

**SOLVING THE DYNAMIC PICKUP AND DELIVERY PROBLEM
WITH TIME WINDOWS USING HYBRID LOCAL SEARCH
METAHEURISTICS**

By
Jawad A. El-Omari

Supervisor
Dr. Sameh Shehabi, Assistant Prof.

**This Thesis was Submitted in Partial Fulfillment of the Requirements for
the Master's Degree in Industrial Engineering – Engineering
Management**

**Faculty of Graduate Studies
The University of Jordan**

تعتمد كلية الدراسات العليا
هذه النسخة من الرسالة
التوقيع: التاريخ: 20/5/2009


May, 2009

ذو عمير
2009

The University of Jordan

Authorization Form

I, Jawad A. El-Omari, authorize the University of Jordan to supply copies of my Thesis/Dissertation to libraries or establishments or individuals on request, according to the University of Jordan regulations.

Signature: 

Date: 25-May-2009


COMMITTEE DECISION

This Thesis (Solving the dynamic pickup and delivery problem with time windows using hybrid local search metaheuristics) was Successfully Defended and Approved on 12-May-2009


Examination Committee

Signature

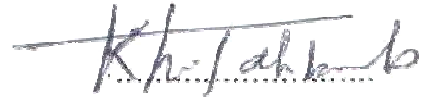
Dr. Sameh Al-Shihabi (Supervisor)
Assist. Prof. of Applied Operations Research


.....

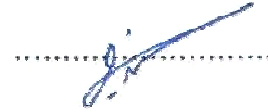
Dr. Ghaleb Abbasi (Member)
Prof. of Project Management


.....

Dr. Khaldoun Tahboub (Member)
Assist. Prof. of Manufacturing and Production Systems


.....

Dr. Nabeel Mandahawi (Member)
Assist. Prof. of Operation Planning and Control
(The Hashemite University)


.....

تعتمد كلية الدراسات العليا
هذه النسخة من الرسالة
التوقيع: التاريخ: 12/5/09

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to Dr. Sameh Shehabi, for his guidance and continued support throughout this work; he has been a great mentor and a good friend. Also many thanks to Mr. Olivier Binckly, from ILOG CP support, for his technical assistance.

LIST OF CONTENTS

| Subject | Page |
|--|------|
| Committee Decision | ii |
| Acknowledgment | iii |
| List of Contents | iv |
| List of Tables | v |
| List of Figures | vi |
| List of Abbreviations | vii |
| List of Appendices | viii |
| Abstract (English) | ix |
| Chapter 1: Introduction | 1 |
| Chapter 2: Literature review | 6 |
| 2.1 Optimization problems | 6 |
| 2.2 Algorithms | 8 |
| 2.3 Computational complexity theory | 10 |
| 2.4 Online versus Offline problems | 15 |
| 2.5 Classification of the dynamic routing problem and related difficulties | 17 |
| 2.6 Solution approaches | 23 |
| 2.7 Previous work | 27 |
| Chapter 3: Methodology | 41 |
| 3.1 Solution approach | 41 |
| 3.2 Overview of the metaheuristics used | 44 |
| 3.3 Experimental procedure | 49 |
| Chapter 4: Results, analysis, and discussion | 54 |
| 4.1 Validation and verification | 54 |
| 4.2 Analysis and conclusion | 66 |
| Chapter 5: Conclusion and recommendations..... | 75 |
| References | 77 |
| Appendices | 83 |
| Abstract (Arabic) | 104 |

LIST OF TABLES

| Number | Table caption | Page |
|--------|---|------|
| 1 | Local optima example | 27 |
| 2 | List of symbols used in the mathematical model representation | 38 |
| 3 | Neighborhood order changes | 50 |
| 4 | Changing search parameters and the neighborhood order | 51 |
| 5 | Results table template | 52 |
| 6 | Paired t-test template | 53 |
| 7 | Problem characteristics for set number 20, trial 6 | 55 |
| 8 | Time windows for visits performed by vehicle 1 | 57 |
| 9 | Validation of the model based on problem set 20 trial 6 – vehicle 1 | 57 |
| 10 | Results table - objective function | 59 |
| 11 | Paired t-test table, Alpha = 0.05 | 62 |
| 12 | Minimum objective function values | 64 |
| 13 | Results table – running time (min) | 67 |
| 14 | Competitive analysis ratios | 69 |
| 15 | Problem sets with different $edod_{nw}$ | 70 |
| 16 | Summary of problem instances | 86 |
| 17 | Results summary | 87 |
| 18 | Time Windows for Problem Set 20 | 92 |
| 19 | Validation of the model based on problem set number 20, trial 6 | 94 |

LIST OF FIGURES

| Number | Figure caption | Page |
|--------|---|------|
| 1 | The vehicle routing problem | 2 |
| 2 | complexity classes | 14 |
| 3 | Dynamic vehicle routing | 18 |
| 4 | Arrival times of dynamic requests | 22 |
| 5 | Effective degree of dynamism with time windows | 23 |
| 6 | Nearest neighbor heuristic | 24 |
| 7 | 2-exchange neighborhood | 25 |
| 8 | Solution approach | 42 |
| 9 | Original VNS algorithm | 47 |
| 10 | Proposed hybrid metaheuristic | 48 |
| 11 | Locations distribution | 55 |
| 12 | Pickup time windows – problem set 20 | 56 |
| 13 | Delivery time windows – problem set 20 | 56 |
| 14 | Normal probability plot of differences – order 1 | 60 |
| 15 | Normal probability plot of differences – order 2 | 60 |
| 16 | Normal probability plot of differences – order 3 | 61 |
| 17 | Normal probability plot of differences – order 4 | 61 |
| 18 | Box plot for neighborhood order 1. Alpha = 0.05 | 63 |
| 19 | Box plot for neighborhood order 2. Alpha = 0.05 | 63 |
| 20 | Box plot for neighborhood order 3. Alpha = 0.05 | 63 |
| 21 | Box plot for neighborhood order 3. Alpha = 0.05 | 64 |
| 22 | Effect of dynamically changing search parameters – neighborhood order 4 | 65 |
| 23 | Johnson transformation for the difference data (trials 2 and 6) | 66 |
| 24 | Pickup time windows - problem set 10 | 71 |
| 25 | Pickup time windows - problem set 11 | 71 |
| 26 | Delivery time windows - problem set 10 | 72 |
| 27 | Delivery time windows - problem set 11 | 72 |
| 28 | Effect of the degree of dynamism on the percentage of rejected requests – problem sets 8 (low $edod_{tw}$) and 9 (high $edod_{tw}$) | 73 |
| 29 | Effect of the degree of dynamism on the percentage of rejected requests – problem sets 12 (low $edod_{tw}$) and 13 (high $edod_{tw}$) | 73 |
| 30 | Effect of the degree of dynamism on the percentage of rejected requests – problem sets 16 (low $edod_{tw}$) and 17 (high $edod_{tw}$) | 74 |
| 31 | Effect of the degree of dynamism on the percentage of rejected requests – problem sets 24 (low $edod_{tw}$) and 25 (high $edod_{tw}$) | 74 |
| 32 | Two-Opt neighborhood | 83 |
| 33 | Or-Opt neighborhood | 84 |
| 34 | Cross neighborhood | 84 |
| 35 | Exchange neighborhood | 85 |

LIST OF ABBREVIATIONS

| Term | Abbreviation |
|---|--------------------|
| Capacitated Pickup and Delivery Problem | CPDP |
| Degree of Dynamism | dod |
| Effective Degree of Dynamism with Time Windows | edod _{tw} |
| General Pickup and Delivery Problem with Time Windows | GPDPWTW |
| Vehicle Routing Problem | VRP |

LIST OF APPENDICIES

| Appendices | Page |
|---|-------------|
| Appendix 1 – Neighborhood Structure Definitions | 82 |
| Appendix 2 – Data Sets and Results Summary | 86 |
| Appendix 3 – Time Windows for Problem Set 20 | 92 |

SOLVING THE DYNAMIC PICKUP AND DELIVERY PROBLEM WITH TIME WINDOWS USING HYBRID LOCAL SEARCH METAHEURISTICS

By
Jawad A. El-Omari

Supervisor
Dr. Sameh Shehabi, Assistant Prof.

ABSTRACT

The General Pickup and Delivery Problem with Time Windows is a broad model enclosing a whole set of problems, in which a fleet of vehicles has to satisfy a set of transportation requests. Hence, a route is constructed for each vehicle detailing which requests to visit, the order of the visits, and the arrival and departure times for each visit (scheduling the visits). All visits are performed within specified time windows, with the objective of minimizing the total traveled distance by all vehicles, but other objectives can be found.

While the General Pickup and Delivery Problem with Time Windows has many variants, all of which are classified as NP -hard problems; which means, there was no known polynomial time algorithm capable of producing an optimal solution, at least for large problem instances. As such, heuristic and metaheuristic methods were applied to gain *near* optimal solutions in reasonable running times. In this study, an online hybrid metaheuristic based on Variable Neighborhood Search, Tabu Search, and Guided Local Search was created and tested on one variant of the general model (i.e. the *Dynamic* Pickup and Delivery Problem with Time Windows). This study was also concerned with determining the effect of dynamically changing the hybrid's search parameters, during the search, on solution quality and running time, and the effect of changing the hybrid's neighborhood order on solution quality and running time. To achieve the aims of this study, the online hybrid was tested against problem instances *based* on the works of Christofides, Fisher, and Taillard (a total of 30 data sets). It should be noted that these sets are modified to include time windows to fit this study. Furthermore, the online hybrid was assessed based on the competitive analysis concept, but not in the exact sense due to the nature of the selected problem (i.e. NP -hard problem).

It was found that, for *large* problem instances, the dynamic change of search parameters produced better solutions more often, although there was no statistical evidence to support this. However, dynamically changing the search parameters had no obvious effect on the running time. Furthermore, the neighborhood order did not seem to have an effect on the solution quality, but the running time was obviously lower for a specific neighborhood order, compared to all other orders. As for the competitive analysis, it was shown that the online hybrid was capable of producing almost as good solutions as its offline algorithm counterpart. This was a good indication of how well the developed hybrid could perform under dynamic conditions.

INTRODUCTION

1.1 Overview and problem statement

The General Pickup and Delivery Problem with Time Windows (GPDPTW) is a generic model representing a whole class of problems, in which a fleet of vehicles is used to satisfy a set of transportation requests. In doing so, a route must be constructed for each vehicle detailing which transportation requests to visit, and the order of the visits. Each visit should be performed within specified time periods. In mathematical modeling terms, a complete weighted digraph $G = (N, E)$ (i.e. a graph in which every pair of distinct vertices is connected by an edge, associated with a cost, and the vertices are ordered pairs) consists of a set of nodes $N = \{1, 2, 3, \dots, n\}$ representing locations, which are often associated with revenues rv_i and time windows $[a_i, b_i]$ during which a visit must be performed, and a set of edges $E = \{(i, j) \mid i, j \in N\}$ representing arcs connecting the locations; edges are associated with weights detailing the cost, distance, or travel time incurred upon traversing them. A fleet of vehicles $V = \{k\}, k = 1, \dots, m$ is used to transport goods from one location to the other, usually from a central depot to customer locations, but that is not necessarily the case. Vehicles can be homogeneous in terms of capacity q_k , traveling costs c_{ijk} , and compatibility, or they can be heterogeneous; the compatibility issue is concerned with the vehicle's ability to serve specific customer requests.

The solution to a vehicle routing problem with time windows is a routing plan (a sequence) $S_k = \{n_1, n_2, n_3, \dots, n_i\}$ for each vehicle specifying the locations to visit, the order of the visits, and the arrival and departure times for each visit (scheduling visits). The solution must satisfy all problem constraints. Finally, an objective function to be optimized is used to guide the search for a solution, usually the objective is a cost function to be minimized (traveling

cost, distance, or time, number of vehicles used, response time... etc.), it may be a service level function to be maximized (number of customers served, revenue generated... etc.), or a combined weighted function (minimizing traveling cost and number of vehicles used). See figure 1.

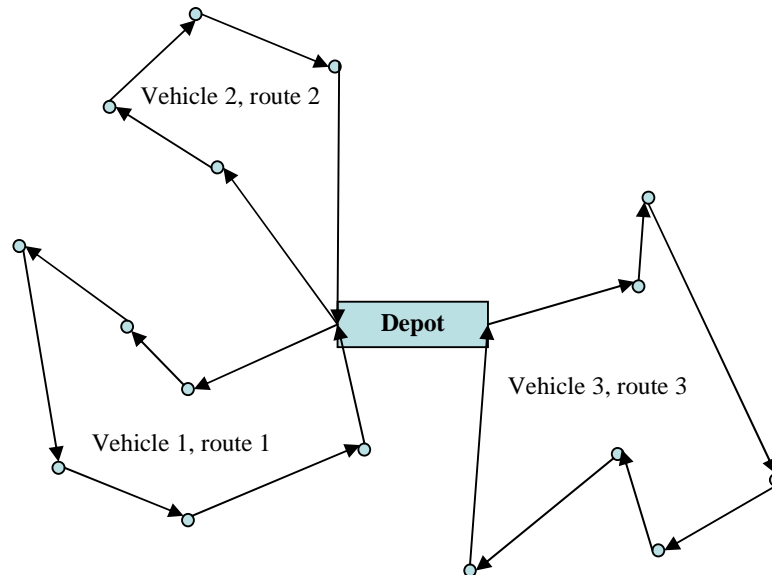


Figure1. The vehicle routing problem

1.2 Importance of the study and area of application

Vehicle routing models can be used to solve many real life applications some examples include but are not limited to: pickup and delivery of courier mail parcels, transportation of handicapped / elderly people, locating emergency medical service facilities, and dispensing salt or grit on snow-covered roads. As with all other models, not all problem characteristics can be captured. However, some models are closer to reality than others, the difference being in the constraints added, relaxed, or removed from the general model, and therefore creating a problem variant, in order to strike a balance between representing reality and solving the model with the given resources (e.g. within reasonable time). The most common problem variants are: the Capacitated Pickup and Delivery Problem (CPDP) in which different vehicles have different capacity limits, Pickup and Delivery Problem with

Time Windows (PDPTW) in which transportation requests must be satisfied within specific time periods, and the Vehicle Routing Problem (VRP) in which vehicles have fixed starting and ending locations, usually at a central depot. Other problem constraints are: precedence and coupling constraints (a pickup request must be served before the delivery request and both must be done by the same vehicle), heterogeneous fleet (vehicles have different capacities, speed, traveling costs...etc.), multi-depot VRP (vehicles are dispatched from multiple centers and each center serves a certain geographical area), vehicle compatibility constraint (certain visits require certain vehicles to perform the service, a common example is technician/repairman dispatching), and fixed or open starting / ending locations for vehicles (not all vehicles are required to begin / finish their routes at a fixed location like the depot).

When some problem variant is used to model a real life application, two major issues must be addressed: the availability of input information, and the certainty of the information. The availability of input information deals with the time when all needed information becomes accessible to the search algorithm. Information can be known entirely before the search begins (i.e. planning the routes), in which case the problem is considered static and the solution found will be the only solution executed for that problem, or, on the other hand, information can be known partially before the search begins and only an initial solution can be found (based on the partial information available at that time), as the remaining part of information becomes available the initial solution must be modified to accommodate such new input if possible; hence, the solution *dynamically* changes whenever new information appears and the problem is considered dynamic. The other major issue, certainty of information, deals with the variability of available information. At one end, information is known for certain and can not vary (i.e. deterministic), and at the other end, information is not known for certain and can vary (i.e. stochastic). The simplest example is the travel times,

they can be assumed to be constant, or, if one wishes to model a more practical situation, travel times should be random variables. With that in mind, any problem variant can be deterministic or stochastic and, at the same time, be static or dynamic. As expected, a stochastic dynamic problem is harder than a deterministic static problem and is closer to modeling real life scenarios.

1.3 Objectives

Given the vast number of problems that can be modeled, the focus of this work is on the dynamic deterministic pickup and delivery problem with time windows; the area of application is the courier mail delivery in which small parcels are picked up from one location and delivered to another on the same working day. The model will be solved using a hybrid local search metaheuristic; specifically, the objectives are three fold:

1. Create a hybrid metaheuristic to solve the dynamic pickup and delivery problem with time windows; the hybrid is based on Tabu Search, Guided Local Search, and Variable Neighborhood Search. A brief on these methods is in the Methodology chapter.
2. Investigate the effect of changing the neighborhood order on solution quality and solution speed (in the Variable Neighborhood Search context).
3. Investigate the effect of dynamically changing search parameters on solution quality and solution speed (in the Tabu Search and Guided Local Search context).

1.4 Outline

The remainder of this study is organized as follows: chapter 2 overviews the theoretical background needed and previous work in literature; section 2.1 overviews optimization problems in general, and specifies which type is used in this study, section 2.2

discusses search algorithms and their classifications, section 2.3 presents the related topics from the computational complexity theory, section 2.4 compares online optimization to offline optimization, and how competitive analysis is used as a performance measure for online algorithms, section 2.5 classifies the application area of this study (i.e. the dynamic pickup and delivery problem with time windows), and points out its major difficulties, section 2.6 gives a brief about solution approaches, and finally, section 2.7 reviews the most important work in literature, and specifies what this study adds along with the mathematical model used to do so. Chapter 3 overviews the metaheuristics used and the experimental methodology followed; section 3.1 specifies the solution approach followed in this study, section 3.2 presents the hybrid metaheuristic used, and section 3.3 lays out the experimental procedure for this study. Chapter 4 validates and discusses the results. And finally, chapter 5 concludes this study and recommends future extensions.

LITERATURE REVIEW

2.1 Optimization problems

The term optimization, or mathematical programming, is a field of mathematics concerned with systematically selecting values for real or integer variables such that a real function, called an objective function, is minimized or maximized. The variables must be chosen from a specified set known as the feasible region or search space; hence, optimization problems are considered *search problems*.

Definition 2.1 The optimization problem (Paepe, 2002)

An optimization problem Π is a three-tuple (I, S, f) such that:

- I is the set of the instances for Π .
- Given an instance $i \in I$, S_i denotes the set of feasible solutions of I .
- Given an instance $i \in I$, f_i is the objective function that attributes to each feasible solution x of I a real number $f_i(x)$, the so-called objective value of x .

The goal is to determine if, for a given an instance $i \in I$, there is a feasible solution, and if so, to find the feasible solution that has the smallest, or largest, objective function value amongst all feasible solutions; that is, a feasible solution x^* such that: $f_i(x^*) = \min/\max \{f_i(x) : x \in S_i\}$.

The major subfields of optimization are: linear programming, non linear programming, integer programming, mixed integer programming, dynamic programming, stochastic programming, and many others. This study focuses on a class of mixed integer programming called combinatorial optimization.

Definition 2.2 Mixed Integer Program (Nemhauser and Wolsey, 1999)

$$\min/\max (cx + hy : Ax + Gy \leq b, x \in Z_+^n, y \in R_+^p) \quad \dots (1)$$

Where Z is the set of nonnegative integral n -dimensional vectors, R is the set of nonnegative real p -dimensional vectors, and x and y are the decision variables. The problem is called a mixed integer program because it contains both integer and real variables. An *instance* of the problem is specified by the matrices / vectors: $c(1 \times n)$, $h(1 \times p)$, $A(m \times n)$, $G(m \times p)$, and $b(m \times 1)$. The set $S = \{x \in Z_+^n, y \in R_+^p : Ax + Gy \leq b\}$ is called the *feasible region*, or *search space*, a pair $(x, y) \in S$ is called a *feasible solution*, the function $z = cx + hy$ is called an *objective function*, and a feasible solution (x^*, y^*) for which the objective function is as small / large as possible (i.e. $cx^* + hy^* \leq cx + hy \forall (x, y) \in S$) is called an *optimal solution*. It is possible to have a problem with no optimal solution, this occurs when either the problem has no feasible solutions, or the constraints do not prevent improving the value of the objective function indefinitely in the direction of one or more of the variables (increasing or decreasing); thus, solving an instance of a mixed integer programming problem means finding an optimal solution, or showing that it is either infeasible or unbounded.

While there is no generally agreed upon definition of combinatorial optimization, it can be stated that combinatorial optimization is a part of integer programming that is concerned with the arrangement, grouping, ordering, or selection of discrete objects (decision variables) from a finite set (search space), such that an objective function is minimized or maximized. The above mathematical formulation also holds for combinatorial optimization.

2.2 Algorithms

Given an optimization problem the question becomes: “how to find the optimal solution.” One needs a search strategy that systematically explores the search space and finds the optimal solution, randomly selecting feasible solutions from the search space will simply not do; it requires a lot of time and there is no guarantee that the optimal solution will be found. The search strategy is known as the search algorithm.

Definition 2.3 Search Algorithm (Paepe, 2002)

An algorithm A for an optimization problem is a general step-by-step procedure that, on every instance, outputs a feasible solution $(x, y) \in S$, or outputs that it cannot find a feasible solution. Search algorithms can be broadly classified as either informed or uninformed algorithms.

An uninformed algorithm, also known as brute-force search, is a very general search strategy that systematically enumerates all possible solutions in the search space, and checks them against the problem's statement (objective function and constraints). By doing so, it guarantees finding an optimal solution, if one exists. However, when the search space is large, which is common for many practical problems, fully enumerating the solutions will take a very long time; consider the example of arranging 10 items, there are 3,628,800 ($10!$) different ways to do so, which can be fully explored within less than one second using an average computer; if the problem is to arrange 15 items, there will be about $1.3 \cdot 10^{12}$ possible solutions, which will take a few minutes to check on a computer; however, if the input to the problem increases to 20 items, there will be about $2.4 \cdot 10^{18}$ possible solutions to explore which will take about 10,000 years! All of this assumes that the computer has all the solutions ready to be tested and that no

solutions have to be generated. The point of this is to show how the search space increases dramatically as the size of the problem input increases; this is known as the *combinatorial explosion*.

An informed algorithm, on the other hand, uses some heuristic information (a *heuristic* is a Latin word meaning "to find" and can be thought of as an educated guess, a rule of thumb, a judgment call... etc.) to guide the search, and reduce the size of the search space; hence, it requires much less time than brute-force search. Nevertheless, it is not concerned with finding an optimal solution, any feasible solution would do. In other words, a heuristic algorithm compromises solution quality for faster running times. An example of heuristic information that can be used to solve an optimization problem is the "shortest processing time" heuristic used in scheduling; consider the problem of scheduling jobs on a machine with the objective of minimize penalties for late deliveries, one way to solve this problem is to work on jobs with shorter processing times first, then on jobs with longer processing times, this way more jobs will be finished in a work shift; while this approach may offer a good solution, it is not necessary the optimal solution; specific penalty costs for each job must be considered. The point is that using heuristics does not guarantee finding an optimal solution, nor does it produce consistent results for different problem instances, still, it can produce good solutions in a short time.

Yet another classification, that is relevant to this study, is the deterministic versus nondeterministic (randomized) algorithms. A deterministic algorithm is an algorithm that, given a specific input, always produces the same output, and the machine used to execute this algorithm will always go through the same sequence of

states (a state describes what a machine is doing at a particular instant of time), and such states are predetermined for each input. However, in many applications deterministic algorithms are very slow and impractical, for example given a list of n elements of which half are labeled with the letter A and the remaining half are labeled with the letter B , and it is required to find any element labeled as A . Using an algorithm that examines each element, and assuming that B -labels are sorted first, it will take $n/2$ operations to find the first A -labeled element, and for large values of n it will take a long time. In fact, for any deterministic algorithm, one can pass a problem instance that will cause the deterministic algorithm to perform in the worst possible manner. On the other hand, if we were to check the elements at random, then we will quickly find an A -labeled element with high probability, regardless of how the problem instance is presented. A nondeterministic algorithm uses some sort of random numbers as part of its input (or execution) and therefore produces a random output (random variable). Such classification will be used to explain the concept of competitive analysis later on.

2.3 Computational complexity theory

Whether an informed or an uninformed algorithm is used to solve an optimization problem, one seeks to find, and use, the “most efficient” algorithm for a given problem. Broadly speaking, the efficiency of an algorithm is measured by the resources needed by the algorithm to solve the *worst-case* problem instance. The branch of computer science that is concerned with this problem is known as the computational complexity theory, it is not to be confused with computability theory which is concerned with determining if a problem can be solved or not, regardless of the resources needed. The most common resources needed by an algorithm are time and space.

The space complexity function simply measures the amount of computer memory required by the algorithm. On the other hand, the time complexity function expresses time requirements by giving, for each possible input length (i.e. the number of bits required to encode an input), the largest amount of time needed by the most efficient algorithm to solve a problem instance of that size (Johnson and Garey, 1979). Of course, this function is not well-defined until one fixes the input encoding scheme, and the computer model used. This is why the Big O notation was created; it is a standard approach for estimating, and comparing, the complexity of algorithms, in spite of the encoding scheme or computer model used. The only requirement is the input size of the problem.

For example, if a problem has an input size of n and it takes an algorithm a total of $2n$ steps to reach a solution, we say that the complexity of the algorithm is a function of n ; whether it is $2n$ or $3n + 9$ does not really matter, what matters is that the complexity is proportional to n not n^2 or n^3 (i.e. it is in the “order” of n). Using the Big O notation, its complexity is written as $O(n)$. Suppose a complexity function is $3n^3 + 2n^2 + 12$, its Big O notation would be $O(n^3)$ as the fastest growing term n^3 will overwhelm all other terms when n is large enough (Kreyszig, 1999).

It should be noted, however, that if, for example, an algorithm has a complexity of $O(n^2)$, then at least one problem instance of size n takes that much time to solve, not all problem instances. In fact, most problem instances will take much less time. Formally speaking, a function $f(n)$ is in the order of another function $g(n)$ (denoted as $f(n) = O(g(n))$) whenever there exists a constant c such that $|f(n)| \leq c|g(n)|$ for large positive values of n . A *polynomial time algorithm* is defined as one whose time

complexity function, for some polynomial function p , is $O(p(n))$; while an algorithm whose time complexity function can not be bounded by a polynomial function, is called an *exponential time algorithm*. Thus, a problem that can not be solved by a polynomial time algorithm is considered *intractable*, for *large* input sizes.

As mentioned earlier, optimization problems (and combinatorial optimization problems) are considered search problems; for each search problem one can associate a decision problem which consists of the original search problem, the three-tuple (I, S, f) , and an additional input parameter known as the *bound* (B); however, the question about a decision problem is not what the optimal solution is, but rather about whether or not there exists a solution $x \in S_i$ such that $f(x) \leq B$, for a given instance I , which is answered with a simple yes or no. Obviously, solving the search problem entails solving the corresponding decision problem, but the opposite is not necessarily true. A decision problem can not be harder than the corresponding search problem; it can be as hard as the search problem, but never harder. Therefore, if a decision problem can be proved to be intractable, its corresponding search problem is also intractable.

One can classify, or group, computational problems and algorithms with related complexities into what is known as *complexity classes*. The theory of NP -completeness, which is a part of the computational complexity theory, differentiates between two complexity classes; P and NP . Of course there are many other classes, but these are the ones that are relevant. To understand how problems are classified as P or NP , consider the subset sum problem, which is this: for a given set of integers, is there a nonempty subset which sums up to zero? This is a decision problem. For example, does the set $\{-1, 6, 3, -9, 1, -5, 4\}$ contain a nonempty set that sums up to zero? Of course, the answer

is yes; the subset $\{6, 3, -9\}$ sums up to zero. Verifying that the subset $\{6, 3, -9\}$ sums up to zero is relatively “easy”; nevertheless, finding such a subset from the original set is not that easy.

The information needed to verify a “yes” answer is called a *certificate*, one can think of a certificate as a given solution; in this case the certificate was the subset $\{6, 3, -9\}$. Now, a decision problem that is given a certificate, for which a positive “yes” answer can be verified within a polynomial time, is placed in the class NP ; on the other hand, a decision problem that can be solved within a polynomial time, but has no certificate, is placed in the class P . Formal definitions of the classes P and NP can be found in Cook (1971) and in Garey and Johnson (1979). The question about whether a solution, to a decision problem, can be computed as easily as it can be verified remains unanswered, in fact it is a well known question in theoretical computer science; if a “yes” answer to a decision problem can be verified "easily" (i.e. in polynomial time), can the answers themselves also be computed in polynomial time? Put differently, does $P = NP$? Clearly, $P \subseteq NP$, but nothing can be said about whether or not $P = NP$.

On this matter, the concept of reduction is important. Basically, reduction is the means by which any instance of one problem can be transformed into an equivalent instance of another problem. If problem A can be solved using a polynomial time algorithm and problem B can be reduced to problem A using a polynomial time reduction (i.e. using a polynomial time algorithm for the reduction), then one can find a polynomial time algorithm to solve problem B. Thus problem A is always harder to solve. We say that a problem is NP -hard, if every other problem in NP can be reduced to it in polynomial time; however, this does not mean that NP -hard problems belong to

the NP class. In fact, an NP -hard problem that is in NP is called NP -complete; thus, NP -complete are the hardest set of problems in NP and if problems in NP -complete can be solved using a polynomial time algorithm, then *all* other problems in NP can be solved in polynomial time (Dorigo and Stutzle, 2004). Likewise, if there is a polynomial time solution for NP -hard problems, then NP -complete, and hence all NP , problems can be solved in polynomial time. See figure 1. An interesting note is that even if $P = NP$, NP -hard problems would still remain outside that complexity class, indicating how hard it is to solve such problems.

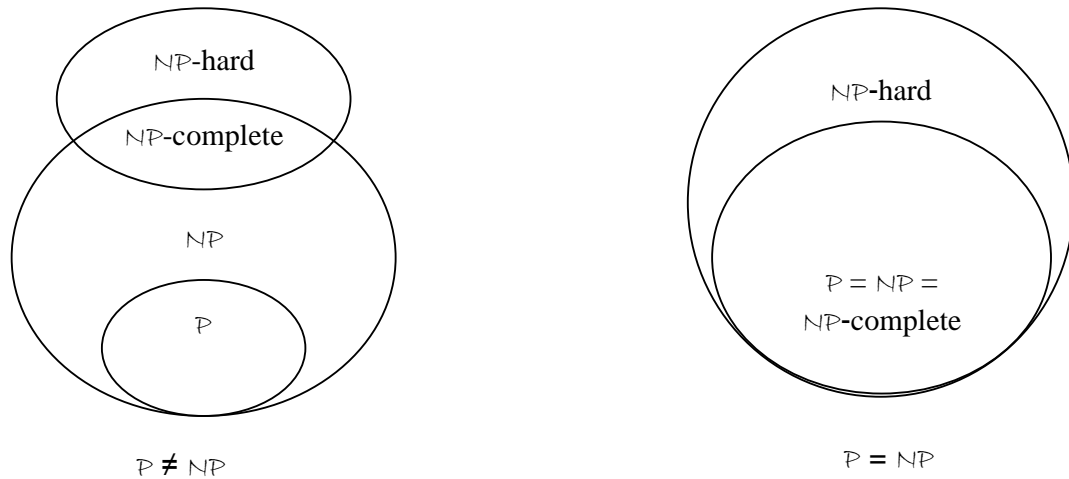


Figure 1. Complexity classes, by Esfahbod (www.wikipedia.org, October 2008)

2.4 Online versus Offline problems

In most real life applications not all input data, or input sequence, required to solve the optimization problem is available / known in advance to the algorithm; information is revealed as time passes, this is known as an *online* optimization problem, as opposed to an *offline* optimization problem where all input data, and sequence, is known a priori. Almost all online problems have an offline counterpart. The solution approach to online problems would be to find an initial solution given the partial information available, executing this initial solution, and then modifying it as more information unfolds over time, using what is known as an *online algorithm*. This means that the online solution can not be better solution than the offline solution, assuming the offline solution is optimal. A common example is the courier mail delivery where a vehicle has to make a set of predetermined delivery stops in addition to making pickup stops that are received throughout the day.

The efficiency of online algorithms can be evaluated using what is known as *competitive analysis*, which was first introduced by Sleator, *et al.* in 1985. A basic concept in competitive analysis is that of the *competitive ratio*, a performance measure for online algorithms, where an online algorithm is compared to an offline algorithm that produces an optimal solution. Let Π be a maximization problem with an objective f , let O be an offline algorithm returning an optimal solution $O(I)$ for f , on a fully revealed input sequence I , and let DA be a deterministic online algorithm. The performance of DA is measured by the ratio between the optimal offline value $f(O(I))$ and the value $f(DA(I))$ over each possible input sequence I (Hentenryck, 2006). That is:

$$\max_I \frac{f(O(I))}{f(DA(I))} \dots (2)$$

Definition 2.3 c -competitiveness for deterministic algorithms (Kallrath, 2005)

A deterministic online algorithm DA is said to be c -competitive if, for all input sequences I , the ratio between the optimal offline value $f(O(I))$ and the value $f(DA(I))$ is bounded by c . That is:

$$\frac{f(O(I))}{f(DA(I))} \leq c + \alpha \quad \dots (3)$$

Where α is a constant added to make up for any initial condition differences between the online and offline algorithm.

As mentioned earlier, any deterministic online algorithm can be set to perform as worst as possible by passing the hardest problem instance(s) to it, this is what is known as an *adversary*. Competitive analysis implicitly assumes a worst-case problem instance is passed to the online deterministic algorithm, and is therefore not representative of the practical performance of the online algorithm. More details on the drawbacks of competitive analysis can be found in Koutsoupias and Papadimitriou 2000. To overcome this issue, randomized algorithms can be used instead; however, this requires a new definition of the competitive ratio which in turn depends on the type of adversary used.

Borodin and El-Yaniv (1998), show that there are three types of adversaries: the *oblivious adversary*, which chooses an input sequence in advance based only on the description of the online algorithm, it cannot adjust its input sequence based on the behavior of the online algorithm afterwards; the *adaptive offline adversary*, which can build / change the input sequence online and can base future requests on the actions of the online algorithm on previous requests; and finally, the *adaptive online adversary*, which can build the input sequence online like the adaptive offline adversary, but it

must also generate its own solution online. Clearly, adaptive adversaries are more powerful than the oblivious adversary.

Definition 2.4. c -competitiveness for randomized algorithms (Paepe, 2002)

A randomized online algorithm RA is said to be c -competitive against an oblivious adversary if, for all input sequences I , the ratio between the optimal offline value $f(O(I))$ and the value $f(RA(I))$ is bounded by c . That is:

$$\frac{f(O(I))}{f(RA(I))} \leq c \quad \dots (4)$$

Only an oblivious adversary will be used throughout this work.

2.5 Classification of the dynamic routing problem and related difficulties

As pointed out in the introduction, this work focuses on the dynamic pickup and delivery problem with time windows. In what follows, classification of the problem, how it differs from its static counterpart, and the related difficulties / issues brought on by being dynamic are presented. Psaraftis (1988) states that: “if the output of a certain formulation is a set of preplanned routes that are not re-optimized and are computed from inputs that do not evolve in real-time, then the problem is considered static” (pp: 3); on the contrary, Psaraftis (1988) states that: “if the output is not a set of routes, but rather a policy that prescribes how the routes should evolve as a function of those inputs that evolve in real-time, then the problem is considered dynamic” (pp: 4). Clearly, time is an essential element in the classification of the problem; based on that, the static routing problem is characterized by:

- All relevant information (e.g. customer locations, service times, travel times, demand... etc.), to solve the problem, is known to the algorithm before the execution of the solution begins.

- All information related to the solution does not change after the problem is solved (routes created).

And the dynamic routing problem is characterized by:

- Not all relevant information, to solve the problem, is known to the algorithm when the execution of the solution begins.
- Some information can change after the initial solution is created.

So, at any time t , in the dynamic vehicle routing problem, there are: completed visits corresponding to the part of the route that is already executed, this part cannot be modified afterwards, current visits corresponding to the location of vehicles, planned visits corresponding to the part of the route that is not yet executed, and new visits that dynamically appear over time and have to be satisfied by one vehicle if possible. See figure 2.

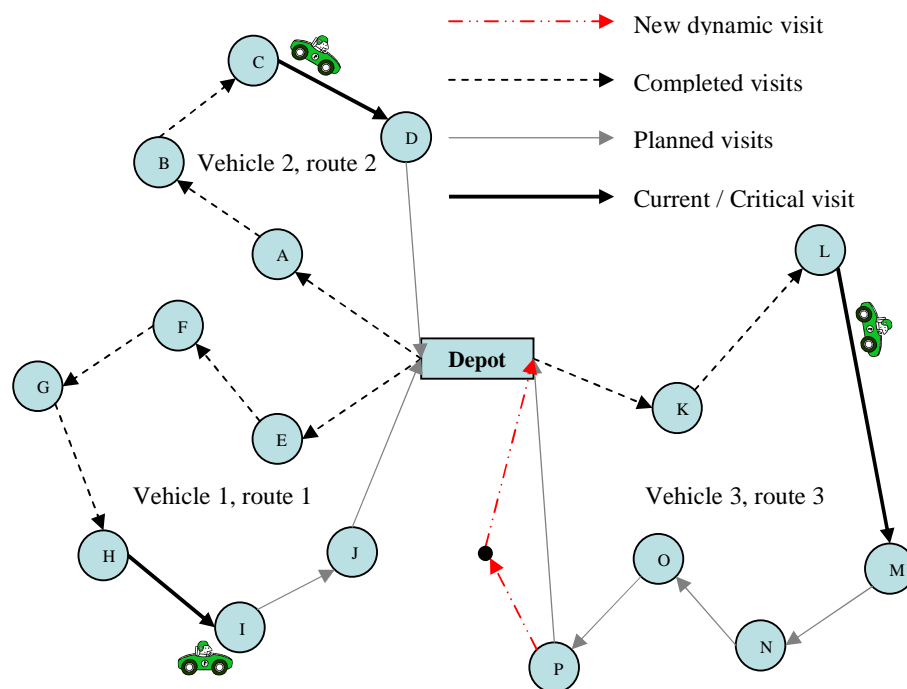


Figure 2. Dynamic vehicle routing

Psaraftis (1988) and Psaraftis (1995) lists twelve issues / difficulties brought on by the dynamic routing problem as opposed to the static routing problem, below is a brief discussion on the most important:

1. Time is critical: when a new request appears, the customer expects an immediate answer whether his / her request can be satisfied on the same day or not. So, the time available to find a feasible insertion point for new requests is seconds, not minutes; in addition, once new requests are inserted, a re-optimization is run, this re-optimized solution is the one dispatched, so re-optimization can not last a long time as this will leave some vehicles idle.
2. The time a new visit appears (τ): a new request arriving a few minutes before its deadline, or before the end of the working day, is much more difficult to satisfy than a request arriving several hours in advance.
3. Time constraints may be soft: in many practical situations a customer requesting an immediate service will most likely tolerate little violations in the time windows, this of course comes at an added extra cost to the objective function.
4. Near-term requests may be more important: in a dynamic setting it would be unwise to immediately commit vehicles to long-term requirements, as new requests may appear at any time. Mitrović-Minić, *et al.* (2004), propose a method to balance between committing to short-term and long term requests, the method is called *double horizon*.
5. Future information may be uncertain or unknown: in a dynamic setting future requests are not known with certainty, at best they are probabilistic.
6. Flexibility to vary vehicle fleet size is lower: in a static setting, the time between finding a solution and executing it allows the adjustment of the fleet; however, in a dynamic setting backup vehicles may not be available.

7. Commit policy: at some point in the execution of a solution, the vehicle must be committed to its next destination(s). Prior to this time, the solution can change, but once a particular part of the solution has been committed to, that part can not change. When to make such a commitment is a fundamental question in dynamic routing, one policy is a “latest commitment” policy, where commitment is left until the last possible moment, the next visit is only communicated to the vehicle at the latest possible time that will allow it to reach that visit (and all subsequent visits in the current temporary solution) by its deadline.
8. Critical node: a critical node is defined as a customer who is currently using a vehicle, or to whom a vehicle is heading. Critical nodes need to be identified instantly when a real-time demand arrives so that the route can be reconstructed. Critical nodes can be easily identified graphically, for example in figure 2 above (page 18) nodes D, I, and M are critical nodes. Mathematically put, if vehicle k is traveling from node i to node j through node h ($i \rightarrow h \rightarrow j$), and the departure times from the nodes are: d_i , d_h , and d_j , then a critical node, at time τ , is identified as h if $d_i < \tau \leq d_h$, or j if $d_h < \tau \leq d_j$
9. Inserting new visits and re-optimization: when new visits appear they have to be inserted into the current solution and the solution is then re-optimized, this can cause problems as the improvements may conflict with the new visits, to deal with this one can:
 - Abort the search whenever new visits are added to the current solution, and then restart it from scratch. This method may degrade the efficiency of the re-optimization as it is not allowed to run for a sufficient amount of time in most cases.

- New visits may be stored until the search finishes and then inserted altogether; however, the feasibility of the new visits may not be known at the time of accepting them.
- New visits could be accepted and included in a tentative solution, once the re-optimization ends, only improvements that do not conflict with the new visits are incorporated.

2.5.1 Measuring the difficulty of a dynamic routing problem; the degree of dynamism:

With the above mentioned issues, a common measure of difficulty is required to compare problem instances, the degree of dynamism is generally accepted as such a measure. Lund, *et al.* (1996) were the first to introduce this concept; basically, the degree of dynamism (*dod*) for a given instance is the number of dynamic / immediate requests appearing throughout the day n_{imm} divided by the total number of request served on that day n_{tot} (i.e. the static requests plus the dynamic requests).

$$dod = \frac{n_{imm}}{n_{tot}} \quad \dots (5)$$

However, this measure does not take into account the time dynamic requests become available; meaning, a problem instance with 25 immediate requests, out of a total of 100, appearing near the beginning of the working day would have the same degree of dynamism as an instance with 25 immediate requests, out of 100, appearing near the end of the working day. See figure 3. Clearly, requests in case 2 are much more difficult to satisfy, and they do not allow much time for the re-optimization algorithm to run; hence, even if such requests could be satisfied, solution quality is

likely to be poor, because there is not much time to search for a solution, as opposed to case 1.

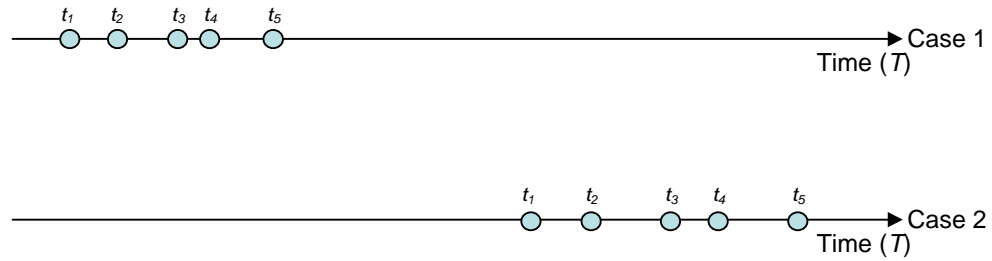


Figure 3. Arrival times of dynamic requests

To overcome this issue, the effective degree of dynamism (*edod*) is defined as the average of how late the requests are received (t_i) compared to the latest possible time (i.e. T) the requests could be received (Larsen, 2001).

$$edod = \frac{\sum_{i=1}^{n_{imm}} \frac{t_i}{T}}{n_{tot}} \quad \dots (6)$$

This measure ranges between 0 (a completely static system) to 1 (a completely dynamic system, where all requests are received at time T), that is:

$$\lim_{t_i \rightarrow T \forall i} edod = 1 \quad \dots (7)$$

s.t. $l_i - t_i \leq T \forall i = 1, 2, 3, \dots, n_{imm}$

Time windows can also be incorporated into this measure as follows; let e_i be the earliest time a service can begin, let l_i be the latest time the service can begin, and let r_i be the response time; difference between the latest time a service can begin and the time the request appears (i.e. $r_i = l_i - t_i$). In figure 4 it is clear that in case 2 the requests are much harder to satisfy than in case 1.

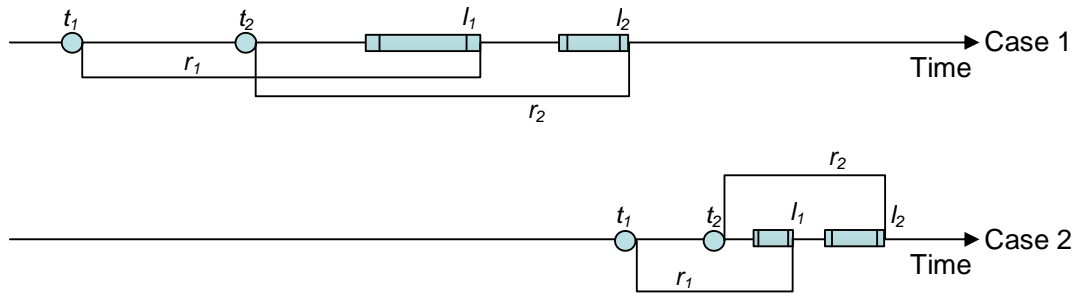


Figure 4. Effective degree of dynamism with time windows

Larsen (2001) extended the *edod* to include time windows like so:

$$edod_{tw} = \frac{1}{n_{tot}} \sum_{i=1}^{n_{imm}} \left(\frac{T - (l_i - t_i)}{T} \right) \quad \dots (8)$$

2.6 Solution approaches

In general, there are two major solution approaches for optimization problems: exact methods, and approximate methods; exact methods guarantee reaching an optimal solution to any problem instance, assuming the required resources are available (i.e. the solution algorithm is allowed to run for a sufficient amount of time, and there is enough computer memory to withhold all necessary data while the algorithm is running). Such methods include: branch and bound, branch and cut, branch and price, Lagrangian relaxation, and column generation. Nevertheless, because of combinatorial explosion (mentioned earlier), and the resources required to reach, if possible, an optimal solution for NP-hard and NP-complete problems, one can not afford the time, or memory, needed to reach such a solution; and so, some sort of a trade off between solution quality (i.e. reaching an optimal solution) and resources required (mostly time), to find such a solution, is needed. In doing so, heuristic and metaheuristic approaches were created as approximate methods.

Approximate methods can be classified as constructive methods and local search methods (Larsen, 2001 and Dorigo and Stutzle, 2004). Step by step and without backtracking, *constructive methods* build a complete solution by incrementally adding solution components to an empty solution. The choice of which solution component to add at each step is usually based on heuristic information; still, solution components can be added at random in some cases. As an example, consider the problem of finding the shortest route that starts and ends at the same location passing through a number of other locations. See figure 5. One constructive method would be to start from a point and then move to a point closest to the current point (this is known as the nearest-neighbor heuristic); starting from point A, the solution will be: $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow A$

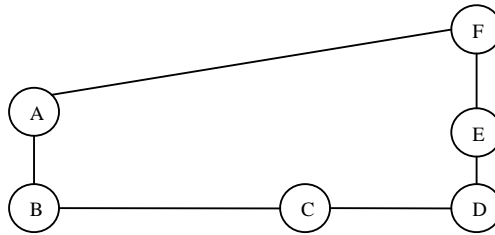


Figure 5. Nearest neighbor heuristic (Dorigo and Stutzle, 2004)

The point to stress is that although heuristic approaches produce solutions fast, the solutions are not optimal and the approach is inconsistent when applied to different instances of the same problem. *Local search*, on the other hand, starts with an initial feasible solution and tries to improve it by making a small change to the current solution, to produce a new solution. This new solution is tested against the problem constraints for feasibility, and its cost is computed. If the new solution is feasible and has a reduced cost, it is accepted as the current one; otherwise, the current solution remains unchanged. This process is repeated (using different solution changes on

subsequent tries) until some stopping condition is met; the stopping condition can be based on time, number of changes attempted, or the fact that no more changes to the solution can be found and accepted under the specified conditions. This is known as a *greedy search* method as it accepts only improving moves, it will be shown later how this could affect the quality of the solution.

Local search requires the definition of a neighborhood structure (a *neighborhood* is a set of solutions that can be reached from the current solution in one single step), and an examination scheme that determines how the neighborhood is searched and which neighbor solutions are accepted. Creating a neighborhood structure is very dependent on the problem and may have many forms, a common example is the *k-exchange* neighborhood structure; basically, the *k* exchange neighborhood of a candidate solution *s* is the set of candidate solutions *s'* that can be obtained from *s* by exchanging *k* solution components; for example, in figure 5 above a 2-exchange neighborhood consists of the set of all the candidate solutions *s'* that can be obtained by exchanging two pairs of arcs in all possible ways. See figure 6.

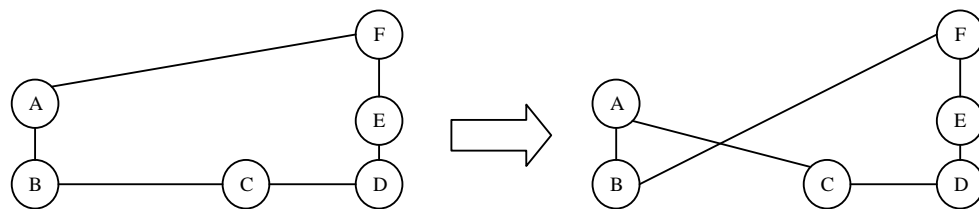


Figure 6. 2-exchange neighborhood (Dorigo and Stutzle, 2004)

The examination scheme either uses the best accept rule, which chooses the neighbor solution giving the largest improvement in the objective function value, or the first accept rule, which accepts the first improving move found. Regardless of which

examination scheme is used, local search can only produce optimal solutions within its defined neighborhood, meaning local search will, at best, terminate at a *local optimal* solution.

To escape local optima, metaheuristics were created. A metaheuristics (“meta” a Latin word meaning “beyond”) is a master strategy, or a general framework, that guides and modifies other heuristics to produce solutions beyond those normally identified by local search heuristics (Glover, 1986; and Glover and Laguna, 1993). Compared to exact methods, such as branch-and-bound, metaheuristics cannot generally ensure the exploration of the entire search space; still, they provide guidance to areas of the search space containing high quality solutions. Well-designed metaheuristics avoid getting trapped in local optima or cycling (sequencing the same visited solutions over and over), and have a mathematical proof of reaching optimal solutions if allowed to run for a sufficient amount of time.

One way to escape local optima is to allow degrading moves to be taken, consider the following example:

$$\min \text{cost} = c(x + y) + (c + 1)z$$

where

$$x, y \in [0,1]$$

$$z = \begin{cases} 0, & x \text{ and } y \text{ are similar} \\ 1, & x \text{ and } y \text{ are different} \end{cases}$$

Variables x and y are the decision variables (binary variables), c is a constant, and z is also a binary variable which equals 0 if both x and y are similar (i.e. either both are 0 or both are 1); otherwise, z will have a value of 1. Now assume that the current solution is at point $x = 1, y = 1$ and assume that only one variable can be changed at a time (i.e. a move from one solution to another in the neighborhood entails changing only one

variable), clearly no one single move can be taken, from the current solution, to reduce the objective function value (in fact, all moves will increase the objective function by 1); nonetheless, a lower cost does exist (i.e. $x = 0, y = 0$) but can not be reached without increasing the objective function value first. See table 1.

Table 1. Local optima example

| x | y | Cost |
|----------|----------|-------------|
| 1 | 1 | $2c$ |
| 1 | 0 | $2c + 1$ |
| 0 | 1 | $2c + 1$ |
| 0 | 0 | 0 |

Clearly, this is a local optima problem which can be overcome by allowing degrading moves to be taken, in hopes that subsequent moves will produce better solutions. Care should be taken, however, about the way in which the objective function is allowed to degrade; to simply accept any move without guidance leads to a random walk through the search space, and is unlikely to bring about a good solution. Instead, metaheuristics use controlled methods of accepting degrading moves, in order to both escape local optima and then go on to find better solutions.

2.7 Previous work

The variety of applications that use routing models to solve routing problems is extensive; on one hand, the problem is to plan a route for one vehicle to deliver a predetermined amount of goods to a specified set of locations, with all needed information being known for certain and known in advance; and on the other hand, the problem is to plan a set of routes for a fleet of heterogeneous vehicles doing both delivery and pickups of goods at locations that are not all known in advance, but rather revealed over time, and with all the needed information being uncertain or probabilistic at best, add to that the time limitations set on performing each pickup / delivery visit.

Needless to say that such a wide range has received a great deal of attention and research, and the amount of literature now available is enormous. In the efforts to provide a comprehensive, well structured review of the pervious related work, this section will be divided into three parts:

- Part I: review of the static vehicle routing problem with exact solution methods.
- Part II: review of the static vehicle routing problem with approximate methods.
- Part III: review of the dynamic vehicle routing problem and its related difficulties.

2.7.1 The static vehicle routing problem – exact methods

As mentioned earlier, the static vehicle routing problem is concerned with finding an optimal route for a fleet of vehicles performing a predetermined set of visits. The vehicle routing problem is classified as NP -hard, and therefore can only be solved, to optimality, for small problem instances which have limited application. Exact methods include: Dynamic Programming, Lagrangian relaxation, and column generation; the most recognized work in each principle is summarized next.

Kolen, *et al.* (1987) were the first to solve the vehicle routing problem with time windows to optimality using Branch-and-Bound and Dynamic Programming methods; each node, in the Branch-and-Bound tree, corresponds to three sets: set F which is the set of feasible routes starting and ending at the depot, set P which is a partially built route starting at the depot, and set C which is a set of customers forbidden to be next on P ; branching is done by selecting a customer that is not forbidden and that does not appear on any route, then two branches are generated: one in which the partially built route P is extended by the selected customer, and one where the selected customer is forbidden to be the next customer on the route, at each Branch-and-Bound node

Dynamic Programming is used to calculate a lower bound on all feasible solutions. A similar approach is used in Caramia, *et al.* (2002) for a multi-cab metropolitan transportation system; basically, a new request is first inserted in one of the planned routes and this route is then optimized using a Dynamic Programming algorithm; here too the approach is applicable only to small problem instances.

Fisher, *et al.* (1982) use Lagrangian relaxation combined with a multiplier adjustment method to solve a mixed integer programming model of a commercial delivery system. Kohl, *et al.* (1997) relax the constraints guaranteeing that every customer is served exactly once and added a penalty to the objective function; the master problem now consists of finding the optimal Lagrangian multipliers and is solved by a sub-gradient optimization method; the sub-problem becomes the shortest path problem with time windows and capacity constraints, and is solved using a Dynamic Programming approach. Kallehauge, *et al.* (2006) use a column generation approach to solve the vehicle routing problem with time windows; a master problem and a sub-problem are created, and a branch-and-bound framework is employed along with acceleration strategies that increase the efficiency of branch-and-bound method. The approach, when tested against Solomon's benchmark, produced the following regarding travel distance: 361.6 for R207.25, 370.7 for R209.25, 350.9 for R211.25, and 1143.2 for R201.100.

Westphal and Krumke (2007), study a large scale real-world vehicle dispatching problem with soft time windows (i.e. time windows can be violated, but with an added extra cost to the objective function); they developed a pruning scheme based on matchings in order to speed up the branch-and-bound enumeration in the column

generation process; computational results on real-world data show that, overall, the computational time is less (only took 38% of the original time), and only 24% of the nodes were explored compared to the old pruning scheme. Laporte, *et al.* (2002) propose a stochastic integer programming framework to solve the stochastic vehicle routing problem (stochastic in the sense of random demand at each customer location), optimal solutions were reached for instances with at most 25 locations and using only 4 vehicles. Letchfor, *et al.* (2006) propose an algorithm based on Branch-and-Cut to solve the open vehicle routing problem with capacity constraints (an open routing problem is one in which vehicles are not supposed to return to a specific location at the end of their routes); the algorithm was tested on several standard instances, small instances, and was able to produce optimal solution. Heuristic methods were then applied to the same instances and their “near-optimal” solutions were compared to the optimal ones. This, in turn, enabled the assessment of the solution quality produced by heuristic methods.

2.7.2 The static vehicle routing problem – approximate methods

With larger problem instances, exact solution methods are impractical and may sometimes never reach an optimal solution; thus, one compromises solution quality for solution speed. In doing so, heuristic and metaheuristic methods are used; in what follows, approximate methods will be categorized into constructive methods and improvement methods; later, some metaheuristics are reviewed.

The first paper on constructive (route-building) heuristics, for the vehicle routing problem with time windows, was that by Edward, *et al.* (1986) in which they present an extension of the traditional Savings heuristic, the algorithm begins with all

possible single customer routes (depot \rightarrow customer $i \rightarrow$ depot) and iteratively calculates which two routes can be combined resulting in the maximum saving (the saving between customers i and j is calculated as $saving_{ij} = d_{i0} + d_{0j} - Gd_{ij}$, where G is referred to as the route form factor). Landeghem (1988) describes a heuristic based on the Savings method with the additional criterion that models an intuitive view of time influence on route building in the vehicle routing problem with time windows; experiments show that this added criterion yields significantly better results to the problem compared to pure routing heuristics, and can compete (in terms of speed) to other heuristics specially built for this kind of problem; however, the savings heuristic generally produces solutions of lower quality as the last un-routed customers tends to be scattered over the geographic area.

To overcome this problem, Potvin and Rousseau (1993) apply an insertion algorithm for the vehicle routing problem with time windows; the algorithm builds routes in parallel and uses a generalized regret measure over all un-routed customers in order to select the next candidate for insertion; numerical results on the standard set of Solomon's benchmark problems compare well with other sequential algorithms presented by Solomon 1987, but are still far from optimal. In addition, Brown, *et al.* (1987) consider the problem of dispatching petroleum tank trucks under various constraints and applied an assignment and routing heuristic on known requests within a rolling horizon; the heuristic first assigns the loads to available vehicles and then solves a traveling salesman problem to optimize each route. Bausch, *et al.* (1995) addressed a similar problem but the heuristic used first generates clusters of customers for each vehicle type, then the total distance traveled is optimized within each cluster using either a heuristic or an exact algorithm, depending on the problem size.

The concept of neighborhoods is central to almost all route improving heuristics, examining some, or all, of the solutions in a neighborhood might reveal solutions that are better than the current one, in which case a move is made to that better solution, and the process is repeated; at some point no better solution(s) can be found and an optimal point is reached, in most cases this will be a local optimum, but it might be a global one. Maybe the most used improvement heuristic in routing problems is the *k-opt* heuristic; in its basic form (i.e. the 2-opt), one seeks a route that crosses over itself and reorders it so that it does not (CROES, 1958). Potvin and Rosseau (1995) present two variants of the k-opt heuristic: the *2-Opt** and the *Or-Opt*; in Or-Opt a segment of the route is moved to another place on the same route; whereas in the *2-Opt** a segment of a route is exchanged with a segment of another route, computational results were only tested against randomly generated data sets.

Osman (1993), created the λ -interchange neighborhood for the vehicle routing problem; here, a subset of customers of a size less than λ in one route is interchanged with a subset of size less than λ in another route; computational results reported on a sample of seventeen benchmark test problems, and nine randomly generated problems; the λ -interchange method improved the solution in terms of the number of vehicles used, and the total distances travelled.

2.7.3 The dynamic vehicle routing problem and related difficulties

The pickup and delivery problem is sometimes modeled as a *dial-a-ride* problem; the name originated from the application of transporting elderly, or handicapped, people from one location to another. A dynamic single-vehicle dial-a-ride problem was first addressed by Psaraftis (1980) with an exact algorithm; based on a

finite time horizon, a series of static problems was solved through a Dynamic Programming algorithm and optimal solutions for small problem instances were found within reasonable time. Later, Psaraftis (1983) generalized his approach to account for time window constraints, and in 1988 Psaraftis introduced a dynamic version of the Vehicle Routing Problem, where it was stated that if the solution is a set of preplanned routes that are not re-optimized and are computed from the inputs which do not evolve in real-time, then the problem is classified as static; on the other hand, if the solution is a policy that recommends how the routes should evolve as a function of the inputs that evolve in real-time, then the problem is classified as dynamic. Psaraftis presented a number of research topics the most important being whether a vehicle should wait at the current location, after finishing the service, in order to “batch” newly arriving requests, or travel immediately after the service; however, Psaraftis did not address these issues back then.

A few models were created in the early nineties which make use of queuing theory; basically, a vehicle is modeled as a traveling server moving from one customer location to the next. Such models required greater computational time and more complex mathematical solutions; therefore, they were not as popular as other models although they were more precise. The most comprehensive work using queuing models is that of Bertsimas and Ryzin (1991); they proposed a generic mathematical model for a single un-capacitated vehicle, traveling at a constant speed in the Euclidean plane. They called this problem the dynamic traveling repairman problem. The objective was to minimize the total time in the system (i.e. waiting time plus service time). Using approaches from queuing theory, simulation, combinatorial optimization, and probability, an optimal routing policy was found for light traffic conditions; in addition,

it was shown that the waiting time grows much faster, than that in traditional queues, as the traffic intensity increases.

Shortly afterwards, Bertsimas and Ryzin (1991) extended their previous work by considering the case of multiple identical vehicles m , with unlimited capacity as well, and the case in which each vehicle can serve at most a predetermined number of customers. They showed that, in heavy traffic conditions, the system time is reduced by a factor of $1/m^2$ over the single-server case, they even extended their work further by using a more general probability distribution to describe the arrival of new requests (a renewal process instead of the Poisson distribution), and the request locations were assumed to be arbitrary, instead of being uniformly distributed. With these “*more realistic*” assumptions, Bertsimas and Ryzin showed that optimal policies used in the static vehicle routing problem can produce near optimal, and in some cases optimal, solutions for the dynamic version of the problem.

Swihart and Papastavrou (1999) extended the dynamic traveling repairman model to include the same day pickup and delivery constraints (i.e. a vehicle must pickup goods from one location and deliver them to another location on the same day), both pickup and delivery locations are independent and uniformly distributed over the service region; a number of routing policies were tested against simulated data, and the nearest neighbor policy performed best in heavy traffic conditions for both the single and multiple vehicle cases. Kilby, *et al.* (1998) divided the working day horizon into fixed time slots during which arriving requests are only considered at the end of each time slot, and the optimization algorithm was thus allowed to run on the current static problem for the duration of a time slot.

A dynamic vehicle routing problem, with time windows, was considered in Gendreau, *et al.* (1998); here, a hybrid approach was used to solve operational problems facing long-distance courier mail companies; first, a simple insertion heuristic is used to insert dynamic requests, and then a Tabu Search with an adaptive memory is applied to the initial solution to improve it until the occurrence of the next event. The same approach was extended in Gendreau, *et al.* (1999) to accommodate for pickup and delivery of parcels in a local express mail company (the addition here is that the same vehicle should be used to do the pickup and delivery, and the pickup should proceed the delivery). A similar approach for a dynamic dial-a-ride problem was used in Attanasio, *et al.* (2004); here, the authors used a parallel implementation of a Tabu Search heuristic previously reported in Cordeau and Laporte (2003) for the static version of the problem; whenever a new service request occurs, an insertion heuristic is first applied, to know if the request can be accepted or not, then Tabu Search is applied to optimize the routes.

A dynamic vehicle routing problem, with no time windows, was considered in Gambardella, *et al.* (2003) and in Montemanni, *et al.* (2005); here, the entire problem was solved as a series of static vehicle routing problems using Ant Colony System metaheuristic; useful information about the solutions produced is transferred from one static problem to the next through a pheromone conservation mechanism.

An interesting approach that is seldom addressed in literature is that of diversion; Regan, *et al.* (1995) were the first to explore this idea; basically, it consists of diverting a vehicle away from its current planned destination to serve a request that has just occurred in its district. The work of Ichoua, *et al.* (2000) propose a broader view of

diversion in the context of a long-distance courier mail service where parcels are picked up from one area, brought back to a central depot, and then delivered to another area on the next business day. Addressing diversion in this context is more challenging, as consolidating and sequencing the requests becomes an important issue. A matter of concern in the diversion approach is the time allocated for the optimization algorithm; since vehicles are moving fast and dynamic requests may appear at any time, diversion opportunities are easily lost; basically, if the time allowed for the algorithm is too large, diversion opportunities can be lost. On the contrary, if the time is too small, solution quality might suffer.

Yet another issue, that is central to dynamic routing problems in general, is that of anticipating future requests and how it affects routing decisions (e.g. relocating idle vehicles, accepting early requests, setting a cut off time for accepting requests... etc.). An approach introduced by Mitrović-Minić, *et al.* (2004), for a pick-up and delivery problem with time windows, is that of the double horizon. Here, both a short-term and a long-term planning horizons are considered, with different objective functions for each horizon; the objective of the short term horizon corresponds to the true objective (e.g. minimizing the total traveled distance), whereas the objective associated with the long-term horizon tries to allocate larger slack times, in the constructed routes, to better accommodate future requests; the optimization is done for both objective functions using a simplified version of Tabu Search.

In routing problems with time windows, scheduling the vehicle's visits is a critical issue. Scheduling, in this context, means the determination of the arrival and departure times at each location; this, in turn, requires setting some waiting strategies at

each location. In static routing problems with time windows, the only waiting strategy is the drive first strategy; a vehicle should move from its current destination as soon as the service ends, thus causing it to wait at the next destination if it arrives before the time window begins. In a dynamic setting, however, the drive first strategy would not apply simply because new requests may appear at any time; it would be better for the vehicle to wait at the current destination, for as long as possible, in order to reach its next destination at its time window's lower bound, thus allowing more new requests to arrive and be better inserted into the current routes. This is known as the wait first strategy, it is expected to produce shorter routes than the drive first strategy, but will require more vehicles to do so.

Between the two extreme strategies (wait first and drive first) Mitrović-Minić and Laporte (2004) showed that a combination of both strategies gives the best results with regard to the number of vehicles and total traveled distance; their approach is based on the dynamic partitioning of planned routes into segments made of close locations; within a segment, a vehicle always departs as soon as possible from its current location; but when it is time to cross a boundary between two segments, the vehicle waits at its current location for a fraction of the time available up to the latest possible departure time. In Ichoua, *et al.* (2001), a vehicle that has completed its service at one location should wait for some amount of time if its next destination is far, and the probability of a new request arriving within its surrounding area, in the near future, is high enough; thus, probability distributions are used in the waiting strategies.

This work is focused on the dynamic pickup and delivery problem with hard time windows and all relevant input data is assumed to be deterministic. As for the

degree of dynamism, some requests are assumed to be known a priori, and initial routing plans will be built upon them; still, there are requests that emerge afterwards which should be accommodated for if possible. The area of application for this model is the courier mail pickup and delivery problem, in which small parcels are picked up from one location and delivered to another location on the same working day; as the parcels are small (mainly documents), the capacity constraint is not considered, and the fleet of vehicles is assumed to be homogeneous. In addition, as the problem is dynamic the vehicle may visit the same location more than once (the customer may request more than one service per day), and hence the vehicle is not constrained to visit the location only once. The objective is to minimize the total distance traveled by all vehicles. Below is the mathematical model along with the notation used.

Table 2. List of symbols used in the mathematical model representation

| Symbol | Representation |
|---|---|
| $G = (N, E)$ | Complete weighted digraph with N nodes and E edges |
| $N^+ = \{1^+, 2^+, 3^+, \dots, n^+\}$ | Set of pickup nodes |
| $N^- = \{1^-, 2^-, 3^-, \dots, n^-\}$ | Set of delivery nodes |
| $N = N^- \cup N^+$ $N = \{0, 1, 2, 3, \dots, 2n\}$ | Set of all nodes with node 0 representing a central depot, n is even |
| $E = \{(i, j) \mid i, j \in N\}$ | Set of edges |
| $V = \{k\}, k = 1, \dots, m$ | Set of vehicles |
| τ | Time at which a new request is available (realization time) |
| $N_c(\tau)$ | Set of critical nodes |
| $N_u(\tau)$ | Set of un-serviced nodes |
| $N_{uc}(\tau)$ | Set of un-serviced or critical nodes |
| r_i | A request for pickup at node i^+ and delivery at node i^- |
| $[e_i, l_i]$ | Time window for request i |
| q_i | Demand at node i |
| t_{ijk} | Travel time from node i to node j using vehicle k |
| c_{ijk} | travel cost\distance from node i to node j using vehicle k |
| s_i | Service time at node i |
| a_i | Arrival time at node i |
| d_i | Departure time from node i |
| $w1_i$ | waiting time before beginning the service at node i (occurs when the vehicle arrives before the time window begins) |
| $w2_i$ | waiting time before the vehicle leaves node i after finishing the service |

Objective function

$$\min \sum_{k=1}^{|V|} \sum_{i=1}^n \sum_{j=1}^n c_{ijk} x_{ijk}$$

Such that:

Flow conversation constraints

$$\sum_{k=1}^{|V|} \sum_{j=1}^n x_{ijk} = 1 \quad \forall i \in N_{c/u}(\tau) \quad \dots (9)$$

$$\sum_{k=1}^{|V|} \sum_{i=1}^n x_{ijk} = 1 \quad \forall j \in N_u(\tau) \quad \dots (10)$$

$$\sum_{i=1}^n x_{ihk} = \sum_{j=1}^n x_{hjk} \quad \forall h \in N_u(\tau), k \in V \quad \dots (11)$$

$$\sum_{j=1}^n x_{0jk} \leq 1 \quad \forall k \in V_0 \quad \dots (12)$$

Time window constraints

$$a_i \leq l_i \quad \forall N_u(\tau) \quad \dots (13)$$

$$a_{0k} \leq l_0 \quad \forall k \in V \quad \dots (14)$$

$$d_i \geq a_i + w1_i + s_i \quad \forall N_{u/c}(\tau) \quad \dots (15)$$

$$a_j = d_i + t_{ij} \text{ if } x_{ijk} = 1 \quad \forall i \in N_{c/o}(\tau), j \in N_u(\tau), k \in V \quad \dots (16)$$

$$w1_i = \max\{0, e_i - a_i\} \quad \forall i \in N_{c/u}(\tau) \quad \dots (17)$$

$$w2_i = d_i - (a_i + w1_i + s_i) \quad \forall i \in N_{c/u}(\tau) \quad \dots (18)$$

$$x_{ijk} = \begin{cases} 0, & \text{if vehicle } k \text{ does not drive from vertex } i \text{ to vertex } j \\ 1, & \text{if vehicle } k \text{ drives from vertex } i \text{ to vertex } j \end{cases} \quad \dots (19)$$

Equation (9) requires that only one vehicle leaves a critical, or un-serviced, node i once. Equation (10) requires that only one vehicle arrives at un-serviced node j once. Equation (11) requires that for each un-serviced node h , the entering vehicle must

eventually leave that node. Equation (12) requires that each vehicle can leave the depot, at most, once. Equation (13) requires that the arrival time, at each node, should be before the end of its time window. Equation (14) requires that all vehicles must return to the depot before it closes. Equation (15) requires that departure time d_i must be later than or equal to the completion time of the service (i.e. $a_i + w1_i + s_i$). Equation (16) requires that the arrival time at a destination node must equal the departure time from the origin node plus the travel time between the two nodes. Equations (17) and (18) define the waiting times before the service (if the vehicle arrives before the time window begins) and after the service (if the vehicle is to wait after the service for new requests to appear). Finally, equation (19) states that x_{ijk} is a binary variable indicating whether (i, j) is used in the solution or not. Of course, the Precedence and coupling constraints (i.e. the pickup must be performed before the delivery, and both must be done by the same vehicle) apply throughout the model.

Methodology

3.1 Solution approach

As mentioned earlier, the area of application is the courier mail pickup and delivery problem, in which small parcels are picked up from one location and delivered to another location on the same working day. The solution approach is as follows: an initial routing plan, for each vehicle, is constructed based on the static data available (i.e. based on the requests known a priori to execution). This routing plan will not be optimal as it is generated using the Nearest Addition heuristic, the reason for not generating an optimal initial routing plan, is because the problem is dynamic. The routing plan is then improved using a hybrid algorithm based on Tabu Search, Guided Local Search, and Variable Neighborhood search.

The initial improved solution is executed, and as new requests unfold throughout the day, they are inserted into some vehicle's routing plan according to the Cheapest Insertion heuristic, *if possible*. Sometime, the time windows on dynamic requests are too tight to satisfy; therefore, such requests may be rejected. The cheapest insertion heuristic is followed by a quick local search based on the Reduced Variable Neighborhood Search (RVNS), for *each* newly arriving request; the idea is to evenly distribute these newly added visits among vehicles. What happens is that when the cheapest insertion heuristic is used, new visits arriving towards the end of the working day are most likely added to un-used vehicles, as it requires much less time to simply use such idle vehicles, than to find a feasible insertion point within the current routes of the used vehicles, and therefore more vehicles will be used, but will also be under utilized.

After this quick local search is done, the dispatcher can inform the customer whether his / her request can be accommodated for on the same day or not; however, requests are not dispatched to the vehicles on the road till after the re-optimization hybrid is run. The re-optimization hybrid algorithm is run after 10 new visits are inserted, and according to its output, new requests are dispatched to the vehicles. Figure 1 illustrates the procedure.

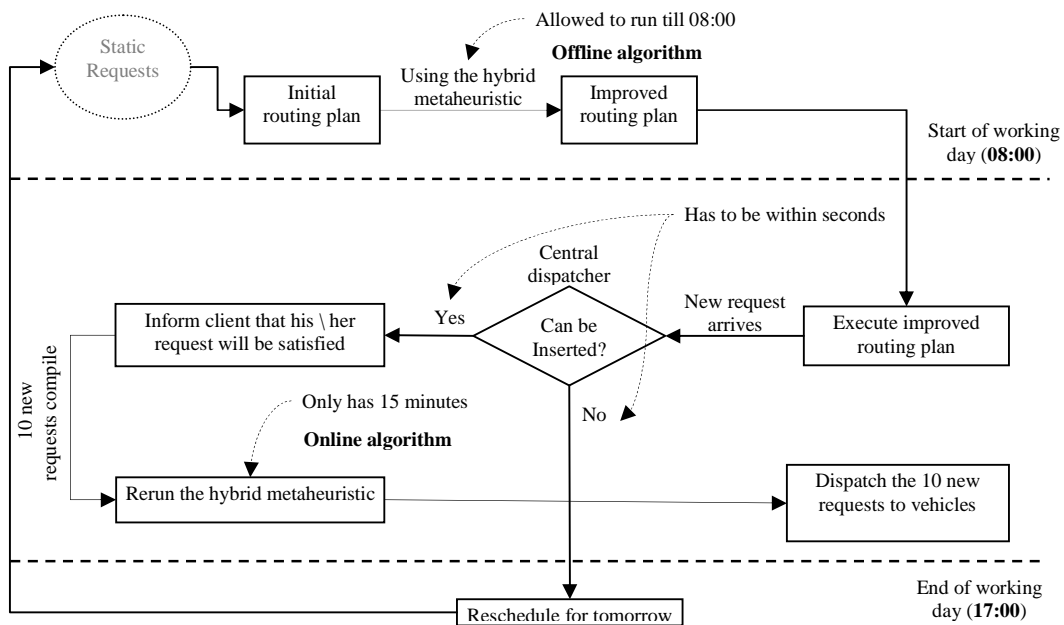


Figure 1. Solution approach

It should be noted, however, that the re-optimization algorithm is only based on ideas from these three metaheuristics; they are not applied in their usual context. The hybrid is then tested against problem sets, originally created to investigate the effectiveness of population based metaheuristics, not local search metaheuristics; as the ones used here. A brief on the heuristic and metaheuristic approaches used is explained, and then the hybrid approach is presented.

3.1.1 Nearest addition heuristic

The nearest addition heuristic builds routes by adding visits closer to the end of the route. The algorithm goes as follows, for all vehicles:

1. Denote the vehicle to be considered by w .
2. Start with a partial route consisting of the departure from the depot.
3. Find a visit v closest (cheapest to get to) to the end of the current partial route of w . If it is not possible to find such a visit, close the current partial route w , choose another empty vehicle and go to step 2.
4. Add v to the end of the partial route.
5. Go to step 3.
6. If all vehicles have been used and there are still visits unperformed, new vehicles must be brought in; otherwise, the algorithm fails.

3.1.2 Cheapest insertion heuristic

Let $c(i, k, j)$ be the cost of inserting node k between nodes i and j that are already part of the route. The cheapest insertion heuristic selects the next node to be the one minimizing $c(i, k, j)$ and is not a part of the current route. The procedure is repeated until all nodes have been inserted. This is relatively a fast heuristic and is practical in a dynamic setting; when a request arrives the customer expects an immediate answer whether his/her request can be satisfied on the same day or not; so, the time available for the algorithm to find a feasible insertion point for new requests is seconds, not minutes.

3.2 Overview of the metaheuristics used

As mentioned before in section 2.6, metaheuristics were created to escape local optima by filtering out proposed moves according to the metaheuristic rule (e.g. a greedy search metaheuristic filters out all moves which do not improve the solution). To do so, metaheuristics were classified into two major categories: those that start with and maintain only one single solution at each iteration, and those that start with multiple solutions (2 solutions or more), the latter is known as population based metaheuristics. It is difficult to say which approach is better, faster, or even has a higher chance of reaching an optimal solution; it actually depends on the problem being solved. All metaheuristics used in this work are of local search nature; these metaheuristics are described next.

3.2.1 Tabu Search

In Tabu Search (TS), degrading moves are accepted during the search to avoid moving to places previously visited; this can be done by, of course, storing every solution visited and forbidding the return to any such point. However, this has a significant memory burden; therefore, TS uses a technique called *Tabu List* (Glover 1986, 1989). The tabu list is a list of "features" of a solution that are forbidden, or alternatively, that must be present. Features are added and dropped from the list when a neighborhood move is made. In routing problems, the features are the arcs of the current solution, and two lists are maintained: one that dictates which arcs must not be part of any new solution (forbid / tabu list), and the other dictates which arcs must be part of any new solution (keep list). Maintaining finite lists encourages exploring solutions with different arcs. The length of time a feature remains on a list (*The Tenure*) is important and affects how the search avoids previous solutions, the tenure is a

parameter of the algorithm, and can be altered dynamically during the search process; which will be done in this work as part of the search algorithm.

Whenever a move is examined, TS looks at the new arcs added to the solution and at the old arcs that left the solution, then the number of new arcs appearing on the forbid list and the number of old arcs that appear on the keep list are added, the summation is called the tabu number of the move. If the tabu number is above a certain value (which varies according to the move type), the move is declared tabu and rejected. This rejection can be overridden in one case: when the cost of this move is better than the best cost solution visited so far. This is known as the aspiration criterion, it prevents the search from being overly inhibited by the tabu list (Glover, 1990 and Gendreau, 2003).

3.2.2 Guided Local Search

Guided Local Search (GLS) can be seen as an alternative to TS in escaping local optima. As in TS, how the search can move around is restricted; GLS makes a series of greedy searches, each to a local minimum, but it optimizes a different cost function from the original. An augmented cost function is created by adding a penalty term to the true cost function, the penalty term is the sum of all penalties for possible “features” of a problem (in this case the features are the arcs of the routing problem), the penalty for each possible arc starts at zero and will only be increased when the local search reaches a local optimum (Voudouris, 2003). Given an objective function g that maps every candidate solution s , GLS defines the augmented function h as:

$$h(s) = g(s) + \lambda \sum_i (p_i I_i(s)) \quad \dots (1)$$

Where λ is a parameter to the GLS algorithm, p_i is the penalty for feature (all p_i values are initialized to 0) and I_i is an indication of whether s exhibits feature i or not, that is:

$$I_i = \begin{cases} 1, & \text{if } s \text{ contains feature } i \\ 0, & \text{otherwise} \end{cases} \quad \dots (2)$$

GLS determines which arcs to penalize based upon the cost of the arc in the solution and how often that arc has previously appeared at local optima. GLS tries to choose a bad or costly arc in the solution to penalize, as removing costly arcs should lead to finding better solutions in subsequent iterations. GLS penalizes an arc for which the *utility* is the highest of all arcs in the current solution, the utility of an arc under a local minimum s_* is:

$$util_i(s_*) = I_i(s_*) \times \frac{c_i}{1 + n_i} \quad \dots (3)$$

c_i is the cost of arc i , n_i is the number of times arc i has been penalized, thus GLS tries to penalize arcs with high cost. However, if an arc has been penalized a number of times, the importance of cost reduces, this is due to the fact that, if an arc has been penalized a large number of times and is still in the solution, there may be no better arc(s) with which to replace it and it is probably best to start looking elsewhere to place penalties. The penalty is equal to the cost of the arc multiplied by the penalty factor (specified to the metaheuristic).

3.2.3 Variable Neighborhood Search

The basic idea of Variable Neighborhood Search (VNS) is the systematic change of the neighborhood structure explored. The key principles of VNS are as follows (Mladenovic and Hansen, 1997, 1999, 2001c, 2005):

- A local minimum with respect to one neighborhood is not necessary so for another.

- A global minimum is a local minimum with respect to all possible neighborhoods.
- For many problems local minima are relatively close to each other. This principle is an empirical one; local optima can provide some insight about the global optima.

VNS is built of two main components: Variable Neighborhood Descent (VND) and Reduced VNS (RVNS). VND is simply a greedy search, in which only improving moves are taken; this will, most likely, produce a local optimum solution which, according to the first principle, differs according to the neighborhood structure used. RVNS is concerned with how to escape a local optima (i.e. which point to move to once a local optima is reached), the easiest answer would be to move to a random point, and, according to the third principle, it would be advisable to move to a point close to the local optima first, then explore further points if required. The original VNS algorithm is as follows (Mladenovic and Hansen, 1997, 1999, 2001c, 2003):

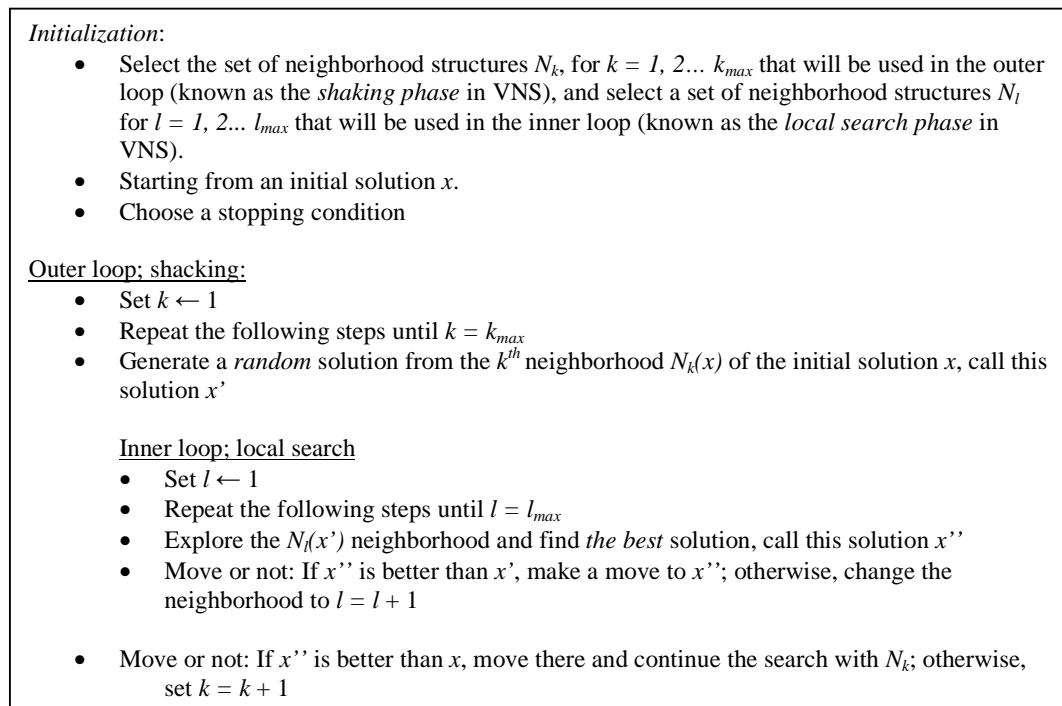


Figure 2. Original VNS algorithm

Clearly, the original VNS algorithm uses the randomly generated solution x' to diversify the search, and uses a greedy search algorithm (best accept) to intensify the search (in the inner loop); in this study, the greedy local search approach is replaced with a metaheuristic combining both TS and GLS filtration criterion, this will allow the inner loop to explore each of the N_l neighborhoods more thoroughly, and reach better local optima. The proposed hybrid is as follows:

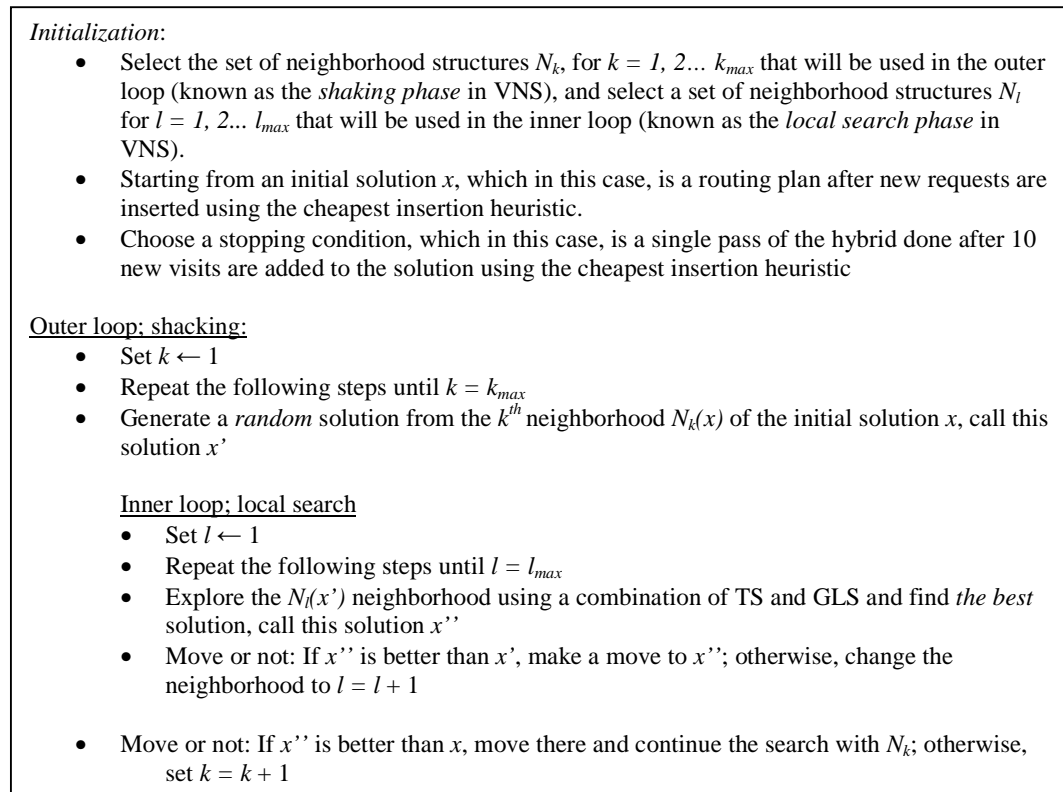


Figure 3. Proposed hybrid metaheuristic

3.3 Experimental procedure

As stated in the introduction, the objectives of this study are:

1. Creating a hybrid metaheuristic to solve the dynamic pickup and delivery problem with time windows; the hybrid is based on Tabu Search, Guided Local Search, and Variable Neighborhood Search. The hybrid will be developed using the ILOG CP classes under a C++ development environment.
2. Investigating the effect of changing the neighborhood order on solution quality and solution speed (in the Variable Neighborhood Search context).
3. Investigating the effect of dynamically changing search parameters on solution quality and solution speed (in the Tabu Search and Guided Local Search context).

The first objective is achieved through the algorithm proposed in figure 3 above. The second objective is achieved through classifying the neighborhoods used in the VNS framework into two groups: those that modify only one route which are known as intra-route neighborhoods (the ones used in this study are: Intra Relocate, Two Opt, and Or Opt), and those that make changes between routes which are known as inter-route neighborhoods (the ones used in this study are: Merge and Relocate Tours, Cross, FP Relocate, Exchange, and Relocate), exact definitions of these neighborhood structures are in the appendix. Table 1 below shows how the order of these neighborhoods is changed on each trial.

Table 1. Neighborhood order changes

| Outer loop; shaking | Inner loop; local search |
|--------------------------|--------------------------|
| Intra Relocate | Merge And Relocate Tours |
| Two-Opt | Cross |
| Or-Opt | FP Relocate |
| | Exchange |
| | Relocate |
| Merge And Relocate Tours | Intra-Relocate |
| Cross | Two-Opt |
| FP-Relocate | Or-Opt |
| Exchange | |
| Relocate | |
| Relocate | Merge And Relocate Tours |
| Or-Opt | Intra-Relocate |
| Exchange | Cross |
| | Two-Opt |
| | FP-Relocate |
| Merge And Relocate Tours | Relocate |
| Intra-Relocate | Or-Opt |
| Cross | Exchange |
| Two-Opt | |
| FP-Relocate | |

The third objective is achieved through dynamically changing the TS and GLS parameters during the local search phase, the changes are as follows:

- The total number of iterations for the local search phase is 150 *per neighborhood*.
- Before the optimization loop is entered set the tenure value (for TS) to 5 and the penalty value (for GLS) to 0.45; meaning when the optimization loop begins the search, towards a local minimum, is intensified.
- When the search reaches iteration number 70 and if there is no improvement on the objective function for the past 10 moves, the search is diversified by increasing the search parameters to 12 for the tenure and 0.8 for the penalty.
- Just before the optimization loop ends (at iteration 120), the search parameters are set back to lower values; tenure = 5 and penalty = 0.45.

The problem will be solved once using these parameter changes, and once without making these changes (i.e. the parameters will be set to: tenure = 5 and penalty = 0.45 at the beginning of the search, and will remain constant till the optimization loop terminates). Table 2 below shows how objectives 2 and 3 are reached.

Table 2. Changing search parameters and the neighborhood order

| Set | Trial | Parameter values | Outer loop; shaking | Inner loop; local search |
|-----|-------|---------------------------|--------------------------|--------------------------|
| X | 1 | tenure = 5 penalty = 0.45 | Intra Relocate | Merge And Relocate Tours |
| X | | tenure = 5 penalty = 0.45 | Two-Opt | Cross |
| X | | tenure = 5 penalty = 0.45 | Or-Opt | FP Relocate |
| X | | tenure = 5 penalty = 0.45 | | Exchange |
| X | | tenure = 5 penalty = 0.45 | | Relocate |
| X | 2 | tenure = 5 penalty = 0.45 | Merge And Relocate Tours | Intra-Relocate |
| X | | tenure = 5 penalty = 0.45 | Cross | Two-Opt |
| X | | tenure = 5 penalty = 0.45 | FP-Relocate | Or-Opt |
| X | | tenure = 5 penalty = 0.45 | Exchange | |
| X | | tenure = 5 penalty = 0.45 | Relocate | |
| X | 3 | tenure = 5 penalty = 0.45 | Relocate | Merge And Relocate Tours |
| X | | tenure = 5 penalty = 0.45 | Or-Opt | Intra-Relocate |
| X | | tenure = 5 penalty = 0.45 | Exchange | Cross |
| X | | tenure = 5 penalty = 0.45 | | Two-Opt |
| X | | tenure = 5 penalty = 0.45 | | FP-Relocate |
| X | 4 | tenure = 5 penalty = 0.45 | Merge And Relocate Tours | Relocate |
| X | | tenure = 5 penalty = 0.45 | Intra-Relocate | Or-Opt |
| X | | tenure = 5 penalty = 0.45 | Cross | Exchange |
| X | | tenure = 5 penalty = 0.45 | Two-Opt | |
| X | | tenure = 5 penalty = 0.45 | FP-Relocate | |
| X | 5 | Dynamically changing | Intra Relocate | Merge And Relocate Tours |
| X | | Dynamically changing | Two-Opt | Cross |
| X | | Dynamically changing | Or-Opt | FP Relocate |
| X | | Dynamically changing | | Exchange |
| X | | Dynamically changing | | Relocate |
| X | 6 | Dynamically changing | Merge And Relocate Tours | Intra-Relocate |
| X | | Dynamically changing | Cross | Two-Opt |
| X | | Dynamically changing | FP-Relocate | Or-Opt |
| X | | Dynamically changing | Exchange | |
| X | | Dynamically changing | Relocate | |
| X | 7 | Dynamically changing | Relocate | Merge And Relocate Tours |
| X | | Dynamically changing | Or-Opt | Intra-Relocate |
| X | | Dynamically changing | Exchange | Cross |
| X | | Dynamically changing | | Two-Opt |
| X | | Dynamically changing | | FP-Relocate |
| X | 8 | Dynamically changing | Merge And Relocate Tours | Relocate |
| X | | Dynamically changing | Intra-Relocate | Or-Opt |
| X | | Dynamically changing | Cross | Exchange |
| X | | Dynamically changing | Two-Opt | |
| X | | Dynamically changing | FP-Relocate | |

To see which of the eight configurations produce the lowest traveling cost, problem sets, based on the work of Christofides (Christofides and Beasley, 1984), 13 data sets; Fisher (Fisher, Jakumar and Wassenhove, 1981), 5 data sets; and Taillard (Taillard, 1994), 12 data sets, will be used. Meaning, a total of 30 data sets will be tested, each set is run eight times for different configurations of parameter values and neighborhood order. It is worth mentioning though that these data sets were originally generated for the dynamic vehicle routing problem with *no* time windows; therefore, the sets had to be modified to fit the scope of this study (i.e. time windows were added to all visits, and pickup and delivery pairs were created). This, in turn, will make it difficult to compare the results with the original data sets, but these data sets were the closest to the problem under study, and therefore were used. All tests will be run on a standard Pentium 4 PC with a CPU clock speed of 2.8 GHz, 1 GB of RAM, and Hyper Threading technology.

After obtaining the results, a paired t-test will be used to investigate the effect of dynamically changing search parameters versus setting the search parameters to fixed values and maintaining them throughout the search; and since there are four neighborhood orders, the paired t-test will be conducted four times, once under each neighborhood order. Tables 3 and 4 below illustrate the procedure. Alpha is set to 0.05.

Table 3. Results table template

| | Search parameters constant | | | | Search parameters dynamically changing | | | |
|-------|----------------------------|-------------------|-------------------|-------------------|--|-------------------|-------------------|-------------------|
| | Order 1 | Order 2 | Order 3 | Order 4 | Order 1 | Order 2 | Order 3 | Order 4 |
| | Trial 1 | Trail 2 | Trail 3 | Trial 4 | Trail 5 | Trial 6 | Trial 7 | Trial 8 |
| Set # | Obj. fun. | Obj. fun. | Obj. fun. | Obj. fun. | Obj. fun. | Obj. fun. | Obj. fun. | Obj. fun. |
| 1 | X ₁₁₁ | X ₁₂₁ | X ₁₃₁ | X ₁₄₁ | X ₁₁₂ | X ₁₂₂ | X ₁₃₂ | X ₁₄₂ |
| 2 | X ₂₁₁ | X ₂₂₁ | X ₂₃₁ | X ₂₄₁ | X ₂₁₂ | X ₂₂₂ | X ₂₃₂ | X ₂₄₂ |
| . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . |
| 30 | X ₃₀₁₁ | X ₃₀₂₁ | X ₃₀₃₁ | X ₃₀₄₁ | X ₃₀₁₂ | X ₃₀₂₂ | X ₃₀₃₂ | X ₃₀₄₂ |

Table 4. Paired t-test template

| Set # | Order 1 difference | Order 2 difference | Order 3 difference | Order 4 difference |
|---|-----------------------|-----------------------|-----------------------|-----------------------|
| 1 | $X_{111} - X_{112}$ | $X_{121} - X_{122}$ | $X_{131} - X_{132}$ | $X_{141} - X_{142}$ |
| 2 | $X_{211} - X_{212}$ | $X_{221} - X_{222}$ | $X_{231} - X_{232}$ | $X_{241} - X_{242}$ |
| . | . | . | . | . |
| . | . | . | . | . |
| . | . | . | . | . |
| 30 | $X_{3011} - X_{3012}$ | $X_{3021} - X_{3022}$ | $X_{3031} - X_{3032}$ | $X_{3041} - X_{3042}$ |
| Average difference (\bar{D}) | \bar{D}_1 | \bar{D}_2 | \bar{D}_3 | \bar{D}_4 |
| Difference standard deviation (S_D) | S_{D1} | S_{D2} | S_{D3} | S_{D4} |
| Test statistic (T_0) | T_1 | T_2 | T_3 | T_4 |
| $t_{\alpha/2, n-1}$ | t_1 | t_2 | t_3 | t_4 |
| p-value | p ₁ | p ₂ | p ₃ | p ₄ |
| 95% CI for mean difference | CI ₁ | CI ₂ | CI ₃ | CI ₄ |

RESULTS, ANALYSIS, AND DISCUSSION

4.1 Validation and verification

As mentioned in section 1.1 the solution to a vehicle routing problem with time windows is a routing plan (a sequence) for each vehicle specifying the locations to visit, the order of the visits, and the arrival and departure times for each visit (scheduling visits); the solution must satisfy all problem constraints. A sample solution is presented next and verified against problem constraints; problem set number 20, trial 6 will be used for this illustration. As shown in table 1 below, the objective function value, for problem set 20, is 2831.4; which is the total distance traveled by all vehicles. It is assumed that 1 unit of distance is equal to 1 unit of cost, and the vehicles have no fixed cost (e.g. a rent cost). Summing the costs of all vehicles gives the total cost of the objective function.

The route of each vehicle, starting and ending at the depot, along with the arrival and departure times to and from each node, are also presented. All arrivals are within the specified time windows of the set. For example, vehicle1 reaches visit33 at time 46.9142, which is within the specified time window for that visit (0 ~ 231), waits 0 time units before the time window opens, does the pickup service in 10 time units, and then travels for 14.8661 time units to reach its next destination (visit34), visit34 is the delivery pair of visit33. Visit34 is reached at 71.7802, which is also within its time window (0 ~ 232). Hence, time window constraints are satisfied. In addition, both the pickup and delivery (visit33 and visit34) are made by the same vehicle and the pickup is made before the delivery. The same reasoning applies to all other visits and all other problem sets. Table 1 and figures 1 – 3 characterize problem set 20, validation details for vehicle 1 are in tables 2 and 3, and the remaining problem instances are in appendix 3.

Table 1. Problem characteristics for set number 20, trial 6

| Set # | Static | Dynamic | edod _{tw} | Parameter values | Shaking Neighborhood | Local search Neighborhood | Initial Solution | Final Solution |
|-------|--------|----------|--------------------|------------------|----------------------|---------------------------|------------------|----------------|
| 20 | 1 ~ 82 | 83 ~ 220 | 0.412 | Dynamic | Merge and Relocate | Intra Relocate | 884.273 | 2831.400 |
| 20 | 1 ~ 82 | 83 ~ 220 | 0.412 | Dynamic | Cross | Two-Opt | | |
| 20 | 1 ~ 82 | 83 ~ 220 | 0.412 | Dynamic | FP-Relocate | Or-Opt | | |
| 20 | 1 ~ 82 | 83 ~ 220 | 0.412 | Dynamic | Exchange | | | |
| 20 | 1 ~ 82 | 83 ~ 220 | 0.412 | Dynamic | Relocate | | | |

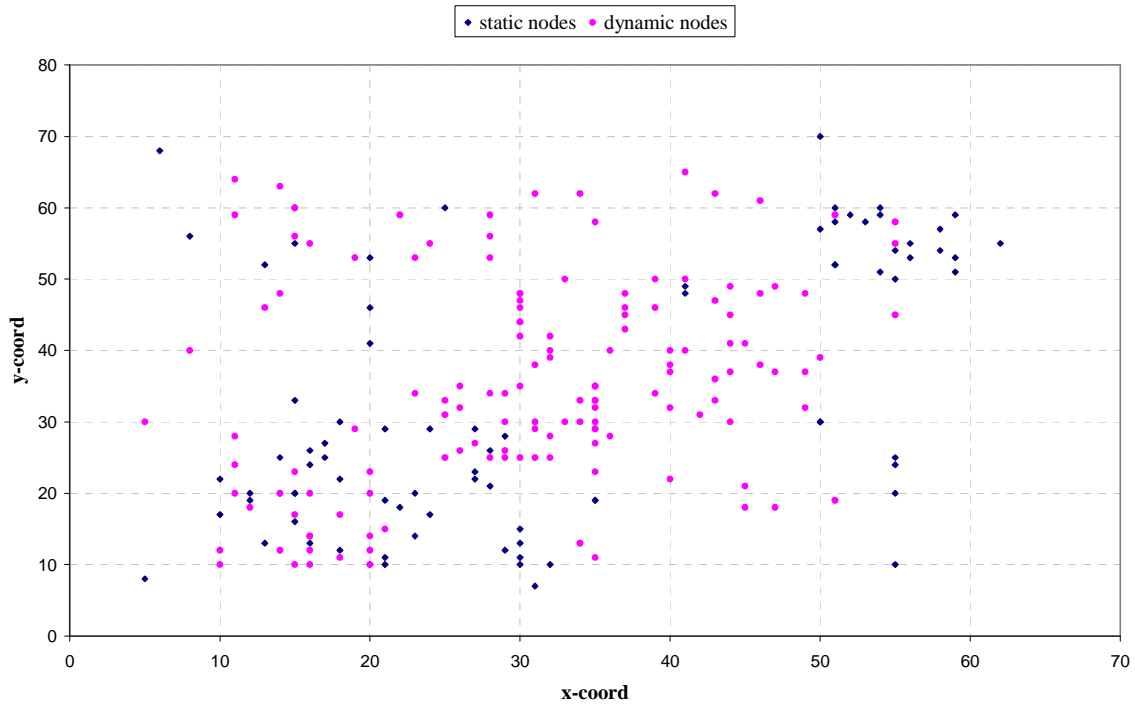


Figure 1. Locations distribution

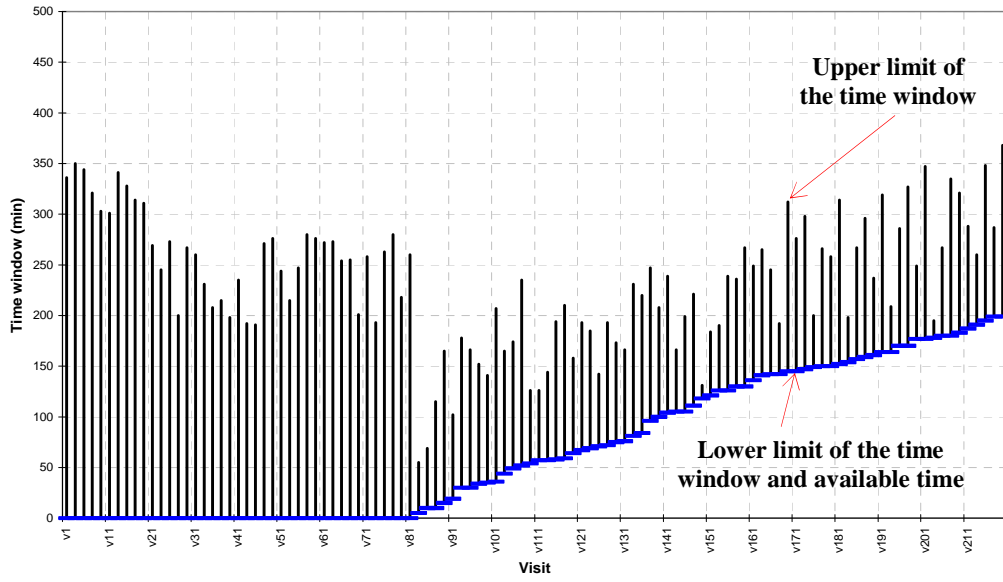


Figure 2. Pickup time windows – problem set 20

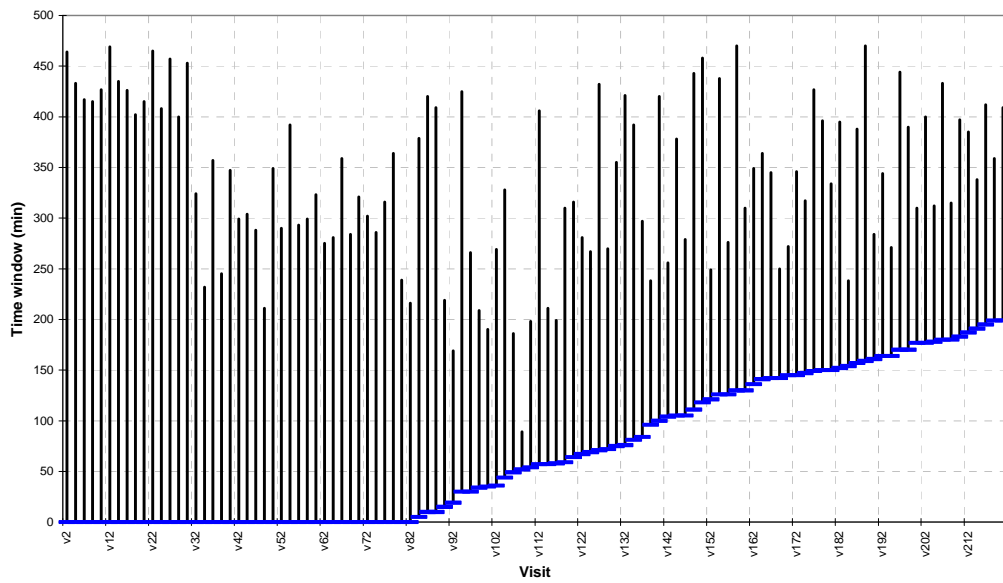


Figure 3. Delivery time windows – problem set 20

Table 2. Time windows for visits performed by vehicle 1

| Pickup | Delivery | Pickup Min Time | Delivery Min Time | Pickup Max Time | Delivery Max Time | Service Time | Drop Time | Available Time |
|---------|----------|-----------------|-------------------|-----------------|-------------------|--------------|-----------|----------------|
| visit33 | visit34 | 0 | 0 | 231 | 232 | 10 | 10 | 0 |
| visit55 | visit56 | 0 | 0 | 247 | 293 | 10 | 10 | 0 |
| visit57 | visit58 | 0 | 0 | 280 | 299 | 10 | 10 | 0 |
| visit65 | visit66 | 0 | 0 | 254 | 359 | 10 | 10 | 0 |

Table 3. Validation of the model based on problem set 20 trial 6 – vehicle 1

| |
|--|
| Vehicle1 Total cost = 148.569 Fixed cost = 0 Cost coefficients = Distance [1] |
| Route: depot → visit57 → visit33 → visit34 → visit65 → visit55 → visit58 → visit66 → visit56 → depot |
| Time: depot [0], delay [0] → travel [23.0217], wait [0] → visit57 [23.0217], delay [10] → travel [13.8924], wait [0] → visit33 [46.9142], delay [10] → travel [14.8661], wait [0] → visit34 [71.7802], delay [10] → travel [41.7732], wait [0] → visit65 [123.553], delay [10] → travel [4.47214], wait [0] → visit55 [138.026], delay [10] → travel [1], wait [0] → visit58 [149.026], delay [10] → travel [11.1803], wait [0] → visit66 [170.206], delay [10] → travel [1], wait [0] → visit56 [181.206], delay [10] → travel [37.3631], wait [0] → depot [228.569] |
| <i>Transit Sum [228.569]</i> |
| Distance: depot [0] delay [0] → travel [23.0217], wait [0] → visit57 [23.0217], delay [0] → travel [13.8924], wait [0] → visit33 [36.9142], delay [0] → travel [14.8661], wait [0] → visit34 [51.7802], delay [0] → travel [41.7732], wait [0] → visit65 [93.5534], delay [0] → travel [4.47214], wait [0] → visit55 [98.0256], delay [0] → travel [1], wait [0] → visit58 [99.0256], delay [0] → travel [11.1803], wait [0] → visit66 [110.206], delay [0] → travel [1], wait [0] → visit56 [111.206] delay [0] → travel [37.3631], wait [0] → depot [148.569] |
| <i>Transit Sum 148.569</i> |
| Vehicle2 Total cost = 192.05 Fixed cost = 0 Cost coefficients = Distance [1] |
| Vehicle3 Total cost = 130.477 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle4 Total cost = 88.3736 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle5 Total cost = 208.919 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle6 Total cost = 108.338 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle7 Total cost = 93.8076 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle8 Total cost = 167.074 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle9 Total cost = 202.768 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle10 Total cost = 21.6734 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle11 Total cost = 140.275 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle12 Total cost = 122.024 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle13 Total cost = 202.49 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle14 Total cost = 90.7877 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle15 Total cost = 72.7513 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle16 Total cost = 42.4945 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle17 Total cost = 48.4243 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle18 Total cost = 127.933 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle19 Total cost = 138.792 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle20 Total cost = 102.149 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle21 Total cost = 111.046 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle22 Total cost = 117.463 Fixed cost = 0 Cost coefficients: Distance [1] |
| Vehicle23 Total cost = 152.724 Fixed cost = 0 Cost coefficients: Distance [1] |
| Total Cost = 2831.4 Number of vehicles used = 23 Number of visits performed = 220 |

4.2 Analysis and discussion

Next, the results are presented and analyzed based on objectives 2 and 3 of this study (i.e. investigating the effect of changing the neighborhood order on solution quality and solution speed, and investigating the effect of dynamically changing search parameters on solution quality and solution speed), this is done in sections 4.2.1 and 4.2.2.

In section 4.2.3, the online algorithm is assessed using competitive analysis (discussed in section 2.4). This is done as follows: for every problem set 8 trials were run (as a result of changing neighborhood orders and search parameters), one of these 8 trials produced the minimum objective function value for that set; the settings at which the minimum value was observed, per set, are used to solve the same problem set again but assuming that all requests are static, and known in advance at the time of route planning. Meaning, the static version will have less constraints as the dynamic requests, which are now assumed to be static, can be inserted into any part of the routing plan with no regards to their realization time (the realization time is now considered to equal zero). Of course, competitive analysis is usually run against a static version of the problem, solved with an offline algorithm capable of producing an *optimal* solution; however, in this case, the static version of the problem is *NP-hard*, there is not a known optimal solution to compare to; hence, the comparison will be made against the solution obtained with less constraints. Such analysis will show how the objective function could have been lower had all the information been available at the route planning phase. Finally, in section 4.2.4, some observations are made about how the degree of dynamism affects the percentage of rejected dynamic requests.

4.2.1 Analysis and discussion – objective function

Based on the experimental procedure (section 3.3), the results, for the objective function, for the 30 trials are in table 4 below.

Table 4. Results table - objective function

| Set # | Search parameters constant | | | | Search parameters dynamically changing | | | |
|-------|----------------------------|-----------|-----------|-----------|--|-----------|-----------|-----------|
| | Order 1 | Order 2 | Order 3 | Order 4 | Order 1 | Order 2 | Order 3 | Order 4 |
| | Trial 1 | Trail 2 | Trail 3 | Trial 4 | Trail 5 | Trial 6 | Trial 7 | Trial 8 |
| | Obj. fun. | Obj. fun. | Obj. fun. | Obj. fun. | Obj. fun. | Obj. fun. | Obj. fun. | Obj. fun. |
| 1 | 2122.20 | 1974.38 | 2063.45 | 2072.71 | 2072.71 | 1974.38 | 2028.52 | 2072.71 |
| 2 | 1250.91 | 1264.39 | 1264.39 | 1201.15 | 1209.44 | 1201.25 | 1317.49 | 1280.97 |
| 3 | 2212.00 | 2290.04 | 1929.91 | 2096.58 | 2226.78 | 2233.58 | 2243.45 | 1972.29 |
| 4 | 2143.21 | 2284.21 | 2055.71 | 2284.21 | 2130.44 | 2284.21 | 2017.60 | 2284.21 |
| 5 | 1718.17 | 1779.20 | 1588.04 | 1536.30 | 1867.43 | 1895.52 | 1722.29 | 1779.14 |
| 6 | 2722.28 | 2664.65 | 2640.65 | 2751.47 | 2722.28 | 2613.88 | 2753.97 | 2572.13 |
| 7 | 1539.09 | 1426.45 | 1542.54 | 1667.97 | 1533.62 | 1602.48 | 1433.39 | 1667.97 |
| 8 | 2715.61 | 2656.26 | 2492.65 | 2584.55 | 2758.65 | 2683.95 | 2655.76 | 2743.94 |
| 9 | 3187.74 | 3539.37 | 3419.05 | 3539.37 | 3373.77 | 3539.37 | 3446.02 | 3231.05 |
| 10 | 2609.37 | 2541.69 | 2738.94 | 2542.92 | 2445.93 | 2542.47 | 2721.51 | 2488.15 |
| 11 | 3361.17 | 3332.61 | 3298.90 | 3098.27 | 3311.17 | 3246.99 | 3134.84 | 3098.27 |
| 12 | 3247.96 | 3297.96 | 3165.24 | 3417.71 | 3233.49 | 3385.23 | 3157.77 | 3345.68 |
| 13 | 3501.75 | 3246.64 | 3370.01 | 3492.33 | 3202.48 | 3457.58 | 3443.55 | 3427.80 |
| 14 | 3613.18 | 3441.99 | 3245.69 | 3227.31 | 3295.72 | 3343.48 | 3153.77 | 3333.34 |
| 15 | 3939.70 | 3857.04 | 3740.19 | 3698.24 | 3774.59 | 3811.09 | 4049.45 | 3698.24 |
| 16 | 2568.28 | 2471.55 | 2552.15 | 2451.48 | 2347.08 | 2127.09 | 2272.98 | 2034.83 |
| 17 | 2457.93 | 2560.21 | 2365.72 | 2537.12 | 2600.02 | 2280.95 | 2217.76 | 2160.65 |
| 18 | 3049.40 | 2973.01 | 3207.35 | 2985.34 | 3181.62 | 3272.90 | 2970.62 | 2905.65 |
| 19 | 4026.65 | 3777.83 | 3921.51 | 3609.90 | 3812.42 | 3575.04 | 3824.77 | 3737.37 |
| 20 | 3060.29 | 2732.91 | 3348.74 | 3047.68 | 3022.75 | 2831.40 | 3098.70 | 2662.99 |
| 21 | 3128.74 | 3353.55 | 3450.76 | 3285.26 | 3386.31 | 3287.07 | 3428.97 | 3385.63 |
| 22 | 4523.01 | 4624.70 | 4154.66 | 4267.13 | 4548.13 | 4728.46 | 4568.10 | 4528.85 |
| 23 | 3537.66 | 3754.80 | 3788.04 | 3614.36 | 3604.43 | 3641.76 | 3459.53 | 3690.06 |
| 24 | 2854.61 | 2842.98 | 2896.05 | 2960.21 | 2939.76 | 2875.42 | 3060.34 | 2828.84 |
| 25 | 3167.00 | 3051.52 | 3033.59 | 3082.85 | 3084.74 | 2884.95 | 3105.17 | 3106.50 |
| 26 | 2134.21 | 2193.18 | 2193.18 | 2355.91 | 2739.79 | 2473.87 | 2226.52 | 2598.65 |
| 27 | 3136.89 | 3342.57 | 3052.60 | 3038.17 | 3105.16 | 2987.35 | 3006.17 | 2992.03 |
| 28 | 3925.67 | 4900.93 | 3932.88 | 4103.12 | 4167.33 | 4028.29 | 4253.63 | 4047.09 |
| 29 | 4176.09 | 4133.42 | 4183.21 | 4205.75 | 4236.64 | 4236.64 | 4262.76 | 4115.77 |
| 30 | 4763.65 | 4601.22 | 4963.99 | 4702.63 | 4914.05 | 4903.35 | 4844.16 | 4723.93 |

Before using the paired t-test, two issues should be checked; the probability distribution of the *differences* (which must be normal), and the correlation between the pairs (this is not a necessary condition, but if a high correlation exists, it justifies using the paired t-test to obtain accurate results even though $n-1$ degrees of freedom will be lost). Starting with

the normality test, a normal probability plot for each of the four differences is created, and the Anderson-Darling test is used with an Alpha of 0.01. See figures 4 – 7.

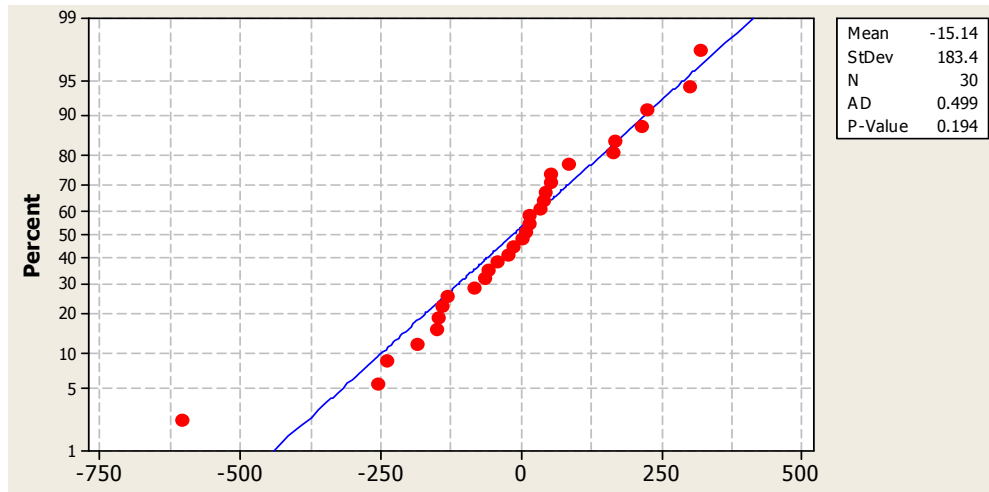


Figure 4. Normal probability plot of differences – order 1

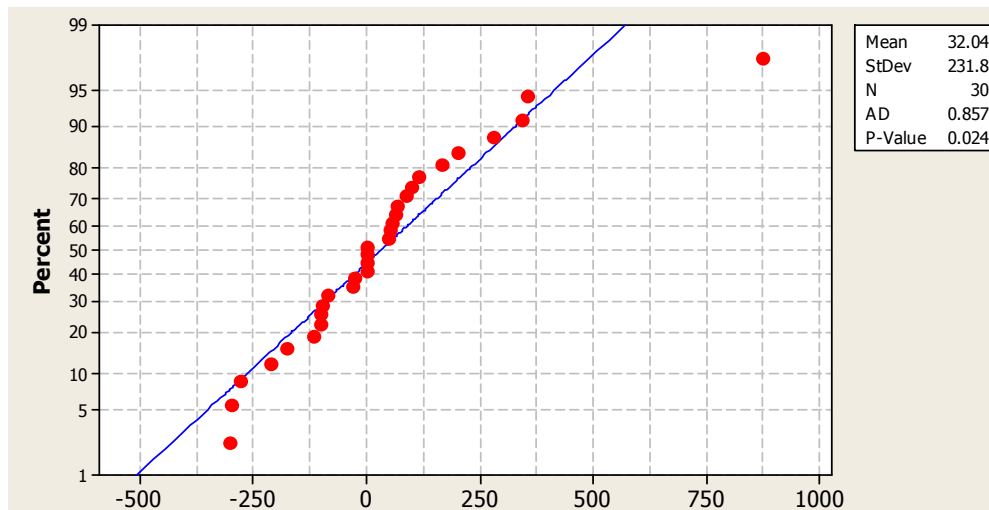


Figure 5. Normal probability plot of differences – order 2

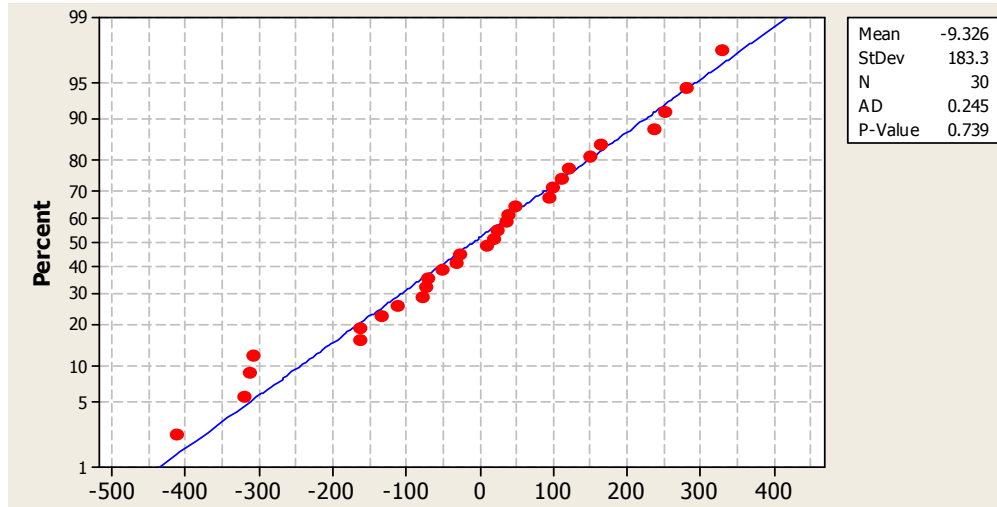


Figure 6. Normal probability plot of differences – order 3

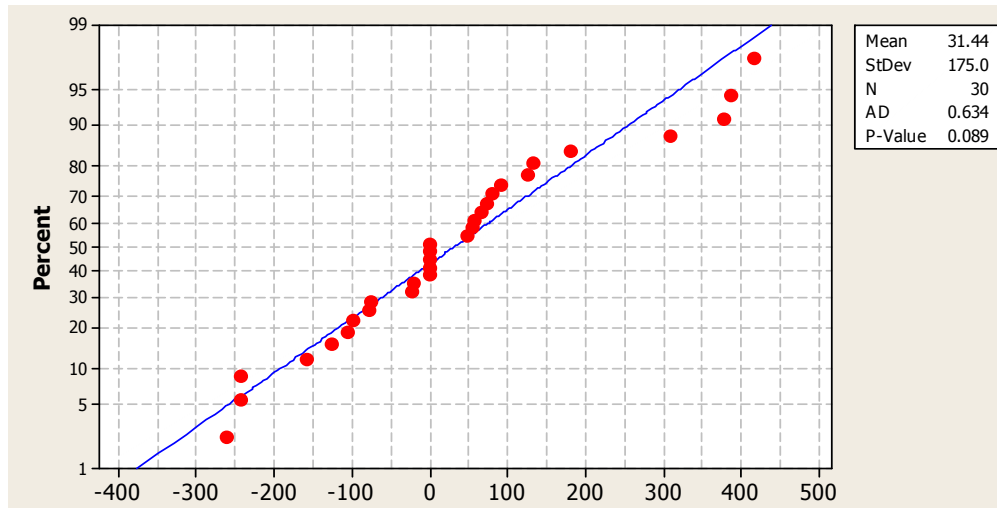


Figure 7. Normal probability plot of differences – order 4

Clearly some outliers exist, but most of the data seem to follow a normal distribution, and all the p-values for the Anderson-Darling test are greater than 0.01, which means there is not enough evidence to reject the null hypothesis of the data being normal. As for the correlation between pairs, the correlation coefficient for each of the order configuration is calculated (i.e. a total of four correlation coefficients).

Pearson correlation of Trial 1 and Trail 5 = 0.977, P-Value = 0.000

Pearson correlation of Trail 2 and Trial 6 = 0.967, P-Value = 0.000

Pearson correlation of Trail 3 and Trial 7 = 0.979, P-Value = 0.000

Pearson correlation of Trial 4 and Trial 8 = 0.978, P-Value = 0.000

The strong correlation necessitates the use of the paired t-test. Results for the tests are in table 5, and box plots are in figures 8 – 11.

Table 5. Paired t-test table, Alpha = 0.05

| Set # | Order 1 difference | Order 2 difference | Order 3 difference | Order 4 difference |
|---|--------------------|--------------------|--------------------|--------------------|
| 1 | 49.49 | 0.00 | 34.93 | 0.00 |
| 2 | 41.47 | 63.14 | -53.10 | -79.82 |
| 3 | -14.78 | 56.46 | -313.54 | 124.29 |
| 4 | 12.77 | 0.00 | 38.11 | 0.00 |
| 5 | -149.26 | -116.32 | -134.25 | -242.84 |
| 6 | 0.00 | 50.77 | -113.32 | 179.34 |
| 7 | 5.47 | -176.03 | 109.15 | 0.00 |
| 8 | -43.04 | -27.69 | -163.11 | -159.39 |
| 9 | -186.03 | 0.00 | -26.97 | 308.32 |
| 10 | 163.44 | -0.78 | 17.43 | 54.77 |
| 11 | 50.00 | 85.62 | 164.06 | 0.00 |
| 12 | 14.47 | -87.27 | 7.47 | 72.03 |
| 13 | 299.27 | -210.94 | -73.54 | 64.53 |
| 14 | 317.46 | 98.51 | 91.92 | -106.03 |
| 15 | 165.11 | 45.95 | -309.26 | 0.00 |
| 16 | 221.20 | 344.46 | 279.17 | 416.65 |
| 17 | -142.09 | 279.26 | 147.96 | 376.47 |
| 18 | -132.22 | -299.89 | 236.73 | 79.69 |
| 19 | 214.23 | 202.79 | 96.74 | -127.47 |
| 20 | 37.54 | -98.49 | 250.04 | 384.69 |
| 21 | -257.57 | 66.48 | 21.79 | -100.37 |
| 22 | -25.12 | -103.76 | -413.44 | -261.72 |
| 23 | -66.77 | 113.04 | 328.51 | -75.70 |
| 24 | -85.15 | -32.44 | -164.29 | 131.37 |
| 25 | 82.26 | 166.57 | -71.58 | -23.65 |
| 26 | -605.58 | -280.69 | -33.34 | -242.74 |
| 27 | 31.73 | 355.22 | 46.43 | 46.14 |
| 28 | -241.66 | 872.64 | -320.75 | 56.03 |
| 29 | -60.55 | -103.22 | -79.55 | 89.98 |
| 30 | -150.40 | -302.13 | 119.83 | -21.30 |
| Average difference (\bar{D}) | -15.1 | 32.0 | -9.3 | 31.4 |
| Difference standard deviation (S_D) | 183.4 | 231.8 | 183.3 | 175.0 |
| Test statistic (T_0) | -0.45 | 0.76 | -0.28 | 0.98 |
| $t_{\alpha/2, n-1}$ | 2.045 | 2.045 | 2.045 | 2.045 |
| p-value | 0.654 | 0.455 | 0.782 | 0.333 |
| 95% CI for mean difference | (-83, 53) | (-54, 118) | (-77, 59) | (-33, 96) |

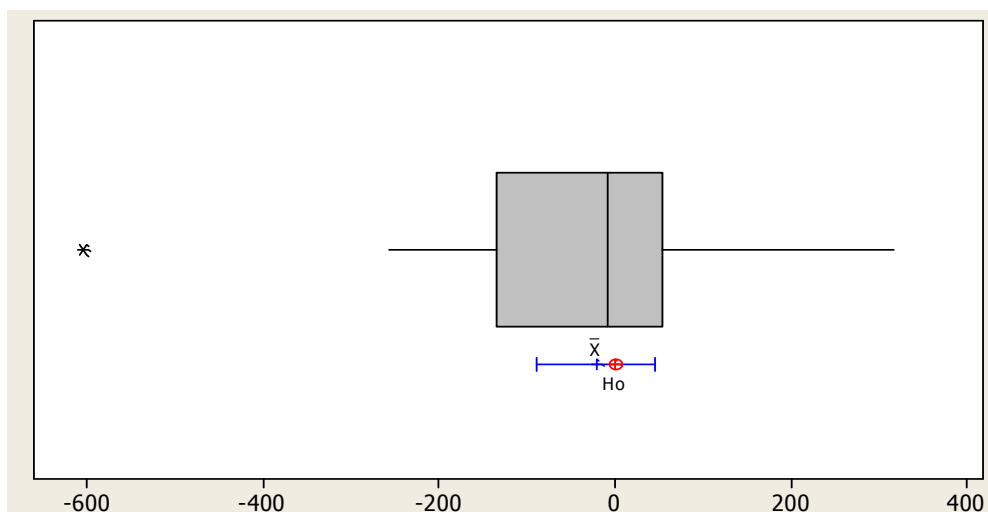


Figure 8. Box plot for neighborhood order 1. Alpha = 0.05

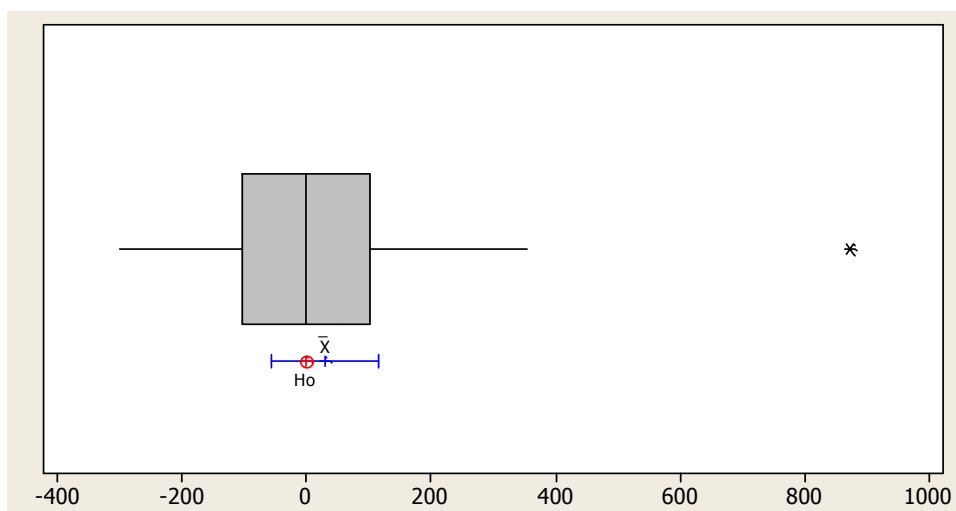


Figure 9. Box plot for neighborhood order 2. Alpha = 0.05

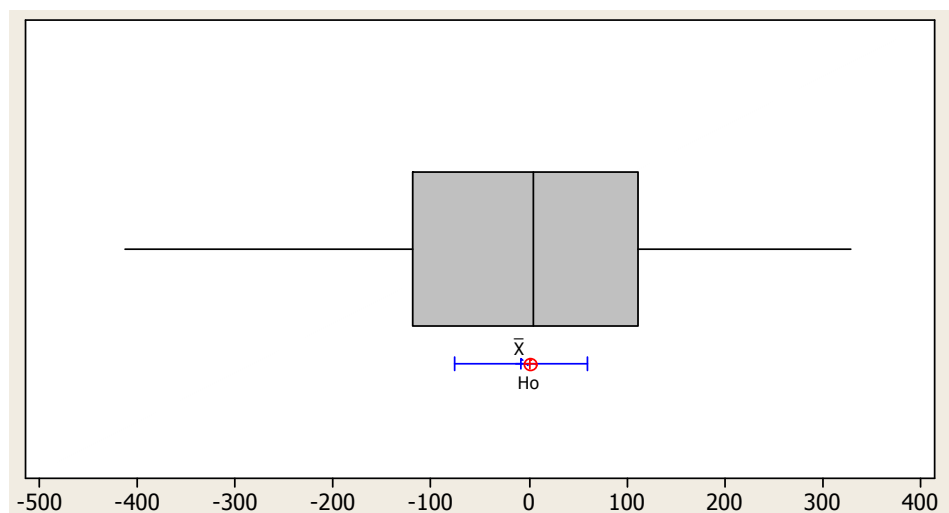


Figure 10. Box plot for neighborhood order 3. Alpha = 0.05

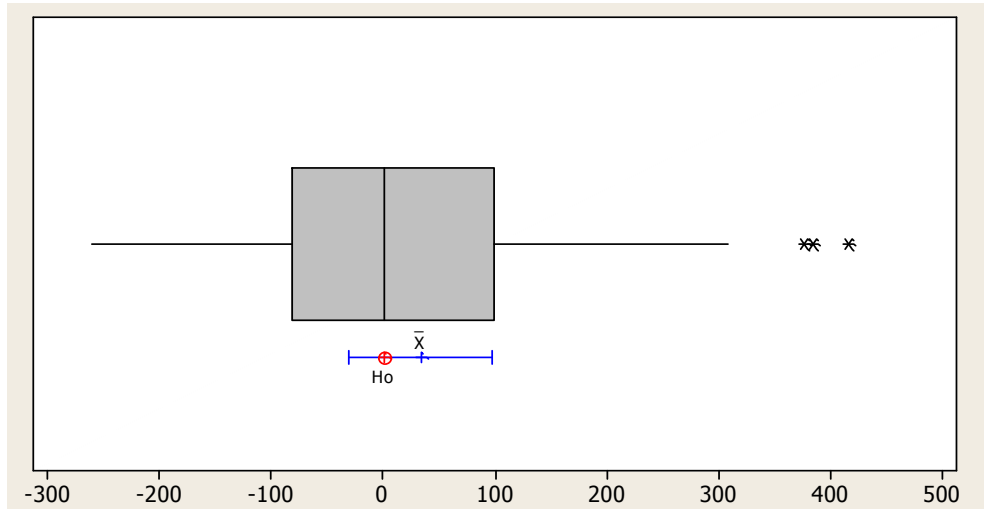


Figure 11. Box plot for neighborhood order 3. Alpha = 0.05

The p-values for all neighborhood orders suggest that there is not enough evidence to reject the null hypothesis of the difference being zero; hence, the results favor the conclusion that the objective function will not improve when the search parameters are dynamically changed, at least in the manner described here, for all four neighborhood orders. However, looking closer at how the dynamic change affects the objective function values for larger problem instances (150 nodes and more), it can be seen that the lowest objective function values occur when search parameters dynamically change; to be more precise, out of 20 problem sets (problem sets 10 through 30), 15 of them had the lowest values when the search parameters were dynamically changing. See table 6 and figure 12. Minimum values are highlighted.

Table 6. Minimum objective function values

| Set # | Fixed parameters | | | | Changing parameters | | | |
|-------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|
| | Trial 1 Obj. fun. | Trial 2 Obj. fun. | Trial 3 Obj. fun. | Trial 4 Obj. fun. | Trial 5 Obj. fun. | Trial 6 Obj. fun. | Trial 7 Obj. fun. | Trial 8 Obj. fun. |
| 1 | 2122.20 | 1974.38 | 2063.45 | 2072.71 | 2072.71 | 1974.38 | 2028.52 | 2072.71 |
| 2 | 1250.91 | 1264.39 | 1264.39 | 1201.15 | 1209.44 | 1201.25 | 1317.49 | 1280.97 |
| 3 | 2212.00 | 2290.04 | 1929.91 | 2096.58 | 2226.78 | 2233.58 | 2243.45 | 1972.29 |
| 4 | 2143.21 | 2284.21 | 2055.71 | 2284.21 | 2130.44 | 2284.21 | 2017.60 | 2284.21 |
| 5 | 1718.17 | 1779.20 | 1588.04 | 1536.30 | 1867.43 | 1895.52 | 1722.29 | 1779.14 |
| 6 | 2722.28 | 2664.65 | 2640.65 | 2751.47 | 2722.28 | 2613.88 | 2753.97 | 2572.13 |
| 7 | 1539.09 | 1426.45 | 1542.54 | 1667.97 | 1533.62 | 1602.48 | 1433.39 | 1667.97 |
| 8 | 2715.61 | 2656.26 | 2492.65 | 2584.55 | 2758.65 | 2683.95 | 2655.76 | 2743.94 |

| | | | | | | | | |
|----|---------|---------|---------|---------|---------|---------|---------|---------|
| 9 | 3187.74 | 3539.37 | 3419.05 | 3539.37 | 3373.77 | 3539.37 | 3446.02 | 3231.05 |
| 10 | 2609.37 | 2541.69 | 2738.94 | 2542.92 | 2445.93 | 2542.47 | 2721.51 | 2488.15 |
| 11 | 3361.17 | 3332.61 | 3298.90 | 3098.27 | 3311.17 | 3246.99 | 3134.84 | 3098.27 |
| 12 | 3247.96 | 3297.96 | 3165.24 | 3417.71 | 3233.49 | 3385.23 | 3157.77 | 3345.68 |
| 13 | 3501.75 | 3246.64 | 3370.01 | 3492.33 | 3202.48 | 3457.58 | 3443.55 | 3427.80 |
| 14 | 3613.18 | 3441.99 | 3245.69 | 3227.31 | 3295.72 | 3343.48 | 3153.77 | 3333.34 |
| 15 | 3939.70 | 3857.04 | 3740.19 | 3698.24 | 3774.59 | 3811.09 | 4049.45 | 3698.24 |
| 16 | 2568.28 | 2471.55 | 2552.15 | 2451.48 | 2347.08 | 2127.09 | 2272.98 | 2034.83 |
| 17 | 2457.93 | 2560.21 | 2365.72 | 2537.12 | 2600.02 | 2280.95 | 2217.76 | 2160.65 |
| 18 | 3049.40 | 2973.01 | 3207.35 | 2985.34 | 3181.62 | 3272.90 | 2970.62 | 2905.65 |
| 19 | 4026.65 | 3777.83 | 3921.51 | 3609.90 | 3812.42 | 3575.04 | 3824.77 | 3737.37 |
| 20 | 3060.29 | 2732.91 | 3348.74 | 3047.68 | 3022.75 | 2831.40 | 3098.70 | 2662.99 |
| 21 | 3128.74 | 3353.55 | 3450.76 | 3285.26 | 3386.31 | 3287.07 | 3428.97 | 3385.63 |
| 22 | 4523.01 | 4624.70 | 4154.66 | 4267.13 | 4548.13 | 4728.46 | 4568.10 | 4528.85 |
| 23 | 3537.66 | 3754.80 | 3788.04 | 3614.36 | 3604.43 | 3641.76 | 3459.53 | 3690.06 |
| 24 | 2854.61 | 2842.98 | 2896.05 | 2960.21 | 2939.76 | 2875.42 | 3060.34 | 2828.84 |
| 25 | 3167.00 | 3051.52 | 3033.59 | 3082.85 | 3084.74 | 2884.95 | 3105.17 | 3106.50 |
| 26 | 2134.21 | 2193.18 | 2193.18 | 2355.91 | 2739.79 | 2473.87 | 2226.52 | 2598.65 |
| 27 | 3136.89 | 3342.57 | 3052.60 | 3038.17 | 3105.16 | 2987.35 | 3006.17 | 2992.03 |
| 28 | 3925.67 | 4900.93 | 3932.88 | 4103.12 | 4167.33 | 4028.29 | 4253.63 | 4047.09 |
| 29 | 4176.09 | 4133.42 | 4183.21 | 4205.75 | 4236.64 | 4236.64 | 4262.76 | 4115.77 |
| 30 | 4763.65 | 4601.22 | 4963.99 | 4702.63 | 4914.05 | 4903.35 | 4844.16 | 4723.93 |

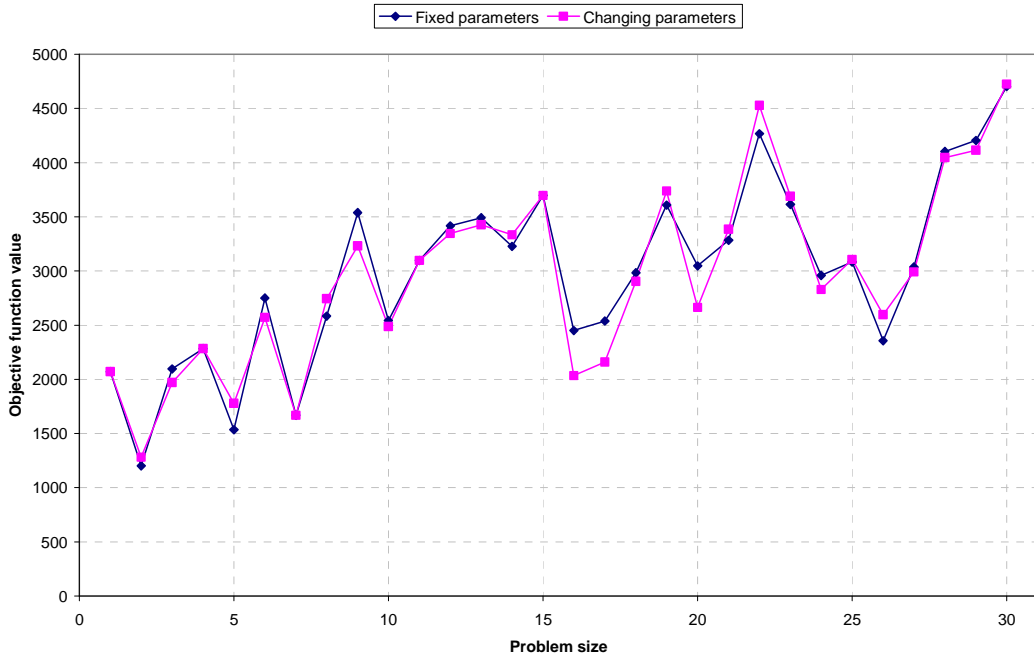


Figure 12. Effect of dynamically changing search parameters – neighborhood order 4

4.2.2 Analysis and discussion – running time

Using the paired t-test, to investigate the effect of dynamically changing search parameters, on the running time, will not be appropriate. As mentioned before, the normality assumption is crucial to conducting the paired t-test; however, in the running time case, the probability distribution of the differences is far from the normal distribution. Data transformation methods were considered, but, again, they were inappropriate. Box-Cox transformation simply did not work, as some of the data (differences) were negative; on the other hand, the Johnson transformation was capable of producing transformed normal data, but the transformation function was very complicated and difficult to interpret. Below is an example of the Johnson transformation made to the differences between trials 2 and 6 of the running time data, clearly, in figure 13, the transformation did produce normal data, but the transformation function is too complicated. The same thing applies for the remaining difference data.

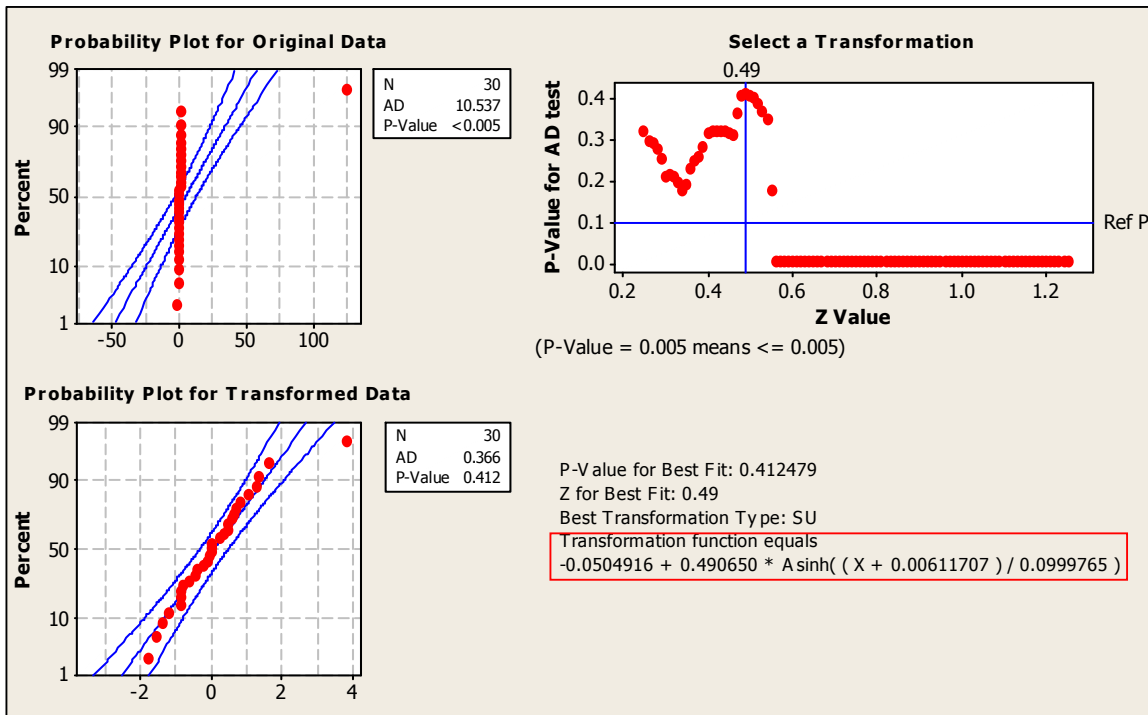


Figure 13. Johnson transformation for the difference data (trials 2 and 6)

In light of the difficulty to use the paired t-test, some other small observations were made instead. It can be seen in table 7 that the lowest running times occurred mostly in trials 2 and 6 (27 times out 30), which both have the same neighborhood order (Shaking Neighborhoods: Merge And Relocate, Cross, FP-Relocate, Exchange, Relocate. Local search Neighborhoods: Intra-Relocate Tours, Two-Opt, Or-Opt). This could indicate that the neighborhood order used in trials 2 and 6 produces the lowest running time for the algorithm.

Table 7. Results table – running time (min)

| Set # | Fixed parameters | | | | Changing parameters | | | |
|-------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|
| | Order 1 Trial 1 Time | Order 2 Trail 2 Time | Order 3 Trail 3 Time | Order 4 Trial 4 Time | Order 1 Trail 5 Time | Order 2 Trial 6 Time | Order 3 Trial 7 Time | Order 4 Trial 8 Time |
| 1 | 0.538 | 0.297 | 1.804 | 0.692 | 0.663 | 0.292 | 1.779 | 0.671 |
| 2 | 1.901 | 0.710 | 1.725 | 1.745 | 1.763 | 0.658 | 1.940 | 1.298 |
| 3 | 1.466 | 1.136 | 3.373 | 1.734 | 1.419 | 0.925 | 2.416 | 1.446 |
| 4 | 0.361 | 0.186 | 0.354 | 0.257 | 0.359 | 0.208 | 0.559 | 0.245 |
| 5 | 2.452 | 0.488 | 2.743 | 1.113 | 3.199 | 0.589 | 1.878 | 0.906 |
| 6 | 1.056 | 0.491 | 2.073 | 0.840 | 1.202 | 0.640 | 2.560 | 1.058 |
| 7 | 0.653 | 0.348 | 0.790 | 0.335 | 0.535 | 0.356 | 0.852 | 0.313 |
| 8 | 0.682 | 0.411 | 0.874 | 0.488 | 0.700 | 0.458 | 0.931 | 0.564 |
| 9 | 0.550 | 0.280 | 0.878 | 0.452 | 0.356 | 0.276 | 0.738 | 0.347 |
| 10 | 2.065 | 0.846 | 1.104 | 1.106 | 1.108 | 1.111 | 1.110 | 1.106 |
| 11 | 1.401 | 0.923 | 1.669 | 0.892 | 1.299 | 0.745 | 1.427 | 0.889 |
| 12 | 5.861 | 2.943 | 11.501 | 2.467 | 10.409 | 3.164 | 6.604 | 3.910 |
| 13 | 5.829 | 2.531 | 12.364 | 2.983 | 9.001 | 2.807 | 6.624 | 3.927 |
| 14 | 10.760 | 2.735 | 9.699 | 2.673 | 7.183 | 1.991 | 9.380 | 2.545 |
| 15 | 5.264 | 2.083 | 4.755 | 2.738 | 5.096 | 1.820 | 5.190 | 2.282 |
| 16 | 18.727 | 2.699 | 8.913 | 4.831 | 14.909 | 3.508 | 8.549 | 7.013 |
| 17 | 13.766 | 2.373 | 7.390 | 3.739 | 15.075 | 3.439 | 8.430 | 3.285 |
| 18 | 19.680 | 3.720 | 30.103 | 5.410 | 13.316 | 3.994 | 15.479 | 4.976 |
| 19 | 7.007 | 2.774 | 7.001 | 4.639 | 5.388 | 2.657 | 5.924 | 2.751 |
| 20 | 9.471 | 1.792 | 6.268 | 2.649 | 5.325 | 1.353 | 5.910 | 1.459 |
| 21 | 9.231 | 1.348 | 6.326 | 2.582 | 5.417 | 1.194 | 5.396 | 1.466 |
| 22 | 4.112 | 0.917 | 3.878 | 1.143 | 3.360 | 1.439 | 6.360 | 1.384 |
| 23 | 1.843 | 0.814 | 1.344 | 0.853 | 1.423 | 0.900 | 132.656 | 0.791 |
| 24 | 16.724 | 1.964 | 28.884 | 2.673 | 20.327 | 1.960 | 29.347 | 3.830 |
| 25 | 17.179 | 2.079 | 28.988 | 3.845 | 20.299 | 1.956 | 29.271 | 3.789 |
| 26 | 96.893 | 133.130 | 130.224 | 12.280 | 62.590 | 9.174 | 99.228 | 11.805 |
| 27 | 35.945 | 1.105 | 48.368 | 3.620 | 22.294 | 2.838 | 33.576 | 3.522 |
| 28 | 79.156 | 3.232 | 67.332 | 7.770 | 21.266 | 1.686 | 18.631 | 3.314 |
| 29 | 17.404 | 2.730 | 13.265 | 3.296 | 3.441 | 1.899 | 13.048 | 4.827 |
| 30 | 3.761 | 1.454 | 4.284 | 2.379 | 3.669 | 1.379 | 4.550 | 2.195 |

4.2.3 Analysis and discussion – competitive analysis

Solving the same problem sets again, under static conditions, produced the results in table 8 (page 66). The following is noted:

- Dynamic problem instances, where some visits could not be inserted (likes of: set 9, 13, 15, 17...), may have confusing competitive analysis ratios; for example, problem set 28 had 17 unperformed visits with a total cost of 3925.67; however, the static version of the same set had only 2 unperformed visits with a total cost of 4011.990, this contradicts what is expected (i.e. a static version of a problem should have a lower cost, as there are less constraints), but given the fact that the dynamic problem was not solved completely (17 unperformed visits), the added extra cost, of the static problem, is attributed to the increased number of satisfied requests (only 2 visits were unperformed).
- Given the first remark, the competitive ratio is interpreted, here, as the portion of the static (lower) solution that the online algorithm can achieve. For example, in problem set 8, the online algorithm can reach 0.76 of the static (lower) solution; meaning, had all the requests been known in advance, the routing plan for problem set 8 would have cost 1891.13 units, instead of 2492.65 (i.e. a total of 601.52 in savings).
- The last column in table 7 shows the value of knowing all relevant information before hand (i.e. how much savings could have been gained).

Table 8. Competitive analysis ratios

| Set # | Dynamic | | | Static | | | Comp. ratio | Gain |
|-------|----------|----------|--------------------|----------|----------|--------------------|-------------|---------|
| | Solution | Vehicles | Unperformed visits | Solution | Vehicles | Unperformed visits | | |
| 1 | 1974.38 | 14 | 0 | 1209.01 | 7 | 0 | 0.61 | 765.37 |
| 2 | 1201.15 | 10 | 0 | 1146.18 | 10 | 0 | 0.95 | 54.97 |
| 3 | 1929.91 | 11 | 0 | 1337.12 | 8 | 0 | 0.69 | 592.79 |
| 4 | 2017.6 | 18 | 0 | 1275.06 | 8 | 0 | 0.63 | 742.54 |
| 5 | 1536.3 | 13 | 0 | 1398.69 | 11 | 0 | 0.91 | 137.61 |
| 6 | 2572.13 | 17 | 0 | 2095.12 | 11 | 0 | 0.81 | 477.01 |
| 7 | 1426.45 | 17 | 0 | 1040.04 | 11 | 0 | 0.73 | 386.41 |
| 8 | 2492.65 | 16 | 0 | 1891.13 | 11 | 0 | 0.76 | 601.52 |
| 9 | 3187.74 | 26 | 1 | 2259.34 | 13 | 0 | 0.71 | 928.40 |
| 10 | 2541.69 | 17 | 0 | 1887.78 | 11 | 0 | 0.74 | 653.91 |
| 11 | 3098.27 | 27 | 0 | 2185.670 | 16 | 0 | 0.71 | 912.60 |
| 12 | 3157.77 | 21 | 0 | 2326.610 | 16 | 0 | 0.74 | 831.16 |
| 13 | 3202.48 | 30 | 4 | 2625.870 | 15 | 0 | 0.82 | 576.61 |
| 14 | 3153.77 | 16 | 0 | 2378.250 | 15 | 0 | 0.75 | 775.52 |
| 15 | 3698.24 | 30 | 13 | 2830.180 | 18 | 0 | 0.77 | 868.06 |
| 16 | 2034.83 | 17 | 0 | 1760.360 | 16 | 0 | 0.87 | 274.47 |
| 17 | 2160.65 | 30 | 10 | 1992.700 | 17 | 1 | 0.92 | 167.95 |
| 18 | 2905.65 | 18 | 0 | 2351.890 | 17 | 0 | 0.81 | 553.76 |
| 19 | 3575.04 | 27 | 0 | 2949.680 | 21 | 0 | 0.83 | 625.36 |
| 20 | 2662.99 | 22 | 0 | 2111.570 | 16 | 0 | 0.79 | 551.42 |
| 21 | 3128.74 | 30 | 6 | 2624.330 | 17 | 2 | 0.84 | 504.41 |
| 22 | 4154.66 | 30 | 10 | 3075.630 | 19 | 0 | 0.74 | 1079.03 |
| 23 | 3459.53 | 30 | 9 | 3076.910 | 23 | 10 | 0.89 | 382.62 |
| 24 | 2828.84 | 27 | 0 | 2112.080 | 19 | 3 | 0.75 | 716.76 |
| 25 | 2884.95 | 30 | 2 | 2277.680 | 17 | 0 | 0.79 | 607.27 |
| 26 | 2134.21 | 22 | 0 | 1811.300 | 14 | 0 | 0.85 | 322.91 |
| 27 | 2987.35 | 28 | 0 | 2117.390 | 15 | 2 | 0.71 | 869.96 |
| 28 | 3925.67 | 30 | 17 | 4011.990 | 23 | 2 | 1.02 | -86.32 |
| 29 | 4115.77 | 27 | 0 | NA | NA | NA | NA | NA |
| 30 | 4601.22 | 30 | 0 | 3406.750 | 24 | 24 | 0.74 | 1194.47 |

* Static problem could not be solved

4.2.4 Analysis and discussion – rejected visits

Newly arriving requests with wide time windows are easily satisfied compared to requests with tight time windows. The $edod_{nw}$, discussed in section 2.5.1, is one measure to assess the hardness of the problem based on the time windows of the newly arriving requests. Most of the data sets used were intentionally modified by changing their $edod_{nw}$. For example, in table 9 and figures 14 ~ 17 (pages 68 and 69), problem sets 10 and 11 are exactly the same, except that problem set 11 has tighter time windows for its dynamic requests; hence, it is

expected to have a higher objective function value and a higher number of rejected dynamic requests. Same thing applies to many other sets. Figures 18 ~ 21 (pages 70 and 71) show how various problem sets with high $edod_{tw}$ have a higher percentage of rejected requests (percentage rejected requests out of the dynamic requests).

Table 9. Problem sets with different $edod_{tw}$

| Set # | Static | Dynamic | $edod_{tw}$ | Type |
|-------|---------|-----------|-------------|------|
| 1 | 1 ~ 20 | 21 ~ 58 | 0.52 | NA |
| 2 | 1 ~ 50 | 51 ~ 68 | 0.17 | NA |
| 3 | 1 ~ 30 | 31 ~ 88 | 0.46 | NA |
| 4 | 1 ~ 18 | 19 ~ 96 | 0.61 | NA |
| 5 | 1 ~ 60 | 61 ~ 98 | 0.26 | NA |
| 6 | 1 ~ 46 | 47 ~ 104 | 0.45 | NA |
| 7 | 1 ~ 12 | 13 ~ 110 | 0.70 | NA |
| 8 | 1 ~ 32 | 33 ~ 130 | 0.52 | easy |
| 9 | 1 ~ 32 | 33 ~ 130 | 0.62 | hard |
| 10 | 1 ~ 16 | 17 ~ 154 | 0.58 | easy |
| 11 | 1 ~ 16 | 17 ~ 154 | 0.69 | hard |
| 12 | 1 ~ 48 | 49 ~ 166 | 0.51 | easy |
| 13 | 1 ~ 48 | 49 ~ 166 | 0.60 | hard |
| 14 | 1 ~ 40 | 40 ~ 178 | 0.48 | easy |
| 15 | 1 ~ 40 | 40 ~ 178 | 0.63 | hard |
| 16 | 1 ~ 58 | 59 ~ 196 | 0.46 | easy |
| 17 | 1 ~ 58 | 59 ~ 196 | 0.60 | hard |
| 18 | 1 ~ 70 | 71 ~ 208 | 0.40 | easy |
| 19 | 1 ~ 70 | 71 ~ 208 | 0.53 | hard |
| 20 | 1 ~ 82 | 83 ~ 220 | 0.41 | easy |
| 21 | 1 ~ 82 | 83 ~ 220 | 0.54 | hard |
| 22 | 1 ~ 66 | 67 ~ 244 | 0.51 | easy |
| 23 | 1 ~ 26 | 27 ~ 244 | 0.70 | hard |
| 24 | 1 ~ 150 | 151 ~ 268 | 0.32 | easy |
| 25 | 1 ~ 150 | 151 ~ 268 | 0.36 | hard |
| 26 | 1 ~ 140 | 141 ~ 298 | 0.26 | easy |
| 27 | 1 ~ 140 | 141 ~ 298 | 0.29 | hard |
| 28 | 1 ~ 180 | 181 ~ 308 | 0.30 | NA |
| 29 | 1 ~ 50 | 51 ~ 308 | 0.55 | NA |
| 30 | 1 ~ 22 | 23 ~ 320 | 0.68 | NA |

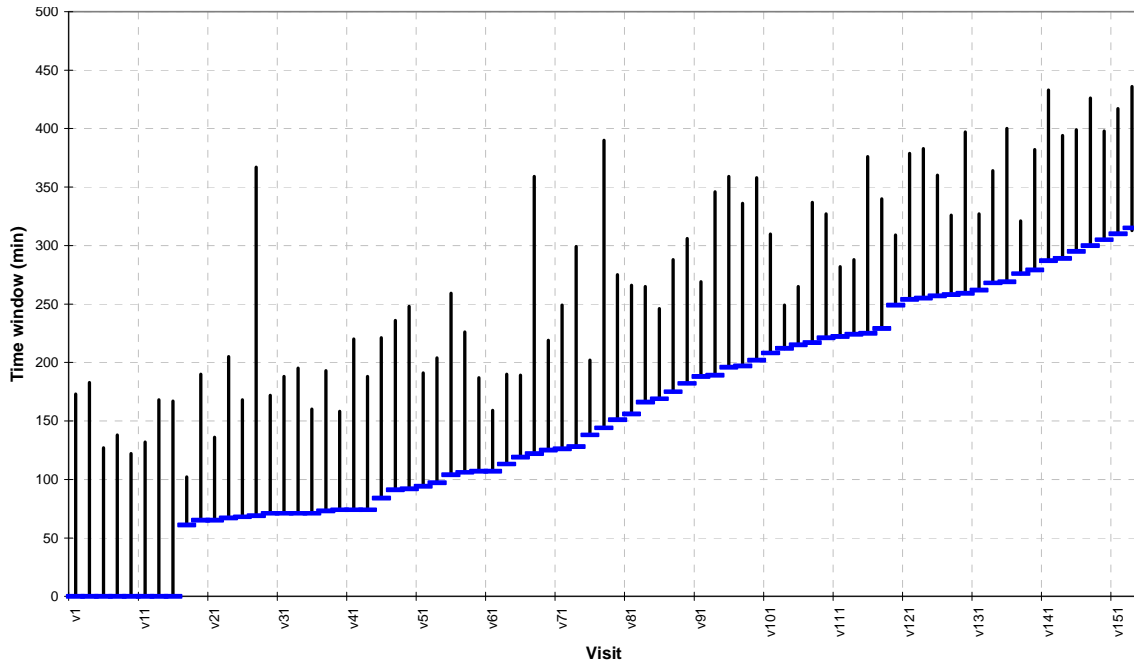


Figure 14. Pickup time windows - problem set 10

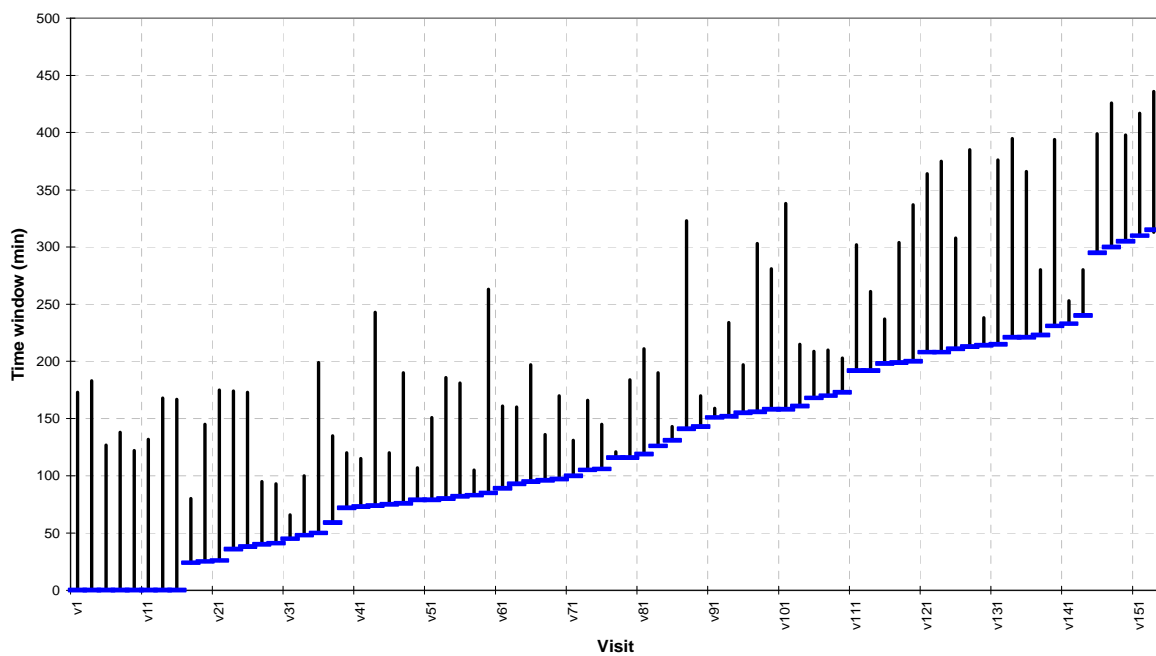


Figure 15. Pickup time windows - problem set 11

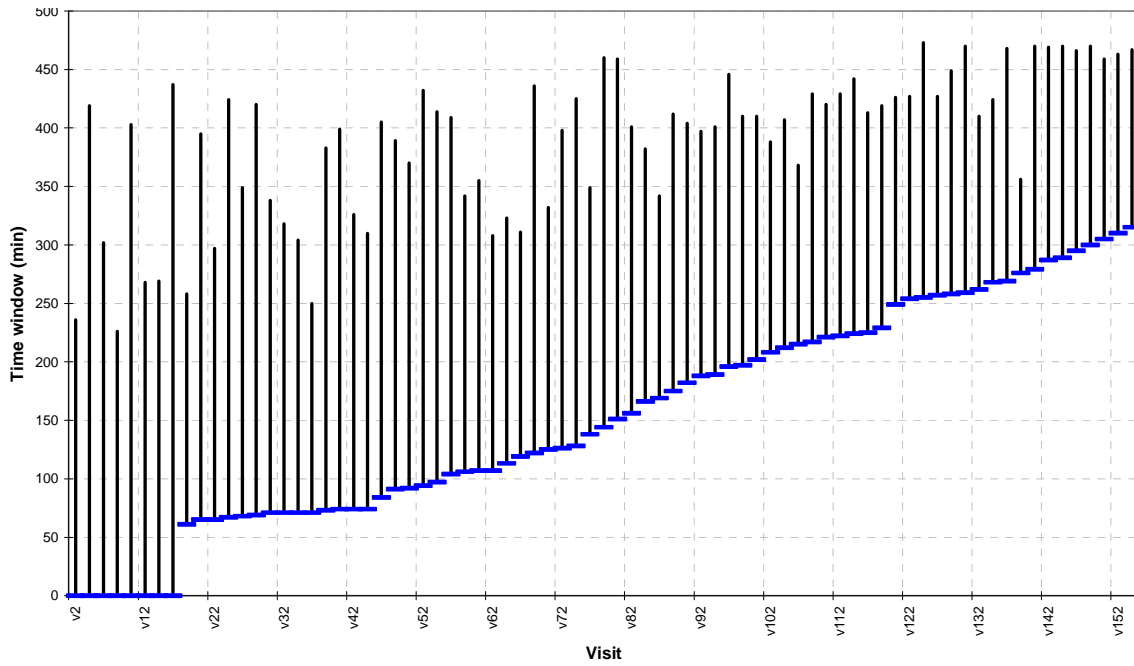


Figure 16. Delivery time windows - problem set 10

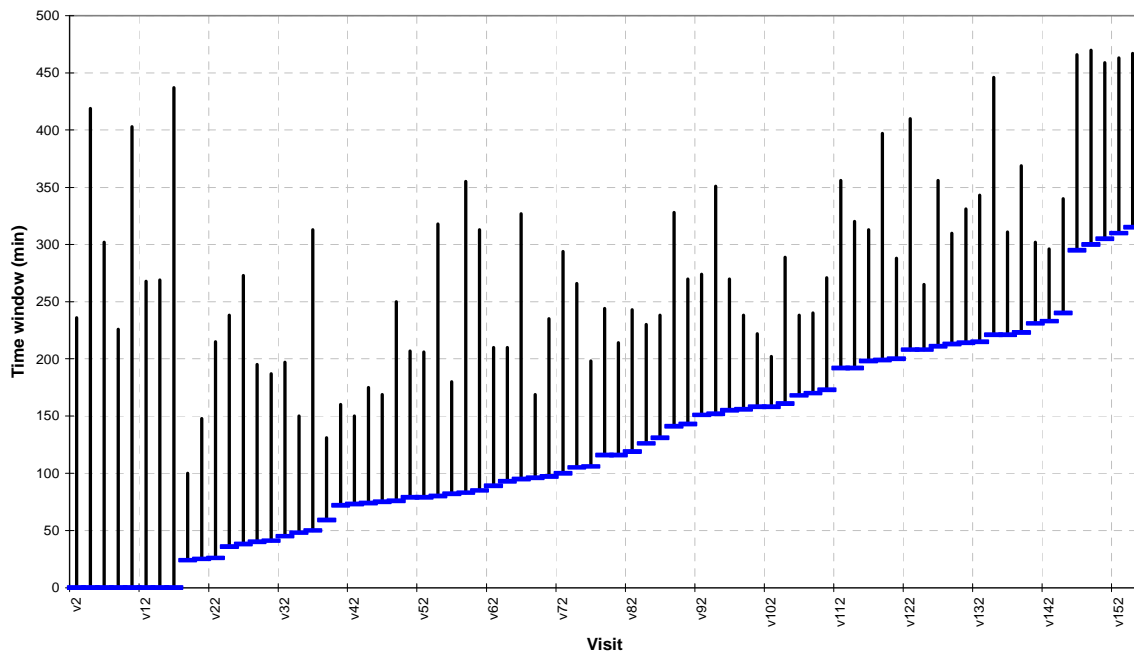


Figure 17. Delivery time windows - problem set 11

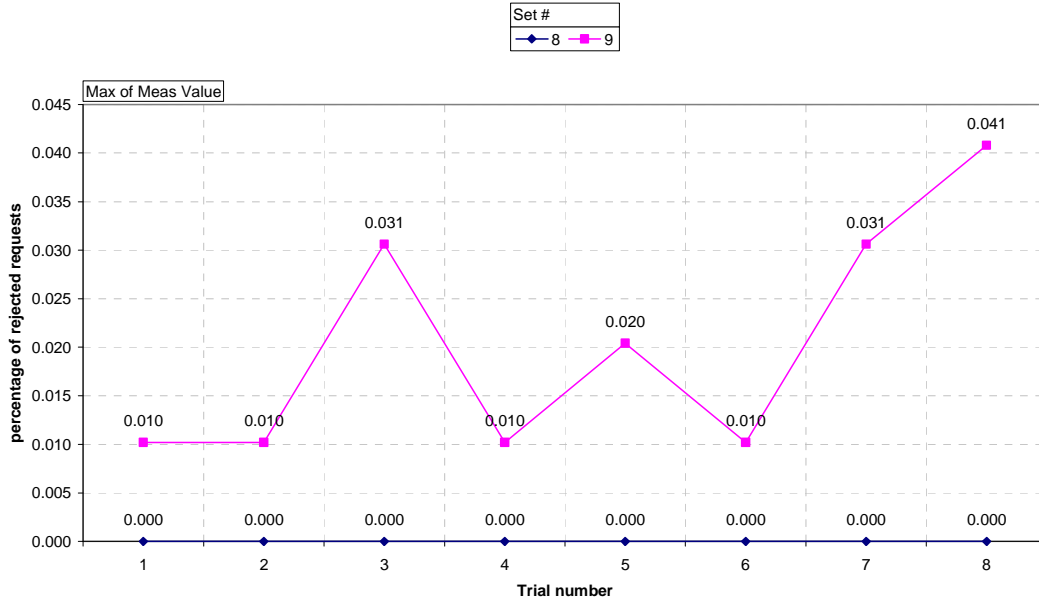


Figure 18. Effect of the degree of dynamism on the percentage of rejected requests – problem sets 8 (low $edod_{tw}$) and 9 (high $edod_{tw}$)

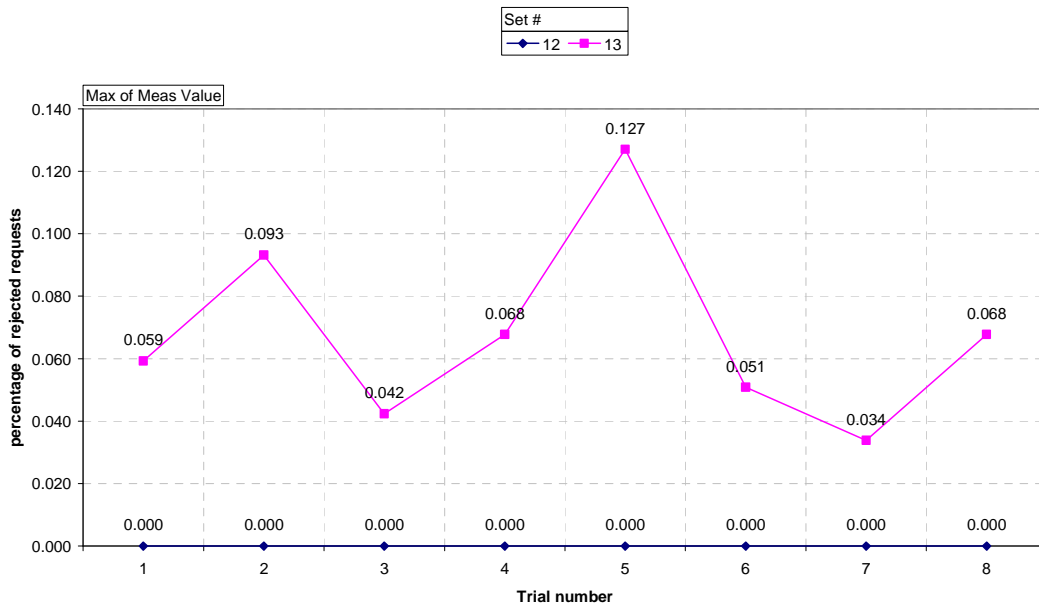


Figure 19. Effect of the degree of dynamism on the percentage of rejected requests – problem sets 12 (low $edod_{tw}$) and 13 (high $edod_{tw}$)

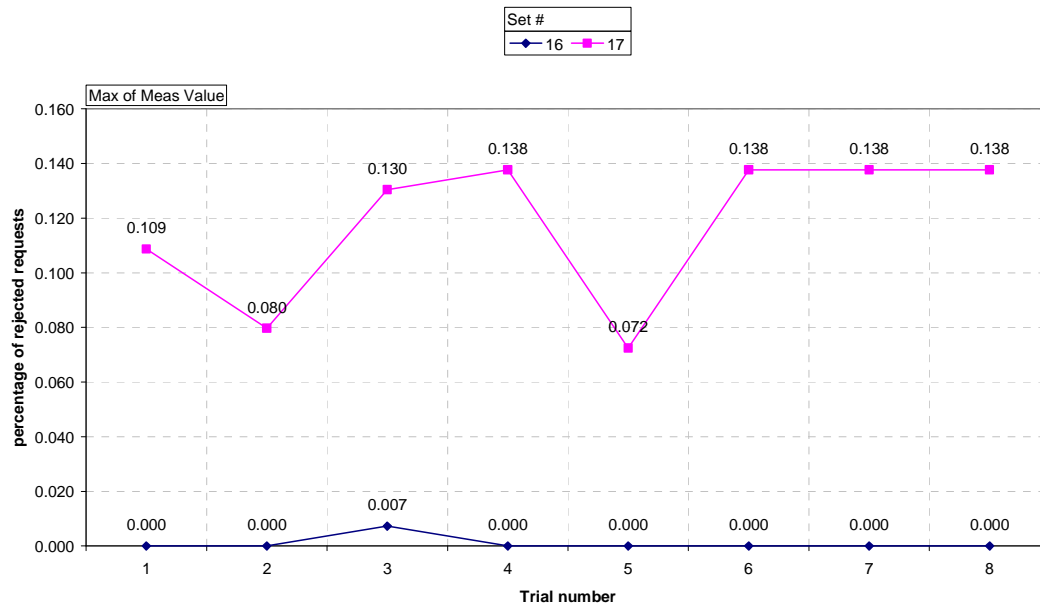


Figure 20. Effect of the degree of dynamism on the percentage of rejected requests – problem sets 16 (low $edod_{tw}$) and 17 (high $edod_{tw}$)

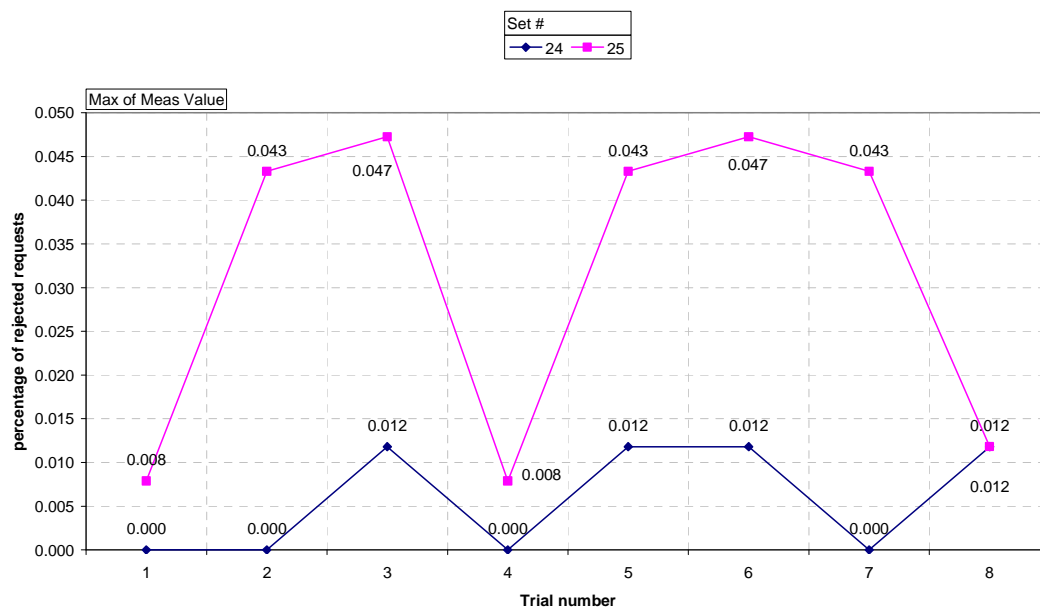


Figure 21. Effect of the degree of dynamism on the percentage of rejected requests – problem sets 24 (low $edod_{tw}$) and 25 (high $edod_{tw}$)

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The General Pickup and Delivery Problem with Time Windows has many variants, all of which are classified as NP -hard problems; which means, there is no known polynomial time algorithm capable of producing an optimal solution, at least for large problem instances. As such, heuristic and metaheuristic methods are usually applied to gain *near* optimal solutions in reasonable running times. In this study, an online hybrid metaheuristic based on Variable Neighborhood Search, Tabu Search, and Guided Local Search was created and tested on one variant of the general model (i.e. the *Dynamic* Pickup and Delivery Problem with Time Windows).

Two major issues were addressed for the online hybrid; the effect of dynamically changing a metaheuristic's search parameters, during the search, on solution quality and algorithm running time, and the effect of changing a metaheuristic's neighborhood order on solution quality and algorithm running time. The algorithm was tested against problem instances based on the works of Christofides, 13 data sets; Fisher, 5 data sets; and Taillard, 12 data sets; however, these sets were modified to include time windows to fit this study.

It was found that, for *large* problem instances, dynamically changing search parameters (starting with Tenure = 5 and Penalty = 0.45, then diversifying the search, when no improving moves are found for the past 10 iterations, to Tenure = 12 and Penalty = 0.8, and finally, intensifying the search again by using the original parameter values) produced better solutions more often, although there was no statistical evidence to support this. However, dynamically changing the search parameters did not have any obvious effect on the

algorithm's running time. In addition, the neighborhood order did not seem to have an effect on the solution quality, but the running time was obviously lower for a specific neighborhood order, compared to all other orders. It was found that arranging the neighborhoods in this manner: Shaking Neighborhoods: Merge And Relocate, Cross, FP-Relocate, Exchange, Relocate; Local search Neighborhoods: Intra-Relocate Tours, Two-Opt, Or-Opt, produced lower running times for *most* problem instances (27 times out of 30). This conclusion could not be supported by statistical tests, as the data required some sort of transformation that are too complicated and will make any kind of conclusions very difficult to interpret.

Furthermore, the online algorithm was assessed based on the competitive analysis concept; although exact adherence to the concept was not possible, due to the nature of the selected problem (i.e. it being NP -hard and no optimal solution is known for the static versions of the problem instances used). All problem instances were solved again under static conditions (i.e. all visits were assumed to be known at the route planning time, and no new visits appeared), this provided benchmark solutions to which the online algorithm can be compared to; meaning, the static solutions are the best possible solutions that the online algorithm can achieve. It was found that, the online algorithm was capable of reaching anywhere between 0.61 and 0.95 of the solution obtained by its offline algorithm counterpart (static solution). This is a good indication of how well the developed algorithm can perform under dynamic conditions.

Other conclusions, although not of primary interest to this work, were made from the results; it was shown that, for almost all problem instances, increasing the degree of dynamism increased the number of rejected dynamic requests. This, of course, is expected, but the way the problem sets were chosen made it difficult to present irrefutable evidence of

this notion; still, such evidence can be easily obtained with the appropriate selection of problem sets.

5.2 Recommendations and future work

Based on the above, the following recommendations should be considered when solving the dynamic pickup and delivery problem with time windows:

- Dynamically changing search parameters, during the search, has a higher chance of producing better solutions for larger problem instances (150 nodes and more).
- When using Variable Neighborhood Search to solve the problem, this neighborhood order has a higher chance of producing lower running times compared to other orders; Shaking Neighborhoods: Merge And Relocate, Cross, FP-Relocate, Exchange, Relocate; Local search Neighborhoods: Intra-Relocate Tours, Two-Opt, Or-Opt.
- Integrating Tabu Search and Guided Local Search into the Variable Neighborhood Search framework does produce higher quality solutions in shorter times.

With all the work that has been done, there are still a lot of issues that require further research, and future extensions can be built on this work. Further research is required in the following areas:

- Using a better methodology to select, and dynamically change, search parameters, likes of adaptive search and racing algorithms.
- Setting the running time of the online algorithm to a limit that suits the problem instance solved; meaning, instead of having the online algorithm run for a fixed time limit (maximum of 15 minutes in this study), the running time should be set such that the number of idle vehicles on the road is minimum.

- The number of dynamic requests that are accepted before running the online algorithm was set, in this study, to 10; however, a better approach is to allow it to change such that the number of accepted requests is maximum, while at the same time preserving some excess capacity for the vehicles to accept other requests that may appear in the near future.
- Using the wait first strategy, instead of the drive first strategy, if it proves to produce shorter routes using the same number of vehicles.

As for future extensions, the following can be added to this work:

- Combining Variable Neighborhood Search with other heuristics that can further enhance its diversification schemes, iterated local search would be a good option to explore.
- Creating a better measure of degree of dynamism to include the dependency of the pickup and delivery pair.
- Applying the same algorithm to a stochastic version of the problem, this will give a better indication of how well it can be applied to a more realistic situation.

REFERENCES

- Attanasio, A. Cordeau, J. Ghiani, G. and Laporte, G. (2004), Parallel Tabu Search Heuristics for the Dynamic Multi-Vehicle Dial-A-Ride Problem, **Parallel Computing**, 30(3): 377 – 387.
- Baker, E. and Schafer J. (1986), Solution Improvement Heuristics for the Vehicle Routing and Scheduling Problem with Time Window Constraints, **American Journal of Mathematical and Management Science** 6(3): 261 – 300.
- Bausch, D. Brown, G. and Ronen, D. (1995), Consolidating And Dispatching Truck Shipments Of Heavy Petroleum Products, **Interfaces** 25(2): 1 – 17.
- Bertsimas, D. and Ryzin, G. (1991), A Stochastic and Dynamic Vehicle Routing Problem In The Euclidean Plane, **Operations Research** 39(4): 601 – 615.
- Bertsimas, D. and Ryzin, G. (1993), Stochastic And Dynamic Vehicle Routing Problem In The Euclidean Plane With Multiple Capacitated Vehicles, **Operations Research** 41(1): 60 – 76.
- Bertsimas, D. and Ryzin, G. (1993), Stochastic And Dynamic Vehicle Routing With General Demand And Interarrival Time Distributions, **Applied Probability** 25(5): 947 – 978.
- Borodin, A. and El-Yaniv, R. (1998), **Online Computation and Competitive Analysis**, 1st edition, UK: Cambridge University Press.
- Brown, G. Ellis, C. Graves, G. and Ronen, D. (1987), Real Time Wide Area Dispatch Of Mobil Tank Trucks, **Interfaces** 17(1): 107 – 120.
- Caramia, M. Italiano, G. Oriolo, G. Pacifici, A. and Perugia, A. (2002), Routing A Fleet Of Vehicles For Dynamic Combined Pick-Up And Delivery Services, **Operations Research Proceedings**, Duisburg, Germany, 3 – 5 September 2002, 3 – 8.
- Chamoni, P. Leisten, R. Martin, A. Minnemann, J. and Stadtler, H. (2007), Planned Route Optimization For Real-Time Vehicle Routing. In: Zeimpekis, Vasileios; Tarantilis, Christos; Giaglis, George; and Minis, Ioannis, **Dynamic Fleet Management**, 1st edition, USA: Springerlink.
- Christofides, N. and Beasley, J. (1984), The period routing problem, **Networks**, 14(2): 237 – 256.
- Cook, S. (1971), The complexity of theorem-proving procedures, **Annual ACM Symposium on Theory of Computing**, 3(3): 151 – 158.
- Cordeau, J. and Laporte, G. (2003), A Tabu Search For The Static Multi-Vehicle Dial-A-Ride Problem, **Transportation Research** 37(6): 579 – 594.

Croes, G. A. (1958), A Method For Solving Traveling Salesman Problems, **Operations Research**, 6(6): 791-812.

Dorigo, M. and Stutzle, T. (2004), **Ant Colony Optimization**, 1st edition, USA: MIT Press.

Fisher M, Greenfield A, Jaikumar R, Kedia P. (1982), **Real-Time Scheduling Of A Bulk Delivery Fleet: Practical Application of Lagrangean Relaxation**, Technical Report, University of Pennsylvania, Department of Decision Sciences, The Wharton School, USA.

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

Gambardella, L.M. Rizzoli, A.E. Oliverio, F. Casagrande, N. Donati, A.V. Montemanni, R. and Lucibello, E. (2003), Ant Colony Optimization for Vehicle Routing In Advanced Logistic Systems, **International Workshop on Modelling and Applied Simulation**, Bergeggi, Italy, 2 – 4 October, 2003, 3 – 9.

Gendreau, M. Guertin, F. Potvin, J.Y. and Séguin, R. (1998), Neighborhood Search Heuristics for A Dynamic Vehicle Dispatching Problem with Pick-Ups And Deliveries, **Transportation Research Part C**, 14(3): 157 – 174.

Gendreau, M. Guertin, F. Potvin, J.Y. and Taillard, É. D. (1999), Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching, **Transportation Science**, 33(4): 381 – 390.

Gendreau, Michel (2003), An Introduction to Tabu Search. In: Glover, Fred W.; Kochenberger, Gary A., **Handbook of Metaheuristics**, (1st Ed.), USA: Kluwer Academic Publishers.

Glover, F. Laguna, M. and Marti, R. (1993), Scatter Search and Path Relinking: Advances and Applications. In: Glover, and Kochenberger, **Handbook Of Metaheuristics**, 1st edition, USA: Kluwer Academic Publishers.

Glover, Fred (1986), Future Paths for Integer Programming and Links To Artificial Intelligence, **Computers and Operations Research**, 13(5): 533 – 549.

Hentenryck, V. (2006), **Online Stochastic Combinatorial Optimization**, 1st edition, USA: MIT Press.

Ichoua, S. Gendreau, M. and Potvin, J.Y. (2000), Diversion Issues in Real-Time Vehicle Dispatching, **Transportation Science**, 34(4): 426 – 438.

Johnson, D. and Garey, M. (1979), **Computers and Intractability A Guide to NP-Completeness**, 1st edition, USA: W. H. Freeman.

Kallehauge, B. Vejleder, M. Larsen, J. and Vejleder, M. (2006), Vehicle Routing Problem with Time Windows, **Computers and Operations Research**, 32(11): 1464 – 1487.

- Kallrath, J. (2005), **Modeling Languages in Mathematical Optimization**, 1st edition, USA: SpringerLink.
- Kilby, P. Prosser, P. and Shaw, P. (1998), **Dynamic VRPs: A Study of Scenarios**, Technical Report APES-06-1998, University of Strathclyde, Glasgow, UK.
- Kolen, A. Rinnooy, A. and Trienekens, H. (1987), Vehicle Routing With Time Windows, **Operations Research**, 35(2): 266 – 273.
- Koutsoupias, E. and Papadimitriou, C. (2000), Beyond Competitive Analysis, **SIAM Journal on Computing**, 30(1): 300 – 317.
- Kreyszig, E. (1999), **Advanced Engineering Mathematics**, 8th edition, USA: John Wiley & Sons.
- Laporte, G. Louveaux, F. V. and Van Hamme, L. (2002), An Integer L-Shaped Algorithm For The Capacitated Vehicle Routing Problem With Stochastic Demands. **Operations Research**, 50(3): 415 – 423.
- Larsen, A. (2001), **The Dynamic Vehicle Routing Problem**, unpublished dissertation, Technical University of Denmark, Denmark.
- Letchford, A. N. Lysgaard, J. and Eglese, R.W. (2006), A Branch-And-Cut Algorithm For The Capacitated Open Vehicle Routing Problem, The Department of Management Science, Lancaster University. UK.
- Lund, K. Madsen, G. and Rygaard, M. (1996), **Vehicle routing problems with varying degrees of dynamism**, unpublished technical report, Technical University of Denmark, Denmark.
- Madsen, O. Hans, F. and Rygaard, J. (1995), A Heuristic Algorithm For A Dial-A-Ride Problem With Time Windows, Multiple Capacities And Multiple Objectives. **Annals of Operations Research**, 60(1): 193 – 208.
- Mitrović-Minić, S. and Laporte, G. (2004), Waiting Strategies For The Dynamic Pickup And Delivery Problem With Time Windows, **Transportation Research B**, 38(7): 635 – 655.
- Mitrović-Minić, S. Krishnamurti, R. and Laporte, G. (2004), Double-Horizon Based Heuristics for The Dynamic Pickup And Delivery Problem With Time Windows, **Transportation Research B**, 38(8): 669 – 685.
- Mladenovic, N. and Hasen, P. (2005), Variable Neighborhood Search. In: Burke, E. and Kendall, G. **Search Methodologies**, USA: Springer.
- Montemanni, R. Gambardella, L. Rizzoli, M. E. and Donati, A. V. (2005), Ant Colony System For A Dynamic Vehicle Routing Problem, **Journal of Combinatorial Optimization** 27(1): 327-343.

- Nemhauser, G. and Wolsey, L. (1999), **Integer and Combinatorial Optimization**, 1st edition, USA: John Wiley & Sons.
- Papee, Willem E. (2002), **Complexity Results and Competitive Analysis for Vehicle Routing Problems**, unpublished masters thesis, University of Technology, Eindhoven, Netherlands.
- Potvin, J.Y. and Rousseau, J.M. (1993), Parallel Route Building Algorithm for The Vehicle Routing And Scheduling Problem with Time Windows, **European Journal of Operational Research**, 66(4): 331 – 340.
- Psaraftis, Harilaos (1983), An Exact Algorithm for The Single Vehicle Many-To-Many Dial-A-Ride Problem with Time Windows, **Transportation Science**, 17(3): 351 – 357.
- Psaraftis, Harilaos (1988), Dynamic Vehicle Routing Problems. In: Golden, B. L. and Assad, A., **Vehicle Routing: Methods and Studies**, Elsevier Science Publishers, North-Holland.
- Psaraftis, Harilaos (1988), Vehicle routing Methods and studies. In: B. Golden, A. Assad, **Dynamic Vehicle Routing Problems**, (1st Ed.) North Holland, Elsevier Science Publishers, the Netherlands.
- Psaraftis, N. (1995), Dynamic vehicle routing: Status and prospects, **Annals of Operations Research**, 61(1): 143 – 164.
- Regan, A. C. Mahmassani, H. S. and Jaillet, P. (1995), Improving Efficiency of Commercial Vehicle Operations Using Real-Time Information: Potential Uses and Assignment Strategies, **Transportation Research Record** 1493: 188 – 197.
- Sleator, D. and Tarjan, R. (1985), Amortized Efficiency of List Update and Paging Rules, **Communications of the ACM**, 28(2): 202 – 208.
- Swihart, M. R. and Papastavrou, J. D. (1999), A Stochastic and Dynamic Model For The Single-Vehicle Pick-Up And Delivery Problem, **European Journal of Operational Research**, 114(3): 447 – 464.
- Taillard, E. (1994), Parallel iterative search methods for vehicle-routing problems, **Networks**, 23(8): 661 – 673.
- Van Landeghem, H. R. G. (1988), A Bi-Criteria Heuristic for The Vehicle Routing Problem with Time Windows, **European Journal of Operational Research**, 36(2): 217 - 226.
- Voudouris, Creg (2003), Guided Local Search. In: Glover, Fred W.; Kochenberger, Gary A., **Handbook of Metaheuristics**, (1st Ed.), Kluwer Academic Publishers, USA.

Appendix 1 – Neighborhood Structures Definition

Intra-route neighborhoods:

- **Intra-Relocate:** This neighborhood modifies the solution by relocating individual visits to a new position in the same route. This neighborhood is similar to Relocate, except that it relocates visits to a new position in the same route. Since it explores fewer options for the relocated visit, this neighborhood is potentially smaller than one created by Relocate.
- **Two-Opt:** This neighborhood modifies the solution by breaking two arcs and starts looking for new neighbors at the place where the last modification took place, in specific the steps are:
 1. Take an initial route.
 2. Remove two arcs from the route, and try the other possible reconnection of the remaining parts of the route.
 3. If the cost has been reduced and if all constraints are satisfied, go back to Step 2.
 4. End.

Example: in the below figure the neighborhood eliminates the crossing by destroying two arcs and creating two new arcs, the resulting route is shorter. With this neighborhood, directional flows between visits may be reversed. However, the presence of tight time constraints can therefore decrease its effectiveness.

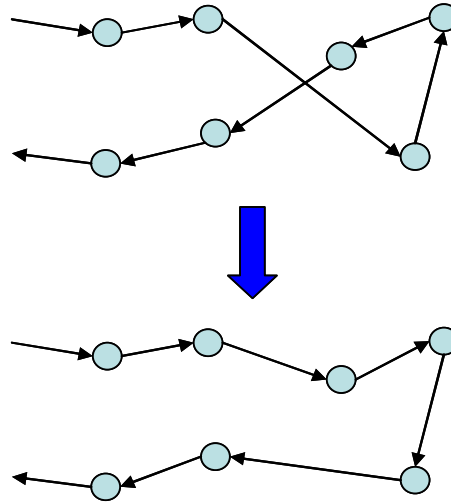


Figure 1. Two-Opt neighborhood

- Or-Opt: This neighborhood modifies the solution by relocating segments of visits in the same route. In specific the steps are:
 1. Start with an initial route.
 2. Move parts composed of one visit elsewhere in the route.
 3. If the cost has been reduced and if all constraints are satisfied, go back to Step 2.
 4. When all such moves have been tested, try moving parts of the route composed of two consecutive visits.
 5. After testing all moves of parts composed of two consecutive visits, try moving parts of the route composed of three consecutive visits.

Example: in the following figure the neighborhood eliminates the crossing by destroying three arcs and creating three new arcs, the resulting route is shorter.

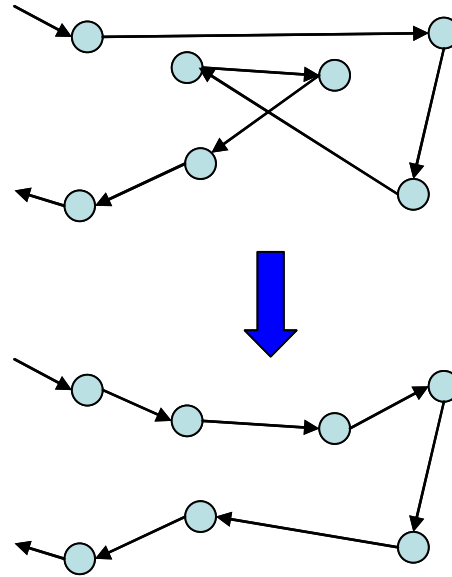


Figure 2. Or-Opt neighborhood

Inter-route neighborhoods:

- **Cross:** in a cross neighborhood the ends of two routes are exchanged: the first part of route A is connected to the last part (end) of route B and the first part of route B is connected to the last part (end) of route A.

Example: in the following figure the neighborhood eliminates the crossing by destroying two arcs and creating two new arcs, the resulting routes are shorter.

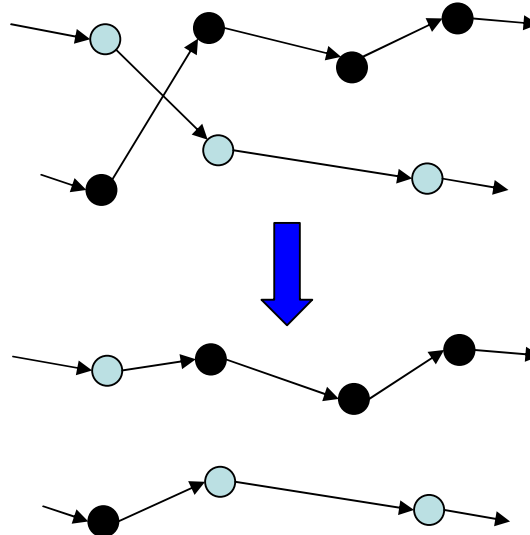


Figure 3. Cross neighborhood

- Exchange: in an exchange neighborhood, two visits of two different routes swap places if all constraints are still satisfied. This method can be generalized if more than one visit of a route is exchanged at the same time. When a pair of visits is exchanged, this neighborhood is useful for optimizing problems such as the Pickup-and-Delivery Problem.

Example: in the following figure the neighborhood eliminates the crossings by destroying four arcs and creating four new arcs, the resulting routes are shorter.

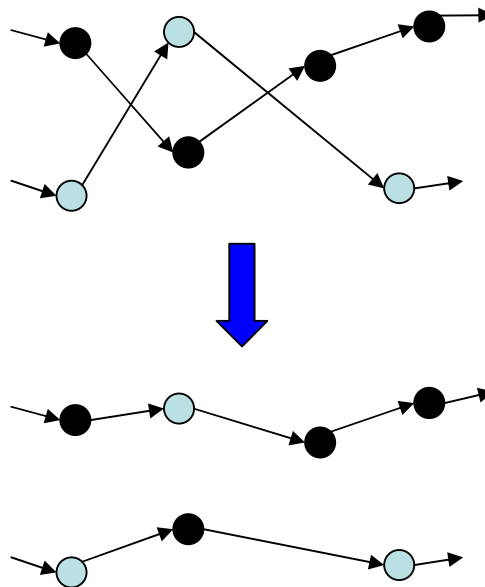


Figure 4. Exchange neighborhood

Appendix 2 – Data Sets and Results Summary

Table 1. Summary of problem instances

| Set # | Static | Dynamic | Total | edod _{tw} |
|-------|---------|-----------|-------|--------------------|
| 1 | 1 ~ 20 | 21 ~ 58 | 58 | 0.52 |
| 2 | 1 ~ 50 | 51 ~ 68 | 68 | 0.17 |
| 3 | 1 ~ 30 | 31 ~ 88 | 88 | 0.46 |
| 4 | 1 ~ 18 | 19 ~ 96 | 96 | 0.61 |
| 5 | 1 ~ 60 | 61 ~ 98 | 98 | 0.26 |
| 6 | 1 ~ 46 | 47 ~ 104 | 104 | 0.45 |
| 7 | 1 ~ 12 | 13 ~ 110 | 110 | 0.70 |
| 8 | 1 ~ 32 | 33 ~ 130 | 130 | 0.52 |
| 9 | 1 ~ 32 | 33 ~ 130 | 130 | 0.62 |
| 10 | 1 ~ 16 | 17 ~ 154 | 154 | 0.58 |
| 11 | 1 ~ 16 | 17 ~ 154 | 154 | 0.69 |
| 12 | 1 ~ 48 | 49 ~ 166 | 166 | 0.51 |
| 13 | 1 ~ 48 | 49 ~ 166 | 166 | 0.60 |
| 14 | 1 ~ 40 | 40 ~ 178 | 178 | 0.48 |
| 15 | 1 ~ 40 | 40 ~ 178 | 178 | 0.63 |
| 16 | 1 ~ 58 | 59 ~ 196 | 196 | 0.46 |
| 17 | 1 ~ 58 | 59 ~ 196 | 196 | 0.60 |
| 18 | 1 ~ 70 | 71 ~ 208 | 208 | 0.40 |
| 19 | 1 ~ 70 | 71 ~ 208 | 208 | 0.53 |
| 20 | 1 ~ 82 | 83 ~ 220 | 220 | 0.41 |
| 21 | 1 ~ 82 | 83 ~ 220 | 220 | 0.54 |
| 22 | 1 ~ 66 | 67 ~ 244 | 244 | 0.51 |
| 23 | 1 ~ 26 | 27 ~ 244 | 244 | 0.70 |
| 24 | 1 ~ 150 | 151 ~ 268 | 268 | 0.32 |
| 25 | 1 ~ 150 | 151 ~ 268 | 268 | 0.36 |
| 26 | 1 ~ 140 | 141 ~ 298 | 298 | 0.26 |
| 27 | 1 ~ 140 | 141 ~ 298 | 298 | 0.29 |
| 28 | 1 ~ 180 | 181 ~ 308 | 308 | 0.30 |
| 29 | 1 ~ 50 | 51 ~ 308 | 308 | 0.55 |
| 30 | 1 ~ 22 | 23 ~ 320 | 320 | 0.68 |

Table 2. Results summary

| Set # | Trial # | Initial Solution | Final Solution | Vehicles | Max CPU Time (min) | Unperformed visits |
|-------|---------|------------------|----------------|----------|--------------------|--------------------|
| 1 | 1 | 520.340 | 2122.200 | 15 | 0.538 | 0 |
| | 2 | 520.340 | 1974.380 | 14 | 0.297 | 0 |
| | 3 | 520.340 | 2063.450 | 15 | 1.804 | 0 |
| | 4 | 520.340 | 2072.710 | 16 | 0.692 | 0 |
| | 5 | 520.340 | 2072.710 | 16 | 0.663 | 0 |
| | 6 | 520.340 | 1974.380 | 14 | 0.292 | 0 |
| | 7 | 520.340 | 2028.520 | 14 | 1.779 | 0 |
| | 8 | 520.340 | 2072.710 | 16 | 0.671 | 0 |
| 2 | 1 | 1291.750 | 1250.910 | 11 | 1.901 | 0 |
| | 2 | 1291.750 | 1264.390 | 11 | 0.710 | 0 |
| | 3 | 1291.750 | 1264.390 | 11 | 1.725 | 0 |
| | 4 | 1291.750 | 1201.150 | 10 | 1.745 | 0 |
| | 5 | 1291.750 | 1209.440 | 10 | 1.763 | 0 |
| | 6 | 1291.750 | 1201.250 | 10 | 0.658 | 0 |
| | 7 | 1291.750 | 1317.490 | 13 | 1.940 | 0 |
| | 8 | 1291.750 | 1280.970 | 12 | 1.298 | 0 |
| 3 | 1 | 626.248 | 2212.000 | 16 | 1.466 | 0 |
| | 2 | 626.248 | 2290.040 | 17 | 1.136 | 0 |
| | 3 | 626.248 | 1929.910 | 11 | 3.373 | 0 |
| | 4 | 626.248 | 2096.580 | 15 | 1.734 | 0 |
| | 5 | 626.248 | 2226.780 | 16 | 1.419 | 0 |
| | 6 | 626.248 | 2233.580 | 16 | 0.925 | 0 |
| | 7 | 626.248 | 2243.450 | 16 | 2.416 | 0 |
| | 8 | 626.248 | 1972.290 | 13 | 1.446 | 0 |
| 4 | 1 | 380.253 | 2143.210 | 19 | 0.361 | 0 |
| | 2 | 380.253 | 2284.210 | 23 | 0.186 | 0 |
| | 3 | 380.253 | 2055.710 | 19 | 0.354 | 0 |
| | 4 | 380.253 | 2284.210 | 23 | 0.257 | 0 |
| | 5 | 380.253 | 2130.440 | 19 | 0.359 | 0 |
| | 6 | 380.253 | 2284.210 | 23 | 0.208 | 0 |
| | 7 | 380.253 | 2017.600 | 18 | 0.559 | 0 |
| | 8 | 380.253 | 2284.210 | 23 | 0.245 | 0 |
| 5 | 1 | 1300.440 | 1718.170 | 15 | 2.452 | 0 |
| | 2 | 1300.440 | 1779.200 | 17 | 0.488 | 0 |
| | 3 | 1300.440 | 1588.040 | 14 | 2.743 | 0 |
| | 4 | 1300.440 | 1536.300 | 13 | 1.113 | 0 |
| | 5 | 1300.440 | 1867.430 | 19 | 3.199 | 0 |
| | 6 | 1300.440 | 1895.520 | 20 | 0.589 | 0 |
| | 7 | 1300.440 | 1722.290 | 16 | 1.878 | 0 |
| | 8 | 1300.440 | 1779.140 | 17 | 0.906 | 0 |
| 6 | 1 | 1075.210 | 2722.280 | 19 | 1.056 | 0 |
| | 2 | 1075.210 | 2664.650 | 19 | 0.491 | 0 |
| | 3 | 1075.210 | 2640.650 | 17 | 2.073 | 0 |
| | 4 | 1075.210 | 2751.470 | 19 | 0.840 | 0 |
| | 5 | 1075.210 | 2722.280 | 19 | 1.202 | 0 |
| | 6 | 1075.210 | 2613.880 | 17 | 0.640 | 0 |
| | 7 | 1075.210 | 2753.970 | 19 | 2.560 | 0 |

| | | | | | | |
|----|---|----------|----------|----|--------|---|
| | 8 | 1075.210 | 2572.130 | 17 | 1.058 | 0 |
| 7 | 1 | 318.032 | 1539.090 | 21 | 0.653 | 0 |
| | 2 | 318.032 | 1426.450 | 17 | 0.348 | 0 |
| | 3 | 318.032 | 1542.540 | 23 | 0.790 | 0 |
| | 4 | 318.032 | 1667.970 | 20 | 0.335 | 0 |
| | 5 | 318.032 | 1533.620 | 19 | 0.535 | 0 |
| | 6 | 318.032 | 1602.480 | 23 | 0.356 | 0 |
| | 7 | 318.032 | 1433.390 | 20 | 0.852 | 0 |
| | 8 | 318.032 | 1667.970 | 20 | 0.313 | 0 |
| 8 | 1 | 741.325 | 2715.610 | 18 | 0.682 | 0 |
| | 2 | 741.325 | 2656.260 | 20 | 0.411 | 0 |
| | 3 | 741.325 | 2492.650 | 17 | 0.874 | 0 |
| | 4 | 741.325 | 2584.550 | 19 | 0.488 | 0 |
| | 5 | 741.325 | 2758.650 | 17 | 0.700 | 0 |
| | 6 | 741.325 | 2683.950 | 20 | 0.458 | 0 |
| | 7 | 741.325 | 2655.760 | 16 | 0.931 | 0 |
| | 8 | 741.325 | 2743.940 | 21 | 0.564 | 0 |
| 9 | 1 | 741.325 | 3187.740 | 26 | 0.550 | 1 |
| | 2 | 741.325 | 3539.370 | 29 | 0.280 | 1 |
| | 3 | 741.325 | 3419.050 | 30 | 0.878 | 3 |
| | 4 | 741.325 | 3539.370 | 29 | 0.452 | 1 |
| | 5 | 741.325 | 3373.770 | 30 | 0.356 | 2 |
| | 6 | 741.325 | 3539.370 | 29 | 0.276 | 1 |
| | 7 | 741.325 | 3446.020 | 30 | 0.738 | 3 |
| | 8 | 741.325 | 3231.050 | 30 | 0.347 | 4 |
| 10 | 1 | 373.718 | 2609.370 | 17 | 2.065 | 0 |
| | 2 | 373.718 | 2541.690 | 17 | 0.846 | 0 |
| | 3 | 373.718 | 2738.940 | 20 | 1.104 | 0 |
| | 4 | 373.718 | 2542.920 | 16 | 1.106 | 0 |
| | 5 | 373.718 | 2445.930 | 15 | 1.108 | 0 |
| | 6 | 373.718 | 2542.470 | 17 | 1.111 | 0 |
| | 7 | 373.718 | 2721.510 | 20 | 1.110 | 0 |
| | 8 | 373.718 | 2488.150 | 15 | 1.106 | 0 |
| 11 | 1 | 373.718 | 3361.170 | 29 | 1.401 | 0 |
| | 2 | 373.718 | 3332.610 | 28 | 0.923 | 0 |
| | 3 | 373.718 | 3298.900 | 30 | 1.669 | 1 |
| | 4 | 373.718 | 3098.270 | 28 | 0.892 | 0 |
| | 5 | 373.718 | 3311.170 | 29 | 1.299 | 0 |
| | 6 | 373.718 | 3246.990 | 27 | 0.745 | 0 |
| | 7 | 373.718 | 3134.840 | 29 | 1.427 | 0 |
| | 8 | 373.718 | 3098.270 | 28 | 0.889 | 0 |
| 12 | 1 | 907.221 | 3247.960 | 21 | 5.861 | 0 |
| | 2 | 907.221 | 3297.960 | 24 | 2.943 | 0 |
| | 3 | 907.221 | 3165.240 | 22 | 11.501 | 0 |
| | 4 | 907.221 | 3417.710 | 23 | 2.467 | 0 |
| | 5 | 907.221 | 3233.490 | 22 | 10.409 | 0 |
| | 6 | 907.221 | 3385.230 | 23 | 3.164 | 0 |
| | 7 | 907.221 | 3157.770 | 26 | 6.604 | 0 |
| | 8 | 907.221 | 3345.680 | 25 | 3.910 | 0 |
| 13 | 1 | 907.221 | 3501.750 | 30 | 5.829 | 7 |

| | | | | | | |
|----|---|----------|----------|----|--------|----|
| | 2 | 907.221 | 3246.640 | 30 | 2.531 | 11 |
| | 3 | 907.221 | 3370.010 | 30 | 12.364 | 5 |
| | 4 | 907.221 | 3492.330 | 30 | 2.983 | 8 |
| | 5 | 907.221 | 3202.480 | 30 | 9.001 | 15 |
| | 6 | 907.221 | 3457.580 | 30 | 2.807 | 6 |
| | 7 | 907.221 | 3443.550 | 30 | 6.624 | 4 |
| | 8 | 907.221 | 3427.800 | 30 | 3.927 | 8 |
| 14 | 1 | 794.224 | 3613.180 | 20 | 10.760 | 0 |
| | 2 | 794.224 | 3441.990 | 18 | 2.735 | 0 |
| | 3 | 794.224 | 3245.690 | 19 | 9.699 | 0 |
| | 4 | 794.224 | 3227.310 | 16 | 2.673 | 0 |
| | 5 | 794.224 | 3295.720 | 16 | 7.183 | 0 |
| | 6 | 794.224 | 3343.480 | 17 | 1.991 | 0 |
| | 7 | 794.224 | 3153.770 | 18 | 9.380 | 0 |
| | 8 | 794.224 | 3333.340 | 18 | 2.545 | 0 |
| 15 | 1 | 794.224 | 3939.700 | 30 | 5.264 | 18 |
| | 2 | 794.224 | 3857.040 | 30 | 2.083 | 15 |
| | 3 | 794.224 | 3740.190 | 30 | 4.755 | 19 |
| | 4 | 794.224 | 3698.240 | 30 | 2.738 | 19 |
| | 5 | 794.224 | 3774.590 | 30 | 5.096 | 19 |
| | 6 | 794.224 | 3811.090 | 30 | 1.820 | 15 |
| | 7 | 794.224 | 4049.450 | 30 | 5.190 | 13 |
| | 8 | 794.224 | 3698.240 | 30 | 2.282 | 19 |
| 16 | 1 | 800.524 | 2568.280 | 27 | 18.727 | 0 |
| | 2 | 800.524 | 2471.550 | 26 | 2.699 | 0 |
| | 3 | 800.524 | 2552.150 | 30 | 8.913 | 1 |
| | 4 | 800.524 | 2451.480 | 25 | 4.831 | 0 |
| | 5 | 800.524 | 2347.080 | 24 | 14.909 | 0 |
| | 6 | 800.524 | 2127.090 | 21 | 3.508 | 0 |
| | 7 | 800.524 | 2272.980 | 23 | 8.549 | 0 |
| | 8 | 800.524 | 2034.830 | 17 | 7.013 | 0 |
| 17 | 1 | 800.524 | 2457.930 | 30 | 13.766 | 15 |
| | 2 | 800.524 | 2560.210 | 30 | 2.373 | 11 |
| | 3 | 800.524 | 2365.720 | 30 | 7.390 | 18 |
| | 4 | 800.524 | 2537.120 | 30 | 3.739 | 19 |
| | 5 | 800.524 | 2600.020 | 30 | 15.075 | 10 |
| | 6 | 800.524 | 2280.950 | 30 | 3.439 | 19 |
| | 7 | 800.524 | 2217.760 | 30 | 8.430 | 19 |
| | 8 | 800.524 | 2160.650 | 30 | 3.285 | 19 |
| 18 | 1 | 1085.520 | 3049.400 | 23 | 19.680 | 0 |
| | 2 | 1085.520 | 2973.010 | 18 | 3.720 | 0 |
| | 3 | 1085.520 | 3207.350 | 26 | 30.103 | 0 |
| | 4 | 1085.520 | 2985.340 | 22 | 5.410 | 0 |
| | 5 | 1085.520 | 3181.620 | 25 | 13.316 | 0 |
| | 6 | 1085.520 | 3272.900 | 23 | 3.994 | 0 |
| | 7 | 1085.520 | 2970.620 | 19 | 15.479 | 0 |
| | 8 | 1085.520 | 2905.650 | 20 | 4.976 | 0 |
| 19 | 1 | 1085.520 | 4026.650 | 30 | 7.007 | 1 |
| | 2 | 1085.520 | 3777.830 | 27 | 2.774 | 0 |
| | 3 | 1085.520 | 3921.510 | 30 | 7.001 | 1 |

| | | | | | | |
|----|---|----------|----------|----|---------|----|
| | 4 | 1085.520 | 3609.900 | 29 | 4.639 | 0 |
| | 5 | 1085.520 | 3812.420 | 30 | 5.388 | 1 |
| | 6 | 1085.520 | 3575.040 | 30 | 2.657 | 4 |
| | 7 | 1085.520 | 3824.770 | 30 | 5.924 | 3 |
| | 8 | 1085.520 | 3737.370 | 30 | 2.751 | 4 |
| 20 | 1 | 1067.650 | 3060.290 | 23 | 9.471 | 0 |
| | 2 | 1067.650 | 2732.910 | 23 | 1.792 | 0 |
| | 3 | 1067.650 | 3348.740 | 28 | 6.268 | 0 |
| | 4 | 1067.650 | 3047.680 | 25 | 2.649 | 0 |
| | 5 | 1067.650 | 3022.750 | 22 | 5.325 | 0 |
| | 6 | 1067.650 | 2831.400 | 23 | 1.353 | 0 |
| | 7 | 1067.650 | 3098.700 | 25 | 5.910 | 0 |
| | 8 | 1067.650 | 2662.990 | 22 | 1.459 | 0 |
| 21 | 1 | 1067.650 | 3128.740 | 30 | 9.231 | 9 |
| | 2 | 1067.650 | 3353.550 | 30 | 1.348 | 8 |
| | 3 | 1067.650 | 3450.760 | 30 | 6.326 | 7 |
| | 4 | 1067.650 | 3285.260 | 30 | 2.582 | 7 |
| | 5 | 1067.650 | 3386.310 | 30 | 5.417 | 7 |
| | 6 | 1067.650 | 3287.070 | 30 | 1.194 | 9 |
| | 7 | 1067.650 | 3428.970 | 30 | 5.396 | 6 |
| | 8 | 1067.650 | 3385.630 | 30 | 1.466 | 6 |
| 22 | 1 | 1377.820 | 4523.010 | 30 | 4.112 | 23 |
| | 2 | 1377.820 | 4624.700 | 30 | 0.917 | 24 |
| | 3 | 1377.820 | 4154.660 | 30 | 3.878 | 39 |
| | 4 | 1377.820 | 4267.130 | 30 | 1.143 | 37 |
| | 5 | 1377.820 | 4548.130 | 30 | 3.360 | 32 |
| | 6 | 1377.820 | 4728.460 | 30 | 1.439 | 10 |
| | 7 | 1377.820 | 4568.100 | 30 | 6.360 | 24 |
| | 8 | 1377.820 | 4528.850 | 30 | 1.384 | 22 |
| 23 | 1 | 680.152 | 3537.660 | 30 | 1.843 | 15 |
| | 2 | 680.152 | 3754.800 | 30 | 0.814 | 9 |
| | 3 | 680.152 | 3788.040 | 30 | 1.344 | 9 |
| | 4 | 680.152 | 3614.360 | 30 | 0.853 | 16 |
| | 5 | 680.152 | 3604.430 | 30 | 1.423 | 12 |
| | 6 | 680.152 | 3641.760 | 30 | 0.900 | 14 |
| | 7 | 680.152 | 3459.530 | 30 | 132.656 | 18 |
| | 8 | 680.152 | 3690.060 | 30 | 0.791 | 14 |
| 24 | 1 | 1297.970 | 2854.610 | 28 | 16.724 | 0 |
| | 2 | 1297.970 | 2842.980 | 27 | 1.964 | 0 |
| | 3 | 1297.970 | 2896.050 | 30 | 28.884 | 3 |
| | 4 | 1297.970 | 2960.210 | 30 | 2.673 | 0 |
| | 5 | 1297.970 | 2939.760 | 30 | 20.327 | 3 |
| | 6 | 1297.970 | 2875.420 | 30 | 1.960 | 3 |
| | 7 | 1297.970 | 3060.340 | 29 | 29.347 | 0 |
| | 8 | 1297.970 | 2828.840 | 30 | 3.830 | 3 |
| 25 | 1 | 1297.970 | 3167.000 | 30 | 17.179 | 2 |
| | 2 | 1297.970 | 3051.520 | 30 | 2.079 | 11 |
| | 3 | 1297.970 | 3033.590 | 30 | 28.988 | 12 |
| | 4 | 1297.970 | 3082.850 | 30 | 3.845 | 2 |
| | 5 | 1297.970 | 3084.740 | 30 | 20.299 | 11 |

| | | | | | | |
|----|---|----------|----------|----|---------|----|
| | 6 | 1297.970 | 2884.950 | 30 | 1.956 | 12 |
| | 7 | 1297.970 | 3105.170 | 30 | 29.271 | 11 |
| | 8 | 1297.970 | 3106.500 | 30 | 3.789 | 3 |
| 26 | 1 | 1220.950 | 2134.210 | 23 | 96.893 | 0 |
| | 2 | 1220.950 | 2193.180 | 22 | 133.130 | 0 |
| | 3 | 1220.950 | 2193.180 | 22 | 130.224 | 0 |
| | 4 | 1220.950 | 2355.910 | 25 | 12.280 | 0 |
| | 5 | 1220.950 | 2739.790 | 27 | 62.590 | 0 |
| | 6 | 1220.950 | 2473.870 | 25 | 9.174 | 0 |
| | 7 | 1220.950 | 2226.520 | 22 | 99.228 | 0 |
| | 8 | 1220.950 | 2598.650 | 29 | 11.805 | 0 |
| 27 | 1 | 1220.950 | 3136.890 | 30 | 35.945 | 6 |
| | 2 | 1220.950 | 3342.570 | 30 | 1.105 | 6 |
| | 3 | 1220.950 | 3052.600 | 30 | 48.368 | 3 |
| | 4 | 1220.950 | 3038.170 | 28 | 3.620 | 0 |
| | 5 | 1220.950 | 3105.160 | 30 | 22.294 | 8 |
| | 6 | 1220.950 | 2987.350 | 30 | 2.838 | 7 |
| | 7 | 1220.950 | 3006.170 | 30 | 33.576 | 8 |
| | 8 | 1220.950 | 2992.030 | 29 | 3.522 | 6 |
| 28 | 1 | 3159.230 | 3925.670 | 30 | 79.156 | 21 |
| | 2 | 3159.230 | 4900.930 | 30 | 3.232 | 22 |
| | 3 | 3159.230 | 3932.880 | 30 | 67.332 | 20 |
| | 4 | 3159.230 | 4103.120 | 30 | 7.770 | 22 |
| | 5 | 3159.230 | 4167.330 | 30 | 21.266 | 21 |
| | 6 | 3159.230 | 4028.290 | 30 | 1.686 | 21 |
| | 7 | 3159.230 | 4253.630 | 30 | 18.631 | 17 |
| | 8 | 3159.230 | 4047.090 | 30 | 3.314 | 17 |
| 29 | 1 | 802.619 | 4176.090 | 27 | 17.404 | 0 |
| | 2 | 802.619 | 4133.420 | 30 | 2.730 | 0 |
| | 3 | 802.619 | 4183.210 | 30 | 13.265 | 3 |
| | 4 | 802.619 | 4205.750 | 30 | 3.296 | 0 |
| | 5 | 802.619 | 4236.640 | 30 | 3.441 | 2 |
| | 6 | 802.619 | 4236.640 | 30 | 1.899 | 0 |
| | 7 | 802.619 | 4262.760 | 30 | 13.048 | 2 |
| | 8 | 802.619 | 4115.770 | 30 | 4.827 | 3 |
| 30 | 1 | 410.932 | 4763.650 | 30 | 3.761 | 27 |
| | 2 | 410.932 | 4601.220 | 30 | 1.454 | 22 |
| | 3 | 410.932 | 4963.990 | 30 | 4.284 | 0 |
| | 4 | 410.932 | 4702.630 | 30 | 2.379 | 35 |
| | 5 | 410.932 | 4914.050 | 30 | 3.669 | 14 |
| | 6 | 410.932 | 4903.350 | 30 | 1.379 | 14 |
| | 7 | 410.932 | 4844.160 | 30 | 4.550 | 9 |
| | 8 | 410.932 | 4723.930 | 30 | 2.195 | 22 |

Appendix 3 – Time Windows for Problem Set 20

Table 3. Time Windows for Problem Set 20

| Pickup | Delivery | Pickup Min Time | Delivery Min Time | Pickup Max Time | Delivery Max Time | Service Time | Drop Time | Available Time |
|---------|----------|-----------------|-------------------|-----------------|-------------------|--------------|-----------|----------------|
| visit1 | visit2 | 0 | 0 | 336 | 464 | 10 | 10 | 0 |
| visit3 | visit4 | 0 | 0 | 350 | 433 | 10 | 10 | 0 |
| visit5 | visit6 | 0 | 0 | 344 | 417 | 10 | 10 | 0 |
| visit7 | visit8 | 0 | 0 | 321 | 415 | 10 | 10 | 0 |
| visit9 | visit10 | 0 | 0 | 303 | 427 | 10 | 10 | 0 |
| visit11 | visit12 | 0 | 0 | 301 | 469 | 10 | 10 | 0 |
| visit13 | visit14 | 0 | 0 | 341 | 435 | 10 | 10 | 0 |
| visit15 | visit16 | 0 | 0 | 328 | 426 | 10 | 10 | 0 |
| visit17 | visit18 | 0 | 0 | 314 | 402 | 10 | 10 | 0 |
| visit19 | visit20 | 0 | 0 | 311 | 415 | 10 | 10 | 0 |
| visit21 | visit22 | 0 | 0 | 269 | 465 | 10 | 10 | 0 |
| visit23 | visit24 | 0 | 0 | 245 | 408 | 10 | 10 | 0 |
| visit25 | visit26 | 0 | 0 | 273 | 457 | 10 | 10 | 0 |
| visit27 | visit28 | 0 | 0 | 200 | 400 | 10 | 10 | 0 |
| visit29 | visit30 | 0 | 0 | 267 | 453 | 10 | 10 | 0 |
| visit31 | visit32 | 0 | 0 | 260 | 324 | 10 | 10 | 0 |
| visit33 | visit34 | 0 | 0 | 231 | 232 | 10 | 10 | 0 |
| visit35 | visit36 | 0 | 0 | 208 | 357 | 10 | 10 | 0 |
| visit37 | visit38 | 0 | 0 | 215 | 245 | 10 | 10 | 0 |
| visit39 | visit40 | 0 | 0 | 198 | 347 | 10 | 10 | 0 |
| visit41 | visit42 | 0 | 0 | 235 | 299 | 10 | 10 | 0 |
| visit43 | visit44 | 0 | 0 | 192 | 304 | 10 | 10 | 0 |
| visit45 | visit46 | 0 | 0 | 191 | 288 | 10 | 10 | 0 |
| visit47 | visit48 | 0 | 0 | 271 | 211 | 10 | 10 | 0 |
| visit49 | visit50 | 0 | 0 | 276 | 349 | 10 | 10 | 0 |
| visit51 | visit52 | 0 | 0 | 244 | 290 | 10 | 10 | 0 |
| visit53 | visit54 | 0 | 0 | 215 | 392 | 10 | 10 | 0 |
| visit55 | visit56 | 0 | 0 | 247 | 293 | 10 | 10 | 0 |
| visit57 | visit58 | 0 | 0 | 280 | 299 | 10 | 10 | 0 |
| visit59 | visit60 | 0 | 0 | 276 | 323 | 10 | 10 | 0 |
| visit61 | visit62 | 0 | 0 | 272 | 275 | 10 | 10 | 0 |
| visit63 | visit64 | 0 | 0 | 273 | 281 | 10 | 10 | 0 |
| visit65 | visit66 | 0 | 0 | 254 | 359 | 10 | 10 | 0 |
| visit67 | visit68 | 0 | 0 | 255 | 284 | 10 | 10 | 0 |
| visit69 | visit70 | 0 | 0 | 201 | 321 | 10 | 10 | 0 |
| visit71 | visit72 | 0 | 0 | 258 | 302 | 10 | 10 | 0 |
| visit73 | visit74 | 0 | 0 | 193 | 286 | 10 | 10 | 0 |
| visit75 | visit76 | 0 | 0 | 263 | 316 | 10 | 10 | 0 |
| visit77 | visit78 | 0 | 0 | 280 | 364 | 10 | 10 | 0 |
| visit79 | visit80 | 0 | 0 | 218 | 239 | 10 | 10 | 0 |
| visit81 | visit82 | 0 | 0 | 260 | 216 | 10 | 10 | 0 |
| visit83 | visit84 | 5 | 25 | 55 | 379 | 10 | 10 | 5 |
| visit85 | visit86 | 13 | 33 | 69 | 420 | 10 | 10 | 10 |
| visit87 | visit88 | 14 | 29 | 115 | 409 | 10 | 10 | 10 |
| visit89 | visit90 | 20 | 23 | 165 | 219 | 10 | 10 | 15 |
| visit91 | visit92 | 19 | 20 | 102 | 169 | 10 | 10 | 19 |

| | | | | | | | | |
|----------|----------|-----|-----|-----|-----|----|----|-----|
| visit93 | visit94 | 31 | 50 | 178 | 425 | 10 | 10 | 30 |
| visit95 | visit96 | 35 | 36 | 166 | 266 | 10 | 10 | 30 |
| visit97 | visit98 | 36 | 47 | 152 | 209 | 10 | 10 | 34 |
| visit99 | visit100 | 35 | 40 | 141 | 190 | 10 | 10 | 35 |
| visit101 | visit102 | 41 | 58 | 207 | 269 | 10 | 10 | 36 |
| visit103 | visit104 | 48 | 60 | 165 | 328 | 10 | 10 | 44 |
| visit105 | visit106 | 54 | 73 | 174 | 186 | 10 | 10 | 49 |
| visit107 | visit108 | 56 | 62 | 235 | 89 | 10 | 10 | 52 |
| visit109 | visit110 | 57 | 71 | 126 | 198 | 10 | 10 | 54 |
| visit111 | visit112 | 62 | 75 | 126 | 406 | 10 | 10 | 57 |
| visit113 | visit114 | 59 | 59 | 144 | 211 | 10 | 10 | 57 |
| visit115 | visit116 | 62 | 72 | 194 | 199 | 10 | 10 | 58 |
| visit117 | visit118 | 63 | 66 | 210 | 310 | 10 | 10 | 59 |
| visit119 | visit120 | 65 | 77 | 158 | 316 | 10 | 10 | 64 |
| visit121 | visit122 | 67 | 67 | 193 | 281 | 10 | 10 | 67 |
| visit123 | visit124 | 69 | 79 | 185 | 267 | 10 | 10 | 69 |
| visit125 | visit126 | 76 | 86 | 142 | 432 | 10 | 10 | 71 |
| visit127 | visit128 | 76 | 92 | 193 | 270 | 10 | 10 | 72 |
| visit129 | visit130 | 77 | 80 | 173 | 355 | 10 | 10 | 75 |
| visit131 | visit132 | 76 | 89 | 166 | 421 | 10 | 10 | 76 |
| visit133 | visit134 | 83 | 84 | 231 | 392 | 10 | 10 | 81 |
| visit135 | visit136 | 87 | 97 | 220 | 297 | 10 | 10 | 84 |
| visit137 | visit138 | 96 | 106 | 247 | 238 | 10 | 10 | 96 |
| visit139 | visit140 | 103 | 110 | 208 | 420 | 10 | 10 | 100 |
| visit141 | visit142 | 104 | 105 | 239 | 256 | 10 | 10 | 104 |
| visit143 | visit144 | 105 | 112 | 166 | 378 | 10 | 10 | 105 |
| visit145 | visit146 | 106 | 116 | 199 | 279 | 10 | 10 | 105 |
| visit147 | visit148 | 114 | 130 | 221 | 443 | 10 | 10 | 111 |
| visit149 | visit150 | 120 | 132 | 131 | 458 | 10 | 10 | 118 |
| visit151 | visit152 | 124 | 126 | 184 | 249 | 10 | 10 | 121 |
| visit153 | visit154 | 128 | 135 | 190 | 438 | 10 | 10 | 126 |
| visit155 | visit156 | 128 | 138 | 239 | 276 | 10 | 10 | 126 |
| visit157 | visit158 | 130 | 147 | 236 | 470 | 10 | 10 | 130 |
| visit159 | visit160 | 131 | 147 | 267 | 310 | 10 | 10 | 130 |
| visit161 | visit162 | 138 | 153 | 249 | 349 | 10 | 10 | 136 |
| visit163 | visit164 | 145 | 156 | 265 | 364 | 10 | 10 | 141 |
| visit165 | visit166 | 145 | 153 | 245 | 345 | 10 | 10 | 142 |
| visit167 | visit168 | 144 | 150 | 192 | 250 | 10 | 10 | 142 |
| visit169 | visit170 | 150 | 170 | 312 | 272 | 10 | 10 | 145 |
| visit171 | visit172 | 150 | 156 | 276 | 346 | 10 | 10 | 145 |
| visit173 | visit174 | 149 | 163 | 298 | 317 | 10 | 10 | 147 |
| visit175 | visit176 | 153 | 160 | 200 | 427 | 10 | 10 | 149 |
| visit177 | visit178 | 151 | 171 | 266 | 396 | 10 | 10 | 150 |
| visit179 | visit180 | 151 | 157 | 258 | 334 | 10 | 10 | 150 |
| visit181 | visit182 | 153 | 160 | 314 | 395 | 10 | 10 | 152 |
| visit183 | visit184 | 156 | 162 | 198 | 238 | 10 | 10 | 154 |
| visit185 | visit186 | 161 | 162 | 267 | 388 | 10 | 10 | 157 |
| visit187 | visit188 | 159 | 176 | 296 | 470 | 10 | 10 | 159 |
| visit189 | visit190 | 163 | 181 | 237 | 284 | 10 | 10 | 161 |
| visit191 | visit192 | 167 | 168 | 319 | 344 | 10 | 10 | 164 |
| visit193 | visit194 | 169 | 176 | 209 | 271 | 10 | 10 | 164 |
| visit195 | visit196 | 175 | 191 | 286 | 444 | 10 | 10 | 170 |

| | | | | | | | | |
|----------|----------|-----|-----|-----|-----|----|----|-----|
| visit197 | visit198 | 170 | 175 | 327 | 390 | 10 | 10 | 170 |
| visit199 | visit200 | 177 | 193 | 249 | 310 | 10 | 10 | 177 |
| visit201 | visit202 | 181 | 185 | 347 | 400 | 10 | 10 | 177 |
| visit203 | visit204 | 178 | 187 | 195 | 312 | 10 | 10 | 178 |
| visit205 | visit206 | 183 | 190 | 267 | 433 | 10 | 10 | 180 |
| visit207 | visit208 | 185 | 203 | 335 | 315 | 10 | 10 | 180 |
| visit209 | visit210 | 183 | 183 | 321 | 397 | 10 | 10 | 183 |
| visit211 | visit212 | 188 | 195 | 288 | 385 | 10 | 10 | 187 |
| visit213 | visit214 | 196 | 210 | 260 | 338 | 10 | 10 | 191 |
| visit215 | visit216 | 199 | 200 | 348 | 412 | 10 | 10 | 195 |
| visit217 | visit218 | 203 | 223 | 287 | 359 | 10 | 10 | 199 |
| visit219 | visit220 | 199 | 201 | 368 | 409 | 10 | 10 | 199 |

Table 4. Validation of the model based on problem set number 20, trial 6

| |
|---|
| Vehicle1 Total cost = 148.569 Fixed cost = 0 Cost coefficients = Distance [1] |
| Route: depot → visit57 → visit33 → visit34 → visit65 → visit55 → visit58 → visit66 → visit56 → depot |
| Time: depot [0], delay [0] → travel [23.0217], wait [0] → visit57 [23.0217], delay [10] → travel [13.8924], wait [0] → visit33 [46.9142], delay [10] → travel [14.8661], wait [0] → visit34 [71.7802], delay [10] → travel [41.7732], wait [0] → visit65 [123.553], delay [10] → travel [4.47214], wait [0] → visit55 [138.026], delay [10] → travel [1], wait [0] → visit58 [149.026], delay [10] → travel [11.1803], wait [0] → visit66 [170.206], delay [10] → travel [1], wait [0] → visit56 [181.206], delay [10] → travel [37.3631], wait [0] → depot [228.569] <i>Transit Sum [228.569]</i> |
| Distance: depot [0] delay [0] → travel [23.0217], wait [0] → visit57 [23.0217], delay [0] → travel [13.8924], wait [0] → visit33 [36.9142], delay [0] → travel [14.8661], wait [0] → visit34 [51.7802], delay [0] → travel [41.7732], wait [0] → visit65 [93.5534], delay [0] → travel [4.47214], wait [0] → visit55 [98.0256], delay [0] → travel [1], wait [0] → visit58 [99.0256], delay [0] → travel [11.1803], wait [0] → visit66 [110.206], delay [0] → travel [1], wait [0] → visit56 [111.206] delay [0] → travel [37.3631], wait [0] → depot [148.569] <i>Transit Sum 148.569</i> |
| Vehicle2 Total cost = 192.05 Fixed cost = 0 Cost coefficients = Distance [1] |
| Route: depot → visit39 → visit1 → visit2 → visit71 → visit81 → visit119 → visit73 → visit74 → visit53 → visit82 → visit120 → visit205 → visit199 → visit200 → visit54 → visit72 → visit40 → visit206 → depot |
| Time: depot [0], delay [0] → travel [18.6011], wait [0] → visit39 [18.6011], delay [10] → travel [6.40312], wait [0] → visit1 [35.0042], delay [10] → travel [18.6815], wait [0] → visit2 [63.6857], delay [10] → travel [1], wait [0] → visit71 [74.6857], delay [10] → travel [9], wait [0] → visit81 [93.6857], delay [10] → travel [19.4165], wait [0] → visit119 [123.102], delay [10] → travel [12.3693], wait [0] → visit73 [145.472..145.472], delay [10] → travel [2], wait [0..1e-010] → visit74 [157.472..157.472], delay [10] → travel [1.41421], wait [0..1e-010] → visit 53 [168.886], delay [10] → travel [11.4018], wait [0] → visit82 [190.288], delay [10] → travel [3.60555], wait [0] → visit120 [203.893], delay [10] → travel [5.38516], wait [0] → visit205 [219.278], delay [10] → travel [8.06226], wait [0] → visit199 [237.34], delay [10] → travel [2.23607], wait [0] → visit200 [249.577], delay [10] → travel [10.0499], wait [0] → visit54 [269.626], delay [10] → travel [10.4403], wait [0] → visit72 [290.067], delay [10] → travel [29.5296], wait [0] → visit40 [329.596], delay [10] → travel [13.4536], wait [0] → visit206 [353.05], delay [10] → travel [9], wait [0] → depot [372.05] <i>Transit Sum [372.05]</i> |
| Distance: depot [0] delay [0] → travel [18.6011], wait [0] → visit39 [18.6011], delay [0] → travel |

| |
|---|
| <p>[6.40312], wait [0] → visit1 [25.0042], delay [0] → travel [18.6815], wait [0] → visit2 [43.6857], delay [0] → travel [1], wait [0] → visit71 [44.6857], delay [0] → travel [9], wait [0] → visit81 [53.6857], delay [0] → travel [19.4165], wait [0] → visit119 [73.1022], delay [0] → travel [12.3693], wait [0] → visit73 [85.4715], delay [0] → travel [2], wait [0] → visit74 [87.4715], delay [0] → travel [1.41421], wait [0] → visit53 [88.8858], delay [0] → travel [11.4018], wait [0] → visit82 [100.288], delay [0] → travel [3.60555], wait [0] → visit120 [103.893], delay [0] → travel [5.38516], wait [0] → visit205 [109.278], delay [0] → travel [8.06226], wait [0] → visit199 [117.34], delay [0] → travel [2.23607], wait [0] → visit200 [119.577], delay [0] → travel [10.0499], wait [0] → visit54 [129.626], delay [0] → travel [10.4403], wait [0] → visit72 [140.067], delay [0] → travel [29.5296], wait [0] → visit40 [169.596], delay [0] → travel [13.4536], wait [0] → visit206 [183.05], delay [0] → travel [9], wait[0] → depot [192.05]</p> <p><i>Transit Sum [192.05]</i></p> |
| <p>Vehicle3 Total cost = 130.477 Fixed cost = 0 Cost coefficients: Distance [1]</p> |
| <p>Route: depot → visit63 → visit69 → visit70 → visit79 → visit61 → visit49 → visit50 → visit64 → visit62 → visit185 → visit193 → visit194 → visit80 → visit186 → depot</p> |
| <p>Time: depot [0], delay [0] → travel [27.5862], wait [0] → visit63 [27.5862], delay [10] → travel [1.41421], wait [0] → visit69 [39.0004], delay [10] → travel [9.05539], wait [0] → visit70 [58.0558], delay [10] → travel [1.41421], wait [0] → visit79 [69.47], delay [10] → travel [9.21954], wait [0] → visit61 [88.6896], delay [10] → travel [5.83095], wait [0] → visit49 [104.521], delay [10] → travel [10.8167], wait [0] → visit50 [125.337], delay [10] → travel [5.09902], wait [0] → visit64 [140.436], delay [10] → travel [4], wait [0] → visit62 [154.436], delay [10] → travel [5.65685], wait [0] → visit185 [170.093], delay [10] → travel [5], wait [0] → visit193 [185.093], delay [10] → travel [2], wait [0] → visit194 [197.093], delay [10] → travel [7], wait [0] → visit80 [214.093], delay [10] → travel [13.0384], wait [0] → visit186 [237.131], delay [10] → travel [23.3452], wait [0] → depot [270.477]</p> <p><i>Transit Sum [270.477]</i></p> |
| <p>Distance: depot [0], delay [0] → travel [27.5862], wait [0] → visit63 [27.5862], delay [0] → travel [1.41421], wait [0] → visit69 [29.0004], delay [0] → travel [9.05539], wait [0] → visit70 [38.0558], delay [0] → travel [1.41421], wait [0] → visit79 [39.47], delay [0] → travel [9.21954], wait [0] → visit61 [48.6896], delay [0] → travel [5.83095], wait [0] → visit49 [54.5205], delay [0] → travel [10.8167], wait [0] → visit50 [65.3372], delay [0] → travel [5.09902], wait [0] → visit64 [70.4362], delay [0] → travel [4], wait [0] → visit62 [74.4362], delay [0] → travel [5.65685], wait [0] → visit185 [80.0931], delay [0] → travel [5], wait [0] → visit193 [85.0931], delay [0] → travel [2], wait [0] → visit194 [87.0931], delay [0] → travel [7], wait [0] → visit80 [94.0931], delay [0] → travel [13.0384], wait [0] → visit186 [107.131], delay [0] → travel [23.3452], wait [0] → depot [130.477]</p> <p><i>Transit Sum [130.477]</i></p> |
| <p>Vehicle4 Total cost = 88.3736 Fixed cost = 0 Cost coefficients: Distance [1]</p> |
| <p>Route: depot → visit13 → visit19 → visit9 → visit23 → visit17 → visit15 → visit14 → visit18 → visit20 → visit16 → visit10 → visit24 → depot</p> |
| <p>Time: depot [0], delay [0] → travel [22.0227], wait [0] → visit13 [22.0227], delay [10] → travel [3.16228], wait [0] → visit19 [35.185], delay [10] → travel [3.16228], wait [0] → visit9 [48.3473], delay [10] → travel [4.24264], wait [0] → visit23 [62.5899], delay [10] → travel [2.23607], wait [0] → visit17 [74.826], delay [10] → travel [6], wait [0] → visit15 [90.826], delay [10] → travel [1.41421], wait [0] → visit14 [102.24], delay [10] → travel [2.23607], wait [0] → visit18 [114.476], delay [10] → travel [4.24264], wait [0] → visit20 [128.719], delay [10] → travel [2.82843], wait [0] → visit16 [141.547], delay [10] → travel [3.16228], wait [0] → visit10 [154.71], delay [10] → travel [10.8167], wait [0] → visit24 [175.526], delay [10] → travel [22.8473], wait [0] → depot [208.374]</p> <p><i>Transit Sum [208.374]</i></p> |
| <p>Distance: depot [0], delay [0] → travel [22.0227], wait [0] → visit13 [22.0227], delay [0] → [4.24264], wait [0] → visit20 [48.7189], delay [0] → travel [2.82843], wait [0] → visit16 [51.5473], delay [0] → travel [3.16228], wait [0] → visit10 [54.7096], delay [0] → travel [10.8167], wait [0] → visit24 [65.5263], delay [0] → travel [22.8473], wait [0] → depot [88.3736]</p> |

| |
|---|
| <i>Transit Sum [88.3736]</i> |
| Vehicle5 Total cost = 208.919 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit29 → visit30 → visit41 → visit35 → visit7 → visit36 → visit67 → visit68 → visit31 → visit43 → visit42 → visit44 → visit5 → visit32 → visit215 → visit216 → visit6 → visit8 → depot |
| Time: depot [0], delay [0] → travel [7.07107], wait [0] → visit29 [7.07107], delay [10] → travel [1], wait [0] → visit30 [18.0711], delay [10] → travel [4], wait [0] → visit41 [32.0711], delay [10] → travel [10], wait [0] → visit35 [52.0711], delay [10] → travel [23], wait [0] → visit7 [85.0711], delay [10] → travel [2], wait [0] → visit36 [97.0711], delay [10] → travel [16.9706], wait [0] → visit67 [124.042], delay [10] → travel [8], wait [0] → visit68 [142.042], delay [10] → travel [4.24264], wait [0] → visit31 [156.284], delay [10] → travel [19.105], wait [0] → visit43 [185.389], delay [10] → travel [6.40312], wait [0] → visit42 [201.792], delay [10] → travel [12.1655], wait [0] → visit44 [223.958], delay [10] → travel [12.0416], wait [0] → visit5 [245.999], delay [10] → travel [10], wait [0] → visit32 [265.999], delay [10] → travel [18.1108], wait [0] → visit215 [294.11], delay [10] → travel [3.16228], wait [0] → visit216 [307.273], delay [10] → travel [9.84886], wait [0] → visit6 [327.121], delay [10] → travel [18.0278], wait [0] → visit8 [355.149], delay [10] → travel [23.7697], wait [0] → depot [388.919] |
| <i>Transit Sum [388.919]</i> |
| Distance: depot [0], delay [0] → travel [7.07107], wait [0] → visit29 [7.07107], delay [0] → travel [1], wait [0] → visit30 [8.07107], delay [0] → travel [4], wait [0] → visit41 [12.0711], delay [0] → travel [10], wait [0] → visit35 [22.0711] delay [0] → travel [23], wait [0] → visit7 [45.0711], delay [0] → travel [2], wait [0] → visit36 [47.0711], delay [0] → travel [16.9706], wait [0] → visit67 [64.0416], delay [0] → travel [8], wait [0] → visit68 [72.0416], delay [0] → travel [4.24264], wait [0] → visit31 [76.2843], delay [0] → travel [19.105], wait [0] → visit43 [95.3892], delay [0] → travel [6.40312], wait [0] → visit42 [101.792] delay [0] → travel [12.1655], wait [0] → visit44 [113.958], delay [0] → travel [12.0416], wait [0] → visit5 [125.999], delay [0] → travel [10], wait [0] → visit32 [135.999], delay [0] → travel [18.1108], wait [0] → visit215 [154.11], delay [0] → travel [3.16228], wait [0] → visit216 [157.273], delay [0] → travel [9.84886], wait [0] → visit6 [167.121], delay [0] → travel [18.0278], wait [0] → visit8 [185.149] delay [0] → travel [23.7697], wait [0] → depot [208.919] |
| <i>Transit Sum [208.919]</i> |
| Vehicle6 Total cost = 108.338 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit83 → visit89 → visit99 → visit85 → visit109 → visit111 → visit110 → visit101 → visit100 → visit102 → visit93 → visit86 → visit94 → visit90 → visit84 → visit112 → depot |
| Time: depot [0], delay [0] → travel [7.61577], wait [0] → visit83 [7.61577], delay [10] → travel [6.08276], wait [0] → visit89 [23.6985], delay [10] → travel [9], wait [0] → visit99 [42.6985], delay [10] → travel [6], wait [0] → visit85 [58.6985], delay [10] → travel [5.83095], wait [0] → visit109 [74.5295], delay [10] → travel [5.09902], wait [0] → visit111 [89.6285], delay [10] → travel [4.12311], wait [0] → visit110 [103.752], delay [10] → travel [3.16228], wait [0] → visit101 [116.914], delay [10] → travel [7], wait [0] → visit100 [133.914], delay [10] → travel [4], wait [0] → visit102 [147.914], delay [10] → travel [2], wait [0] → visit93 [159.914], delay [10] → travel [4], wait [0] → visit86 [173.914], delay [10] → travel [0], wait [0] → visit94 [183.914], delay [10] → travel [17.72], wait [0] → visit90 [211.634], delay [10] → travel [2.23607], wait [0] → visit84 [223.87], delay [10] → travel [6.08276], wait [0] → visit112 [239.953], delay [10] → travel [18.3848], wait [0] → depot [268.338] |
| <i>Transit Sum [268.338]</i> |
| Distance: depot [0], delay [0] → travel [7.61577], wait [0] → visit83 [7.61577], delay [0] → travel [6.08276], wait [0] → visit89 [13.6985] delay [0] → travel [9], wait [0] → visit99 [22.6985], delay [0] → travel [6], wait [0] → visit85 [28.6985], delay [0] → travel [5.83095], wait [0] → visit109 [34.5295], delay [0] → travel [5.09902], wait [0] → visit111 [39.6285], delay [0] → travel [4.12311], wait [0] → visit110 [43.7516], delay [0] → travel [3.16228], wait [0] → visit101 [46.9139], delay [0] → travel [7], wait [0] → visit100 [53.9139], delay [0] → travel [4], wait [0] → visit102 [57.9139], delay [0] → travel [2], wait [0] → visit93 [59.9139], delay [0] → travel [4], wait [0] → visit86 [63.9139], delay [0] → |

| |
|--|
| travel [0], wait [0] → visit94 [63.9139] delay [0] → travel [17.72], wait [0] → visit90 [81.6339] delay [0] → travel [2.23607], wait [0] → visit84 [83.87], delay [0] → travel [6.08276], wait [0] → visit112 [89.9528] delay [0] → travel [18.3848], wait [0] → depot [108.338] <i>Transit Sum [108.338]</i> |
| Vehicle7 Total cost = 93.8076 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit91 → visit11 → visit21 → visit25 → visit26 → visit12 → visit22 → visit131 → visit92 → visit179 → visit180 → visit132 → depot |
| Time: depot [0], delay [0] → travel [19.9249], wait [0] → visit91 [19.9249], delay [10] → travel [11.4018], wait [0] → visit11 [41.3266], delay [10] → travel [1.41421], wait [0] → visit21 [52.7408], delay [10] → travel [2.23607], wait [0] → visit25 [64.9769], delay [10] → travel [1], wait [0] → visit26 [75.9769], delay [10] → travel [9.05539], wait [0] → visit12 [95.0323], delay [10] → travel [4.47214], wait [0] → visit22 [109.504], delay [10] → travel [6.40312], wait [0] → visit131 [125.908], delay [10] → travel [12.8062], wait [0] → visit92 [148.714], delay [10] → travel [8.544], wait [0] → visit179 [167.258], delay [10] → travel [4], wait [0] → visit180 [181.258], delay [10] → travel [3.60555], wait [0] → visit132 [194.863], delay [10] → travel [8.94427], wait [0] → depot [213.808] <i>Transit Sum [213.808]</i> |
| Distance: depot [0], delay [0] → travel [19.9249], wait [0] → visit91 [19.9249], delay [0] → travel [11.4018], wait [0] → visit11 [31.3266], delay [0] → travel [1.41421], wait [0] → visit21 [32.7408], delay [0] → travel [2.23607], wait [0] → visit25 [34.9769], delay [0] → travel [1], wait [0] → visit26 [35.9769], delay [0] → travel [9.05539], wait [0] → visit12 [45.0323], delay [0] → travel [4.47214], wait [0] → visit22 [49.5044], delay [0] → travel [6.40312], wait [0] → visit131 [55.9075], delay [0] → travel [12.8062], wait [0] → visit92 [68.7138], delay [0] → travel [8.544], wait [0] → visit179 [77.2578], delay [0] → travel [4], wait [0] → visit180 [81.2578], delay [0] → travel [3.60555], wait [0] → visit132 [84.8633] delay [0] → travel [8.94427], wait [0] → depot [93.8076] <i>Transit Sum 93.8076</i> |
| Vehicle8 Total cost = 167.074 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit103 → visit95 → visit127 → visit115 → visit116 → visit167 → visit96 → visit104 → visit168 → visit128 → depot |
| Time: depot [0], delay [0] → travel [12.083], wait [0] → visit103 [48], delay [10] → travel [4.12311], wait [0] → visit95 [62.1231], delay [10] → travel [26.9258], wait [0] → visit127 [99.0489], delay [10] → travel [18.7883], wait [0] → visit115 [127.837], delay [10] → travel [5], wait [0] → visit116 [142.837], delay [10] → travel [16.5529], wait [0] → visit167 [169.39], delay [10] → travel [16.9706], wait [0] → visit96 [196.361], delay [10] → travel [2.23607], wait [0] → visit104 [208.597], delay [10] → travel [16.7631], wait [0] → visit168 [235.36], delay [10] → travel [22.1359], wait [0] → visit128 [267.496], delay [10] → travel [25.4951], wait [0] → depot [302.991] <i>Transit Sum 302.991</i> |
| Distance: depot [0], delay [0] → travel [12.083], wait [0] → visit103 [12.083] delay [0] → travel [4.12311], wait [0] → visit95 [16.2062], delay [0] → travel [26.9258], wait [0] → visit127 [43.132], delay [0] → travel [18.7883], wait [0] → visit115 [61.9203], delay [0] → travel [5], wait [0] → visit116 [66.9203], delay [0] → travel [16.5529], wait [0] → visit167 [83.4732], delay [0] → travel [16.9706], wait [0] → visit96 [100.444], delay [0] → travel [2.23607], wait [0] → visit104 [102.68], delay [0] → travel [16.7631], wait [0] → visit168 [119.443] delay [0] → travel [22.1359], wait [0] → visit128 [141.579] delay [0] → travel [25.4951], wait [0] → depot [167.074] <i>Transit Sum [167.074]</i> |
| Vehicle9 Total cost = 202.768 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit107 → visit108 → visit171 → visit173 → visit175 → visit172 → visit219 → visit177 → visit187 → visit188 → visit174 → visit176 → visit3 → visit220 → visit178 → visit4 → depot |

| |
|--|
| <p>Time: depot [0], delay [0] → travel [20.6155], wait [0] → visit107 [56], delay [10] → travel [4.47214], wait [0] → visit108 [70.4721], delay [10] → travel [12.53], wait [56.9979] → visit171 [150], delay [10] → travel [3], wait [0] → visit173 [163], delay [10] → travel [5.65685], wait [0] → visit175 [178.657], delay [10] → travel [2.23607], wait [0] → visit172 [190.893], delay [10] → travel [10.4403], wait [0] → visit219 [211.333], delay [10] → travel [25.0799], wait [0] → visit177 [246.413], delay [10] → travel [4.47214], wait [0] → visit187 [260.885], delay [10] → travel [3.60555], wait [0] → visit188 [274.491], delay [10] → travel [14.8661], wait [0] → visit174 [299.357], delay [10] → travel [12.083], wait [0] → visit176 [321.44], delay [10] → travel [5.38516], wait [0] → visit3 [336.825], delay [10] → travel [21.2132], wait [0] → visit220 [368.038], delay [10] → travel [6.08276], wait [0] → visit178 [384.121], delay [10] → travel [17.0294], wait [0] → visit4 [411.15], delay [10] → travel [34], wait [0] → depot [455.15]</p> <p><i>Transit Sum 455.15</i></p> |
| <p>Distance: depot [0] delay [0] → travel [20.6155], wait [0] → visit107 [20.6155], delay [0] → travel [4.47214], wait [0] → visit108 [25.0877], delay [0] → travel [12.53], wait [0] → visit171 [37.6176], delay [0] → travel [3], wait [0] → visit173 [40.6176], delay [0] → travel [5.65685], wait [0] → visit175 [46.2745], delay [0] → travel [2.23607], wait [0] → visit172 [48.5106], delay [0] → travel [10.4403], wait [0] → visit219 [58.9509], delay [0] → travel [25.0799], wait [0] → visit177 [84.0307], delay [0] → travel [4.47214], wait [0] → visit187 [88.5029], delay [0] → travel [3.60555], wait [0] → visit188 [92.1084], delay [0] → travel [14.8661], wait [0] → visit174 [106.974], delay [0] → travel [12.083], wait [0] → visit176 [119.058], delay [0] → travel [5.38516], wait [0] → visit3 [124.443], delay [0] → travel [21.2132], wait [0] → visit220 [145.656], delay [0] → travel [6.08276], wait [0] → visit178 [151.739], delay [0] → travel [17.0294], wait [0] → visit4 [168.768], delay [0] → travel [34], wait [0] → depot [202.768]</p> <p><i>Transit Sum 202.768</i></p> |
| <p>Vehicle10 Total cost = 21.6734 Fixed cost = 0 Cost coefficients: Distance [1]</p> |
| <p>Route: depot → visit97 → visit98 → depot</p> |
| <p>Time: depot [0], delay [0] → travel [6], wait [30] → visit97 [36], delay [10] → travel [8.60233], wait [0] → visit98 [54.6023], delay [10] → travel [7.07107], wait [0] → depot [71.6734]</p> <p><i>Transit Sum 71.6734</i></p> |
| <p>Distance: depot [0] delay [0] → travel [6], wait [0] → visit97 [6], delay [0] → travel [8.60233], wait [0] → visit98 [14.6023], delay [0] → travel [7.07107], wait [0] → depot [21.6734], delay [0] → travel [0], wait [0]</p> <p><i>Transit Sum [21.6734]</i></p> |
| <p>Vehicle11 Total cost = 140.275 Fixed cost = 0 Cost coefficients: Distance [1]</p> |
| <p>Route: depot → visit117 → visit45 → visit27 → visit145 → visit143 → visit141 → visit144 → visit118 → visit146 → visit142 → visit46 → visit28 → depot</p> |
| <p>Time: depot [0], delay [0.471886] → travel [15.8114], wait [0] → visit117 [63], delay [10] → travel [12.2066], wait [0] → visit45 [85.2066], delay [10] → travel [22.1359], wait [0] → visit27 [117.342], delay [10] → travel [14.3178], wait [0] → visit145 [141.66], delay [10] → travel [9], wait [0] → visit143 [160.66], delay [10] → travel [4.12311], wait [0] → visit141 [174.783], delay [10] → travel [6.08276], wait [0] → visit144 [190.866], delay [10] → travel [5.09902], wait [0] → visit118 [205.965], delay [10] → travel [1], wait [0] → visit146 [216.965], delay [10] → travel [1], wait [0] → visit142 [227.965], delay [10] → travel [18.2483], wait [0] → visit46 [256.213], delay [10] → travel [12.6491], wait [0] → visit28 [278.863], delay [10] → travel [18.6011], wait [0] → depot [307.464]</p> <p><i>Transit Sum [307.464]</i></p> |
| <p>Distance: depot [0], delay [0] → travel [15.8114], wait [0] → visit117 [15.8114], delay [0] → travel [12.2066], wait [0] → visit45 [28.0179], delay [0] → travel [22.1359], wait [0] → visit27 [50.1539], delay [0] → travel [14.3178], wait [0] → visit145 [64.4717], delay [0] → travel [9], wait [0] → visit143 [73.4717], delay [0] → travel [4.12311], wait [0] → visit141 [77.5948], delay [0] → travel [6.08276], wait [0] → visit144 [83.6776], delay [0] → travel [5.09902], wait [0] → visit118 [88.7766], delay [0] → travel [1], wait [0] → visit146 [89.7766], delay [0] → travel [1], wait [0] → visit142 [90.7766], delay [0]</p> |

| |
|--|
| → travel [18.2483], wait [0] → visit46 [109.025], delay [0] → travel [12.6491], wait [0] → visit28 [121.674], delay [0] → travel [18.6011], wait [0] → depot [140.275] <i>Transit Sum [140.275]</i> |
| Vehicle12 Total cost = 122.024 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit105 → visit87 → visit129 → visit37 → visit113 → visit114 → visit106 → visit88 → visit130 → visit38 → depot |
| Time: depot [0], delay [0] → travel [26.2488], wait [0] → visit105 [54], delay [10] → travel [1], wait [0] → visit87 [65], delay [10] → travel [12.083], wait [0] → visit129 [87.083], delay [10] → travel [1], wait [0] → visit37 [98.083], delay [10] → travel [8], wait [0] → visit113 [116.083], delay [10] → travel [10.2956], wait [0] → visit114 [136.379], delay [10] → travel [8.94427], wait [0] → visit106 [155.323], delay [10] → travel [9.05539], wait [0] → visit88 [174.378], delay [10] → travel [21.1896], wait [0] → visit130 [205.568], delay [10] → travel [24.2074], wait [0] → visit38 [239.775], delay [10] → travel [0], wait [0] → depot [249.775] <i>Transit Sum [249.775]</i> |
| Distance: depot [0] delay [0] → travel [26.2488], wait [0] → visit105 [26.2488], delay [0] → travel [1], wait [0] → visit87 [27.2488], delay [0] → travel [12.083], wait [0] → visit129 [39.3319], delay [0] → travel [1], wait [0] → visit37 [40.3319], delay [0] → travel [8], wait [0] → visit113 [48.3319], delay [0] → travel [10.2956], wait [0] → visit114 [58.6275], delay [0] → travel [8.94427], wait [0] → visit106 [67.5718], delay [0] → travel [9.05539], wait [0] → visit88 [76.6271] delay [0] → travel [21.1896], wait [0] → visit130 [97.8168], delay [0] → travel [24.2074], wait [0] → visit38 [122.024], delay [0] → travel [0], wait [0] → depot [122.024] <i>Transit Sum [122.024]</i> |
| Vehicle13 Total cost = 202.49 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit121 → visit123 → visit122 → visit125 → visit135 → visit133 → visit134 → visit124 → visit136 → visit126 → depot |
| Time: depot [0], delay [0] → travel [48.8365], wait [18.1635] → visit121 [67], delay [10] → travel [3.16228], wait [0] → visit123 [80.1623], delay [10] → travel [24.1868], wait [0] → visit122 [114.349], delay [10] → travel [16.1555], wait [0] → visit125 [140.505], delay [10] → travel [22.1359], wait [0] → visit135 [172.64], delay [10] → travel [16.5529], wait [0] → visit133 [199.193], delay [10] → travel [10.4403], wait [0] → visit134 [219.634], delay [10] → travel [15], wait [0] → visit124 [244.634], delay [10] → travel [13.4536], wait [0] → visit136 [268.087], delay [10] → travel [4.12311], wait [0] → visit126 [282.21], delay [10] → travel [28.4429], wait [0] → depot [320.653], delay [0] → travel [0], wait [0] <i>Transit Sum [320.653]</i> |
| Distance: depot [0], delay [0] → travel [48.8365], wait [0] → visit121 [48.8365], delay [0] → travel [3.16228], wait [0] → visit123 [51.9987], delay [0] → travel [24.1868], wait [0] → visit122 [76.1855], delay [0] → travel [16.1555], wait [0] → visit125 [92.341], delay [0] → travel [22.1359], wait [0] → visit135 [114.477], delay [0] → travel [16.5529], wait [0] → visit133 [131.03], delay [0] → travel [10.4403], wait [0] → visit134 [141.47], delay [0] → travel [15], wait [0] → visit124 [156.47], delay [0] → travel [13.4536], wait [0] → visit136 [169.924], delay [0] → travel [4.12311], wait [0] → visit126 [174.047] delay [0] → travel [28.4429], wait [0] → depot [202.49], delay [0] → <i>Transit Sum [202.49]</i> |
| Vehicle14 Total cost = 90.7877 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit51 → visit59 → visit52 → visit60 → depot |
| Time: depot [0], delay [0] → travel [36.7151], wait [0] → visit51 [36.7151], delay [10] → travel [2.23607], wait [0] → visit59 [48.9512], delay [10] → travel [10.8167], wait [0] → visit52 [69.7678], delay [10] → travel [19.9249], wait [0] → visit60 [99.6927], delay [10] → travel [21.095], wait [0] → |

| |
|--|
| depot [130.788] <i>Transit Sum [130.788]</i> |
| Distance: depot [0], delay [0] → travel [36.7151], wait [0] → visit51 [36.7151] delay [0] → travel [2.23607], wait [0] → visit59 [38.9512], delay [0] → travel [10.8167], wait [0] → visit52 [49.7678], delay [0] → travel [19.9249], wait [0] → visit60 [69.6927], delay [0] → travel [21.095], wait [0] → depot [90.7877], delay [0] → travel [0], wait [0] <i>Transit Sum [90.7877]</i> |
| Vehicle15 Total cost = 72.7513 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit149 → visit159 → visit155 → visit151 → visit139 → visit153 → visit154 → visit150 → visit156 → visit152 → visit160 → visit140 → depot |
| Time: depot [0], delay [0] → travel [21.3776], wait [0] → visit149 [120] delay [10] → travel [7.2111], wait [0] → visit159 [137.211], delay [10] → travel [1], wait [0] → visit155 [148.211], delay [10] → travel [0], wait [0] → visit151 [158.211], delay [10] → travel [3], wait [0] → visit139 [171.211], delay [10] → travel [5], wait [0] → visit153 [186.211], delay [10] → travel [3.60555], wait [0] → visit154 [199.817], delay [10] → travel [4.12311], wait [0] → visit150 [213.94], delay [10] → travel [2.82843], wait [0] → visit156 [226.768], delay [10] → travel [3.60555], wait [0] → visit152 [240.374], delay [10] → travel [4], wait [0] → visit160 [254.374], delay [10] → travel [1], wait [0] → visit140 [265.374], delay [10] → travel [16], wait [0] → depot [291.374] <i>Transit Sum [291.374]</i> |
| Distance: depot [0], delay [0] → travel [21.3776], wait [0] → visit149 [21.3776], delay [0] → travel [7.2111], wait [0] → visit159 [28.5887], delay [0] → travel [1], wait [0] → visit155 [29.5887], delay [0] → travel [0], wait [0] → visit151 [29.5887], delay [0] → travel [3], wait [0] → visit139 [32.5887], delay [0] → travel [5], wait [0] → visit153 [37.5887], delay [0] → travel [3.60555], wait [0] → visit154 [41.1942], delay [0] → travel [4.12311], wait [0] → visit150 [45.3173], delay [0] → travel [2.82843], wait [0] → visit156 [48.1457], delay [0] → travel [3.60555], wait [0] → visit152 [51.7513], delay [0] → travel [4], wait [0] → visit160 [55.7513], delay [0] → travel [1], wait [0] → visit140 [56.7513], delay [0] → travel [16], wait [0] → depot [72.7513] <i>Transit Sum [72.7513]</i> |
| Vehicle16 Total cost = 42.4945 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit147 → visit137 → visit148 → visit138 → depot |
| Time: depot [0], delay [0] → travel [19.0263], wait [94.9737] → visit147 [114], delay [10] → travel [5.65685], wait [0] → visit137 [129.657], delay [10] → travel [0], wait [0] → visit148 [139.657], delay [10] → travel [2], wait [0] → visit138 [151.657], delay [10] → travel [15.8114], wait [0] → depot [177.468] <i>Transit Sum [177.468]</i> |
| Distance: depot [0], delay [0] → travel [19.0263], wait [0] → visit147 [19.0263], delay [0] → travel [5.65685], wait [0] → visit137 [24.6832], delay [0] → travel [0], wait [0] → visit148 [24.6832], delay [0] → travel [2], wait [0] → visit138 [26.6832], delay [0] → travel [15.8114], wait [0] → depot [42.4945] <i>Transit Sum [42.4945]</i> |
| Vehicle17 Total cost = 48.4243 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit161 → visit162 → depot |
| Distance: depot [0], delay [0] → travel [24.0832], wait [0] → visit161 [24.0832], delay [0] → travel [8.06226], wait [0] → visit162 [32.1454], delay [0] → travel [16.2788], wait [0] → depot [48.4243] <i>Transit Sum [48.4243]</i> |
| Vehicle18 Total cost = 127.933 Fixed cost = 0 Cost coefficients: Distance [1] |

| |
|--|
| Route: depot → visit163 → visit47 → visit164 → visit48 → visit207 → visit208 → depot |
| Time: depot [0], delay [0] → travel [24.3311], wait [0] → visit163 [145], delay [10] → travel [2], wait [0] → visit47 [157], delay [10] → travel [13.3417], wait [0] → visit164 [180.342], delay [10] → travel [2.23607], wait [0] → visit48 [192.578], delay [10] → travel [31.0161], wait [0] → visit207 [233.594], delay [10] → travel [42.2019], wait [0] → visit208 [285.796], delay [10] → travel [12.8062], wait [0] → depot [308.602] |
| <i>Transit Sum [308.602]</i> |
| Distance: depot [0], delay [0] → travel [24.3311], wait [0] → visit163 [24.3311], delay [0] → travel [2], wait [0] → visit47 [26.3311], delay [0] → travel [13.3417], wait [0] → visit164 [39.6727], delay [0] → travel [2.23607], wait [0] → visit48 [41.9088], delay [0] → travel [31.0161], wait [0] → visit207 [72.9249], delay [0] → travel [42.2019], wait [0] → visit208 [115.127], delay [0] → travel [12.8062], wait [0] → depot [127.933] |
| <i>Transit Sum [127.933]</i> |
| Vehicle19 Total cost = 138.792 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit165 → visit189 → visit169 → visit170 → visit190 → visit166 → depot |
| Time: depot [0], delay [0] → travel [7.61577], wait [137.384] → visit165 [145], delay [10] → travel [39.4081], wait [0] → visit189 [194.408], delay [10] → travel [4.12311], wait [0] → visit169 [208.531], delay [10] → travel [11.4018], wait [0] → visit170 [229.933], delay [10] → travel [34.0588], wait [0] → visit190 [273.992], delay [10] → travel [30.4795], wait [0] → visit166 [314.471], delay [10] → travel [11.7047], wait [0] → depot [336.176], delay [0] → travel [0], wait [0] |
| <i>Transit Sum [336.176]</i> |
| Distance: depot [0], delay [0] → travel [7.61577], wait [0] → visit165 [7.61577], delay [0] → travel [39.4081], wait [0] → visit189 [47.0239], delay [0] → travel [4.12311], wait [0] → visit169 [51.147], delay [0] → travel [11.4018], wait [0] → visit170 [62.5488], delay [0] → travel [34.0588], wait [0] → visit190 [96.6075], delay [0] → travel [30.4795], wait [0] → visit166 [127.087], delay [0] → travel [11.7047], wait [0] → depot [138.792] |
| <i>Transit Sum [138.792]</i> |
| Vehicle20 Total cost = 102.149 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit183 → visit181 → visit182 → visit184 → depot |
| Time: depot [0], delay [0] → travel [40.3113], wait [115.689] → visit183 [156], delay [10] → travel [34.0147], wait [0] → visit181 [200.015], delay [10] → travel [13.4536], wait [0] → visit182 [223.468], delay [10] → travel [2], wait [0] → visit184 [235.468], delay [10] → travel [12.3693], wait [0] → depot [257.838] |
| <i>Transit Sum [257.838]</i> |
| Distance: depot [0], delay [0] → travel [40.3113], wait [0] → visit183 [40.3113], delay [0] → travel [34.0147], wait [0] → visit181 [74.326], delay [0] → travel [13.4536], wait [0] → visit182 [87.7796], delay [0] → travel [2], wait [0] → visit184 [89.7796], delay [0] → travel [12.3693], wait [0] → depot [102.149], delay [0] → travel [0], wait [0] |
| <i>Transit Sum [102.149]</i> |
| Vehicle21 Total cost = 111.046 Fixed cost = 0 Cost coefficients: Distance [1] |
| Route: depot → visit195 → visit191 → visit201 → visit197 → visit75 → visit198 → visit196 → visit202 → visit192 → visit76 → depot |

| |
|---|
| <p>Time: depot [0], delay [0] → travel [39.8497], wait [135.15] → visit195 [175], delay [10] → travel [3.16228], wait [0] → visit191 [188.162], delay [10] → travel [5], wait [0] → visit201 [203.162], delay [10] → travel [7.61577], wait [0] → visit197 [220.778], delay [10] → travel [4.12311], wait [0] → visit75 [234.901], delay [10] → travel [7.2111], wait [0] → visit198 [252.112], delay [10] → travel [2], wait [0] → visit196 [264.112], delay [10] → travel [2], wait [0] → visit202 [276.112], delay [10] → travel [4], wait [0] → visit192 [290.112], delay [10] → travel [1.41421], wait [0] → visit76 [301.526], delay [10] → travel [34.6699], wait [0] → depot [346.196]</p> <p><i>Transit Sum [346.196]</i></p> |
| <p>Distance: depot [0], delay [0] → travel [39.8497], wait [0] → visit195 [39.8497], delay [0] → travel [3.16228], wait [0] → visit191 [43.012], delay [0] → travel [5], wait [0] → visit201 [48.012], delay [0] → travel [7.61577], wait [0] → visit197 [55.6278], delay [0] → travel [4.12311], wait [0] → visit75 [59.7509], delay [0] → travel [7.2111], wait [0] → visit198 [66.962], delay [0] → travel [2], wait [0] → visit196 [68.962], delay [0] → travel [2], wait [0] → visit202 [70.962], delay [0] → travel [4], wait [0] → visit192 [74.962], delay [0] → travel [1.41421], wait [0] → visit76 [76.3762], delay [0] → travel [34.6699], wait [0] → depot [111.046]</p> <p><i>Transit Sum [111.046]</i></p> |
| <p>Vehicle22 Total cost = 117.463 Fixed cost = 0 Cost coefficients: Distance [1]</p> |
| <p>Route: depot → visit203 → visit77 → visit204 → visit78 → visit217 → visit218 → depot</p> |
| <p>Time: depot [0], delay [0] → travel [34], wait [144] → visit203 [178], delay [10] → travel [5], wait [0] → visit77 [193], delay [10] → travel [5], wait [0] → visit204 [208], delay [10] → travel [9.84886], wait [0] → visit78 [227.849], delay [10] → travel [13.6015], wait [0] → visit217 [251.45], delay [10] → travel [18.2483], wait [0] → visit218 [279.699], delay [10] → travel [31.7648], wait [0] → depot [321.463]</p> <p><i>Transit Sum [321.463]</i></p> |
| <p>Distance: depot [0], delay [0] → travel [34], wait [0] → visit203 [34], delay [0.1e-010] → travel [5], wait [0] → visit77 [39], delay [0] → travel [5], wait [0] → visit204 [44], delay [0] → travel [9.84886], wait [0] → visit78 [53.8489], delay [0] → travel [13.6015], wait [0] → visit217 [67.4503], delay [0] → travel [18.2483], wait [0] → visit218 [85.6986], delay [0] → travel [31.7648], wait [0] → depot [117.463]</p> <p><i>Transit Sum [117.463]</i></p> |
| <p>Vehicle23 Total cost = 152.724 Fixed cost = 0 Cost coefficients: Distance [1]</p> |
| <p>Route: depot → visit209 → visit213 → visit157 → visit211 → visit210 → visit214 → visit212 → visit158 → depot</p> |
| <p>Time: depot [0], delay [0] → travel [12.2066], wait [170.793] → visit209 [183], delay [10] → travel [8.24621], wait [0] → visit213 [201.246], delay [10] → travel [12.3693], wait [0] → visit157 [223.616], delay [10] → travel [11.6619], wait [0] → visit211 [245.277], delay [10] → travel [32.28], wait [0] → visit210 [287.557], delay [10] → travel [16.2788], wait [0] → visit214 [313.836], delay [10] → travel [18.1108], wait [0] → visit 212 [341.947], delay [10] → travel [19.2094], wait [0] → visit158 [371.156], delay [10] → travel [22.3607], wait [0] → depot [403.517]</p> <p><i>Transit Sum [403.517]</i></p> |
| <p>Distance: depot [0], delay [0] → travel [12.2066], wait [0] → visit209 [12.2066], delay [0] → travel [8.24621], wait [0] → visit213 [20.4528], delay [0] → travel [12.3693], wait [0] → visit157 [32.8221], delay [0] → travel [11.6619], wait [0] → visit211 [44.484], delay [0] → travel [32.28], wait [0] → visit210 [76.764], delay [0] → travel [16.2788], wait [0] → visit214 [93.0428], delay [0] → travel [18.1108], wait [0] → visit212 [111.154], delay [0] → travel [19.2094], wait [0] → visit158 [130.363], delay [0] → travel [22.3607], wait [0] → depot [152.724]</p> <p><i>Transit Sum [152.724]</i></p> |

حل مشكلة تحديد مسار المركبات الديناميكي باستخدام اساليب البحث العشوائية

إعداد
جواد عاصم العمري

المشرف
سامح الشهابي

ملخص

تتكون مشكلة تحديد المسار الأمثل لمجموعة من المركبات من: عدد من المواقع الجغرافية، و محطة مركزية، و عدد من المركبات، بحيث ينبغي على المركبات زيارة جميع هذه المواقع خلال فترات زمنية محددة لكل موقع. يتكون الحل لمثل هذه المشاكل من مجموعة من المسارات، تبدأ كل منها من المحطة المركزية، و تنتهي أيضا عند تلك المحطة، بشرط أن تتم زيارة كل المواقع في الفترات الزمنية المحددة، و تكون مجموع المسافات التي قطعها جميع المركبات أقل ما يمكن.

هناك العديد من اساليب البحث التي تستعمل لحل مثل هذه المشاكل، لكن لا يمكن لأى منها الوصول الى الحل الأمثل خلال فترة زمنية معقولة، لذلك تستبدل هذه الاساليب بطرق البحث العشوائية القادرة على ايجاد حلول قريبة من الحل الأمثل لكن خلال فترة زمنية قصيرة. مثل هذه الطرق سوف تستخدم في هذه الدراسة.

تهدف هذه الدراسة الى بتحديد المسار الامثل لمجموعة من المركبات، باستخدام مجموعة من اساليب البحث العشوائية. كما تهدف هذه الدراسة الى تحديد مدى تأثير التغير المنتظم في بارامترات البحث، واماكن البحث، على نوعية الحل و على سرعة الحصول عليها. يتم هذا بانشاء مجموعة من اساليب البحث العشوائية، و اختبارها على احدى التطبيقات العملية.

تدل نتائج البحث بأن تأثير التغير المنتظم في بارامترات البحث يحسن نوعية الحل عندما يتجاوز عدد المواقع الجغرافية التي يجب زيارتها عن 150 موقع، لكن البيانات الاحصائية لا تؤيد هذا الاستنتاج. كما لا يوجد أي علاقة بين التغير المنتظم في بارامترات البحث وسرعة الحصول على الحل. تبين أيضا أن هناك ترتيب محدد لأماكن البحث الذي يؤدي الى الحصول على النتائج في أقل وقت ممكن.