{-1} A_j - c_j\right) \geq 0 \textbf{ THEN STOP}}$$

3. Determine the departing variable by choosing the variable with the smallest non-negative θ-ratio.

$$\boxed{t_p = B^{-1} A_p}$$

$$\theta - ratio = {}^{\mathrm{x}_{i_r}}/_{t_{rp}} \quad for\ every\ t_{rp} > 0$$

$$where\ p\ is\ the\ index\ of\ the\ entering\ variable$$

If there is not any value with non-negative **θ-ratio** the problem does not have a feasible solution, so stop execution.

IF θ-ratio≥ 0 THEN NO FEASIBLE SOLUTION

4. Determine the new **B**$^{-1}$ from the previous one and the departing and entering variables.

$$B_{new}^{-1} = B_{old}^{-1}E$$

$$E = \begin{bmatrix} 1 & A_{q1} & 0 \\ \vdots & \vdots & \vdots \\ 0 & A_{qm} & 1 \end{bmatrix}$$

$$where\ q\ is\ the\ index\ of\ the\ departing\ variable$$

5. Determine the new **x**$_{\mathbf{B}}$ and go back to step 1.

$$\mathrm{x}_B = B^{-1}b$$

2.3.2 Branch and Bound

Branch and Bound (B&B) is by far the most widely used tool for solving large scale NP-hard combinatorial optimization problems. It is used in cases that we cannot afford to enumerate all possible combinations and get the solution. It is a divide and conquer method where we divide a large problem into a few smaller ones, which is the “branch” part. The conquering part is done by estimating how good solution could be given by the selected node’s sub-tree, which is the “bound” part.

The idea is to partition the feasible region into more manageable subdivisions and then, if required, to further partition the 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.

The efficiency of the method depends strongly on the node-splitting procedure and on the upper and lower bound estimators. Ideally the procedure stops when all nodes of the search tree are either pruned or solved. At that point, all non-pruned sub regions will have their upper and lower bounds equal to the global minimum of the function.

In general, there are a number of ways to divide the feasible region, and as a consequence there are a number of B&B algorithms. Thus, B&B is an algorithm that each time that is implemented needs several parameters to be filled in and there are numerous available choices for each case. These parameters should be carefully chosen in order to get the desired behavior. This is the reason that several techniques for the design of efficient B&B algorithms have emerged over the years.

Definitions

- **Node**: represents any partial or complete solution. Partial solution is represented with a node when each tree level defines a specific part-variable of the problem.
- **Leaf node**: a complete solution where all variables are known and no other branching is possible. Every leaf node has associated an actual value of the objective function along with the corresponding values for all the variables of the problem. Leaf nodes are also those that have given an infeasible solution.
- **Bud node**: a partial solution, either feasible or infeasible. It is a node that might grow further but we don’t know it for sure, yet.
- **Bounding function**: Each bud node has associated a bounding function. The bounding function is used to estimate the best value of the objective function that we could get by growing the specific node. It is an estimator of the values that we will compute for the child nodes and it is very important to be an optimistic estimator. Thus when we have a minimization problem it must underestimate the actual best achievable result and the opposite in the maximization case.
- **Branching**, **growing** or **expanding a node**: is the process of creating the child nodes of a bud node. For every possible value range we create a child node. If we have a binary tree then when we branch we always create two child nodes one for the zero value and one associated to the value one. If we have an integer programming problem, for each variable that is not integer we create a child node with its floor value and one for the corresponding ceil value.
- **Incumbent**: the currently best feasible solution. Each time that we find a feasible solution we compare it with the incumbent and in case that is better than the current incumbent we update it with the new value. Usually when the process begins we don’t have an incumbent and thus, the first feasible solution found becomes the incumbent.
- **Partitioning policy:** the rules used to decide when and how the branching will be effectuated.
- **Node selection policy**: the rules to select which is the next node to be visited.
- **Variable selection** policy: the rules to select which variables should be analyzed and how in order to create branches.

- **Fathoming**: when should be stopped the growth of a node, it depends on the current solution and its magnitude relation with the incumbent along with the type of variables that constitute that solution.
- **Terminate rule**: when should we terminate the algorithm execution either a solution has been found or not.

Algorithm

Let us have an integer linear programming problem, to solve it we will use the Branch and Bound algorithm (Forrest and Tomlin, 2007), as it is depicted in Figure 3.

❖ **Step 1:** Having an integer programming model we create its linear programming relaxation by dropping the requirement that all variables must be integers.

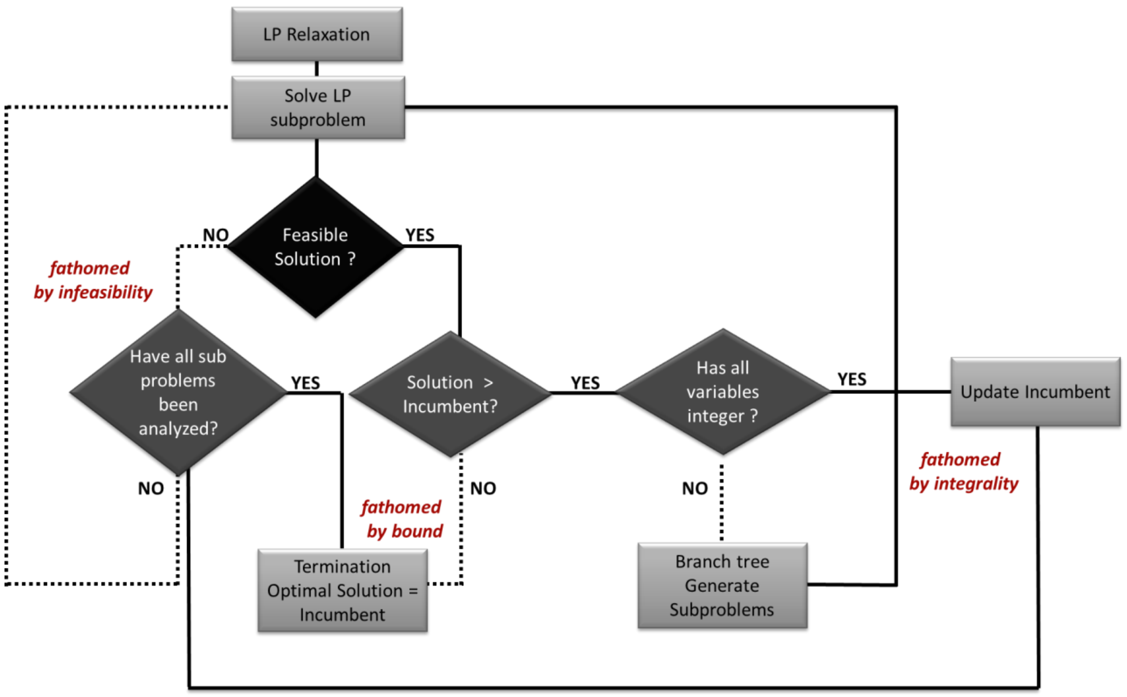

LP Relaxation
Solve LP subproblem
Feasible Solution ?
NO
YES
fathomed by infeasibility
Have all sub problems been analyzed?
YES
NO
Solution > Incumbent?
YES
NO
fathomed by bound
Has all variables integer ?
YES
NO
Update Incumbent
fathomed by integrality
Termination Optimal Solution = Incumbent
Branch tree Generate Subproblems

Figure 4 Branch and Bound Flowchart

❖ **Step 2:** Solve the corresponding linear programming problem.

  - **Step 2.1:** If no feasible solution exists, then the algorithm stops. The response returned is that the initial integer linear problem does not have a feasible solution.

❖ **Step 3:** Compare the optimal solution to the best known feasible solution.

  - **Step 3.1:** If all the variables of the solution are integer then this is a possible optimal solution of the initial problem. Check if the found solution is greater from the last known feasible solution (for a maximization problem, the opposite in case of minimization). If yes, then set the current solution as the best solution we know (incumbent) else fathom the branch.

- **Step 3.2:** If not all the variables of the solution are integers, check if the found solution is greater from the last known feasible solution (for a maximization problem, the opposite in case of minimization). If yes, then divide this subproblem further and repeat.

Consequently, **a subproblem is fathomed** in the following cases:

- The relaxation of the subproblem has an optimal solution but it is not greater of the current best solution (in case of maximization).
- The relaxation of the subproblem has no feasible solution.
- The relaxation has an optimal solution with all the variable values integer (or binary in case of binary problem).

A **subproblem is branched** when the solution found is greater (in case of maximization) than the incumbent but not all the variable values are integer (or binary for binary problems). The way that is branched is defined by the variable selection policy.

Partitioning policy

As we discussed above the partitioning policy deals with all the issues related to branching a node of the tree. Based on the selected policy it is set the condition based on which the branching will take place. There are several partitioning policies to select from. The selection depends on the problem type and often is more empirical than rule based.

Variable Dichotomy

Suppose that solving the relaxed linear problem returns an optimal solution where not all variables are integer. The idea in variable dichotomy is to create for each non integer variable x_k two branches, one corresponding to the ceiling value of the variable$\lceil x_k \rceil$, let it be d+1 and the other to the floor value$\lfloor x_k \rfloor$, d. The linear problem of each branch will be updated by adding a new constraint, $x_k \leq d$ and $x_k \geq d+1$. We then repeat the procedure for each of the two linear problems obtained(Dakin, 1965).

Generalized-Upper-Bound Dichotomy (GUB)

Suppose that solving the relaxed linear problem returns an optimal solution where there are one or more fractional variables and the initial problem contains the constraint$\sum_{t\in Q} x_j = 1$, then exist Q_1 andQ_2 such that $Q = Q_1 \cup Q_2$ and$\sum_{t\in Q_1} x_j = 1$,$\sum_{t\in Q_2} x_j = 0$, (Beale and Tomlin, 1970). This way each time we branch by creating two sets of possible values, all variables that can get the value 0 go to the Q_2 set and the rest to the Q_1 set.

Multiple branches for bounded integer variable

If the optimal solution computed by solving the relaxed linear problem contains fractional variables $x_j \in \{0, \dots, l\}$ then $l + 1$ equations and corresponding branches can be created. Each branch will have$x_j = k$ where $k = 0, .., l$ (Land and Doig, 1960) . Actually, this method creates branches for each fractional variable. The number of branches per variable is defined by all the possible integer values that the variable can get. It is very slightly better than the full enumeration of the possible solutions.

Node selection policy

Node selection policy refers to the way that the next node to be visited in the B&B tree will be selected. The selected node contains the next sub-problem that will be solved in the process to find a global optimum for the initial problem. It is a critical policy due to the fact that strongly affects the computational time of the algorithm. Many general and problem specific policies have been proposed in the literature. Here in we present the most commonly used both due to their low computational complexity and their good results in practical size problems.

Breadth-First Search method

The Breadth-First Search method is a FIFO way of traversing a tree, since we examine each level's nodes and then continue to the next level, starting from the top of the tree and moving towards the leafs.

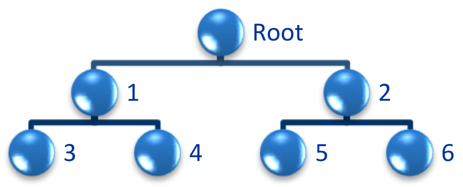


Figure 5 BFS Search in Binary Tree

Depth-First Search with Backtracking method

This method examines in depth the tree, branch by branch and each time that encounters a leaf node goes to the closest unexplored node till no unexplored nodes have been left. Starting from the root node selects a child node and examines it; the process is repeated till no unexamined children exist. Then it goes back to one or more past levels to find the next unexamined node and start the same process again, essentially it is a LIFO method, as shown in Figure 5.

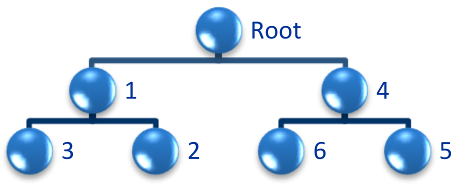


Figure 6 DFS Search in Binary Tree

Best-Bound method

This policy examines the linear objective value of the nodes and chooses the one that has the best value. All nodes that have linear objective value less than the incumbent will be fathomed. It is based on the principle that if the value of the linear problem is less than the incumbent then no integer value of the variables will give a greater result and thus there is no need to examine its children nodes. This policy minimizes the total number of explored nodes.

Sum of Integer Infeasibilities method

The sum of integer infeasibilities at a node is calculated as:

$$s_i = \sum \min\left(x_j - \lfloor x_j \rfloor, 1 - (x_j - \lfloor x_j \rfloor)\right)$$

Based on this policy the node to be visited is the one having either minimum sum of infeasibilities in the maximization problem or maximum sum in the opposite case. If the solution is feasible the sum of the infeasibilities is equal to zero and no further branching of the node is needed.

Variable selection policy

After having defined how are we going to examine the nodes we have to define the rules based on which from a node we will create children nodes. Clearly it is a critical the choice of branching variables and obviously affects the running time of the algorithm. Many different approaches have been developed and tested on different types of integer programs. Some common approaches are listed below.

Driebeck-Tomlin Penalties(Driebeek, 1966)/ (Tomlin, 1971)

Penalties give a lower bound on the degradation of the objective value when the under examination node will be branched. The penalties are the cost of the dual pivot needed to remove the fractional variable from the basis. Once the penalties have been computed, a variety of rules can be used to select the branching variables. A penalty can be used to eliminate a branch if the LP objective value for the parent node minus the penalty is worse than the incumbent integer solution. Penalties are out of favor because their cost is considered too high for their benefit.

Pseudo-Cost Estimate (Benichou et al., 1971)

Pseudo-costs provide a way to estimate the degradation to the objective value by forcing a fractional variable to an integral value. Pseudo-costs attempt to reflect the total cost, not just the cost of the first pivot, as with penalties. This is a sophisticated rule in the sense that it keeps a history of the success of the variables on which already has been branched. Pseudo-costs are not considered to be beneficial on problems where there are large percentages of integer variables.

Pseudo-Shadow Prices (Land and Powell, 1979)

Similar to pseudo-costs, pseudo-shadow prices estimate the total cost to force a variable to an integral value. The branching variable is chosen using criteria similar to penalties and pseudo-costs.

Strong Branching (Applegate et al., 1995)

The idea of Strong Branching is to test which of the fractional candidates gives the best progress before actually branching on any of them. This test is done by temporarily introducing a lower bound and subsequently an upper bound for the examined variable with fractional LP value, and solving the corresponding linear relaxations.

Most/Least Infeasible Integer Variable (Brunetta, Conforti and Fischetti, 2000)

In this approach, the integer variable whose fractional value is farthest from (closest to) an integral value is chosen as the branching variable.

Priorities Selection (Smith, 2004)

Variables are selected based on their priorities. Priorities can be user-assigned, or based on objective function coefficients, or on pseudo- costs. This policy strongly depends on the kind of problem to be solved and has been used mainly in telecommunications problems.

3 The Optimal Solution for computing the frequency of service in retail distribution networks

3.1 Introduction

This problem was initially introduced by Asimakopoulos (2006) as an Undergraduate thesis developed by the DeOpSys team of the University of the Aegean. The main idea is to define the frequency of service over a given set of customers and fleet of vehicles in order to maximize the total profit.

To define the frequency of service we need to know the product needs of each customer assigned to the route, or in other words the demand. In our case, the demand is not a constant number but it is proportional to the number of visits to the client. To be more precise, we have a minimum demand for each client and the possibility of extra sales after the first visit. However, there is an upper limit to the possible number of visits to each client during a specific time window. This upper limit is given as input.

This limitation arises from the fact that there is a specific amount of available products to sale and at the same time when the rate of demand reaches its peak all next visits to the same client will not add any surplus of profit. This relationship between the frequency of service to a customer and the rise of the demand is shown in Figure 6.

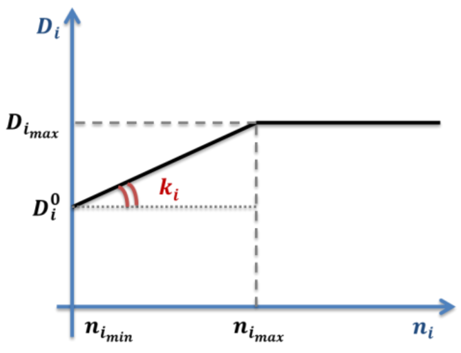


Figure 7 Relation of demand to visits per client

The following assumptions that form our operating scenario are considered:

- If the number of visits equals the minimum possible, $n_{i_{min}}$ then the demand isD_i^0that is the initial demand of this client covering the customer's needs for the referred time frame.
- Each subsequent visit will cause an increase on the demand $D_i$with a standard slopek_i.

- When a customer has been serviced for the maximum required number of visits, i.e.$n_{i_{max}}$, then the demand $D_{i_{max}}$ would be the maximum possible for that specific customer and time frame.
- All subsequent visits after the $n_{i_{max}}$ will not affect the demand.

The objective is to define the number of visits per client in order to maximize the overall profit in a specific time horizon (e.g. 5 days). Figure 7 represents the period and incremental income from a client i in relation to the number of visits. The following assumptions are considered regarding the incomes involved:

- Let $\mathrm{I_i}$ denote **the total income** of a specific client in a predefined time period. The value of this income is proportional to$\mathrm{D_i}$.
- Thus, $\mathrm{I_i = pD_i}$ where p is the profit parameter.
- **Incremental incomes** are given by the equation: $I_i^0 = pD_i^0$.
- When $n_i \leq n_{i_{max}}$ then $I_i^0 = pk_i$.

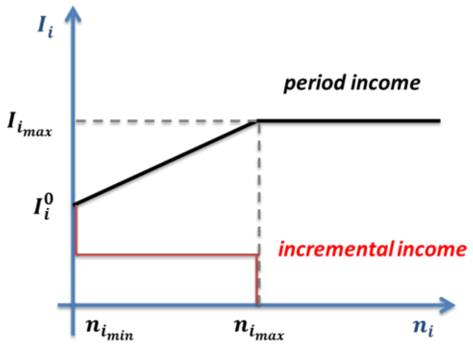


Figure 8 Relation of Income to Visits

From the above, it is obvious that by visiting each customer $n_{i_{max}}$times the sold goods are maximized and thus the profit. However visiting each customer $n_{i_{max}}$ times is not easy to realize when there is limited number of available vehicles and there are time constraints.

Based on the above considerations, the following section describes a simplified version of the problem, focusing on the function that describes the profit and the limitations that are set by the problem constraints. Furthermore, the mathematical formulation of the problem will be presented and analyzed.

3.2 Description of the Problem

Let N be a network consisting of a set of nodes$V = \{0,1,\dots,n\}$, being represented by a graph G and A a set of links that interconnect the nodes, which we will call edges from now on. The set of nodes represents the customers of the retail network that should be visited with the exception of node 0 that represents the depot that is the base of the fleet of vehicles. Every edge $(i, j) \in A$ is related to the cost c_{ij} to go from node i to node$j, \forall\, i, j \in V$ except when $i = j$that$c_{ij} = 0$. Let T be the time window being analyzed and n_d the number of time periods in which is divided. Based on the duration of each time period is defined the maximum distance C that can be driven in that time period by any track of the fleet.

Definitions

Graph $G = (V, A)$	describing the retail network
V	is the set of vertices, representing all the customers of the network plus the depot
A	is the set of edges, representing the possible routes for going from one customer to the other or the depot
$V_u = V\backslash\{0\}$	represents all the customers of the retail network
T	is the time window horizon
$d \in \{1,2,\dots,n_d\}$	Is a time period of T. Each route should be completed in one time period and not more.
i	Is a customer represented by a node of graph G, $i \in V_u$, and thus a selling point where goods are delivered.
p	Is a unitary profit assigned to each unit of goods.
c_{ij}	Is the cost to go from node i to node$j$, $\forall\ i,j \in V$ or in other words the cost of the vehicle to drive from customer i to customer$j$.
n_i	Is the number of times that the customer i will get delivery of goods during a specific time window, T.
$n_{i_{max}}$	Is the maximum number of visits to customer i, after which no extra profit will be earned.
$n_{i_{min}}$	Is the minimum number of visits to customer i, to keep the required service level.
D_i^0	Is the demand corresponding to $n_{i_{min}}$ visits to the customer.
D_i^d	Is the demand of customer i during time period$d$.
k_i	Is the increase of demand rate, for each visit after the minimum number of visits.
C	Is the maximum total cost that can be spent during a time period.
Q	Is the capacity of the vehicle in terms of units of sellable goods.
$x_{ij}^d = \begin{Bmatrix} 1 \\ 0 \end{Bmatrix}$	$if\ i \rightarrow j\ \in route\ for\ time\ section\ d$
	$otherwise$

$y_i^d = \begin{Bmatrix} 1 \\ 0 \end{Bmatrix}$	$if customer\ i\ will\ be\ served\ during\ time\ section\ d$
	$otherwise$

Objective function

The following model has been developed by the DeOpSys team of the University of the Aegean (Asimakopoulos, 2006). The objective function aims at the optimization (i.e. maximization) of the total profit gained by delivering goods to the customers during a specified time window and within the overall planning horizon (e.g. shift). Thus, the objective function is formed by two main parts and can be written in a very simple form as follows:

$$\max(profit) = \max(income - cost)$$

Assuming that cost is denoted as c_{ij} and there is a decision variable x_{ij} that represents the product of distance between customer $i$and customer j and the cost per unit of distance, then the total cost can be seen as:

$$cost = \sum_{d=1}^{n_d} \sum_{i \in V} \sum_{\substack{j \in V \\ i \neq j}} c_{ij} x_{ij}^d$$

Let D_i^0 be the initial demand of customer i,y_i^d a binary decision variable that gets as value 1 when the customer i will be visited the day d, $n_{i_{min}}$ the minimum number of visits to customer i, the raise of demandk_i, after the first $n_{i_{min}}$visits and p the profit per unit, then the income part of the objective can be stated as:

$$income = \sum_{i \in V_u} \left[D_i^0 + \left(\sum_{d=1}^{n_d} y_i^d - n_{i_{min}} \right) k_i \right] p$$

Consequently, the mathematical expression describing this objective is presented below:

$$max \left[\sum_{i \in V_u} \left[D_i^0 + \left(\sum_{d=1}^{n_d} y_i^d - n_{i_{min}} \right) k_i \right] p - \sum_{d=1}^{n_d} \sum_{i \in V} \sum_{\substack{j \in V \\ i \neq j}} c_{ij} x_{ij}^d \right]$$

Now by breaking down each of the two parts of the objective function we can get a more detailed view of its components.

Firstly, to compute the incomes we need to know the number of visits $\sum_{d=1}^{n_d} y_i^d$that will be effectuated to each customer, during the time window and how many of those are above the minimum, $\sum_{d=1}^{n_d} y_i^d - n_{i_{min}}$. The two cases are distinguished because on those visits that are above the minimum we should apply the incremental factor k_i to get the extra sales that we will due to visiting the customer more times than the essential. Thus the sum of incremental incomes and period incomes will be:

$$\left[D_i^0 + \left(\sum_{d=1}^{n_d} y_i^d - n_{i_{min}}\right) k_i\right] p = D_i^0 p + \left(\sum_{d=1}^{n_d} y_i^d - n_{i_{min}}\right) k_i p$$

This way we have computed the incomes from one customer and all the days of the time window in question. By applying$\sum_{i \in V_u}$, we can calculate the total income for all the customers of the retail distribution network that is being examined.

Secondly, in order to compute the total cost we need to compute the cost of going from node i to node j multiplied to the existence or not of this edge in the route of the specific time period as$c_{ij} x_{ij}^d$, then compute it for all route edges ending at node$i$ by findint the$\sum_{\substack{j \in V \\ i \neq j}}$ and the same for all edges starting from node i as $\sum_{i \in V}$ and finally computing the total of all time period comprising the specified time window as$\sum_{d=1}^{n_d}$.

Constraints

$\sum_{j \in V, i \neq j} x_{ji}^d = y_i^d$	$\forall i \in V_u, \forall d \in \{1, \dots, n_d\}$	(3.1)
$\sum_{j \in V_u, i \neq j} x_{ij}^d = y_i^d$	$\forall i \in V, \forall d \in \{1, \dots, n_d\}$	(3.2)
$\sum_{j \in V} x_{0j}^d = 1$	$\forall d \in \{1, \dots, n_d\}$	(3.3)
$\sum_{j \in V} x_{j0}^d = 1$	$\forall d \in \{1, \dots, n_d\}$	(3.4)
$\sum_{i \in S} \sum_{j \in S} x_{ij}^d \leq \sum_{i \in S} y_i^d - 1$	$\forall S \text{ subset of } V_u, \forall d \in \{1, \dots, n_d\}$	(3.5)
$\sum_{i \in V} \sum_{j \in V, i \neq j} c_{ij} x_{ij}^d \leq C$	$\forall d \in \{1, \dots, n_d\}$	(3.6)
$n_{i_{min}} \leq \sum_{d=1}^{n_d} y_i^d \leq n_{i_{max}}$	$\forall i \in V_u$	(3.7)
$\sum_{i \in V_u} D_i^d y_i^d \leq Q$	$\forall d \in \{1, \dots, n_d\}$	(3.8)
$x_{ij}^d = \{0,1\}$		(3.9)
$y_i^d = \{0,1\}$		(3.10)

Constraint (3.1) specifies if the vehicle visits customer i during time period d then no other edge contained in the route will end to this node in the same time period. Next, constraint (3.2) ensure that If the vehicle visits customer i during time period d then no other edge contained in the route will start from this node in the same time period. Constraints (3.3) and (3.4)ensure that only one edge contained in the route of a specific time period starts from and ends to the depot. Constraint (3.5) specifies that each time period's route should not contain cyclic paths. Constraint (3.6) force that the total cost of a time period's route cannot be greater than the limit cost C set by the problem's initial data. Constraint (3.7)specifies that the number of visits to each customer should be between a minimum of $n_{i_{min}}$ and a maximum of $n_{i_{max}}$.The constraint (3.8) guarantee that no day the capacity of the vehicle is surpassed by the goods that should be delivered based on the designed route. Here should be noted that:

$$D_i^d = \begin{cases} D_i^0 / n_{i_{min}} & n_i \le n_{i_{min}} \\ D_i^0 + \left(\sum_{d=1}^{n_d} y_i^d - n_{i_{min}} \right) k_i n_{i_{min}} & \le n_i \le n_{i_{max}} \end{cases}$$

This constraint is not taken into consideration in the herein proposed solution. Finally, constraints (3.9) ensure that each edge $i \to j$ either it is contained to the route corresponding to time period d or not, thus the variables x_{ij} are binary and constraint (3.10) that each customer i either gets a delivery in time periodd or not, thus the variables y_i are binary.

3.3 Optimization Method

The above described problem is a binary linear programming optimization problem and our aim is to solve it using a low complexity and high velocity algorithm in order to get quick solutions in practical sized instances of the problem. Aiming at this goal the proposed solution combines branch and bound method with revised simplex and relaxation techniques. More specifically, the steps followed to solve any instance of the above problem are depicted below:

8. Get instance's input
   a. D_i^0:Initial demand per customer, which is the demand corresponding to $n_{i_{min}}$
   b. $n_{i_{min}}$ and $n_{i_{max}}$: the minimum and maximum allowed number of visits per customer.
   c. k_i: Rate of raise of demand for each visit to customer after the first $n_{i_{min}}$ visits.
   d. c_{ij}: Distance in km between each pair of customers per cost/km, usually the coordinates of each customer's location are given and the distance is computed based on the equation $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.
   e. p: The incomes from each visit to any customer.
9. Formalize the problem in order to acquire the form shown in previous section:

$$max\left[\sum_{i\in V_u}\left[D_i^0+\left(\sum_{d=1}^{n_d} y_i^d - n_{i_{min}}\right)k_i\right]p-\sum_{d=1}^{n_d}\sum_{i\in V}\sum_{\substack{j\in V\\ i\neq j}} c_{ij}x_{ij}^d\right]$$

$$under\ constraints:$$

$$\sum_{j\in V, i\neq j} x_{ji}^d - y_i^d = 0$$

$$\sum_{j\in V, i\neq j} x_{ij}^d - y_i^d = 0$$

$$\sum_{j\in V} x_{0j}^d = 1$$

$$\sum_{j\in V} x_{j0}^d = 1$$

$$\sum_{i\in S} y_i^d - \sum_{i\in S}\sum_{j\in S} x_{ij}^d \geq 1$$

$$\sum_{i\in V}\sum_{j\in V, i\neq j} c_{ij}x_{ij}^d \leq C$$

$$\sum_{d=1}^{n_d} y_i^d \leq n_{i_{max}}$$

$$\sum_{d=1}^{n_d} y_i^d \geq n_{i_{min}}$$

10. Transform the problem in its augmented form by adding slack, artificial variables and subtracting surplus variables as needed. Specifically, constraints 1,2,3,4 are equalities and thus an artificial variable will be added to each of them; constraint 6 and the second part of 7 are less equal and thus only a slack variable will be added for each one and all the rest are greater than and will be needed to subtract a surplus variable and add an artificial. Here is noted that whenever a negative value exists in the right part of the inequality, first we multiply it by -1 and then we start doing the above process. After all these transformations our optimization problem will have the following form:

$$max \left[\sum_{i \in V_u} \left[D_i^0 + \left(\sum_{d=1}^{n_d} y_i^d - n_{i_{min}} \right) k_i \right] p - \sum_{d=1}^{n_d} \sum_{i \in V} \sum_{\substack{j \in V \\ i \neq j}} c_{ij} x_{ij}^d \right] + A_1 + A_2 + A_3 + A_4 + A_5 + A_6$$

$$under\ constraints:$$

$$\sum_{j \in V, i \neq j} x_{ji}^d - y_i^d + \boldsymbol{A_1} = 0$$

$$\sum_{j \in V, i \neq j} x_{ij}^d - y_i^d + \boldsymbol{A_2} = 0$$

$$\sum_{j \in V} x_{0j}^d + \boldsymbol{A_3} = 1$$

$$\sum_{j \in V} x_{j0}^d + \boldsymbol{A_4} = 1$$

$$\sum_{i \in S} y_i^d - \sum_{i \in S} \sum_{j \in S} x_{ij}^d + \boldsymbol{A_5} - \boldsymbol{P_1} \geq 1$$

$$\sum_{i \in V} \sum_{j \in V, i \neq j} c_{ij} x_{ij}^d + \boldsymbol{S_1} \leq C$$

$$\sum_{d=1}^{n_d} y_i^d + \boldsymbol{S_2} \leq n_{i_{max}}$$

$$\sum_{d=1}^{n_d} y_i^d + \boldsymbol{A_6} - \boldsymbol{P_2} \geq n_{i_{min}}$$

$$x_i^d, y_i^d = \{0,1\}$$

11. Relax the initial problem by removing the constraint that demands the solution variables to be binary variables.
12. Solve the relaxed problem using Revised Simplex Method. Original simplex method calculates and stores all numbers in the tableau without needing all of them. On the other hand the Revised Simplex Method is more efficient for computing that does its heavy calculation at the beginning of each iteration. The goal of this method is the ordering of all calculations so that no unnecessary calculations are performed. It uses Gauss – Jordan to compute the inverse of a matrix and the rest are simple multiplications between matrices.

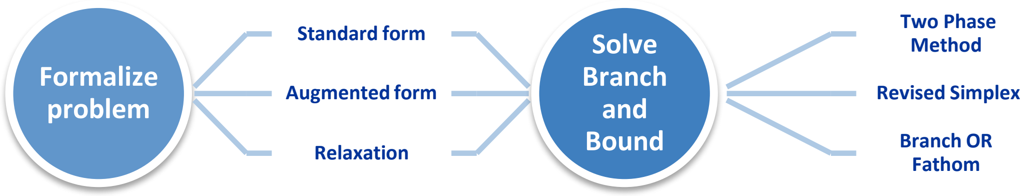


Figure 9 Solution Steps

13. Furthermore, in our case the problem always contains artificial variables and thus we need to either use Big M method or Two Phase to solve it. In order to have more accurate calculations through our software, the Two Phase method was selected. Using this method first a minimization problem using only the artificial variables is formulated and solved, if the result and all artificial variables get a zero value then the initial problem has feasible solution and all columns corresponding to artificial variables are dropped and the initial problem is solved using revised simplex otherwise the problem does not have a feasible solution.
14. Use the solution along with the relaxed problems as initial values for the branch and bound algorithm and start the branching and bounding process. This is done by defining the branching policy and fathoming policy. Branching policy, describes how is decided which branches should be created for each node, that in our case is to create for each binary variable with value different from 0 or 1 two branches, one for each case. Fathoming policy defines how is decided which branches should be fathomed based on the results that give, if the current solution is less that the incumbent or no feasible solution was found then the branch is fathomed.
15. If no solution has been found and we have reached a tree depth equal to the number of unknown variables in the objective function then return the current incumbent solution and terminate the execution.

4 Experimental Results

4.1 Introduction

In this section are presented the results of the implemented method when compared to the execution of the corresponding built-in libraries of MATLAB, as they are presented in Asimakopoulos (2006) thesis. To be more precise, the source code used in Asimakopoulos thesis was run with the same input data with the herein presented software tool and the same environment in order to get the comparison results. The goal is to prove that by implementing the Branch and Bound algorithm using a middle level language as C++, we get faster and more reliable results than using standardized high level development environments, as MATLAB. However, the ultimate target is to achieve solutions of practical sized problems in meaningful execution time.

The machine used to test the performance of the two applications has the following profile:

- Intel Core Duo T2300 (1.66 GHz, 667 MHz FSB, 2MB L2 cache)
- 2048 MB DDR 266
- Windows 7 Prof.

Performance was tested by counting CPU time for each routine of the algorithm and excluding user time and I\O operations. I\O operations were not included due to the different implementation of the two application, the one using input and output files and the other providing results exclusively to the display.

Furthermore, to get reliable results each set of input data were executed 100 times using each application and the average time was used as execution time, in order to avoid external interferences that could cause misleading results.

Here, should be noted that the source code given, as basis for the comparison, when run in MATLAB didn’t return any results for any case with more than 6 clients, thus the corresponding values from Asimakopoulos’ thesis were used for the comparison.

4.2 Results

4.2.1 2 - 4 Customers

Table 2 Two - Four Customers Results

	2 CUSTOMERS		3 CUSTOMERS		4 CUSTOMERS	
DAYS	V-RED (Sec)	MATLAB (Sec)	V-RED (Sec)	MATLAB (Sec)	V-RED (Sec)	MATLAB (Sec)
1	0,01	0,04	0,01	0,06	0,03	0,12
2	0,04	0,05	0,04	0,08	0,09	0,22
3	0,07	0,06	0,08	0,12	0,21	0,38
4	0,08	0,08	0,15	0,17	0,39	0,59
5	0,09	0,08	0,23	0,26	0,64	0,81

4.2.2 5 - 6 Customers

Table 3 Five - Six Customers Results

	5 CUSTOMERS			6 CUSTOMERS		
DAYS	V-RED (Sec)	MATLAB (Sec)	PROFIT (€)	V-RED (Sec)	MATLAB (Sec)	PROFIT (€)
1	0,07	3,43	2031,68	0,30	14,22	2831,68
2	0,38	65,54	3091,35	0,23	75,43	4703,35
3	0,99	145,55	4151,03	2,60	178,65	6575,03
4	1,88	287,54	5210,71	4,96	312,43	8446,70
5	3,15	642,86	6270,38	8,18	709,33	10318,38

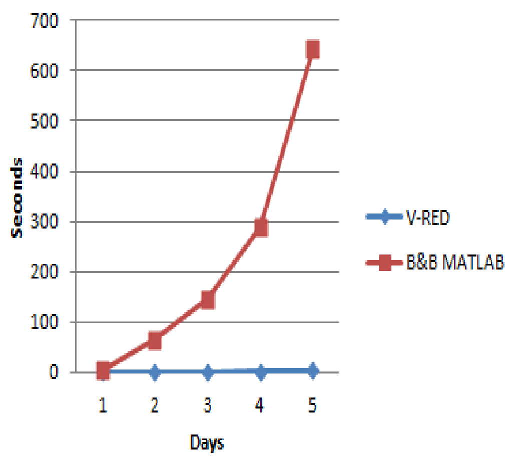


Figure 10 Execution Time Comparison 5 Customers

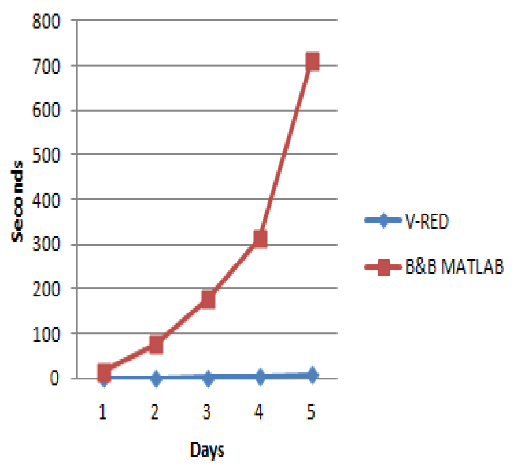


Figure 11 Execution Time Comparison 6 Customers

4.2.3 7 - 8 Customers

Table 4 Seven - Eight Customers Results

	7 CUSTOMERS			8 CUSTOMERS		
DAYS	V-RED (Sec)	MATLAB (Sec)	PROFIT (€)	V-RED (Sec)	MATLAB (Sec)	PROFIT (€)
1	0,85	25,56	3356,43	2,61	30,42	3576,00
2	3,04	87,64	5612,32	6,65	98,62	6018,01
3	5,87	189,54	7534,78	18,60	204,45	8460,01
4	12,88	349,53	9145,65	27,32	379,73	10902,01
5	20,87	789,54	12126,03	29,14	986,42	13344,02

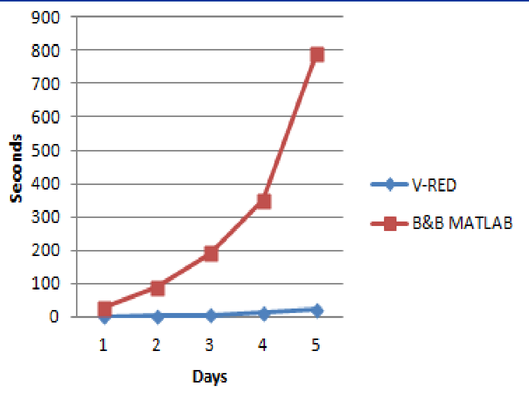


Figure 12 Execution Time Comparison 7 Customers

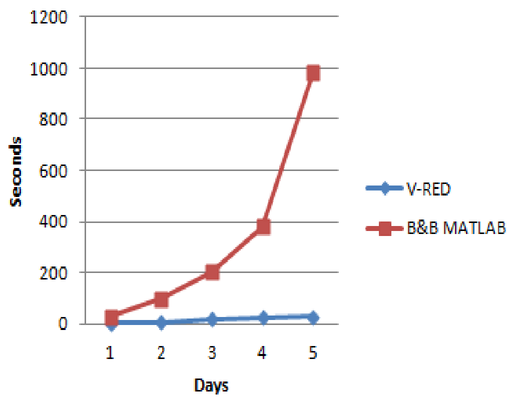


Figure 13 Execution Time Comparison 8 Customers

4.2.4 Solution of new instances of more than 8 Customers

Table 5 Nine - Twelve Customers Results

	9 CUSTOMERS		10 CUSTOMERS		11 CUSTOMERS		12 CUSTOMERS	
DAYS	TIME (Sec)	PROFIT (€)	TIME (Sec)	PROFIT (€)	TIME (Sec)	PROFIT (€)	TIME (Sec)	PROFIT (€)
1	4,40	3920,33	10,20	4220,33	30,80	4795,14	98,50	5487,10
2	12,20	6716,66	35,20	7316,66	120,60	8270,27	444,10	14914,20
3	30,60	9512,99	93,40	10412,99	245,90	11745,41		
4	54,90	12309,32	158,40	13509,32	406,60	14845,74		
5	93,80	15105,65	235,00	16487,65				

4.3 Execution Time versus Number of Clients

It was experimentally proven that the best results for the proposed branch and bound method are given when using the new implementation of the proposed method.

Comparing the above described implementations, the new implementation method, seems to need significantly less computational effort and it provides better computational results even when the number of clients is increased. Moreover, tests have been done till 12 clients and no hanging or long waiting for the results was verified. Further research and appropriate, more complex, test cases are needed to verify the upper bound of the problem's size that can be handled by this implementation.

5 Software Development

5.1 Introduction

In the previous sections the theoretic basis of a software tool capable of solving maximization problems that can be described by chapter's four equations was in detail described. In this chapter we will focus on the software engineering process used to implement the software. More specifically, we will describe the used software engineering methods, in other words the structured approaches to software development which include description of the system through graphical models along with all the rules and constraints applied to the system's model.

The herein described software will be analyzed based on the following axes:

1. **Specification**: what the system should do and its development constraints.
2. **Development**: production of the software system.
3. **Validation**: checking that the software is what the customer wants.

The software model used to develop this system is called Component-based and it stands on the ground that the system is composed by components that are developed independently and represent specific aspects and functionality of the system and following are created interfaces to make the components interact to each other and get the integrated system (Sommerville, 2007).

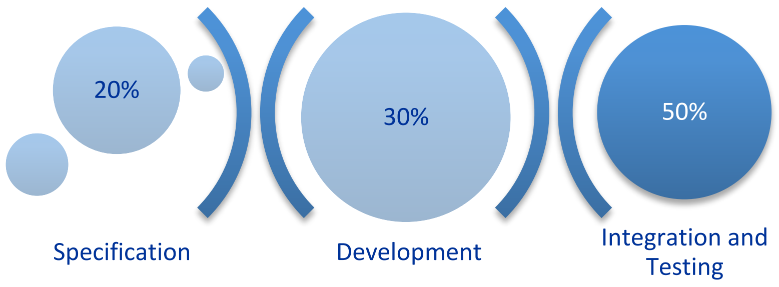


Figure 14 Time spent for each stage of component based engineering

In this case, as shown in Figure 13, after the specification of the requirements and the development of individual components, there is a great amount of time spent in the components integration and the system's testing.

Final aim is to create a good software tool and by saying that we mean that it should deliver the required functionality and performance to the user and should be maintainable, dependable and acceptable. More analytically, maintainability is needed in order to have a tool that can easily be updated so to evolve to meet changing needs; depend ability so its computational steps and results can be trustworthy and efficiency because it should not make wasteful use of system resources (Xu, 2006).

In the following sections will be described the step by step process that was followed in order to create the desired software; starting from the initial requirements given by the mathematical problem itself, the requirement analysis needed to find out what was the functional specification that would fulfill the end user's needs, and finally the system's design and architecture described using Unified Modeling Language (UML) models.

5.1.1 Use Case Model

Use-cases are scenario-based techniques in the UML which identify the actors in an interaction and which describe the interaction itself. A set of use cases should describe all possible interactions with the system.

More specifically, a use case is a technique for documenting the potential requirements of a new system or software change. Each use case provides one or more scenarios that convey how the system should interact with the end user or another system to achieve a specific business goal. Use cases typically avoid technicalities, preferring instead the language of the end user or domain expert. Use cases are often co-authored by requirements engineers and stakeholders(Zündorf, 2001).

A use case contains a textual description of all of the ways which the intended users could work with the software or system. Use cases do not describe any internal workings of the system, nor do they explain how that system will be implemented. They simply show the steps that a user follows to perform a task(MacKinnon, 2003). All the ways that users interact with a system can be described in this manner.

In the herein presented case each basic step of the process correspond to a use case and there are both unilateral and bilateral communications among the components, as shown in Figure 14.

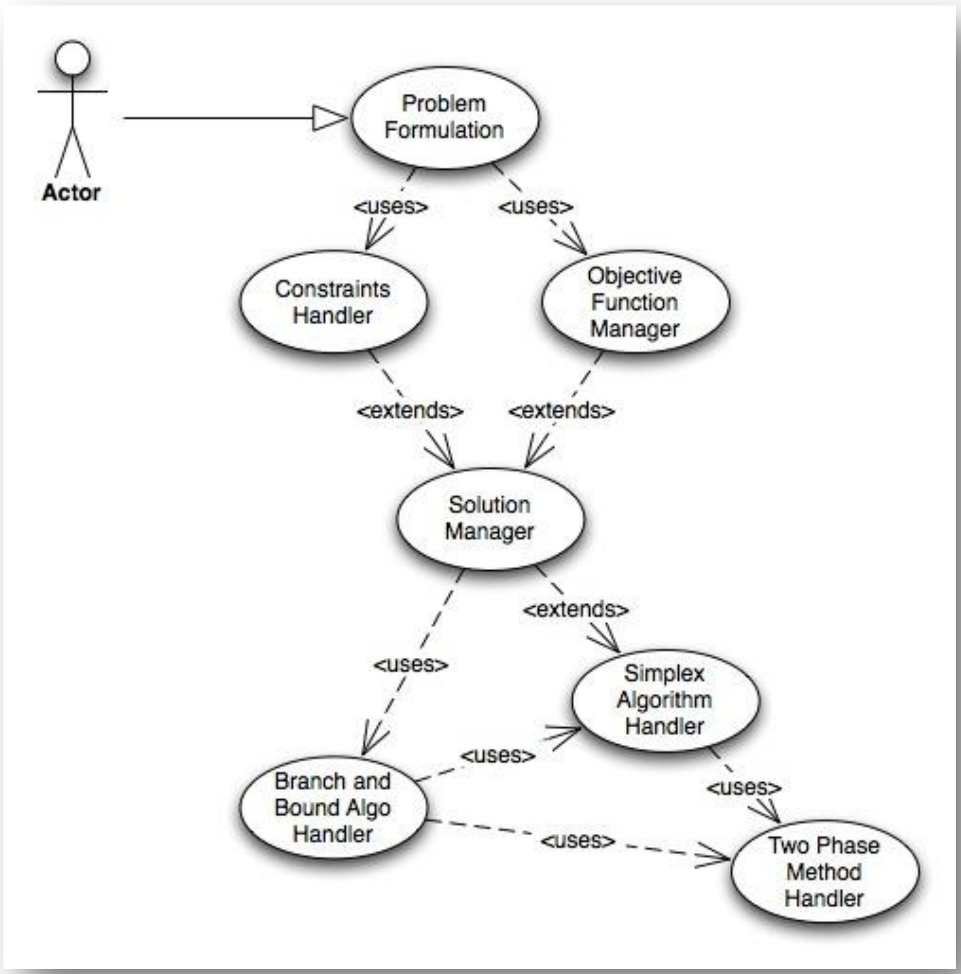


Figure 15 Use Case Model

The system aims in computing the optimum frequency of service in retail distribution networks and in order to achieve this employs the following use cases:

- Problem Formulation: Responsible for handling inputs and based on the given mathematical formulas transforming those inputs to a valid objective function and constraints.
- Constraints Handler: It is used by the problem formulation and handles each constraint in order to dynamically create all the relevant to the specific case constraints.
- Objective Function Handler: It transforms the generic mathematic formula of the objective function to an instance specific equation to be used in the following steps.
- Solution Manager: It is the orchestrator of the optimization process which calls the rest of the components when and if needed.

- Branch and Bound Algorithm Handler: It handles the creation of branches and the fathoming of those that are not helpful, given an initial solution of the relaxed linear problem.
- Simplex Algorithm Handler: It is a standalone component used to solve general optimization problems using the simplex algorithm, in cases that inequalities with greater than or equalities exist in the given constraints, it is extended by the Two Phase Method Handler.
- Two Phase Method Handler: It is used by the Simplex Algorithm Handler in order to transform the optimization problem to its standard form.

5.1.2 System's Modules

The herein described software was designed following the modular design technique. The re-usable components were taken apart and set to be modules with corresponding interfaces and a coordinator module was used to orchestrate the process and provide the logic sequences and flow conditions, as shown in Figure 15.

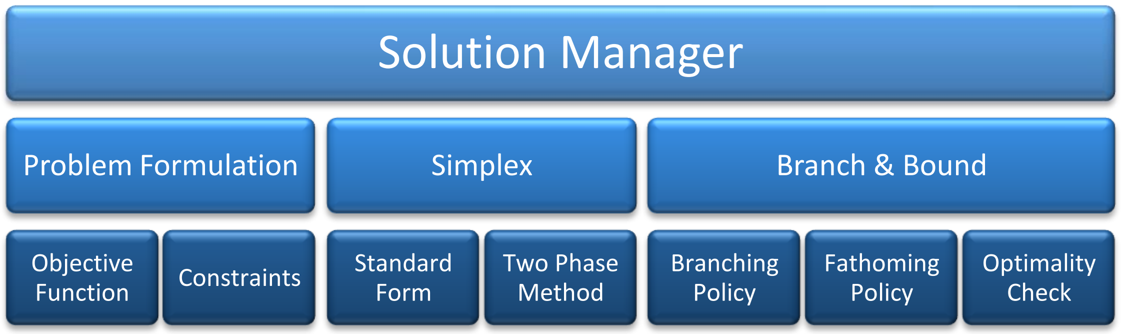


Figure 16 System's Modules

Essentially, we have on top a **Solution Manager** module that handles I\O and calls the rest of the modules as needed based on the results that each of them returns and on the coded logic of the system.

Then we have three main components used to implement the optimization method concerning the frequency of service in retail distribution networks. The first one is the **Problem Formulation** module that gets from Solution Manager the initial inputs and creates all the internal structures needed to represent the problem in a manner that the rest of the components can handle it. More precisely, this component uses two other modules the **Objective Function** and **Constraints** modules to achieve its goals.

The **Objective Function** module is responsible for creating an array line containing the objective function coefficients of the specific problem's instance along with an array containing the corresponding variable names.

The **Constraints** module does the same for all the problem's constraints, saving them to two arrays, one used for less than inequalities and one for equalities and greater than inequalities. Again there is a vertex for the corresponding basic variables that will be needed by the simplex module.

The **Simplex** module, as expected, implements the simplex algorithm, supported by two sub-modules **Standard Form** and **Two Phase Method.** The first one is used for problems that are in general standard form whether the second one is used when greater than and equalities constraints exist.

Finally, the **Branch and Bound** module uses three sub-modules, the **Branching Policy**, **Fathoming Policy** and **Optimality Check** to implement the well-known algorithm. The sub-modules are pretty much self-explanatory, **Branching Policy**, used to decide which branches should be created for each node, **Fathoming Policy** used to decide which branches should be fathomed based on the results that give, and **Optimality Check** that handles the comparisons of current solution to incumbent and the termination rules, in the case that no solution has been found and we have reached a tree depth equal to the number of unknown variables in the objective function then returns as result the current incumbent solution and terminates the execution.

5.1.3 Sequence Diagram

UML sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes(Glinz, 2000).

Sequence diagrams are typically used to model usage scenarios. Sequence diagrams are typically used to model usage scenarios. A usage scenario is a description of a potential way your system is used. The logic of a usage scenario may be part of a use case, perhaps an alternate course. It may also be one entire pass through a use case, such as the logic described by the basic course of action or a portion of the basic course of action, plus one or more alternate scenarios(Bernardi, 2002).

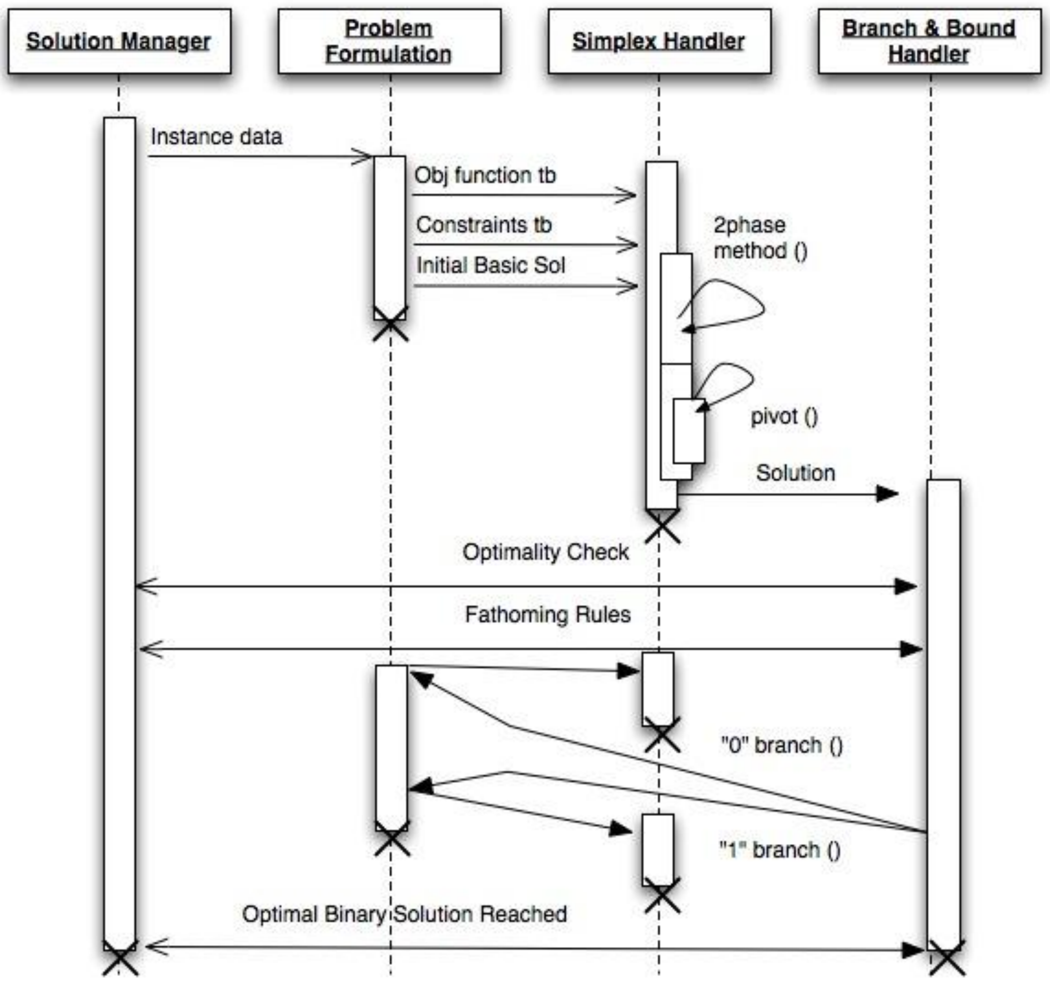


Figure 17 System's UML sequence diagram

In Figure 16 is depicted the main flow of events for the system in a generic usage scenario. As shown the Solution Manager initiates the process by posting the input to the Problem Formulation and receives the partial results each time that Branch and Bound object checks for optimality.

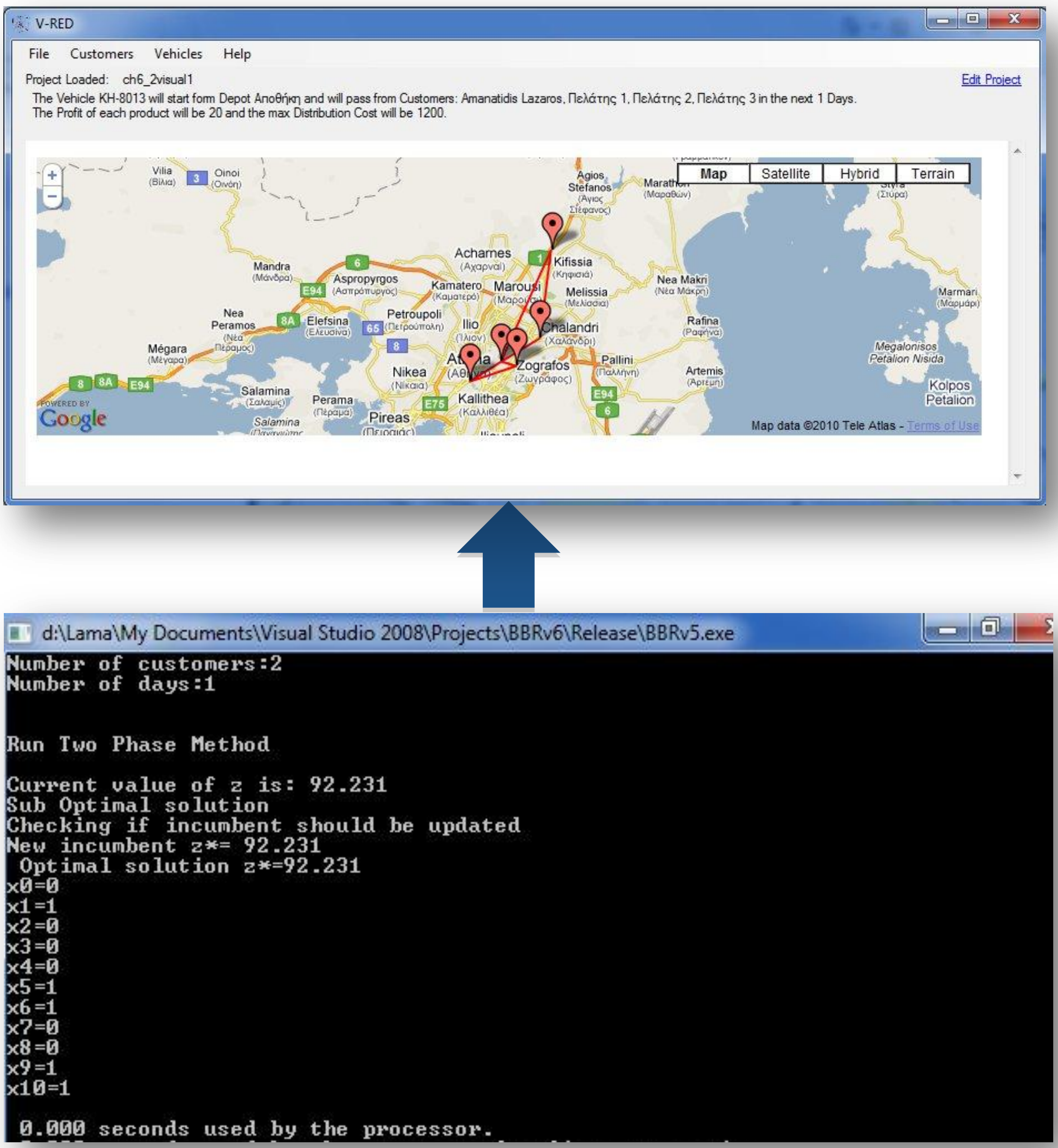
The Problem Formulation problem is used once per problem solved in order to transform the given input to specific input data for the rest of the objects. Essentially, based on the given mathematical model are created the appropriate objective functions and constraints and saved in arrays in order to be easily handled.

Furthermore, the Simplex Handler is called for the solution of the initial relaxed problem and each time that a new branch is created in order to generate the solutions. Each solution is either set to be incumbent in case that is greater than the current incumbent and fulfills the binary requirement for all the variables or it is discarded through the fathoming policy of Branch and Bound Object.

Finally, the Branch and Bound object besides checking each time the optimality of the given solution creates the branches and validates the status for each node and validates whether it should be fathomed or not. However, it is the main component of the system and interacts with the Solution Manager till the end of the process either it concludes with finding an optimal solution or by reaching the maximum depth of the Branch and Bound tree.

5.2 Software Description

The V-RED application that is the software tool developed to compute the optimal frequency of service in retail distribution networks, is a standalone application coded in Visual C++ and with an add-on interface developed in Visual C#, to make the whole process more user friendly.



Project Loaded: ch6_2visual1

The Vehicle KH-8013 will start form Depot Αποθήκη and will pass from Customers: Amanatidis Lazaros, Πελάτης 1, Πελάτης 2, Πελάτης 3 in the next 1 Days.
The Profit of each product will be 20 and the max Distribution Cost will be 1200.

d:\Lama\My Documents\Visual Studio 2008\Projects\BBRv6\Release\BBRv5.exe

```
Number of customers:2
Number of days:1


Run Two Phase Method

Current value of z is: 92.231
Sub Optimal solution
Checking if incumbent should be updated
New incumbent z*= 92.231
 Optimal solution z*=92.231
x0=0
x1=1
x2=0
x3=0
x4=0
x5=1
x6=1
x7=0
x8=0
x9=1
x10=1


 0.000 seconds used by the processor.
```

Figure 18 V-RED Interface & Console Application

As shown in Figure 17, the V-RED application initially was developed as a console application where input data were given through the keyboard and output was returned to the display. Subsequently, was added on the top level a window based interface to simplify the input of data and make more descriptive the results that the system generates. This was achieved adding a map based system for adding customers and their positions and the same was done for the presentation of the optimal route along with its expected cost. Essentially, in both cases the core implementation it’s the same and there are no time consuming process because the C++ code used in the standalone version is used as an embedded CLR in the C# code, making the application to have the exact same performance. However, here we will focus on the console application in order to describe the core system functionality and not the features of the interface.

The V-Red application has an object oriented design, thus the components are objects that communicate through interfaces and react to the specifics of the given problem instance.

There are 3 main objects that constitute the backbone of the whole system and several minor in order to implement needed functionality. Here we have to note that no libraries were used for the implementation of either the Simplex or the Branch and Bound Algorithm. The purpose of that was to have full access to each step and ability to optimize the implementation of these algorithms based on the specific problem's features without having to load and handle libraries that are generic and thus would bring heavy computational effort. Consequently, there is the core object that implements the Simplex and Two Phase Algorithm, the object that implements the Branch and Bound Algorithm and the Problem Formulation object that is responsible for transforming the input data to a valid set of constraints and an objective function. On top of those there is the Solution Manager that calls as needed the rest of the objects and returns to the end user intermediate results, error messages and\or the final solution.

5.3 V-RED Algorithm

In this section the algorithm used to implement the V-RED core application will be presented. Furthermore, a graphical view of the algorithm is shown in the Flow Chart of Figure 19. In order to make this section easier to read initially the backbone of the system is presented as a basic algorithm and then each component is analyzed through its own algorithm.

Begin V-RED Application

```
1. Read input data from file
2. Call Problem Formulation
    2.1. For each customer
        2.1.1. Read D, $n_{min}$, $n_{max}$, k , p, (x,y)
    2.2. Read depot coordinates ($x_0$,$y_0$)
    2.3. For each customer i and the depot
        2.3.1. For each customer j and the depot
        2.3.2. Compute distance $c_{ij}=\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$
    2.4. Construct Objective Function
    **2.5. Call Constraint #1- Constraint #10**
    2.6. Save data to ***InitiaProblem.txt*** file
    2.7. Call Branch and Bound (InitialProblem.txt)
3.  Printout Error Messages
4.  Printout Results
End V-RED Application
```

5.3.1 Constraints

Constraint #1$\sum_{j\in V, i\neq j} x_{ji}^{d} = y_i^{d}$

```
1.  For each day d
    1.1. For each customer i
         1.1.1.  For each customer j
                 Sum+= X[d][j][i]
         Sum -Y[d][i]=0
```

Constraint #2$\sum_{j\in V_u, i\neq j} x_{ij}^{d} = y_i^{d}$

```
1.  For each day d
    1.1. For each customer i
         1.1.1.  For each customer j
                 Sum+= X[d][i][j]
         Sum -Y[d][i]=0
```

Constraint #3$\sum_{j\in V} x_{0j}^{d} = 1$

```
1.  For each day d
    1.1. For each customer j
         Sum+= X[d][0][j]
    Sum-1=0
```

Constraint #4$\sum_{j\in V} x_{j0}^{d} = 1$

```
1.  For each day d
    1.1. For each customer i
         Sum+= X[d][i][0]
    Sum-1=0
```

Constraint #5$\sum_{i\in S}\sum_{j\in S} x_{ij}^{d} - \sum_{i\in S} y_i^{d} \leq 1$

```
1.  Compute combinations of possible S
2.  For each day d
    2.1. For each item i of set S
         2.1.1.  For each item j of set S different from i
```

```
            SumPartA+=x[d][i][j]
      SumPartB+=y[d][i]
SumPartA[d]-SumPartB[d]<=1
```

Constraint #6 $\sum_{i \in V} \sum_{j \in V, i \neq j} c_{ij} x_{ij}^d \leq C$

```
1. For each day d
   1.1. For each customer i
        1.1.1. For each customer j
               Sum C[i][j]*x[d][i][j]<=C
```

Constraint #7 $n_{i_{min}} \leq \sum_{d=1}^{n_d} y_i^d$

```
1. For each day d
   1.1. For each customer i
   Y[d][i]>=nmin[i]
```

Constraint #8 $\sum_{d=1}^{n_d} y_i^d \leq n_{i_{max}}$

```
2. For each day d
   2.1. For each customer i
   Y[d][i]<=nmax[i]
```

Constraint #9 Binary variables

This constraint is initially relaxed to having all variables between 0 and 1 to start the Branch and Bound Algorithm

5.3.2 Optimizer

```
Begin Optimizer (InitialProblem.txt)
   1.1. Check for =, >= constraints
        1.1.1. If yes Call Two Phase Method
        1.1.2. Else Call Simplex Method
   1.2. RETURN
End Optimizer
```

5.3.3 Two Phase Method

```
Begin Two Phase Method
Begin Phase 1
    1.1. Create new objective function z=-Sum of(Artificial Var) and same constraints
    1.2. Call Simplex Method
    1.3. If method returns non FS solution or solution different from zero then return
         non FS and Error Message “Phase 1 Failed”
         Else if method returns solution equal to zero
        1.3.1. Check if there are artificial variables in base
            1.3.1.1.1. If yes then for each one check if the corresponding
                       right side is zero
                1.3.1.1.1.1. If yes then Call Pivot to get them out of base
                             Else if none is in the base Call Phase 2
                       Else return non FS and Error Message “There are non-zero
                       artificial variables in base”
End Phase 1
Begin Phase 2
    2.1. Replace Objective function with initial and keep table of Phase 1
    2.2. Call Simplex Method
    2.3. Get Results from Simplex Method
    2.4. RETURN
End Phase 2
End Two Phase Method
```

5.3.4 Simplex Method

```
    Begin Simplex Method
    1.1. If the line corresponding to objective function has not any negative values
         optimal solution found RETURN
         Else if maximum number of allowed pivots reached STOP
         Else select the first most negative value and mark its column as pivot column
    1.2. Compute θ-ratio and mark the line with minimum value and positive divisor
         as pivot line
    1.3. Call Pivot for the item that corresponds to (marked line, marked column)
         Begin Pivot
        1.3.1. Divide pivot line by pivot element
        1.3.2. For each line besides the pivot line compute new values as New line =
               Old line – Pivot Item*Old line
        End Pivot
    1.4.  Go to 1.1
    End Simplex Method
```

5.3.5 Branch and Bound Method

The idea is to create a tree starting from the root node that is the initial relaxed problem and each time that is solved, make two new branches one giving a 0 value (left branch) and one giving the value 1 (right branch) to the first variable of the problem that is not in binary form. Before creating the branches we check if the currently examined node produced a non-feasible solution thus we need to fathom the branch or the solution found is greater than or equal to the incumbent and in that case the incumbent should be updated and again the branch fathomed.

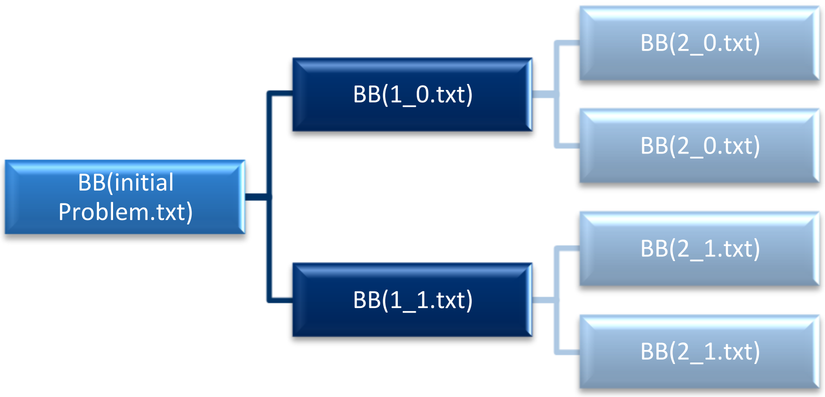


Figure 19 Branch and Bound Calls

```
Begin Branch and Bound Method (Input.txt)
1.1. Call Optimizer(Input.txt) // read input data from file
1.2. Check optimality of current solution (returned by 1.1)
    1.2.1.  If all variables binary
        1.2.1.1.  Check incumbent and update if current >= incumbent
        1.2.1.2.  Fathom Branch
    1.2.2.  Else
        1.2.2.1.  Add new constraint =0// create left branch
        1.2.2.2.  Update Input File to "$count_0.txt"//save new problem's data
        1.2.2.3.  Call Branch and Bound ("$count _0.txt")//repeat algorithm
                  Add new constraint =1
        1.2.2.4.  Update Input File to "$count _1.txt"//save new problem's data
        1.2.2.5.  Call Branch and Bound ("$count _1.txt")//repeat algorithm
1.3. If no FS was returned then Fathom Branch
1.4. RETURN (when arrives here no other branches are available)
End Branch and Bound Method
```

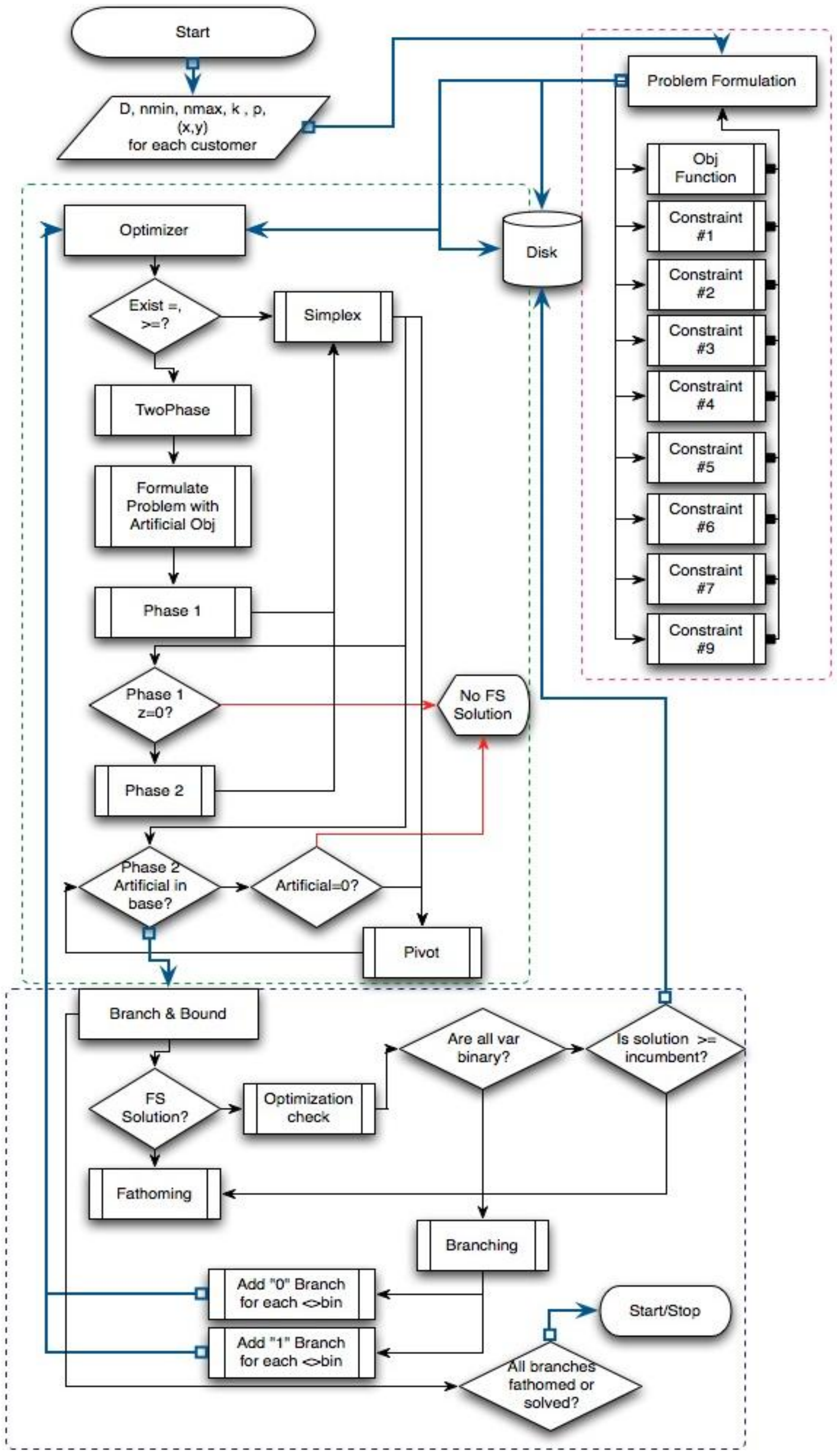


Figure 20 V-RED Flow Chart

5.4 Software Validation and Testing

The principal objectives of testing are the discovery of defects in a system and the assessment of whether or not the system is useful and useable in an operational situation. Both verification and validation are concerned with establishing the existence of defects in a program but not the source of that error(Sommerville, 2007).

More specifically, software testing is concerned with exercising and observing product behavior that is a dynamic way of verifying the product. The ordinary verification procedure consists in executing the system with test data and then observe its operational behavior.

Attention should be given to the fact that testing can reveal the presence of errors but not their absence. This actually, is also the only validation technique for non-functional requirements as the software has to be executed to see how it behaves.

There are two types of testing(Sommerville, 2007):

- **Defect testing** that consists of tests designed to discover system defects. A successful defect test is one which reveals the presence of defects in a system. However, defect testing and debugging are distinct processes. Debugging is concerned with locating and repairing these errors and involves formulating a hypothesis about program behavior then testing these hypotheses to find the system error
- **Validation testing** that is intended to show that the software meets its requirements. A successful validation test is one that shows that one or more requirements have been properly implemented.

Concluding, software testing can be implemented at any time in the development process. Usually, most of the test effort occurs after the requirements have been defined and the coding process has been completed. However, different software development models will focus the test effort at different points in the development process and will use different methodologies.

5.5 Testing Process

In this section the followed testing process will be presented, focusing on release testing - where the complete system to be delivered as a black-box is tested. Of course, it is well known that only exhaustive testing can show a program is free from defects. However, exhaustive testing is impossible. Thus best practices are used in selecting system tests. A general rule is that all functions accessed through menus or user selection should be tested and where user input is required; all functions must be tested with correct and incorrect input. However we should always keep in mind that testing can show the presence of faults in a system but it cannot prove there are no remaining faults(Sommerville, 2007).

Analyzing the testing process used to validate the V-RED software, shown in Figure 20, we see that the first step is to create test cases that will describe what features of the system should be checked, what will be the exact input data and the expected results. Here should be noted that test cases should be created not only for standard situations but also for all those that is expected to return error messages. Test cases aim to find situations that will reveal defects in the system, that's why:

- Should be chosen inputs that force the system to generate all error messages;
- Should be designed inputs that cause buffers to overflow;
- Each input or input series should be repeated a number of times;
- Invalid outputs should be forced to be generated;
- Computation results should be forced to be too large or too small

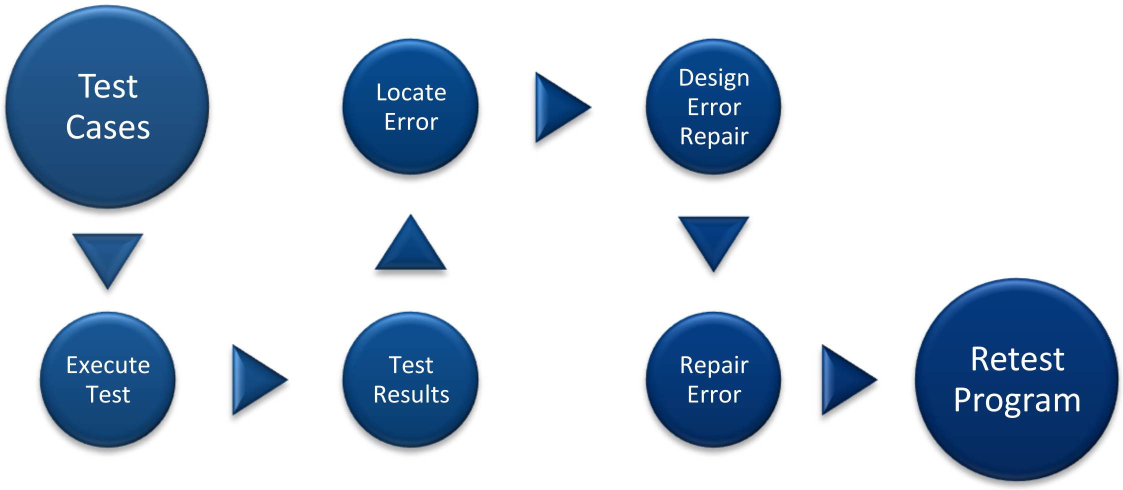


Figure 21 Testing Process

After creating the test cases, each one should be executed and the given results compared to the correct ones. In case of complex systems that the expected results cannot easily be computed, a different software tool should be used, that we already know that works correctly, in order to create the comparison output data.

In cases of erroneous results, a debug session will be initiated to locate the errors in the code or the logic of the system. To do so, first of all the test case and input data causing the error will be used to step by step check the main variables and the intermediate results in order to find out if it is an error caused by the way it was coded the system or an error in the algorithm that is being implemented. In both cases as soon as the error is located, an error repair is designed and implemented. When the code is ready, it is going to be retested with all the available test cases and not only the one that caused the error, because it is not uncommon by fixing an error to create some other bugs, in situations that previously the system worked perfectly.

As the last step of software development the performance of the code should be tested, or even better benchmarked. Generally speaking benchmark is a standard by which something is evaluated or measured. In computing, benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it. Benchmarking is usually associated with assessing performance characteristics of computer hardware, but there are circumstances when the technique is also applicable to software.

The key objective of software benchmarking is to help the developers completely understand how the program performs during execution. This is usually achieved by using an integrated set of performance and debugging profilers and then collecting all crucial performance, memory and resource allocation information at runtime. Final goal is to easily isolate and eliminate all performance issues, memory leaks and resource leaks within the source code and certifying a specific standard of performance in any situation.

6 Discussion and Further Research

The focus of the proposed master thesis was set on the definition of the frequency of service in a distribution network by maximizing the profit in a specific time window. The problem is modeled using integer mathematical programming and is solved using the Brach and Bound algorithm in order to find the optimal solution.

This problem had already been solved within an Undergraduate thesis developed by the DeOpSys team of the University of the Aegean. In that effort the focus was given in solving practical sized problems using a specific heuristic algorithm. The actual solution of the problem using the Branch and Bound algorithm was narrowed down to a small number of clients due to the high complexity of the problem (number of variables and constraints).

Therefore, primary goal of this thesis was to optimally solve the problem for a greater number of clients. More specifically, research was performed on the restrictions and parameters that raise the problem's complexity and solution time. Finally, a set of methods and techniques were implemented in order to reduce the complexity and get the optimal solution faster for a larger set of clients. The resulting product is a software application, V-RED, that takes all the delivery network's parameters as input data and returns the optimal routing solution.

The benchmarking results lead us to the conclusion that is possible to use the Branch and Bound Algorithm in practical sized problems but it needs very careful implementation and customization of the tricky steps in order to have minimum execution times.

However, as seen the execution time depends more on the number of days that the route contains than the number of customers. This happens due to the fact that the number of customer adds an initial overhead to the system but this overhead remains unchanged no matter how many days are in the route and it is probable that we could prove that this amount grows by a stable parameter c.

On the contrary the overhead given by the number of days grows exponentially and leads us to assume that there is a specific upper bound for the number of days that we can get a scheduled routing using this algorithm.

It would be interesting to find out which specific functions of the designed system are exponentially related to the problem size and even more important if there are ways to narrow down the execution time and thus elevate the size of the solvable problems by using more specific fathoming and branching policies.

Furthermore, by evaluating the time used by each constraint we can say that there are two very time consuming constraints. The one refers to the cost per day and it is a limiting constraint in all the executed cases and even better it has a normal rate of growth in relation to the problem size. The other one refers to cyclicity and using computation of all possible combinations make the whole system much slower. It probably would be a good idea for future research to remove the constraint from the optimization problem and add it as a part of the fathoming policy used by the Branch and Bound algorithm. This way the execution time would be minimized.

Finally, some issues about the mathematical modeling appeared to exist. First of all the peculiar handling of the case of 2 customers and 2 or more route days where the system instead of returning a positive result greater or equal to the one given to 2 customers 1 day it keeps giving negative profit. Investigating this case was found that the distribution cost was greater than the income. If the minimum number of visits (n_{min}) for at least one of the customers was zero (0), then the profit would have not been negative.

7 References

Ambrosino, D. and M. G. Scutellà (2005). "Distribution network design: New problems and related models." European Journal of Operational Research**165**(3): 610-624.

Amiri, A. (2006). "Designing a distribution network in a supply chain system: Formulation and efficient solution procedure." European Journal of Operational Research**171**(2): 567-576.

Applegate, D., R. E. Bixby, et al. (1995). Finding cuts in the TSP (a preliminary report).Technical Report 95-05, DIMACS, Rutgers University, New Brunswick, NJ 08903.

Beale, E. M. L. and J. A. Tomlin (1970). Special Facilities in a General Mathematical Programming System for Non-convex Problems Using Ordered Sets of Variables. Proceedings of the Fifth International Conference on Operations Research, Tavistock.

Benichou, M., J. M. Gauthier, et al. (1971). "Experiments in mixed integer linear programming." Mathematical Programming**1**(76-94).

Bernardi, S., S. Donatelli, et al. (2002). From UML sequence diagrams and statecharts to analysable petri net models. Proceedings of the 3rd international workshop on Software and performance. Rome, Italy, ACM**:** 35-45.

Brunetta, L., M. Conforti, et al. (2000). "A polyhedral approach to an integer multicommodity flow problem." Discrete Applied Mathematics**101**(1): 13-36.

Choon Tan, K. (2001). "A framework of supply chain management literature." European Journal of Purchasing & Supply Management**7**(1): 39-48.

Crainic, T. G. and G. Laporte (1997). "Planning models for freight transportation." European Journal of Operational Research**97**(3): 409 -438.

Dakin, R., J. (1965). "A tree-search algorithm for mixed integer programming problems." The Computer Journal(8): 250 - 255.

Driebeek, N. J. (1966). "An algorithm for the solution of mixed integer programming problems." Management Science**21**: 576-587.

Eskigun, E., R. Uzsoy, et al. (2005). "Outbound supply chain network design with mode selection, lead times and capacitated vehicle distribution centers." European Journal of Operational Research**165**(1): 182-206.

Forrest, J. J. H. and J. A. Tomlin (2007). "Branch and bound, integer, and non-integer programming." Annals of Operations Research**149**(1): 81-87.

Ghezavati, V. R., M. S. Jabal-Ameli, et al. (2009). "A new heuristic method for distribution networks considering service level constraint and coverage radius." Expert Systems with Applications(36): 5620-5629.

Gill, P., W. Murray, et al. (1982).Practical Optimization Academic Press

Glinz, M. (2000). Problems and Deficiencies of UML as a Requirements Specification Language.Proceedings of the 10th International Workshop on Software Specification and Design, IEEE Computer Society**:** 11.

Goel, A. and V. Gruhn (2008). "A General Vehicle Routing Problem." European Journal of Operational Research**191**(3): 650-660.

Hilier, F. S. and G. J. Lieberman (2005). Introduction to Operations Research, McGraw Hill.

Kolman, B. and R. E. Beck (1980). Elementary Linear Programming with Applications.London, Academic Press.

Land, A. H. and A. G. Doig (1960). "An Automatic Method for Solving Discrete Programming Problems." Econometrica(28): 497-520.

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

Laporte, G. (1992). "The vehicle routing problem: An overview of exact and approximate algorithms." European Journal of Operational Research**59**(3): 345-358.

Lindgaard, G., R. Dillon, et al. (2006). "User Needs Analysis and requirements engineering: Theory and practice." Interacting with Computers**18**(1): 47-70.

MacKinnon, N. and S. Murphy (2003). Designing UML diagrams for technical documentation. Proceedings of the 21st annual international conference on Documentation. San Francisco, CA, USA, ACM**:** 105-112.

Ritchie, D. M. (1993).The development of the C language.The second ACM SIGPLAN conference on History of programming languages. Cambridge, Massachusetts, United States, ACM**:** 201-208.

Smith, J. C. (2004). "Algorithms for distributing telecommunication traffic on a multiple-ring sonet-based network." European Journal of Operational Research**154**(3): 659-672.

Sommerville, I. (2007). Software Engineering, Addison-Wesley Publishing Company.

Sommerville, I. and G. Dewsbury (2007). "Dependable domestic systems design: A socio-technical approach." Interacting with Computers**19**(4): 438-456.

Sutcliffe, A., G. Papamargaritis, et al. (2006). "Comparing requirements analysis methods for developing reusable component libraries." Journal of Systems and Software**79**(2): 273-289.

Tomlin, J. A. (1971). "An improved branch and bound method for integer programming." Opeations Research**19**: 1070-1075.

Xu, H., P. Sawyer, et al. (2006). "Requirement process establishment and improvement from the viewpoint of cybernetics." Journal of Systems and Software**79**(11): 1504-1513.

Zündorf, A. (2001). From use cases to code - rigorous software development with UML.Proceedings of the 23rd International Conference on Software Engineering. Toronto, Ontario, Canada, IEEE Computer Society**:** 711-712.

A. Appendix

A.1 Software Requirement Analysis

Requirements in systems engineering and software engineering, encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders, such as beneficiaries or users.

“Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families”(Lindgaard, 2006).

The primary measure of success of a software system is the degree to which it meets the purpose for which it was intended. Broadly speaking, software systems requirements engineering (RE) is the process of discovering that purpose, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation. There are a number of inherent difficulties in this process. Stakeholders (including paying customers, users and developers) may be numerous and distributed. Their goals may vary and conflict, depending on their perspectives of the environment in which they work and the tasks they wish to accomplish. Their goals may not be explicit or may be difficult to articulate, and, inevitably, satisfaction of these goals may be constrained by a variety of factors outside their control.

Requirements analysis is critical to the success of a development project. Requirements must be documented, actionable, measurable, testable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design. Requirements can be functional and non-functional (Sutcliffe, 2006).

Below are depicted the initial requirements as provided by the DeOPSys research team. The non-functional requirements of the herein discussed system can be summarized as follows:

- The system should be able to make all computations based on each instance’s inputs.
- The only given inputs should be:
  - Initial demand per customer.
  - Minimum and maximum allowed number of visits per customer.

    - Rate of raise of demand for each visit to customer after the minimum allowed number of visits.
    - Coordinates of each customer's location are given in order to compute the distance between each pair of customers and between each customer and the depot.
    - Incomes from each visit to any customer.
- The system should be able to solve any linear optimization problem with the objective function and constraints described in section 4.3.
- The mathematical algorithms used to solve the problem should be the Branch and Bound Algorithm and the Simplex method.

As expected these data are general guidelines about what the software tool should be able to do and not how is going to be implemented. The only non-habitual about the given requirements is the fact that the algorithms that are going to be used to solve the problem are well known and predefined. This happens due to the fact that the final aim of the software tool is to verify the computational time needed when using these algorithms and not to create an innovative product.

A.2 Software Specification Process

A software requirements specification is a complete description of the behavior of the system to be developed. In order to represent the herein defined system from different perspectives three different models will be used. An external perspective will be used to show the context of the system that is being modeled. This will be achieved by designing the system's use case model that is going to present all the interactions that the users will have with the software. A behavioral perspective will be given using a sequence diagram to show the main flow of events and critical alternate flows and a structural perspective will be presented through the architecture of the system where the functionality of the main system modules will be described(Sommerville, 2007).

The steps taken in order to gather the requirements to build the system's architecture, as seen in Figure 21, can be summarized as follows:

1. Problem Definition: Crafting the problem statement is always the first step in any design. Here the goal is to state succinctly, but accurately, the problem keeping focused in what and why, but not how. Any design problem begins with research of the area in order to get acquainted with the domain. Another very important step of this process is the definition of all the terms that are going to be used throughout the systems documentation, in order to create a common base for the communication among the basic actors.

2. Business Requirements: Concerns the features that either the end users and/or the stakeholders of the project would like to see in the new system.
3. Software Model Description: System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers. Different models present the system from different perspectives.

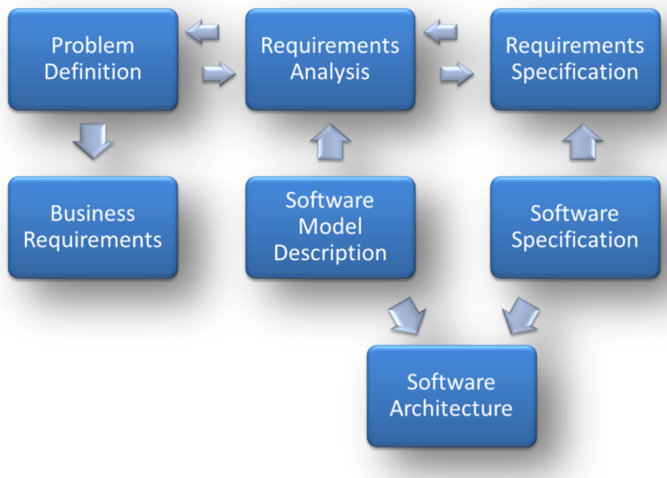


Figure 22 Requirements Analysis Process

4. Requirement Analysis: Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system’s operational constraints.
5. Software Specification: The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
6. Requirements Specification: A detailed view of the requirements that gives analytical descriptions of the system services and constraints that are generated during the requirements engineering process.
7. Software Architecture: The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is architectural design. The output of this design process is a description of the software architecture.

A.3 Stages of Development

The software development life cycle is different for every project and every development team, but there are key stages that must be carried out. The V-RED application was developed using agile methods that reflect the close collaboration of the development team to the end users and the frequent change of requirements and functionality. Below are outlined the distinctive features of this development method(Sommerville, 2007) :

- **Customer involvement:** The customer is closely involved throughout the development process with primary role to provide and prioritize new system requirements and to evaluate the iterations of the system.
- **Incremental delivery:** The software is developed in increments with the customer specifying the requirements to be included in each increment.
- **Adaptable**: It is expected that the system requirements will change and thus the system is designed so that it can accommodate these changes.
- **Simplicity**: Focus should be given on simplicity in both the software being developed and in the development process used.

Perhaps the best-known and most widely used agile method is Extreme Programming (XP) that takes an 'extreme' approach to iterative development where new versions may be built several times per day. In order to achieve the standards set by Extreme Programming, below are summarized the best practices that were followed during the development:

- Small Releases: The minimal useful set of functionality was developed first and releases of the system were frequent and incrementally added functionality to the first release.
- Simple Design: Enough design was carried out to meet the current requirements and no more.
- Test first development: A unit test is used for each new piece of functionality before that functionality itself is implemented.
- Refactoring: The code was being refactored each time that code improvements were found.

Concluding, the Extreme Programming agile method was used in order to have a rapid development that would give early releases and would accommodate requirements change.

A.4 Implementation Tools

The V-RED application was developed as a standalone windows application using the Microsoft Visual Studio 2010 as development environment and more specifically Visual C++ and .NET framework 4.0. The reasons that led as to such decision were from one hand the user friendly environment given from the Microsoft Visual Studio and from the other hand the need to have fast code as it is given only by middle and low level languages as C++ and the portability of the application given by the framework on which was developed. Below are briefly described the main features of the development tools that were used along with the analytical reasoning that made us selected them for this specific implementation.

Microsoft Visual Studio

Microsoft Visual Studio is an integrated development environment from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft.

Visual Studio includes a code editor supporting IntelliSense as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that enhance the functionality at almost every level—including adding support for source-control systems and adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle.

Visual Studio supports different programming languages by means of language services, which allow the code editor and debugger to support nearly any programming language, provided a language-specific service exists. Built-in languages include C/C++, VB.NET, C#, and F#. Support for other languages such as M, Python, and Ruby among others is available via language services installed separately.

Microsoft Visual C++

Selecting a programming language requires many different considerations to be taken. The first step is to select the level of the programming language. The level determines the proximity of the programming language to the hardware. In the lower level languages, instructions are written as a direct interface with the underneath hardware, while in high level languages a more abstract code is written.

Generally, high level code is more portable, thus it can be used in different machines although sometimes a small number of modifications could be needed, whereas a low level language is limited by the specific features of the hardware on which it was written. Nevertheless, the undoubtable advantage of low level code is that it is faster due to the fact that it is written taking advantage of the possibilities of a specific machine. A higher or lower level of programming is to be chosen for a specific project depending on the type of program that is being developed.

There are languages that are clearly low level, like Assembly, whose instruction sets are adapted to each machine the code is made for, and other languages are inherently high level, like the Java, that is designed to be totally independent of the platform where is going to run. However, the C++ language is in a middle position, since it can interact directly with the hardware almost with no limitations, and can as well abstract lower layers and work like one of the most powerful high level languages.

Concluding, C++ has certain characteristics over other programming languages that led us to select it for the V-RED development. The most remarkable are (Ritchie, 1993):

- **Object-oriented programming:** The possibility to orientate programming to objects allows the programmer to design applications as communication between objects rather than a structured sequence of code. In addition it allows a greater reusability of code in a more logical and productive way.
- **Portability:** The same C++ code can be compiled in almost any type of computer and operating system without making any changes.
- **Brevity**: Code written in C++ is very short in comparison with other languages.
- **Modular programming**: An application's body in C++ can be made up of several source code files that are compiled separately and then linked together. Saving time since it is not necessary to recompile the complete application when making a single change but only the file that contains it. In addition, this characteristic allows to link C++ code with code produced in other languages.
- **Speed**: The resulting code from a C++ compilation is very efficient, due indeed to its duality as high-level and low-level language and to the reduced size of the language itself.

Microsoft .NET Framework

The .NET Framework is Microsoft's comprehensive and consistent programming model for building applications that have visually elevated user experiences, seamless and secure communication, and the ability to model a range of business processes.

A.5 Test Cases

To test the V-RED application as a whole were selected three test cases representing the minimal input having two customers and just one day route, and two ordinary cases, of two customers two days and three customers two days. It was also effectuated a stress test with eight customers and five days that will be presented in chapter 8.

Each test case consists of manually computing the objective function and constrains and comparing it with those given by the V-RED software and then checking the optimization results of the software compared to the results given by the Microsoft Excel Solver 2010.

Microsoft Excel Solver is a numerical optimization add-in of Microsoft Excel. Solvers, or optimizers, are software tools that help users find the best way to allocate scarce resources. The resources may be raw materials, machine time or people time, money, or anything else in limited supply. The best or optimal solution may mean maximizing profits, minimizing costs, or achieving the best possible quality. There are different optimization model that can be used with the Excel Solver. In this case the linear optimization model was used to be able to confront the results with V-RED.

An optimization model in Microsoft Excel Solver has three parts: the target cell, the changing cells, and the constraints.

- Target Cell: represents the objective or goal. We want to either minimize or maximize the target cell.
- Changing cells: are the spreadsheet cells that we can change or adjust to optimize the target cell.
- Changing cells: are the spreadsheet cells that we can change or adjust to optimize the target cell.

Test Case 1: 2 Customers 1 Day

Input Data

Variables	Customer 1	Customer 2
Depot Coordinates	[200,-200]	
Customer Coordinates	[0,100]	[0,200]
n_{min}	1	1
n_{max}	5	5
Demand D	20	30
Demand Raise k	20	20.2

Profit per unit	20
Cost	1
Maximum Cost per day	1200
Distances	C_{01}=360.5 C_{02}=447.2 C_{12}=100

Problem Formulation

Objective Function

$$z = 400y_1 + 404y_2 + 196 - 360.5x_{01} - 447.2x_{02} - 360.5x_{10} - 100x_{12} - 447.2x_{20} - 100x_{21}$$

Under constraints:

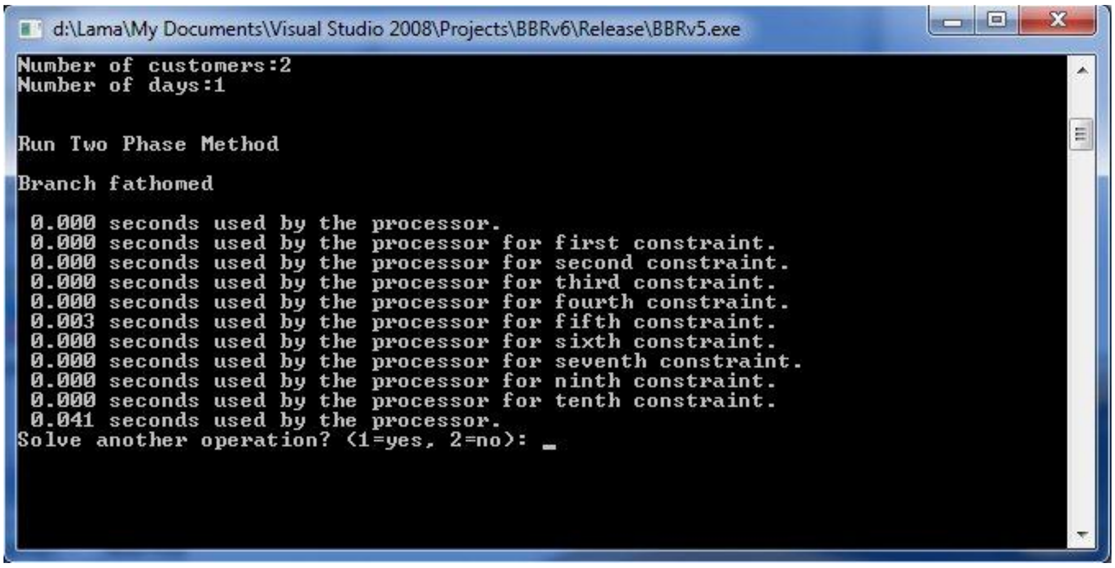
Constraint #1:	$\boldsymbol{x_{10} + x_{12} = y_1}$ $\boldsymbol{x_{20} + x_{21} = y_2}$
Constraint #2:	$x_{01} + x_{21} = y_1$ $x_{02} + x_{12} = y_2$
Constraint #3:	$x_{01} + x_{02} = 1$
Constraint #4:	$x_{10} + x_{12} = 1$
Constraint #5 :	$-x_{21} - x_{12} + y_1 + y_2 \geq 1$
Constraint #6:	$360.5x_{01} + 447.2x_{02} + 360.5x_{10} + 100x_{12} + 447.2x_{20} + 100x_{21} \leq 1200$
Constraint #7:	$1 \leq y_1 \leq 5$ $1 \leq y_2 \leq 5$

Excel Solver Results

RULE	x00	x01	x02	x10	x11	x12	x20	x21	x22	y1	y2	INEQ	RIGHT	computed
	0	0	1	1	0	0	0	1	0	1	1			
1	0	0	0	1	0	1	0	0	0	-1	0	=	0	0
1	0	0	0	0	0	0	1	1	0	0	-1	=	0	0
2	0	1	0	0	0	0	0	1	0	-1		=	0	0
2	0	0	1	0	0	1	0	0	0	0	-1	=	0	0
3	0	1	1	0	0	0	0	0	0	0	0	=	1	1
4	0	0	0	1	0	0	1	0	0	0	0	=	1	1
5	0	0	0	0	0	-1	0	-1	0	1	1	>=	1	1
6	0	360.5	447.2	360.5	0	100	447.2	100	0	0	0	<=	1200	907.7
7	0	0	0	0	0	0	0	0	0	1	0	<=	5	1
7	0	0	0	0	0	0	0	0	0	1	0	>=	1	1
7	0	0	0	0	0	0	0	0	0	0	1	<=	5	1
7	0	0	0	0	0	0	0	0	0	0	1	>=	1	1
OBJ	0	-360.5	-447.2	-360.5	0	-100	-447.2	-100	0	400	404	196	result	92.3

Figure 23 Excel Solver Results for 2 customers 1 day

V-RES Results



```
d:\Lama\My Documents\Visual Studio 2008\Projects\BBRv6\Release\BBRv5.exe
Number of customers:2
Number of days:1

Run Two Phase Method

Branch fathomed

 0.000 seconds used by the processor.
 0.000 seconds used by the processor for first constraint.
 0.000 seconds used by the processor for second constraint.
 0.000 seconds used by the processor for third constraint.
 0.000 seconds used by the processor for fourth constraint.
 0.003 seconds used by the processor for fifth constraint.
 0.000 seconds used by the processor for sixth constraint.
 0.000 seconds used by the processor for seventh constraint.
 0.000 seconds used by the processor for ninth constraint.
 0.000 seconds used by the processor for tenth constraint.
 0.041 seconds used by the processor.
Solve another operation? (1=yes, 2=no): _
```

Figure 24 V-RED Results for 2 customers 1 day

Conclusions

As shown in Figures 22 and 23, both software tools return the same result and route using different direction of movement. In Figure 24 is depicted the proposed route. The expected optimal profit is 92.3.

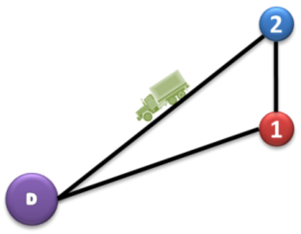


Figure 25 Case 2 - 1 Route

Test Case 2: 2 Customers 2 Days

Input Data

Variables	Customer 1	Customer 2
Depot Coordinates	[200,-200]	
Customer Coordinates	[0,100]	[0,200]
n_{min}	1	1
n_{max}	5	5
Demand D	20	30
Demand Raise k	20	20.2

Profit per unit	20
Cost	1
Maximum Cost per day	1200
Distances	C_{01}=360.5 C_{02}=447.2 C_{12}=100

Problem Formulation

Objective Function

$$z = 400y_1^1 + 404y_2^1 + 400y_1^2 + 404y_2^2 + 196 - (360.5x_{01}^1 + 447.2x_{02}^1 + 360.5x_{10}^1 + 100x_{12}^1 + 447.2x_{20}^1 + 100x_{21}^1) - (360.5x_{01}^2 + 447.2x_{02}^2 + 360.5x_{10}^2 + 100x_{12}^2 + 447.2x_{20}^2 + 100x_{21}^2)$$

Under constraints:

Constraint #1:

$$\mathbf{x_{10}^1 + x_{12}^1 = y_1^1}$$
$$\mathbf{x_{20}^1 + x_{21}^1 = y_2^1}$$
$$\mathbf{x_{10}^2 + x_{12}^2 = y_1^2}$$
$$\mathbf{x_{20}^2 + x_{21}^2 = y_2^2}$$

Constraint #2:

$$x_{01}^1 + x_{21}^1 = y_1^1$$
$$x_{02}^1 + x_{12}^1 = y_2^1$$
$$x_{01}^2 + x_{21}^2 = y_1^2$$
$$x_{02}^2 + x_{12}^2 = y_2^2$$

Constraint #3:

$$x_{01}^1 + x_{02}^1 = 1$$
$$x_{01}^2 + x_{02}^2 = 1$$

Constraint #4:

$$x_{10}^1 + x_{20}^1 = 1$$
$$x_{10}^2 + x_{20}^2 = 1$$

Constraint #5 :

$$-x_{21}^1 - x_{12}^1 + y_1^1 + y_2^1 \geq 1$$
$$-x_{21}^2 - x_{12}^2 + y_1^2 + y_2^2 \geq 1$$

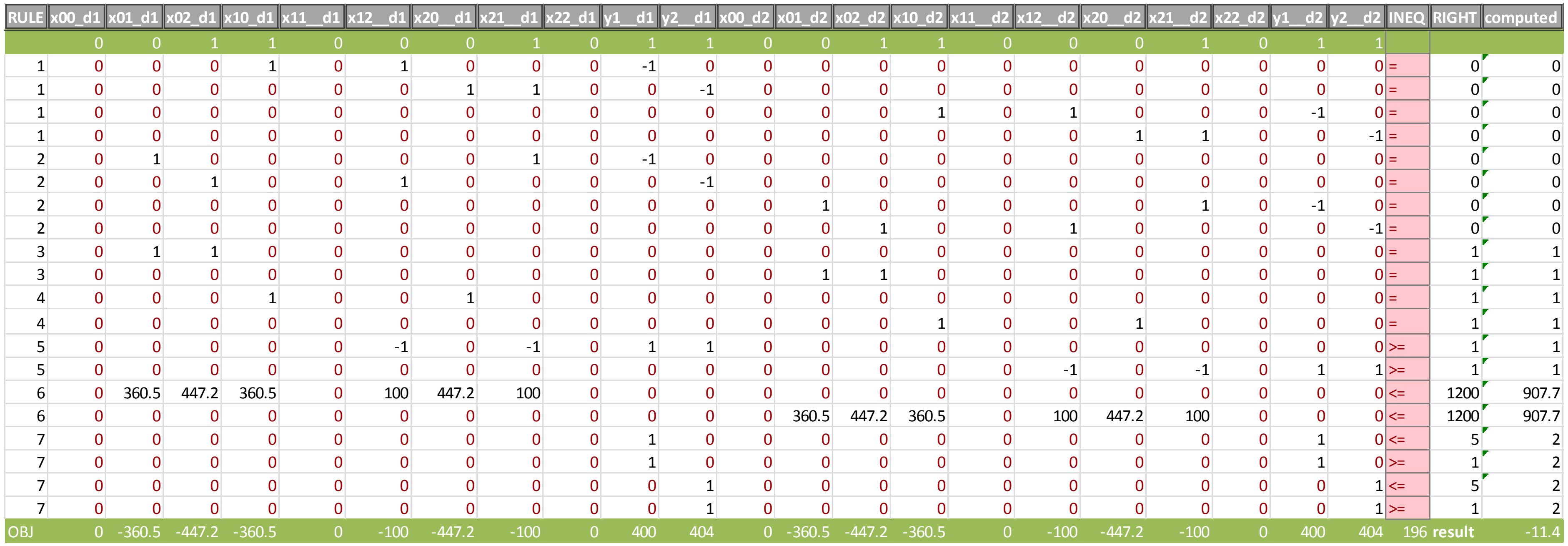
Constraint #6:

$$360.5x_{01}^1 + 447.2x_{02}^1 + 360.5x_{10}^1 + 100x_{12}^1 + 447.2x_{20}^1 + 100x_{21}^1 \leq 1200$$
$$360.5x_{01}^2 + 447.2x_{02}^2 + 360.5x_{10}^2 + 100x_{12}^2 + 447.2x_{20}^2 + 100x_{21}^2 \leq 1200$$

Constraint #7:

$$1 \le y_1^1 + y_1^2 \le 5$$
$$1 \le y_2^1 + y_2^2 \le 5$$

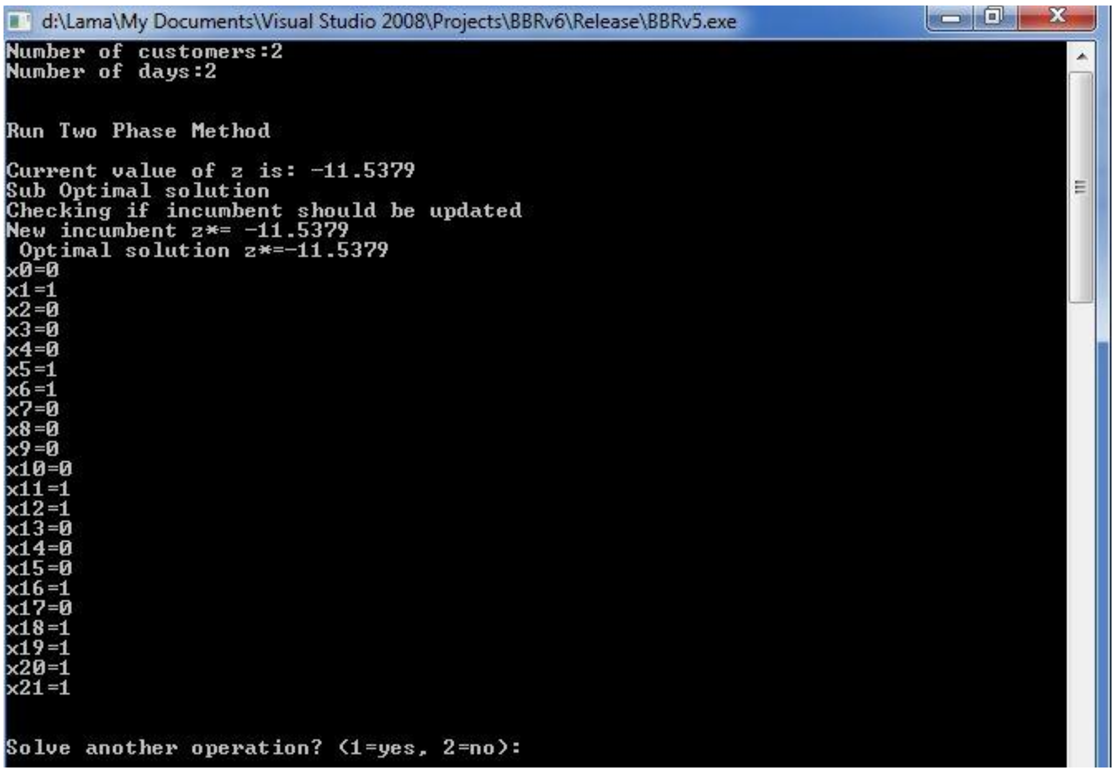
Excel Solver Results



RULE	x00_d1	x01_d1	x02_d1	x10_d1	x11__d1	x12__d1	x20__d1	x21__d1	x22_d1	y1__d1	y2__d1	x00_d2	x01_d2	x02_d2	x10_d2	x11__d2	x12__d2	x20__d2	x21__d2	x22_d2	y1__d2	y2__d2	INEQ	RIGHT	computed
	0	0	1	1	0	0	0	1	0	1	1	0	0	1	1	0	0	0	1	0	1	1			
1	0	0	0	1	0	1	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	=	0	0
1	0	0	0	0	0	0	1	1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	=	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	-1	0	=	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	-1	=	0	0
2	0	1	0	0	0	0	0	1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	=	0	0
2	0	0	1	0	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	=	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	-1	0	=	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	-1	=	0	0
3	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	1	1
3	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	=	1	1
4	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	1	1
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	=	1	1
5	0	0	0	0	0	-1	0	-1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	>=	1	1
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	-1	0	1	1	>=	1	1
6	0	360.5	447.2	360.5	0	100	447.2	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	<=	1200	907.7
6	0	0	0	0	0	0	0	0	0	0	0	0	360.5	447.2	360.5	0	100	447.2	100	0	0	0	<=	1200	907.7
7	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	<=	5	2
7	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	>=	1	2
7	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	<=	5	2
7	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	>=	1	2
OBJ	0	-360.5	-447.2	-360.5	0	-100	-447.2	-100	0	400	404	0	-360.5	-447.2	-360.5	0	-100	-447.2	-100	0	400	404	196	result	-11.4

Figure 26 Excel Solver Results for 2 customers 2 days

V-RES Results


d:\Lama\My Documents\Visual Studio 2008\Projects\BBRv6\Release\BBRv5.exe

```
Number of customers:2
Number of days:2


Run Two Phase Method

Current value of z is: -11.5379
Sub Optimal solution
Checking if incumbent should be updated
New incumbent z*= -11.5379
 Optimal solution z*=-11.5379
x0=0
x1=1
x2=0
x3=0
x4=0
x5=1
x6=1
x7=0
x8=0
x9=0
x10=0
x11=1
x12=1
x13=0
x14=0
x15=0
x16=1
x17=0
x18=1
x19=1
x20=1
x21=1


Solve another operation? (1=yes, 2=no):
```

Figure 27 V-RED Results for 2 customers 2 days

Conclusions

As shown in Figures 25 and 26, both software tools return the same result and route using different direction of movement. In Figure 24 is presented the proposed route that is the same that was in Case 2-1 for each day although it gives as result a loss of profit.

Ideally it should give a result that worst case would be equal to the one given in Case 2- 1. This doesn’t happen because this is a case that the given mathematical model doesn’t cover. If we don’t have route for the 2nd day, then the sum of incoming/outgoing vertices to the depot will be equal to zero that goes against constraints #3 and #4. Furthermore, minor loss we would have even if the route contained only one and not both customers, but again this time the cyclicity constraint #5 $-x_{21}^2 - x_{12}^2 + y_1^2 + y_2^2 \geq 1$ would not be satisfied. Thus, the model used cannot give optimal results when customer number is 2 and route days are more than 1.

Test Case 3: 3 Customers 2 Days

Input Data

Variables	Customer 1	Customer 2	Customer 3
Depot Coordinates	[200,-200]		
Customer Coordinates	[0,100]	[0,200]	[200,300]
n_{min}	1	1	1
n_{max}	5	5	5
Demand D	20	30	40
Demand Raise k	20	20.2	30.3
Profit per unit	20		
Cost	1		
Maximum Cost per day	1200		
Distances	C_{01}=360.5 C_{02}=447.2 C_{03}=500 C_{12}=100 C_{13}=282.8 C_{23}=223.6		

Problem Formulation

Objective Function

$$z = 400y_1^1 + 404y_2^1 + 606y_3^1 + 400y_1^2 + 404y_2^2 + 606y_3^2 + 390 - (360.5x_{01}^1 + 447.2x_{02}^1 + 360.5x_{10}^1 + 100x_{12}^1 + 447.2x_{20}^1 + 100x_{21}^1 + 223.6x_{23}^1 + 223.6x_{32}^1 + 282.8x_{31}^1 + 282.8x_{13}^1 + 500x_{03}^1 + 500x_{30}^1) - (360.5x_{01}^2 + 447.2x_{02}^2 + 360.5x_{10}^2 + 100x_{12}^2 + 447.2x_{20}^2 + 100x_{21}^2 + 223.6x_{23}^1 + 223.6x_{32}^1 + 282.8x_{31}^1 + 282.8x_{13}^1 + 500x_{03}^1 + 500x_{30}^1)$$

Under constraints:

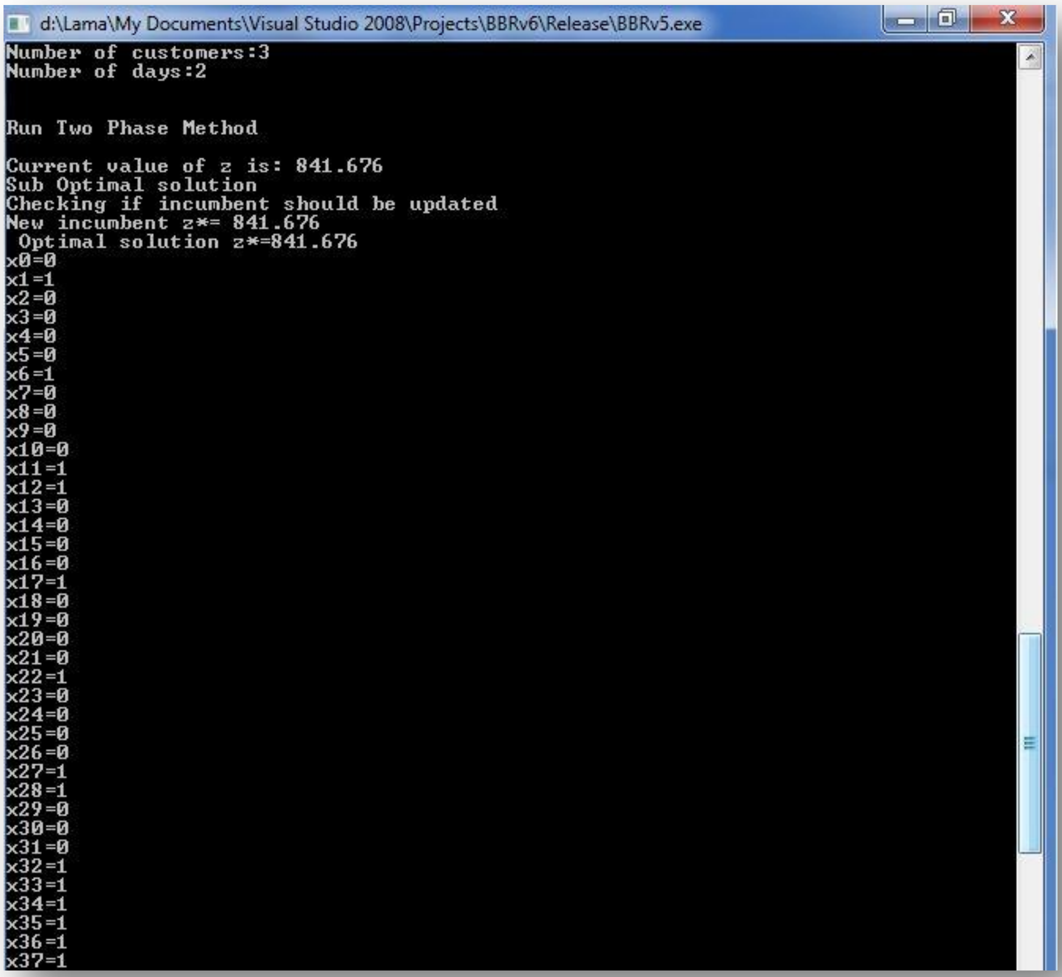
Constraint #1:	$\mathbf{x_{10}^1 + x_{12}^1 + x_{13}^1 = y_1^1}$
	$\mathbf{x_{20}^1 + x_{21}^1 + x_{23}^1 = y_2^1}$
	$\mathbf{x_{30}^1 + x_{31}^1 + x_{32}^1 = y_3^1}$
	$\mathbf{x_{10}^2 + x_{12}^2 + x_{13}^2 = y_1^2}$
	$\mathbf{x_{20}^2 + x_{21}^2 + x_{23}^2 = y_2^2}$
	$\mathbf{x_{30}^2 + x_{31}^2 + x_{32}^2 = y_3^2}$
Constraint #2:	$x_{01}^1 + x_{21}^1 + x_{31}^1 = y_1^1$
	$x_{02}^1 + x_{12}^1 + x_{32}^1 = y_2^1$
	$\mathbf{x_{03}^1} + x_{13}^1 + \mathbf{x_{23}^1} = \mathbf{y_3^1}$
	$x_{01}^2 + x_{21}^2 + x_{31}^2 = y_1^2$
	$x_{02}^2 + x_{12}^2 + x_{32}^2 = y_2^2$
	$\mathbf{x_{03}^2} + x_{13}^2 + \mathbf{x_{23}^2} = \mathbf{y_3^2}$
Constraint #3:	$x_{01}^1 + x_{02}^1 + x_{03}^1 = 1$
	$x_{01}^2 + x_{02}^2 + x_{03}^2 = 1$
Constraint #4:	$x_{10}^1 + x_{20}^1 + x_{30}^1 = 1$
	$x_{10}^2 + x_{20}^2 + x_{30}^2 = 1$
Constraint #5 :	$-x_{21}^1 - x_{12}^1 - x_{31}^1 - x_{13}^1 - x_{23}^1 - x_{32}^1 + y_1^1 + y_2^1 + y_3^1 \geq 1$
	$-x_{21}^2 - x_{12}^2 - x_{31}^2 - x_{13}^2 - x_{23}^2 - x_{32}^2 + y_1^2 + y_2^2 + y_3^2 \geq 1$
Constraint #6:	$360.5x_{01}^1 + 447.2x_{02}^1 + 360.5x_{10}^1 + 100x_{12}^1 + 447.2x_{20}^1 + 100x_{21}^1 + 223.6x_{23}^1 + 223.6x_{32}^1 + 282.8x_{31}^1 + 282.8x_{13}^1 + 500x_{03}^1 + 500x_{30}^1 \leq 1200$
	$360.5x_{01}^2 + 447.2x_{02}^2 + 360.5x_{10}^2 + 100x_{12}^2 + 447.2x_{20}^2 + 100x_{21}^2 + 223.6x_{23}^2 + 223.6x_{32}^2 + 282.8x_{31}^2 + 282.8x_{13}^2 + 500x_{03}^2 + 500x_{30}^2 \leq 1200$
Constraint #7:	$1 \leq y_1^1 + y_1^2 \leq 5$
	$1 \leq y_2^1 + y_2^2 \leq 5$
	$1 \leq y_3^1 + y_3^2 \leq 5$

Excel Solver Results

RULE	x00	x01	x02	x03	x10	x11	x12	x13	x20	x21	x2	x23	x30	x31	x32	x3	y1	y2	y3	x(	x01	x02	x03	x10	x1	x12	x13	x20	x21	x22	x23	x30	x31	x32	x3	y1	y2	y3	INEQ	RIGHT	comp
	0	1	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	1	0	1	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	1			
1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	0	0
1	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	-1	0	0	=	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	-1	0	=	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	-1	=	0	0
2	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	0	0
2	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	0	0
2	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	-1	0	0	=	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	-1	0	=	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	-1	=	0	0
3	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	1	1
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	1	1
4	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	1	1
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	=	1	1
5	0	0	0	0	0	0	-1	0	0	-1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	>=	1	1
5	0	0	0	0	0	0	0	-1	0	0	0	0	0	-1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	>=	1	2
5	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	-1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	>=	1	1
5	0	0	0	0	0	0	-1	-1	0	-1	0	-1	0	-1	-1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	>=	1	1
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	-1	0	0	0	0	0	0	1	1	0	>=	1	1
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	-1	0	0	1	0	1	>=	1	2
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	-1	0	0	1	1	>=	1	1
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-1	0	-1	0	-1	0	-1	-1	0	1	1	1	>=	1	1
6	0	-360.5	-447.2	-500	-360.5	0	100	282.8	447.2	100	0	223.6	500	282.8	223.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	<=	1200	463
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360.5	447.2	500	360.5	0	100	282.8	447.2	100	0	223.6	500	282.8	223.6	0	0	0	0	<=	1200	1184
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	<=	5	2
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	<=	5	2
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	<=	5	2
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	>=	1	2
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	>=	1	2
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	>=	1	2
OBJ	0	-360.5	-447.2	-500	-360.5	0	-100	-282.8	-447.2	-100	0	-223.6	-500	-282.8	-223.6	0	400	404	606	0	-360.5	-447.2	-500	-360.5	0	-100	-283	-447.2	-100	0	-223.6	-500	-282.8	-223.6	0	400	404	606	390	result	842

Figure 28 Excel Solver Results for 3 customers 2 days

V-RES Results



d:\Lama\My Documents\Visual Studio 2008\Projects\BBRv6\Release\BBRv5.exe

```
Number of customers:3
Number of days:2


Run Two Phase Method

Current value of z is: 841.676
Sub Optimal solution
Checking if incumbent should be updated
New incumbent z*= 841.676
 Optimal solution z*=841.676
x0=0
x1=1
x2=0
x3=0
x4=0
x5=0
x6=1
x7=0
x8=0
x9=0
x10=0
x11=1
x12=1
x13=0
x14=0
x15=0
x16=0
x17=1
x18=0
x19=0
x20=0
x21=0
x22=1
x23=0
x24=0
x25=0
x26=0
x27=1
x28=1
x29=0
x30=0
x31=0
x32=1
x33=1
x34=1
x35=1
x36=1
x37=1
```

Figure 29 V-RED Results for 3 customers 2 days

Conclusions

As shown in Figures 27 and 28, both software tools return the same result and route using different direction of movement. In Figure 29 is depicted the proposed route that is the same for the first and second day. The expected optimal profit is 841.76.

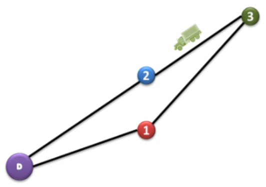


Figure 30 Route for Case 3-2