

USING POPULATION-BASED
INCREMENTAL LEARNING TO OPTIMIZE
FEASIBLE DISTRIBUTION LOGISTIC
SOLUTIONS

by

Tobie Lourens

Thesis submitted in partial fulfilment of the
requirements for the degree of

Master of Science in Industrial Engineering

University of Stellenbosch

April 2005



DECLARATION

I, the undersigned, hereby declare that the work contained in this document is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: _____

Date: _____



ACKNOWLEDGEMENTS

The author wishes to thank the following persons:

Professor W van Wijck

His inextinguishable passion for this field of study has been a great source of inspiration in this project. Thanks also to the many invaluable insets that came without asking. There are not many better study leaders.

Mr J Bekker

At the very last moment, nobody could have given better support and insights on finishing and finalizing the project.

Professor J van Vuuren

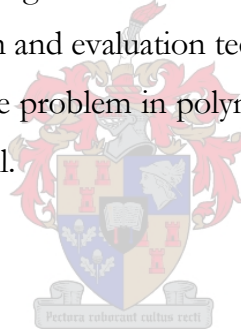
Thank you for the many “quick info” sessions. They helped in forming the fundamental basis of this research work.



In the end, all praise must ultimately go to the Lord, without whom this work would never have even started. He granted me this opportunity.

ABSTRACT

This thesis introduces an adaptation of the Population-Based Incremental Learning (PBIL) meta-heuristic implemented on a variant of the General Pickup and Delivery Problem. The mapping of the customers in the problem and the vehicle routes on a time grid enables the utilization of the powerful genetic search that the PBIL algorithm provides in liaison with competitive learning. The problem consists of a number of customers who may at any time of the day place an order on another customer for some package. The fleet of vehicles travelling between the customers must then combine powers to pickup and deliver the package as fast as possible without ever leaving their assigned routes. The solution to this problem then, is a set of routes for the fleet that will minimize some percentile of the delivery times between customers. The PBIL meta-heuristic provides the blueprint of the final algorithm, where the final algorithm is actually just a normal PBIL algorithm with some external solution generation and evaluation techniques employed. The final algorithm can easily solve an instance of the problem in polynomial time, given that the resolution of the time grid used is not too small.



OPSOMMING

Hierdie dokument stel 'n aangepaste weergawe van die Populasie Gebaseerde Inkrementele Leer (PBIL) meta-heuristiek voor wat geïmplementeer is op 'n variant van die Algemene Optel en Afleveringsprobleem. Die uitstipping van die kliënte in die problem sowel as die afleveringsvoertuie se roetes op 'n tydrooster maak die gebruik van die kragtige genetiese soekfunksies in samewerking met mededingende leer moontlik. Die problem bestaan uit 'n aantal kliënte wat op enige tydstep van die dag 'n bestelling vir 'n pakkie by 'n ander kliënt kan plaas. Die voertuie wat deur die dag tussen die kliënte rondry, moet dan hulle kragte saamspan om die pakkie so gou as moontlik af te lewer sonder dat enigeen sy toegekende roete verlaat. 'n Oplossing vir die problem bestaan dus uit 'n stel roetes wat een of ander persentiel van die afleveringstye tussen kliënte sal minimeer. Die PBIL meta-heuristiek is die ruggraat van die finale algoritme, waar die finale algoritme in werklikheid net 'n PBIL algoritme is met eksterne tegnieke vir die konstruksie van roetes en die evaluasie van oplossings ingebou. Die finale algoritme kan 'n geval van die problem maklik binne polinoomtyd oplos, mits die resolusie van die tydrooster wat gebruik word nie te klein is nie.

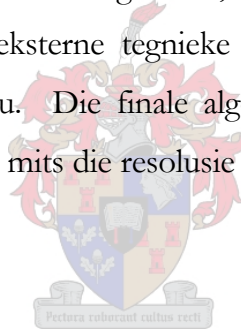


TABLE OF CONTENTS

Chapter 1	Introduction.....	1
1.1	Project Background.....	1
1.2	Problem Statement.....	1
1.3	Project Aim and Scope.....	2
1.4	Layout and Content	4
Chapter 2	Literature Review	5
2.1	Introduction.....	5
2.2	Computational Complexity Theory.....	5
2.2.1	Algorithms and Algorithmic Complexity	5
2.2.2	Classification of Problems	9
2.3	Vehicle Routing and Scheduling Problems.....	11
2.3.1	The Capacitated Vehicle Routing Problem.....	12
2.3.2	The Vehicle Routing Problem with Time Windows.....	14
2.3.3	The Multiple Depot Vehicle Routing Problem.....	15
2.3.4	The Stochastic Capacitated Vehicle Routing Problem	16
2.3.5	The Pickup and Delivery Problem.....	17
2.4	The Population-Based Incremental Learning Algorithm	19
2.4.1	Choice of algorithm.....	20
2.4.2	Representation.....	21
2.4.3	The Population Concept.....	22
2.4.4	The Probability Vector	23
2.4.5	Intelligence in Searching.....	24
2.4.6	Evaluation	24
2.4.7	Mutation	25
2.4.8	Termination.....	26
2.4.9	Summary.....	26
2.5	Dijkstra's Shortest Path Algorithm.....	29
2.6	Literature Review Summary.....	34
Chapter 3	The Adapted Pickup and Delivery Problem.....	35
3.1	Introduction.....	35
3.2	Route Interactivity.....	35
3.3	Time Grid Mapping.....	36
3.4	Travel, Loading, Unloading & Waiting Times	37
3.5	Route Cycling	41
3.6	Package Delivery Times	42
3.7	Variability of the Grid Spacing.....	43
3.8	A Mathematical Model.....	45
3.9	Conclusion	47

Chapter 4 The Tailored PBIL Algorithm	48
4.1 Introduction.....	48
4.2 The Three-Dimensional Probability Vector.....	48
4.3 Solution Vectors Population.....	50
4.3.1 Route Construction.....	51
4.3.2 Route Translation	54
4.3.3 The Solution Graph System	54
4.3.4 Solution Evaluation.....	55
4.4 Probability Vector Update and Mutation.....	57
4.5 Termination	58
Chapter 5 Validation and Verification.....	59
5.1 Introduction.....	59
5.2 Test Problems Setup and Execution	59
5.2.1 First Test Problem.....	59
5.2.2 Second Test Problem.....	60
5.2.3 Third Test Problem.....	62
5.2.4 Fourth Test Problem	64
5.2.5 Fifth Test Problem	65
5.2.6 Sixth Test Problem.....	67
5.2.7 Conclusion	68
5.3 The Effects of Input Variables.....	68
5.3.1 Number of Stocking Points.....	69
5.3.2 Vehicle Fleet Size.....	70
5.3.3 Grid Spacing	71
5.3.4 Conclusion	71
Chapter 6 Recommendations	72
6.1 Alternative Evaluation Technique	72
6.2 Automation of Choosing Grid Spacings Size.....	72
6.3 Turning Off the Time Grid.....	72
6.4 Further Research	73
Chapter 7 Conclusions.....	74
7.1 Success of the Project.....	74
Appendix A Solution Vectors For Test Problems.....	A1
Solution Vector for First Test Problem.....	A1
Solution Vector for Second Test Problem.....	A1
Solution Vector for Third Test Problem.....	A2
Appendix B Visual Basic Code	B1
Declaration Section	B1

Algorithm Starting Sub Procedure	B1
Initialization Sub Procedure	B1
Data Loading Sub Procedure	B2
Routes Construction Sub Procedure	B3
Function Returning Random SP	B4
Route Conversion Sub Procedure.....	B4
Graph System Evaluation Sub Procedure.....	B5
Dijkstra Path-Finding Function.....	B6
Probability Vector Update Sub Procedure.....	B7
Probability Vector Mutate Sub Procedure	B8



LIST OF ILLUSTRATIONS

List of Tables

Table 2-1: Running times for different algorithmic classes	9
Table 2-2: First iteration in executing Dijkstra's algorithm	32
Table 2-3: Second iteration in executing Dijkstra's algorithm	32
Table 2-4: Third iteration in executing Dijkstra's algorithm	32
Table 2-5: Fourth iteration in executing Dijkstra's algorithm.....	33
Table 2-6: Last iteration in executing Dijkstra's algorithm.....	33
Table 4-1: The collections of starting and ending stocking points for three routes.....	51
Table 5-1: Travel times between the six stocking points of the third test problem.....	62
Table 5-2: Start and end stocking points for third test problem	62
Table 5-3: Travel times between the six stocking points of the fourth test problem	64
Table 5-4: Travel times between SPs of fifth test problem	66
Table 5-5: Input variable values for the effects experiment.....	69

List of Figures

Figure 1-1: The general flow of optimization with a meta-heuristic	3
Figure 2-1: An illustration of asymptotic notation.....	7
Figure 2-2: Graphs of different algorithmic complexity orders	9
Figure 2-3: The relation of the different complexity classes	10
Figure 2-4 Two connected graphs with and without a bridge.....	11
Figure 2-5: The Capacitated Vehicle Routing Problem.....	12
Figure 2-6: The PBIL algorithm.....	20
Figure 2-7: The crossover of routes	21
Figure 2-8: Binary representation of two variables.....	22
Figure 2-9: A population of solution vectors	22
Figure 2-10: Probability representation of a population of solution vectors.....	23
Figure 2-11: The function $f(x) = (x-6)^3 - (x-6)^2 - 30x + 180$ for $0 \leq x \leq 15$	27
Figure 2-12: A population of solution vectors generated from a probability vector	28
Figure 2-13: Dijkstra's Shortest Path Algorithm.....	30
Figure 2-14: A weighted graph with six vertices	31
Figure 3-1: A time grid mapping.....	36
Figure 3-2: Mapping time on the time grid.....	36
Figure 3-3: Travel times between stocking points.....	37
Figure 3-4: A time matrix for every part of the day	37
Figure 3-5: Travel times expressed by functions of time	38
Figure 3-6: Travel times between two stocking points as a function of the time of day ...	38
Figure 3-7: Visitation of vehicles to stocking points.....	39
Figure 3-8: Events on two vehicles' routes.....	40
Figure 3-9: Loading/Unloading times distributed normal with a mean of 10 minutes and standard deviation of 2 minutes	40

Figure 3-10: The constraining of route start and end points	41
Figure 3-11: Possible paths for a package to follow.....	43
Figure 3-12: A graph showing the optimizing of the grid spacing.....	45
Figure 3-13: Vehicle routes with the time grid turned off	45
Figure 4-1: The tailored PBIL algorithm.....	48
Figure 4-2: The three-dimensional probability vector	49
Figure 4-3: A three-dimensional solution vector	50
Figure 4-4: The forming of a population of solution vectors	51
Figure 4-5: Travel times between three stocking points and the probability vector	52
Figure 4-6: Choosing the correct SP to insert based on the probability vector	52
Figure 4-7: The length of a route determined on a timeline	53
Figure 4-8: Routes translated into a solution vector	54
Figure 4-9: Package paths in the absence of vehicle routes.....	55
Figure 4-10: The complete graph system	55
Figure 4-11: The probabilistic distribution of order arrivals	56
Figure 4-12: A histogram of shortest path lengths.....	57
Figure 5-1: Expected solution of first test problem.....	59
Figure 5-2: Convergence output for first test problem	60
Figure 5-3: Expected solutions of second test problem.....	60
Figure 5-4: Convergence output for second test problem.....	61
Figure 5-5: Optimal solution to second test problem.....	61
Figure 5-6: Graph representation of third test problem	62
Figure 5-7: Expected solution of third test problem	63
Figure 5-8: Convergence output for third test problem.....	63
Figure 5-9: Alternate solution of third test problem.....	64
Figure 5-10: Convergence output for fourth test problem.....	65
Figure 5-11: Graph representation of fifth test problem	65
Figure 5-12: Convergence output for fifth test problem	66
Figure 5-13: Returned optimal solution to fifth test problem	67
Figure 5-14: Convergence output for sixth test problem.....	67
Figure 5-15: Effect of number of stocking points on algorithm running time	70
Figure 5-16: Effect of vehicle fleet size on algorithm running time.....	70
Figure 5-17: Effect of grid spacing on algorithm running time.....	71

GLOSARRY

BPP	Bin Packing Problem
CVRP	Capacitated Vehicle Routing Problem
DARP	Dial-A-Ride Problem
EGA	Equilibrium Genetic Algorithm
GA	Genetic Algorithm
GPDP	General Pickup and Delivery Problem
IP	Integer Programming
JIT	Just In Time
MDVRP	Multiple Depot Vehicle Routing Problem
MP	Mutation Probability
MS	Mutation Shift
MTSP	Multiple Travelling Salesman Problem
PBIL	Population-Based Incremental Learning
PDP	Pickup and Delivery Problem
PV	Probability Vector
SP	Stocking Point
SVRP	Stochastic Vehicle Routing Problem
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem
VRPPD	Vehicle Routing Problem with Pickup and Deliveries
VRPTW	Vehicle Routing Problem with Time Windows

INTRODUCTION

1.1 Project Background

Distribution logistics is a vital and very real component of supply chains in general. For some companies it is so much integrated into the business structure, a large part of the accomplishment of the company depends on it. In South Africa, this is no exception.

One of the large South African groups playing an important role in the automotive industry is concerned with the way distribution of spare parts is currently taking place. The cost and service performance of their current delivery process have not performed to plan since its implementation, and a new or improved model on which deliveries can be based has become a necessity for this company. This presented an opportunity to conduct research in the field of distribution logistics and to offer a new or improved model for deliveries.

1.2 Problem Statement

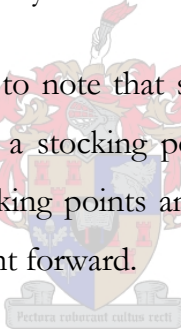
The company uses a fleet of delivery vehicles to perform the delivery of parts to dealers. Dealers can be subdivided into two categories: stocking points and demand points. From a central distribution store (mainly virtual) inventory are distributed to a number of regions. In such a region, there are many stocking and demand points and the concern is with the distribution of parts between these points.

Stocking points are mainly the larger dealers in a region. It is only at these points that inventory are located. Each stocking point has a number of unique clients and forms the preferred stocking point of these clients. Usually these clients will fall within the near vicinity of the preferred stocking point *i.e.* the assignment of unique clients is based on geographical location. These unique clients are also called demand points. The distribution of spare parts takes place between stocking points and demand points or between multiple stocking points.

Focusing on one such region's statistics, the availability of parts within this region is above the target of 95%. On the other hand, deliveries from preferred stocking points to demand points are 87%, just below the target of 90%. Fifteen percent of the deliveries of parts are taking more than two working hours, the original target being zero percent. In addition, 32 delivery vehicles are deployed versus an original target of 22. Although most of the concern is abounding from this specific region, the problem and so also the solution is much more generic. Intuitively it should be obvious that many other businesses outside the automotive industry could also benefit from a solution to this type of problem. The possible applications of such a solution, however, fall beyond the scope of this text.

The main reason why the mentioned figures are not on target is believed to be this: the scheduled routes might not be an optimal combination of routes for the delivery vehicles to follow. The problem, then, to which this thesis provides a solution, is to find an optimal set of routes that the delivery vehicles may follow.

At this point it might be worthy to note that since there are conceptually no difference between a delivery made between a stocking point and a demand point, and one made between two stocking points, stocking points and demand points are both referred to as stocking points (SPs) from this point forward.



1.3 Project Aim and Scope

The aim of this project is to propose and implement a meta-heuristic that will find a near-optimal set of routes from the vast expanse of possible sets of routes. Meta-heuristics are algorithms* that implement some strategy that enables it to search the solution space in an intelligent way for an optimal solution (Michalewicz & Zbigniew, 2002). It focuses on combinatorial problems and stands in contrast to exact methods like linear- or dynamic programming (which are essentially both algorithms). For reasons that will become clear later in the text, the Population-Based Incremental Learning algorithm is the chosen meta-heuristic that will be implemented on the problem.

* Note that, respecting the context, the words *algorithm* and *meta-heuristic* may often be used interchangeably in the text.

Verifying and validating the proposed method is never to be left out. The method is tested on a number of seemingly small, yet complex enough problems of which either the optimal answer is already known, or a fairly good one can be composed from one's gutt-feal. The main validation and verification of the method are done by testing it on the problem that initiated this research work. The specific region mentioned previously has twelve stocking points in total, so the proposed method and this research work will be deemed successful if it can find a solution to this problem consisting of twelve stocking points.

Looking at the general approach of a meta-heuristic, the scope of this project can be clearly defined. Figure 1-1 shows the general flow of optimization using a meta-heuristic. A number of solutions are randomly generated based on information contained within the meta-heuristic. These solutions are evaluated using some sort of fitness function and the fittest, or best, solution is returned. This solution is then used to train the system before generating another set of solutions. This cycle continues until some terminating condition has been reached, after which the cycle will be exited and the algorithm stopped. In this project, the way in which a given solution is evaluated to compare it with other solutions has been proposed by Van Wijck (2004). This method is a given and is used as proposed for the evaluation of alternative solutions and will be thoroughly discussed in this text to present the final solution as a whole.

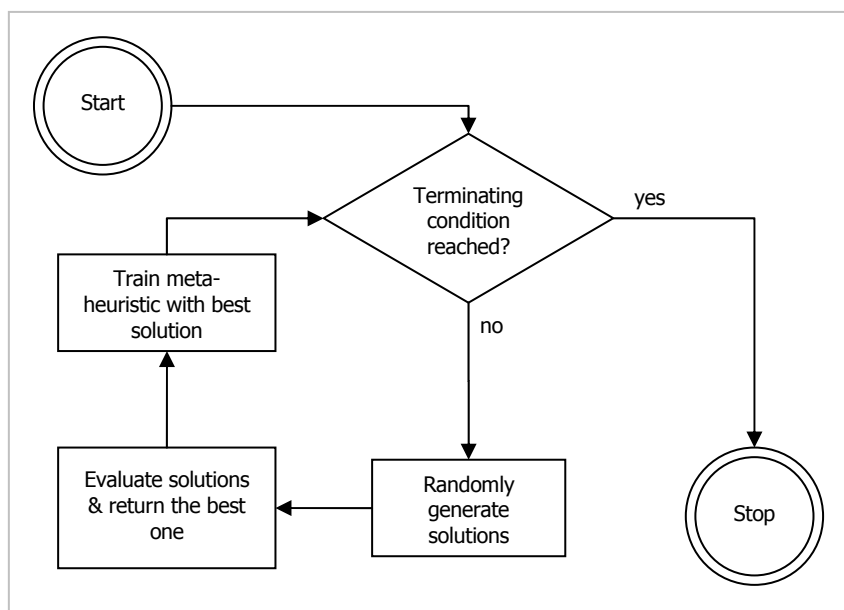
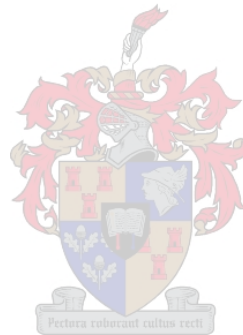


Figure 1-1: The general flow of optimization with a meta-heuristic

The main scope of this project consists of the remaining aspects of the cycle. That is, the generation of solutions, and the training of the meta-heuristic to find better and better solutions as it goes through its iterations.

1.4 Layout and Content

To understand the underlying theories of vehicle scheduling, pickup and delivery problems etc., a study of the literature is undertaken. The literature review is presented in Chapter 2. The translation of the real-world problem into a mathematical model forms the first step towards a solution. Presented in Chapter 3 is this translation. Chapter 4 contains a discussion of a tailored version of the Population-Based Incremental Learning (PBIL) algorithm for this specific problem. Verification and validation follows in Chapter 5, and thereafter the recommendations in Chapter 6. Finally, in Chapter 7, conclusions are drawn as to the viability and success of the proposed method and the project as a whole.



LITERATURE REVIEW*

2.1 Introduction

The literature study conducted consists of the following main categories:

- ◆ Complexity Theory – to put algorithms and problems in perspective with other algorithms and problems, the theories behind computational complexity form a definitive basis from which to do this.
- ◆ Vehicle Routing and Pickup & Delivery Problems – the necessity to study these comes from the fact that the distribution problem under study will fit in somewhere within the scope of these fields.
- ◆ Population-Based Incremental Learning Algorithm – the meta-heuristic decided upon in this project, tailored to fit and implemented to solve the distribution problem.
- ◆ Dijkstra's Shortest Path Algorithm – this algorithm is implemented in the solution evaluation technique proposed by Van Wijck (2004).

2.2 Computational Complexity Theory

Complexity theory is part of the theory of computation and a central and important field of theoretical computer science. The aim of complexity theory is to understand the intrinsic complexity of computational tasks. Complexity theory studies natural computational resources, like time or space, and tries to understand the class of problems that are solvable with a given limited amount of these resources (Bläzer, 2003).

2.2.1 Algorithms and Algorithmic Complexity

The word *algorithm* comes ultimately from the name of the 9th-century mathematician Abu Abdullah Muhammad bin Musa al-Khwarizmi. The word *algorism* originally referred only to the rules of performing arithmetic using Arabic numerals but evolved into *algorithm* by the

*The informed reader may omit this chapter with no loss of continuity.

18th century. The word has now evolved to include all definite procedures for solving problems or performing tasks.

2.2.1.1 *Classes of Algorithms*

There are many ways to classify algorithms and this topic has been the subject of ongoing debate over the years. One way to do this is by complexity classes. Before discussing the classification of algorithms by means of algorithmic complexity, one other way of classifying them is by their design methodology or paradigm. The most commonly found paradigms are listed below.

- ◆ Divide and Conquer – A divide-and-conquer algorithm recursively subdivides a problem into smaller instances of the same problem by dividing the solution space of the problem. The smaller problems are then solved and put together to form a solution to the original problem.
- ◆ Dynamic Programming – When the optimal solution to a problem can be constructed from optimal solutions to subproblems and the subproblems are used to solve many instances of the same problem, dynamic programming is used. This type of algorithm avoids recomputing solutions that have already been computed.
- ◆ Greedy Algorithms – They are similar to dynamic programming algorithms, with the difference that solutions to subproblems are not necessary. At each stage, the algorithm makes a greedy choice of what looks best at that moment.
- ◆ Linear Programming – When a problem can be modelled by a linear objective function subjected to some linear constraints, such a problem can be minimized or maximized by linear programming.
- ◆ Heuristics – The definition of heuristics have already been stated earlier in the text. This class includes algorithms such as the genetic algorithm, simulated annealing, PBIL etc.

The more mathematically inclined classification model of algorithmic complexity will be used and discussed in this text. While complexity theory is concerned with showing that a certain problem cannot be solved with a certain amount of computational resources, the theory of algorithms tries to develop algorithms that use as few of these resources as

possible to solve the problem. Algorithmic complexity is usually measured by time and space complexity, while other resources can also be considered.

2.2.1.2 Asymptotic Notation

The amount of resources used by a specific instance of an algorithm is a function of the input size of the problem it is trying to solve. Expressing these functions, which may be very complex sometimes, calls for some method of notation. Asymptotic notation is an order notation used to express the amount of resources used (complexity) by an algorithm. The advantage of this measuring system is that it is not dependant upon any specific computer system. The order of complexity of a problem is fixed, regardless of the processing capabilities of the computer on which the algorithm is implemented.

Let $f(n)$ and $g(n)$ be two functions. By definition (Goodrich & Tamassia, 2001), $f(n)$ is $O(g(n))$ if there exists a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $0 \leq f(n) \leq c \cdot g(n)$ for every $n \geq n_0$. Another way of writing this is $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq c$. This type of asymptotic notation is referred to as the O notation (pronounced Big-Oh). Another way to describe the notation is possible: the order of magnitude of the algorithmic complexity of a function $f(n)$ is given by the term in $f(n)$ which grows fastest in the number of basic operations performed by the algorithm as the input size n grows. In effect, any polynomial $a_k n^k + a_{k-1} n^{k-1} + \dots + a_0$ will always be $O(n^k)$. In Figure 2-1, the function $f(n)$ is $O(g(n))$, for $f(n) \leq cg(n)$ when $n \geq n_0$.

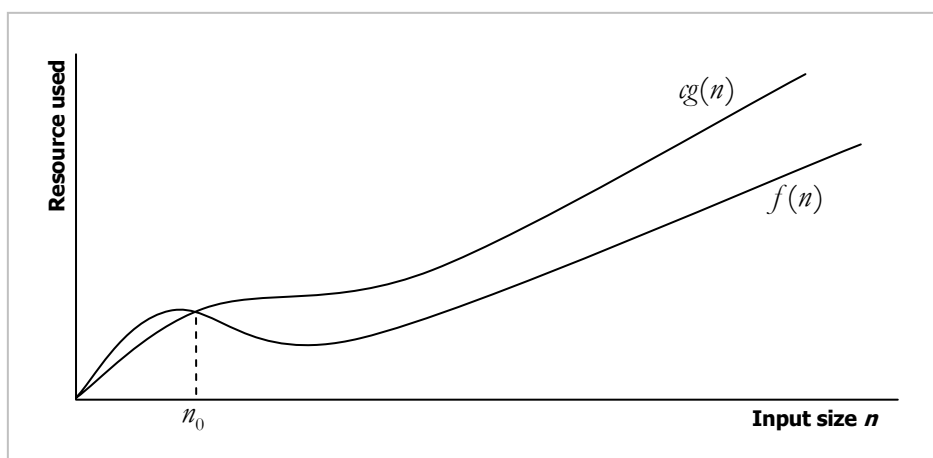


Figure 2-1: An illustration of asymptotic notation

Some properties of order notation are in order at this point. In order to understand the analysis of algorithms in terms of complexity, these properties might be useful.

1. If $f = O(b)$ & $g = O(b)$, then $(f + g) = O(b)$
2. If $f = O(b)$ & $g = O(l)$, then $(f \cdot g) = O(bl)$
3. $f = O(f)$ [*Reflexivity*]
4. If $f = O(g)$ & $g = O(b)$, then $f = O(b)$ [*Transitivity*]
5. For all $k > 0$, $kf = O(f)$

2.2.1.3 Space Complexity

Space complexity refers to the amount of storage space used as a resource in the execution of an algorithm. Nowadays, computers have far more memory compared to their counterparts of a decade ago, so space complexity is not as much a problem as it used to be. The amount of space used to solve large instances of problems may indeed still render an algorithm impractical. For the sake of brevity, only the concept of time complexity will be covered in this text.

2.2.1.4 Time Complexity

If $T(n) = n^2 + 6n - 36$ measures the time complexity of some algorithm, then $T(n) = O(n^2)$, or $T(n)$ is $O(n^2)$ as stated previously. Different classifications of time complexity suggest itself, since a function's fastest growing term in terms of the input size can literally be anything. If the complexity is independent of the input size n , it is classified as *constant* and denoted $O(1)$. If it grows linearly with n it is classified as *linear* and denoted $O(n)$. If it grows exponentially with n it is classified as *exponential*. This is denoted $O(a^n)$. For sufficiently large values of n , as per definition, the following hierarchy of complexity classes can be drawn (the notation $f(n) \prec g(n)$ means that $f(n) = O(g(n))$, or $f(n)$ is $O(g(n))$):

$$1 \prec \log n \prec n \prec n \log n \prec n^2 \prec n^3 \prec 2^n \prec n! \quad \dots(2.1)$$

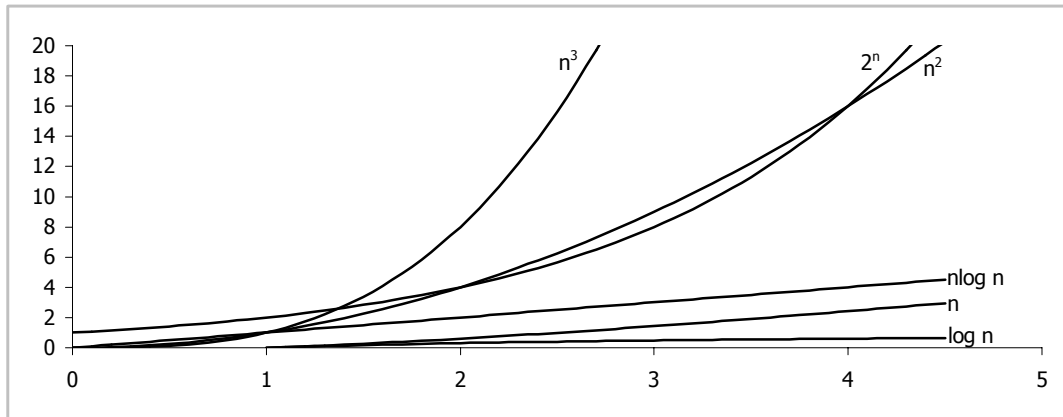


Figure 2-2: Graphs of different algorithmic complexity orders

Figure 2-2 (Chartrand & Oellermann, 1993) shows the graphs of some of the complexity classes in which an algorithm may fall. To give an indication of the concept of time complexity of an algorithm, consider an algorithm that has an input size given by $n = 10^6$. At our disposal is a computer capable of processing one million operations per second. This equals an operating speed of one microsecond per basic operation. Table 2-1 from Chartrand & Oellermann (1993) lists the running times of algorithms from different complexity classes with this type of input.

Class	Complexity	Operations	Time
Constant	$O(1)$	1	1 microsecond
Logarithmic	$O(\log n)$	10	10 microseconds
Linear	$O(n)$	10^6	1 second
$n \log n$	$O(n \log n)$	10^7	10 seconds
Quadratic	$O(n^2)$	10^{12}	11.574 days
Cubic	$O(n^3)$	10^{18}	31688 years
Exponential	$O(2^n)$	$10^{301\,030}$	$10^{301\,006}$ times the age of the universe

Table 2-1: Running times for different algorithmic classes

2.2.2 Classification of Problems

Complexity theory usually involves determining an estimate of the resources required to solve the hardest instance of a problem on a theoretical computer. Many difficulties for mathematicians and logicians of the earlier centuries were solved with the description of such a theoretical computer, known as a Turing machine. A Turing machine is an abstract model of a computer formulated by Alan Turing. It is a finite-state machine with an

infinite read-write memory tape. If a polynomial time algorithm is known that can solve a problem on a Turing machine, then it is said that the problem is *tractable*. Conversely, if no such algorithm is known*, then the problem is called *intractable* or hard (Chartrand & Oellermann, 1993).

A subset of complexity theory deals with decision problems. A decision problem is one that is interpreted as a binary question that may be answered as either *yes* or *no*, and fall into sets of comparable complexity called complexity classes. The class P consists of all decision problems that can be solved on a deterministic sequential machine in an amount of time that is polynomial in the size of the input. The class NP consists of the decision problems whose solution can be found in polynomial time on a non-deterministic machine. The complementary class of class NP is $co-NP$. A decision problem L is said to be NP -complete if $L \in NP$ and $L_1 \preceq L$ for all $L_1 \in NP$. A decision problem L is called NP -hard if a problem L' exists that is NP -complete for which $L' \preceq L$ (Wikipedia, 2004). Figure 2-3 depicts how all of these complexity classes relate to each other.

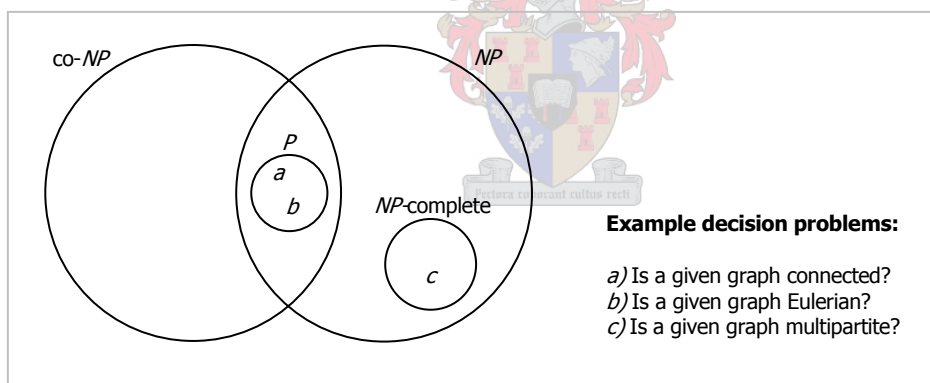


Figure 2-3: The relation of the different complexity classes

As an example from Chartrand & Oellermann (1993), suppose that a given graph is connected and has some edge e . The problem of deciding whether or not the edge e is a bridge of the graph is a decision problem. This problem is easily classified to be in $co-NP$ since it is possible to verify in polynomial time that e is not a bridge if given a cycle in the graph containing the edge e . It is also possible to verify in polynomial time that e is indeed a bridge of the graph if given some pair of vertices (u, v) of the graph such that no $u - v$

* Note that such an algorithm may exist. It is just not known.

path exists in $G - e$ where G is the graph. Therefore, the decision problem is also in the class NP . Figure 2-4 shows how and when an edge is a bridge of a connected graph.

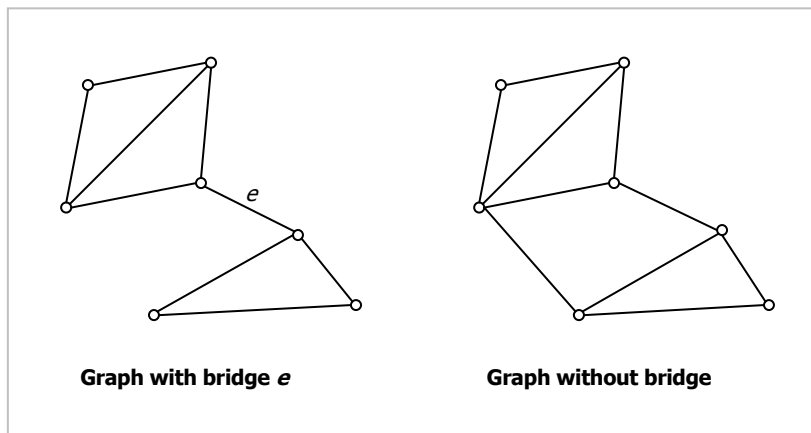


Figure 2-4 Two connected graphs with and without a bridge

Some interesting theorems from decision theory are listed below (Chartrand & Oellermann, 1993):

1. If L_1 is NP -complete & $L_1 \in P$, then $P = NP$
2. If $L_1 \in NP$, L_2 is NP -complete & $L_2 \preceq L_1$, then L_1 is NP -complete
3. If L_1 is NP -complete & $L_1 \in co-NP$, then $NP = co-NP$

The significance of the second theorem above is that, in order to prove the NP -completeness of a problem, it is only necessary to show that it is NP -hard. All of these theorems and statements on complexity classes lead to the most important open question in theoretical computer science. So far no way has been found to tell whether $P = NP$ or $P = co-NP$. However, the level of mathematics needed to tackle such a problem is not needed in this text. A moderate understanding of complexity and decision theory will certainly be enough to get some insight into the placement of the problem under study within the huge number of combinatorial problems.

2.3 Vehicle Routing and Scheduling Problems

The vehicle routing problem (VRP) is actually a generic name given to a class of problems in which customers are visited by vehicles. The standard problem was first formulated by

Dantzig and Ramser in 1959. Due to the complex combinatorial nature of the problem, it is not practical to use exact approaches for large instances of the VRP. Most approaches to the VRP rely on heuristics and give approximate solutions to the problem. Some of the most familiar cases of the VRP will be discussed in the next few sections.

2.3.1 The Capacitated Vehicle Routing Problem

The most elementary case of the VRP is the Capacitated Vehicle Routing Problem (CVRP), also called the Classic Vehicle Routing Problem. Given a certain depot and several customers, unique demands for the customers and a fixed fleet of vehicles with uniform capacity, the goal of the problem is to determine a set of routes for the vehicles to follow that will minimize the total route cost and leave all customers serviced without exceeding the capacity limit of any vehicle. A route must start and end at the depot. The problem is depicted in Figure 2-5.

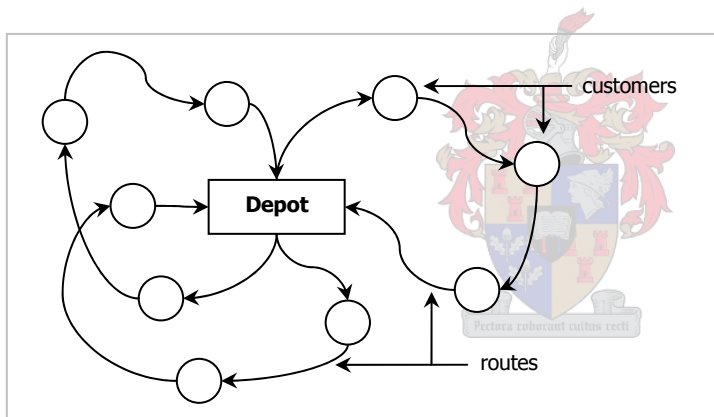


Figure 2-5: The Capacitated Vehicle Routing Problem

Formally, the CVRP can mathematically be expressed in the following way. One central depot v_0 uses m independent delivery vehicles, each with an identical delivery capacity C . These vehicles have to service known demands d_i from n customers v_i , $i = 1, \dots, n$. The vehicles must accomplish the delivery with a minimum total cost where the length w_{ij} is the time from customer i to customer j with $i, j \in [1, n]$. The times between customers is symmetric *i.e.* $w_{ij} = w_{ji}$, and $w_{ii} = 0$. Note that time can easily be substituted by distance. A solution to the CVRP would be a partition $\{R_1, \dots, R_m\}$ of the n demands into m

routes, each route R_s ($1 \leq s \leq m$) satisfying $\sum_{p \in R_s} d_p \leq C$. Associated with each R_s is a permutation of the demands belonging to it, specifying the delivery order of the vehicles. The length of a route $R_i = (v_0, \dots, v_{m(i)}, v_{m(i)+1})$ with $v_0 = v_{m(i)+1}$ is given by

$$\delta(R_i) = \sum_{p=0}^{m(i)} w_{p,p+1} \quad \dots(2.2)$$

This problem is naturally associated with the complete undirected graph consisting of vertices $V \cup \{v_0\}$, edges E and edge-traversal costs w_{ij} , $\{i, j\} \in E$. The solution in this graph will be the union of m cycles that all intersect at the depot, v_0 . If a binary variable x_e is associated with each edge e of the graph, the following integer programming (IP) problem is obtained (Ralphs *et al.*, 2002).

Minimize $\sum_{e \in E} w_e x_e$ subject to the following constraints:

$$\sum_{e=\{0,j\} \in E} x_e = 2m \quad \dots(2.3)$$

$$\sum_{e=\{i,j\} \in E} x_e = 2 \quad \forall i \in V \quad \dots(2.4)$$

$$\sum_{\substack{e=\{i,j\} \in E \\ i \in S, j \in S}} x_e \geq 2\tau(S) \quad \forall S \subset V, |S| > 1 \quad \dots(2.5)$$

$$0 \leq x_e \leq 1 \quad \forall e = \{i, j\} \in E, i, j \neq 0$$

$$0 \leq x_e \leq 2 \quad \forall e = \{0, j\} \in E$$

$$x_e \text{ integral } \forall e \in E$$

Constraints (2.3) and (2.4) are the degrees constraints. An obvious lower bound on the number of vehicles m needed to service all the customers in set S is defined by $\tau(S) = \lceil (\sum_{i \in S} d_i) / C \rceil$. Now constraint (2.5) enforces the connectivity of the solution and ensures that no route has total demand exceeding the capacity C .

The CVRP has both the Bin Packing Problem (BPP) and the Travelling Salesman Problem (TSP) as special cases and conceptually lies at the intersection of these two well-studied NP -complete problems. For example, by setting $C = \infty$ we get an instance of the Multiple Travelling Salesman Problem (MTSP). Since the TSP is NP -complete (Michalewicz & Fogel, 2002) and the BPP is NP -hard (De Marco *et al.*, 2003), it is an indication that the CVRP might at least be an NP -complete problem. Rinaldi *et al.* (1995) have proved that it is NP -complete. In fact, just solving the complete relaxation of the formulated IP problem is a NP -complete problem. After multiple attempts to solve the CVRP, both exact and approximate methods have emanated from research. In 1994, Fisher proposed a branch and bound approach to find an exact solution to the instance where the number of customers does not exceed a hundred. A number of approximate methods have been developed, like those by Clark & Wright (1964), Taillard (1999) and Kinderwater & Savelsbergh (1997). Tabu search (Kelly & Xu, 1996), ant systems (Gambardella, 1999) and Constraint programming (Shaw, 1998) meta-heuristics respectively provide some smarter techniques to solve the VRP.

2.3.2 The Vehicle Routing Problem with Time Windows

The Vehicle Routing Problem with Time Windows (VRPTW) has the same definition as the plain CVRP, but has the extra constraints in that each customer has a time window associated with it that defines an interval in which it must be serviced. Even the depot has a time window. All vehicles must start and end within the depot's time interval. This variation of the CVRP is actually a much better representation of what real-world problems encompass. For companies with Just-in-Time (JIT) manufacturing strategies, time is of great importance; it will simply not do to model their distribution system with an ordinary CVRP. Even bank-deliveries, school bus routings and postal deliveries will be modelled much better with the VRPTW.

Two types of time windows are used as constraints in the VRPTW: soft time windows and hard time windows. Hard time windows are the general strict constraints that will not allow a customer to be serviced outside of the interval. Soft time windows have the attribute that the constraint will not prohibit a customer to be serviced after the end of the interval. If a service does occur after the end of a soft time window, it is merely penalized in some

fashion. In the case where a vehicle arrives at a customer before the start of a soft time window, a waiting time arises. A hard time window constraint will not allow any early arrivals and so eliminate all waiting times, although it is possible to modify the definition of the time windows to combine soft and hard time windows.

Define the time interval $[e_i, l_i]$ for customer i of n customers and let b_i denote the service start time at customer i , $i \in [1, n]$. The interval $[e_0, l_0]$ is that of the depot and is generally referred to as the scheduling horizon. Typically, this horizon can be the start and end times of a general working day. A waiting time of $t_p = \max\{0, b_p - b_{p-1} - w_{p-1,p}\}$ is now induced at customer v_p . The cost in terms of time of route R_i is now given by

$$\delta(R_i) = \sum_{p=0}^{m(i)} w_{p,p+1} + \sum_{p=1}^{m(i)} t_p \quad \dots(2.6)$$

In addition to all this, a service time s_i at customer i will leave the waiting time $t_p = \max\{0, b_p - b_{p-1} - s_{p-1} - w_{p-1,p}\}$ and the time cost of route R_i

$$\delta(R_i) = \sum_{p=0}^{m(i)} w_{p,p+1} + \sum_{p=1}^{m(i)} t_p + \sum_{p=1}^{m(i)} s_p \quad \dots(2.7)$$

2.3.3 The Multiple Depot Vehicle Routing Problem

The Multiple Depot Vehicle Routing Problem (MDVRP) is an extension of the classic CVRP. The constraints of the MDVRP are similar to those of the CVRP, except for the requirements that each route starts and finishes at the same depot.

In the MDVRP, the single depot v_0 of the CVRP is substituted with the set a_i , $i \in [1, k]$ of k depots. The most common technique to solve this problem is by way of a two-phase method. First, the customers v_i , $i \in [1, n]$, are clustered into Z different clusters by means of some clustering method. Thangiah & Salhi (2001) introduced an adaptive genetic clustering method that maps a geometric shape on the chromosome of a Genetic Algorithm (GA)*. This method aims to minimize the number of vehicles used in the

*The interested reader may refer to the work of Holland (1992) on artificial systems and genetic algorithms.

routing process, which result in the fact that Z will not necessarily equals k . Secondly, within each of these Z clusters of customers, a simple TSP is then solved, whose solution is then fed to a third and final post-optimization phase.

2.3.4 *The Stochastic Capacitated Vehicle Routing Problem*

The single fundamental difference between the problem under study and the variations of the CVRP discussed so far is the matter of stochastic demand. In all of the mentioned variations, the demand d_i for customer v_i is deterministic. In reality, there are not many applications where demand is deterministic. The Stochastic Vehicle Routing Problem (SVRP) provides a welcome approach to model those situations where the delivery vehicle will only know the customer's demand when it arrives at the customer.

Gendreau *et al.* (1997) presented a model for the SVRP and in addition a technique to solve the SVRP in two consecutive steps. Assume that all demands are discreet and independent, then each customer v_i has a stochastic demand \mathcal{G}_i as opposed to the deterministic demand d_i of the CVRP. Each customer also has a probability p_i of being present. When $p_i = 0$, that is to say customer v_i is absent, then \mathcal{G}_i is zero. It is a good question to ask why the demand and presence of a customer are treated separately. The reason is this: since $\mathcal{G}_i = 0$ implies that $p_i = 0$ when $\mathcal{G}_i = 0$ is discrete, the state of the customer is in principle fully described by \mathcal{G}_i . However, information about customers where $\mathcal{G}_i = 0$ is known at an earlier time and these customers are never visited. As a result, a customer's demand and presence affect the solution differently.

In the CVRP, routes are constructed so that

1. Each route starts and ends at the depot,
2. Each customer is serviced exactly once by one vehicle,
3. The total deterministic customer demand on a route does not exceed C .

In the first phase of the two-phased technique, routes are constructed in the same way it is for a CVRP, satisfying the first two conditions. The presence or absence of a customer is revealed at the latest upon leaving the preceding customer, but the nonnegative demands of

the remaining customers become known only when the vehicle arrives at the customer. In this manner, the certainty of deterministic demands is replaced by a stochastic demand for each customer. In the second phase, the routes are followed as planned, with the following exceptions (assuming the vehicles are collecting goods from the customers):

1. Any absent customer is not visited.
2. Whenever the vehicle capacity is exceeded or exactly attained, it returns to the depot to unload, and resumes collections starting at the last visited customer whose goods have not yet been fully collected.

A solution to the SVRP will then be a first stage solution that minimizes the cost of the second stage solution. This brings in effect the extra cost of trips back to the depot due to a full vehicle. It is obvious that the probability of a vehicle's capacity being exceeded or attained is not known in the first stage of the solution. Only in the second stage, and then only when the vehicle arrives at the customer, will the vehicle know if its capacity is going to be exceeded or attained after collecting the customers' goods.

Suppose, however, that the demand at the customers is samples from certain identifiable probability distributions. Now it might be possible to construct routes in the first phase that satisfies the third constraint of the CVRP. The expected demand $E(X) = \sum_{x \in A} x f_{X,i}(x)$ (obtained from the specific customer's demand distribution $f_{X,i}(x)$) can be used as a deterministic demand value only for the first phase. The second phase will then continue in the usual fashion, drawing the customer's demand randomly from the distribution $f_{X,i}(x)$.

2.3.5 The Pickup and Delivery Problem

Where some cases of the VRP was discussed in the last few sections, the Pickup and Delivery Problem (PDP) is discussed here. In the PDP, vehicles have to transport loads from origins to destinations without transshipment at intermediate locations (Savelsbergh & Sol, 1995). Again, a fleet of m vehicles is available to perform the tasks, and each vehicle has a capacity C , a start and a finish location. A transportation request consists of the following:

1. The size of the load to be transported.
2. The location where the load has to be picked up.
3. The location where the load has to be dropped off.

Each load may be transported by only one vehicle from its origin to its destination. Three variations of the PDP exist. The General Pickup and Delivery Problem (GPDP) specifies only one origin and one destination for each vehicle. All the vehicles also depart from and return to a depot. This variation is equivalent to the VRP with Pickup and Deliveries (VRPPD). The Dial-a-Ride Problem (DARP) is a PDP where people represent the loads. The size of a load can then only be one, and only one origin and one destination can be specified. The third variation is the VRP, which is actually a PDP where either all the origins or all the destinations are located at the depot.

The following problem formulation for the PDP is taken from the work of Savelsbergh & Sol (1995). Let N be the set of transportation requests. For each transportation request $i \in N$, a load of size $\bar{q}_i \in \mathbb{N}^+$ has to be transported from a set of origins N_i^+ to a set of destinations N_i^- . Each load is subdivided as follows:

$$\bar{q}_i = \sum_{j \in N_i^+} q_j = - \sum_{j \in N_i^-} q_j \quad \dots(2.8)$$

This suggests positive quantities for pickups and negative quantities for deliveries. Define $N^+ = \cup_{i \in N} N_i^+$ as the set of all origins and $N^- = \cup_{i \in N} N_i^-$ as the set of all destinations. Let $V = N^+ \cup N^-$. Furthermore, let M be the set of vehicles. Each vehicle $k \in M$ has a capacity $Q_k \in \mathbb{N}^+$, a start location k^+ and an end location k^- . Define $M^+ = \{k^+ \mid k \in M\}$ as the set of start locations and $M^- = \{k^- \mid k \in M\}$ as the set of end locations. Let $W = M^+ \cup M^-$. For all $i, j \in V \cup W$ let d_{ij} denote the travel distance, t_{ij} the travel time and c_{ij} the travel cost. The dwell time at origins and destinations can easily be incorporated in the travel time and therefore will not be considered explicitly.

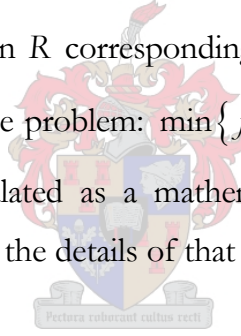
A pickup and delivery route R_k for vehicle k is a directed route through a subset $V_k \subset V$ such that the following six conditions hold:

1. R_k starts in k^+
2. $(N_i^+ \cup N_i^-) \cap V_k = \emptyset$ or $(N_i^+ \cup N_i^-) \cap V_k = N_i^+ \cup N_i^-$ for all $i \in N$
3. If $N_i^+ \cup N_i^- \subseteq V_k$, then all locations in N_i^+ are visited before all locations in N_i^-
4. Vehicle k visits each location in V_k exactly once
5. The vehicle load never exceeds C_k
6. R_k ends in k^-

A pickup and delivery plan is a set of routes $R = \{R_k \mid k \in M\}$ such that the following two conditions hold:

1. R_k is a pickup and delivery route for vehicle k , for each $k \in M$
2. $\{V_k \mid k \in M\}$ is a partition of V

Define $f(R)$ as the price of plan R corresponding to a certain objective function f . We can now define the GPDP as the problem: $\min \{f(R) \mid R \text{ is a pickup and delivery plan}\}$. The GPDP can also be formulated as a mathematical program. The formulation is, however, a very tedious one, and the details of that are left for exploration by the interested reader.



This concludes the discussion of vehicle routing and scheduling problems. The next section covers the Population-Based Incremental Learning algorithm.

2.4 The Population-Based Incremental Learning Algorithm

The PBIL algorithm was developed by Shumeet Baluja in 1994. The PBIL algorithm is an extension to the Equilibrium Genetic Algorithm (EGA) and combines several components from genetic search algorithms and competitive learning for function optimization (Baluja, 1994). The version of the PBIL algorithm described in this section is a very elementary one, and serves only as a base from which to understand the tailored version presented later on in the text. Also, note that many of the concepts in this section will be explained by example and that most of the concepts presented originate from the work by Baluja.

The algorithm is presented in Figure 2-6 (Bekker & Daronnat, 2004) by pseudo-code. Each of the components mentioned in the figure will subsequently be introduced and discussed.

```

1 Start Algorithm
2 Initialization
3 While ( $i < \text{NumberOfGenerations}$ )
4   For  $j = 1$  To  $\text{PopulationSize}$ 
5     Generation of solution vectors
6     Evaluation of solution vectors
7   Next  $j$ 
8   Find the best solution vector  $B$ 
9   Update probability vector  $P$  according to  $B$ 
10  For  $j = 1$  To  $\text{NumberOfElementsOfP}$ 
11    If ( $\text{Random}(0,1) < \text{MutationProbability}$ ) Then
12      Mutate  $P(j)$ 
13    End If
14  Next  $j$ 
15   $i = i + 1$ 
16  If Converged Then
17    End Algorithm
18  End If
19 End While
20 End Algorithm

```

Figure 2-6: The PBIL algorithm

2.4.1 Choice of algorithm

The question may be asked whether any other meta-heuristic would also have been suitable for the problem under study. In the author's opinion, the answer is debatable. The reason is one that has been the topic of many discussions in the past. Other evolutionary algorithms implement a population diversification step called crossover that is not present in the PBIL algorithm. It is the presence of this crossover, in the author's opinion, in some other meta-heuristics that make them infeasible for implementation in the manner presented in this thesis, for the problem under study. However, the implementation presented here might ultimately be improved by introducing some sort of crossover. Since this topic falls outside the scope of the thesis, no further comments are made on it, and is left for further research in future.

The most elementary case of crossover, single-point crossover, is best described by means of an example and is done here to give a simple understanding of what crossover constitutes of and why there might be difficulties bringing it and the problem under study together. Suppose that two routes are represented as shown in Figure 2-7, where each

letter represents a stop-over. Note that the first and last stop-over of each route is the same, making the routes circular routes. The single-point crossover of these two routes constitutes that a random position in the routes be found, and the parts of the routes be swap. This process now present two totally new routes. Although this is a good way to introduce new solutions, it is not an organized way with which to search for alternative solutions and translates into a random search of the solution space when the solutions is routes as in Figure 2-7.

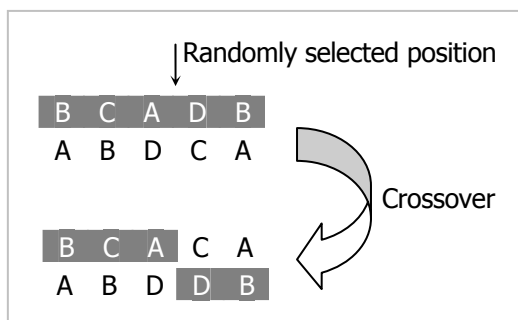


Figure 2-7 The crossover of routes

The PBIL algorithm provides a way to search the solution space in an organized way, not using crossover but using competitive learning and mutation. The representation of a route as shown in Figure 2-7 can be used in the PBIL, and will be discussed next.

2.4.2 Representation

In the PBIL algorithm, a solution is represented as a binary encoded string. When solution to a problem is some decimal value, the encoding of this value into a binary string is certainly a very popular method. Although the PBIL algorithm can be applied to higher-cardinality representations and also other types of solutions, the binary alphabet will be considered for the remainder of this section. Although the problem under study does not have a decimal value for a solution, a binary representation is also used to represent a solution of routes (see section 4.2).

The equivalent binary string representation of the decimal value $x = 9$, is “1001” using only four bits. Using two bits, the value $x = 9$ cannot be encoded into a binary string, since the maximum decimal value that can be represented with a binary string of two bits is

$2^2 - 1 = 3$. If the sign of the value has to be taken into account, an extra bit will be necessary to accommodate it. The maximum decimal value that can be represented with a binary string of four bits is $2^4 - 1 = 15$.

An encoded binary string can also be seen as a linear array of binary digits in which case it will generally be referred to as a solution vector. If in some problem, two variables x_1 and x_2 have to be encoded, each with four bits, it can be represented as shown in Figure 2-8 by concatenating two binary strings.

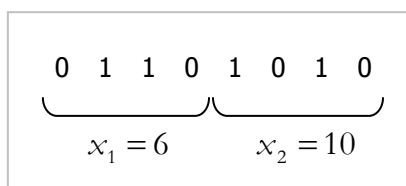


Figure 2-8: Binary representation of two variables

These binary strings, or solution vectors, are also sometimes referred to as *chromosomes*, although this terminology is more generally used in terms of genetic algorithms. Each bit in such a chromosome is then called an allele.

2.4.3 The Population Concept

Figure 2-9 shows a set of solution vectors. The solution vectors are four-bit encoded and the corresponding minimum and maximum values that can be represented by this encoding are zero and fifteen ($2^4 - 1$). Note that the direct decoding of a binary number will always result in an integer value. This set of solution vectors is called a population of solution vectors.

Population of solution vectors

0	0	1	1
1	1	0	0
1	1	0	0
1	0	1	0
0	1	0	1
0	0	1	1

Figure 2-9: A population of solution vectors

2.4.4 The Probability Vector

Additional to the three populations in Figure 2-10 is a probability representation for each of the populations. Such a probability representation is called a probability vector (PV). The process just described in the previous paragraph is used in reverse by the PBIL algorithm when constructing a population of solution vectors. Rather than passively transforming each population into a probability vector, from which solution vectors are generated, the aim of the algorithm is to actively create a probability vector that, with high probability, represents a population of high evaluation solution vectors.

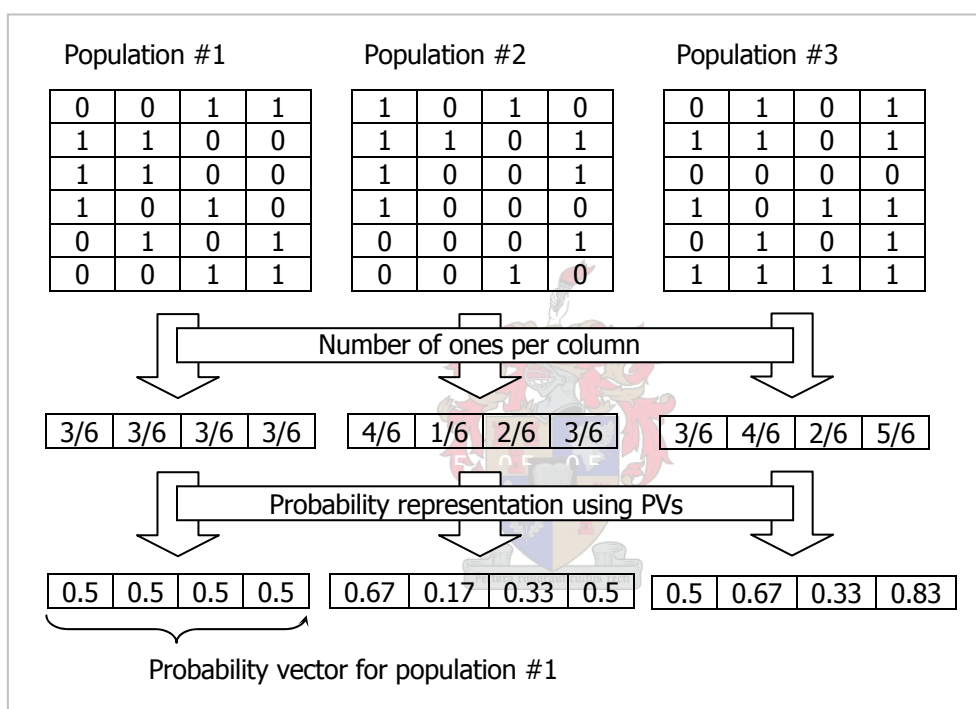


Figure 2-10 Probability representation of a population of solution vectors

In other words, the algorithm attempts to create a probability vector that can be considered a prototype for high evaluation vectors for the function space being explored. The initial probability vector, from which the first population's solution vectors will be generated, is initialized with each bit equal to 0.5. In this way, the search process is started completely randomly (Baluja, 1994).

In a binary encoded solution string, the probability vector specifies the probability of each bit position containing a '1'. The probability of the bit position containing a '0' is obtained

by subtracting the probability specified in the probability vector from one (Baluja, 1994). It is on this PV that all further operations in the algorithm are done. Note that one probability vector can specify many different populations.

2.4.5 Intelligence in Searching

There is no sense in continually searching the same area of a solution space, since a better solution may occur elsewhere in the solution space. In some way, the random search of the solution space has to be led in preferably the direction of high evaluation solutions. The algorithm should intelligently search the solution space. When an algorithm based on intelligent search methods starts, it is basing its decisions upon random information. Methods exist that will take the output generated by the algorithm, and train it to improve its intelligence. The method implemented in the PBIL algorithm is drawn from the weight update rule in a competitive learning network. The same formula is also used in some forecasting models, where the decision to be made is how much weight is assigned to historical data.

The probability vector provides the information upon which the algorithm bases its decisions, and it is the fittest solution vector from the last generated population that is used to train or update this probability vector. Then, based on this trained or updated probability vector, a new population is generated and the cycle is continued. The fitness of a solution vector is how well, after being decoded, it optimizes the objective function.

$$probability_i^{new} = (probability_i^{old} \times (1.0 - LR)) + (LR \times vector_i) \quad \dots(2.9)$$

In (2.9), $probability_i$ is the probability of generating a '1' in bit position i of a solution vector. LR is the learning rate and $vector_i$ is the i^{th} bit position in the solution vector which the probability vector is moved towards. The values in the probability vector will move away from 0.5, towards 0.0 or 1.0 as the search progresses. This progression in search causes the probability vector to converge around a single point.

2.4.6 Evaluation

Each solution vector in a population has to be evaluated to see whether it is a good potential solution or a bad one. In our example, the evaluation of a solution vector will

simply be the decoding of the vector to a decimal value for x after which the objective function value $f(x)$ can be determined. The fitness of solution vectors are measured relative to each other, so that the one leading to the most optimal objective function value is the fittest. This potential solution is then used to train the probability vector.

Objective functions and the development of a fitting objective function is a field of study on its own. Care must be taken in simply stating an objective function. In the field of numerical function optimization, the objective function is in most cases a given. In scheduling and routing type problems, the possibilities for an objective function are limited. In some cases, the distance that a package is to travel can be minimized to curtail damages to the package. In still other cases, the time taken for a vehicle to travel between two points can be minimized, or the cost or distance. Utilization of vehicles may also serve in an objective function, and all of these possibilities may even be combined in some way. As will be discussed in section 3.8, the problem under study is optimized using an objective function that aims to minimize the time that is taken to deliver a package.

2.4.7 Mutation

One would notice that the further the search continues, the nearer the values in the probability vector will get to either 0.0 or 1.0. In effect, the probability that similar solution vectors will be created in a population increases *i.e.* the genetic diversity of the population is decreasing. The role of mutation is more pronounced in the latter stages of search, when diversity in the population is lost (Spears, 1992). This phenomenon might sometimes lead to premature convergence, sometimes around a local optimum in the solution space. The ideal situation would be to maintain diversity in the populations, right throughout the search process, letting only the probability vector to converge to an optimal solution. An extensive discussion of the role of mutation in extensive search can be found in Spears (1992) and Fogel (1993).

In the PBIL algorithm, the use of a probability vector that learns from fit solutions in itself provides some comfort in this situation. The learning rate has a great effect on the speed of convergence. In addition, since the population represented by the probability vector is not unique, the diversity of the search is maintained. There is, however, one other method to

utilize and ensure that genetic diversity is maintained within a solution. This is called mutation.

Mutation can be implemented in two ways. The first is to perform the mutation directly on the generated solution vectors, the other to perform the mutation on the probability vector itself. The second method will be discussed here, since this version will be implemented later in our problem.

$$probability_i^{new} = (probability_i^{old} \times (1.0 - MS)) + (random(0,1) \times MS) \quad \dots(2.10)$$

In (2.10), $probability_i$ is the probability of generating a '1' in bit position i of a solution vector. MS is the mutation shift which is the amount that the mutation affects the probability vector. A typical value for the mutation shift is 0.05 (Bekker & Daronnat, 2004). A mutation at bit position i will occur with a predefined probability specified by the mutation probability (MP). A typical value for the MP is 0.02.

2.4.8 Termination

As one can deduce from the pseudo-code listed in Figure 2-6, the algorithm is terminated in one of two ways. Either the algorithm terminates because it has converged to a final solution, or the number of iterations (also called generations in genetic search) has reached a set limit. The condition that has to be satisfied before it can be said that the algorithm has converged, is this: all values in the probability vector have to be either higher than a certain value (upper threshold) or lower than a certain value (lower threshold). The margins at the high and low thresholds are typically set to 0.05, so that when the algorithm terminates with all of the probability vectors' values being either below 0.05 or above 0.95, the low values are set to 0.0 and the high values to 1.0. This updated probability vector then represents the final solution vector.

2.4.9 Summary

Meta-heuristics are usually used for problems of which the solution space is very large. In these cases, it is usually impossible to tell if the found solution vector is a global optimum or a local optimum, so it is vital to implement the correct heuristic in a correct manner.

The concept of a global optimum versus a local optimum and the working of the algorithm is now demonstrated by an example. Suppose that a function in x is given as $f(x) = (x-6)^3 - (x-6)^2 - 30x + 180$ with the objective to find the integer value of x for which $f(x)$ will be a maximum for $x \in [0,15]$. Now, it is certainly not necessary to use a meta-heuristic to find the solution since it can be solved analytically. One can easily see that the function reaches a maximum of $f(x) = 378$ at $x = 15$. This is called the global optimum. At $x = 3$, however, the function also reaches a local optimum of $f(x) = 54$. A graph of the function is shown in Figure 2-11 for the integer values $0 \leq x \leq 15$.

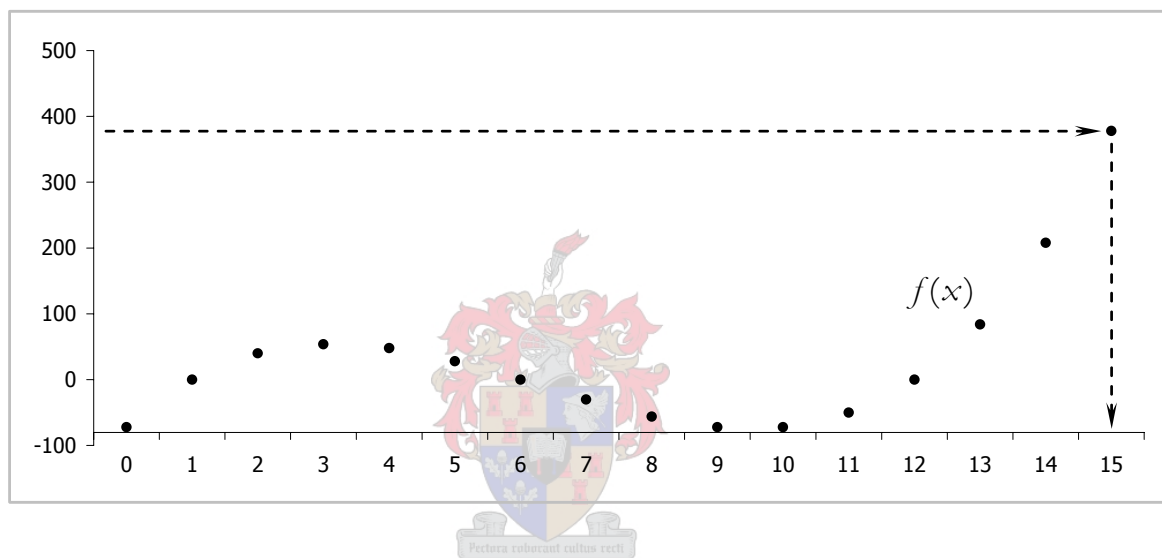


Figure 2-11: The function $f(x) = (x-6)^3 - (x-6)^2 - 30x + 180$ for $0 \leq x \leq 15$

Note that the PBIL algorithm does not have any *a priori* knowledge of what the optimal solution to the problem is. In this case, the problem would have already been solved. The goal of the PBIL algorithm is to search the solution space (possible integer values of x over the definition range of $f(x)$) and return the optimal solution.

The first step in the execution of the algorithm is to generate a population of possible solutions that is based on the probability vector. The probability vector is initialized with a value of 0.5 in each cell. For this example, let the size of the population be six. Figure 2-12(a) shows the first population generated based on the probability vector. Note that the population's columns do not always obey the probability vector, since each bit position is sampled randomly. Of course, one can expect the population to obey the probability

vector more strictly with an increase of the population size. Each of the solution vectors in the first population is now evaluated. The binary strings are decoded to decimal values, and the objective function value $f(x)$ is determined for each decimal value. In the first population, the third solution decodes to a value of $x=15$, and delivers a value of $f(15)=378$. Of all the solution vectors in this population, this one evaluates to the highest value. As mentioned before, this is also the global maximum for $f(x)$ (the algorithm does not know it yet), and so the probability vector will be updated according to the formula in (2.9) using this solution vector (Figure 2-12(b)). A learning rate of 0.3 will be used for this example.

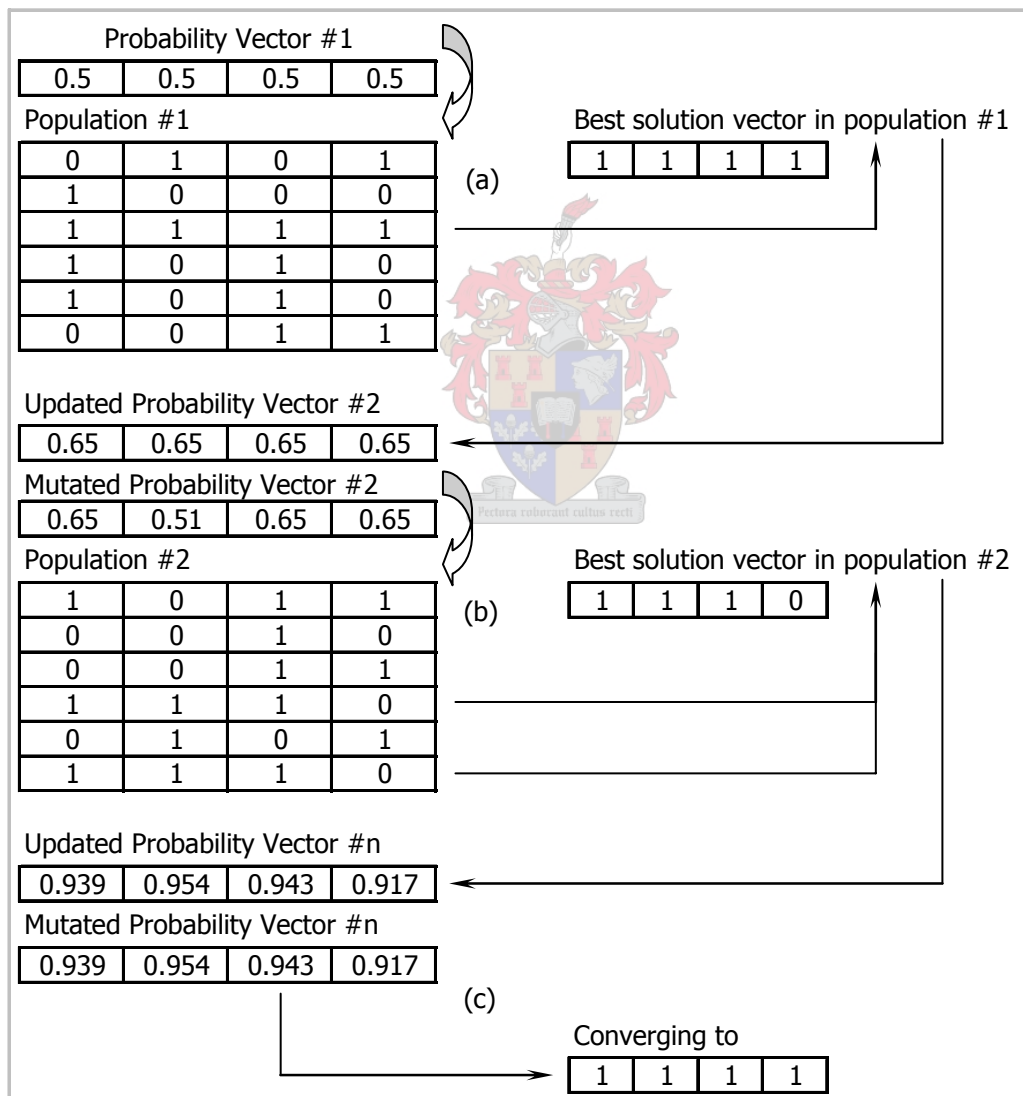


Figure 2-12: A population of solution vectors generated from a probability vector

After the probability vector is updated, each cell of the PV will be given a chance to mutate (based on a mutation probability of 0.02 and a mutation shift of 0.05 for this example) according to the formula in (2.10). By chance, it is only the second cell that is mutated. In fact, the probability that at least one out of the four cells will mutate with a mutation probability of 0.02 is shown in the calculation in (2.11).

$$\begin{aligned}
 P &= \sum_{i=1}^4 \binom{4}{i} (0.02)^i (0.98)^{4-i} \\
 &= 1 - \binom{4}{0} (0.02)^0 (0.98)^4 \quad \dots(2.11) \\
 &= 1 - (0.98)^4 \\
 &= 0.0776 \approx \frac{1}{13}
 \end{aligned}$$

The mutated probability vector is now used to generate the second population of solution vectors for which the whole evaluation procedure is repeated. The cycle continues until some terminating condition has been reached, as indicated in Figure 2-12(c). Note that no mutation took place in the n^{th} updated probability vector. In the n^{th} probability vector, all the values are above the upper threshold of 0.95. From this, we can conclude that the values in all four cells are converging to a value of '1'. These values as indicated represents a decimal value of $x = 15$, which returns the maximum value $f(15) = 378$ as determined analytically.

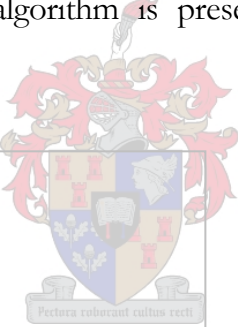
It could have been that the populations created in an instance of the algorithm did not have enough genetic diversity. In such a case the algorithm could have converged to the local optimum of $f(3) = 54$. This is where mutation plays a vital role in the PBIL, even in an elementary problem such as this example.

2.5 Dijkstra's Shortest Path Algorithm

In a given network of routes, streets, railways etc. the question of finding a shortest path between certain given points in the network is always present. It costs money to drive on a longer route than is necessary. In optimizing a set of routes for delivery vehicles to follow, this shortest path question also surfaces. As will be explained in section 3.6 and more formally in section 3.8, the method of finding a shortest path in a network will be applied in

the optimization of the problem under study. The main reason why it is included here is to let the reader gain a better understanding of what such an algorithm does, and what the complexity of it is.

Dijkstra's algorithm, named after its inventor, the Dutch computer scientist Edsger Dijkstra, is an algorithm that solves the shortest path problem for a directed graph with nonnegative edge weights (Wikipedia, 2004). Given a graph with n nodes, the worst-case running time of the algorithm is $O(n^2)$. This worst-case time complexity is usually seen with dense graphs, while the running time with sparse graphs can be considerably faster. The algorithm is used to find the shortest (or lowest cost) path from one vertex to another in a graph. In fact, it constructs a shortest-path tree from an initial vertex to every other vertex in the graph, leaving a shortest path from the initial vertex to every other vertex. In short, this means that a set of shortest paths $\{(v_0, v_i)\}$ with $1 \leq i \leq n - 1$ is generated, where v_0 is the initial vertex. The algorithm is presented in pseudo-code in Figure 2-13 (Wikipedia, 2004).



```

1 Start Algorithm
2 For Each Vertex  $v$  in Graph
3    $d[v] = \text{infinity}$ 
4    $\text{previous}[v] = \text{undefined}$ 
5 Next  $v$ 
6  $d[v_0] = 0$ 
7  $S = \text{Empty Set}$ 
8  $Q = \text{Set of all Vertices}$ 
9 Do While  $Q$  is not an empty set
10   Extract Vertex  $u$  with minimum  $d[u]$ 
11    $S = S \text{ union } u$ 
12   For Each Edge  $(u, v)$  outgoing from  $u$ 
13     If  $d[v] > d[u] + w(u, v)$  Then
14        $d[v] = d[u] + w(u, v)$ 
15        $\text{previous}[v] = u$ 
16     End If
14   Next  $(u, v)$ 
15 End While
16 End Algorithm

```

Figure 2-13: Dijkstra's Shortest Path Algorithm

The algorithm works by keeping for each vertex v_i the cost $d(v_i)$ of the shortest path found so far. Initially, this value is zero for the source vertex v_0 and infinity for all other vertices, representing the fact that no path is known leading to those vertices. When the

algorithm finishes, $d(v_i)$ will be the cost of the shortest path from v_0 to v_i , or infinity if no such path exists. The algorithm maintains two sets of vertices: S and Q . Set S contains all vertices for which we know that the value $d(v_i)$ is already the cost of the shortest path and set Q contains all other vertices. Set S starts empty, and in each step one vertex is moved from Q to S . This vertex is chosen as the vertex with lowest value of $d(v_i)$.

Again, the working of the algorithm will be discussed and demonstrated with an example. Consider a park where a network of small streams divides the park into six separate parts. Over these streams are small bridges providing a way to cross them. Consider a graph as shown in Figure 2-14 mapped on the park and its streams. The distance to walk between two parts of the park is indicated on the edges. One would want to know the shortest path to get from one part of the park to another, so let v_1 be the first part from where the shortest path to all other parts is to be found.

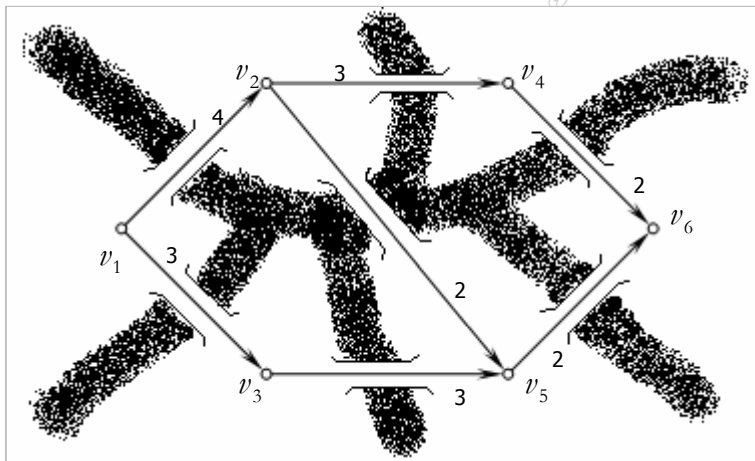


Figure 2-14: A weighted graph with six vertices

According to the algorithm, some information of each part (vertex) of the park should be remembered somehow, so plant a small flag in each part indicating the shortest distance found so far to that part (vertex), and also the last visited part (vertex) en-route to that part (vertex) on the shortest path. A representation of this in table form is shown in **Error! Reference source not found.** Note that for the vertices that has not yet been visited, the distance is infinite and the previous vertex is undefined.

Vertex	Distance	Previous	Visited
v_1	0	-	Yes
v_2	∞	-	No
v_3	∞	-	No
v_4	∞	-	No
v_5	∞	-	No
v_6	∞	-	No

Table 2-2: First iteration in executing Dijkstra's algorithm

From v_1 there are only two ways to go, and both these ways have to be followed to get the shortest distances from v_1 to v_2 and v_3 . In fact, the distances of these two vertices is updated according to the statements found in lines 13 through 16 in Figure 2-13. The distances to these two vertices is only the distance walked from v_1 .

Vertex	Distance	Previous	Visited
v_1	0	-	Yes
v_2	4	v_1	Yes
v_3	3	v_1	Yes
v_4	∞	-	No
v_5	∞	-	No
v_6	∞	-	No

Table 2-3: Second iteration in executing Dijkstra's algorithm

From these last two visited vertices, the one with the flag indicating the smallest distance is chosen, in this case being v_3 . From v_3 one can only go to v_5 , so the data on the flag at v_5 is updated.

Vertex	Distance	Previous	Visited
v_1	0	-	yes
v_2	4	v_1	yes
v_3	3	v_1	yes
v_4	∞	-	no
v_5	6	v_3	yes
v_6	∞	-	no

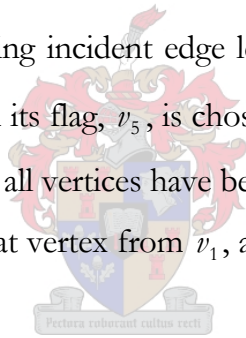
Table 2-4: Third iteration in executing Dijkstra's algorithm

Now, v_1 and v_3 have no outgoing incident edges leading to unvisited vertices, so from $\{v_2, v_5\}$ the one with the smallest distance is chosen. This will be v_2 from where one can go to either v_4 or v_5 . The data of these two vertices is also updated. Note that the distance to v_5 is the same if it is accessed via v_2 or v_3 , so it is not changed. Changing it, however, would not have made a difference to the proper working of the algorithm.

Vertex	Distance	Previous	Visited
v_1	0	-	yes
v_2	4	v_1	yes
v_3	3	v_1	yes
v_4	7	v_2	yes
v_5	6	v_3	yes
v_6	∞	-	no

Table 2-5: Fourth iteration in executing Dijkstra's algorithm

Both v_4 and v_5 have an outgoing incident edge leading to an unvisited vertex, v_6 . The one with the smallest distance on its flag, v_5 , is chosen. From v_5 one can only go to v_6 , so its data is updated. At this point, all vertices have been visited, and have a flag that indicates the shortest distance found to that vertex from v_1 , as well as the previous vertex visited en-route to the vertex.



Vertex	Distance	Previous	Visited
v_1	0	-	yes
v_2	4	v_1	yes
v_3	3	v_1	yes
v_4	7	v_2	yes
v_5	6	v_3	yes
v_6	8	v_5	yes

Table 2-6: Last iteration in executing Dijkstra's algorithm

So, to know the shortest path from, say v_1 to v_6 , it has to be traversed in reverse. Starting at v_6 , one would go to the vertex indicated on its flag, v_5 . From v_5 one would walk to v_3 , and from v_3 to v_1 . This whole procedure then returns a shortest path from a given part of the park (v_1 in this case) to every other part of the park.

2.6 Literature Review Summary

This then concludes the literature review. In this chapter, the most important theories needed to understand the problem under study and the method implemented to solve it is presented. It is not destined to be a fully- and mathematically comprehensive presentation of these theories, however, and serves only as a guide. More information on any of the topics can be found in the documents and textbooks listed in the references.

In the next two chapters, the more specific detail of the problem under study and the implementation of the PBIL algorithm will be discussed.



THE ADAPTED PICKUP AND DELIVERY PROBLEM

3.1 Introduction

The definition of the general Pickup and Delivery Problem and that of the Vehicle Routing Problem provides a useful basis from where a proper definition for the problem under study can be deducted. In this chapter, the technical aspects of the problem will be explained in more detail, and it will formally be expressed as a mathematical model.

3.2 Route Interactivity

One of the aspects of an ordinary Pickup and Delivery Problem causes it to be incompatible with the problem under study. In the PDP, each package that is to be picked up and delivered may only be delivered by the vehicle that picked it up. One might say that no interactivity between delivery vehicles or their routes exists. Vehicles in the PDP also do not have predefined routes to follow. The routes emerge as requests for pickup and delivery emerge. The fact is that if there are a number of delivery vehicles criss-crossing an area, each on its own pre-assigned route, it is very likely that more than one vehicle will visit a certain stocking point in a given time window, either it being for a pickup or a delivery.

A framework of reference where the delivery vehicle is the focal object of attention, may be somewhat limiting in this case. If the focus were shifted from the delivery vehicle to a specific package, what would be the consequence? It might be possible to make use of more than one delivery vehicle to get the package to its delivery point. It can then be seen as if the package is assigned a route comprising of delivery vehicles and stocking points where it is to get on and get off the vehicles. We say that interactivity between vehicle routes now become possible. This suggests that vehicles be able to visit a single SP more than once on its pre-defined route.

Adhering to this framework enables one to minimize the total travel time (section 3.8) of a certain package by assigning different routes to the delivery vehicles. The key part of the problem is still the assignment of routes to vehicles, but the assignments may now be evaluated in terms of package travel times.

3.3 Time Grid Mapping

The one problem with having vehicles to interact with each other is that the times they visit a stocking point may not always overlap. When a package should change from one vehicle to another, it has to be dropped at a stocking point to be picked up later by another vehicle. This allows for some dead time on the package. By introducing a time grid in the problem, a way to synchronize vehicles at stocking points in order for them to exchange packages directly emerges. Vehicles are synchronized at stocking points on certain times of the day. These synchronization times are called the grid points, where the time elapsed between two grid points is called the grid spacing. For instance, look at the example in Figure 3-1. Suppose a normal working day starts at 08:00 and ends at 18:00, then the time 08:00 will be mapped on the grid point 0.

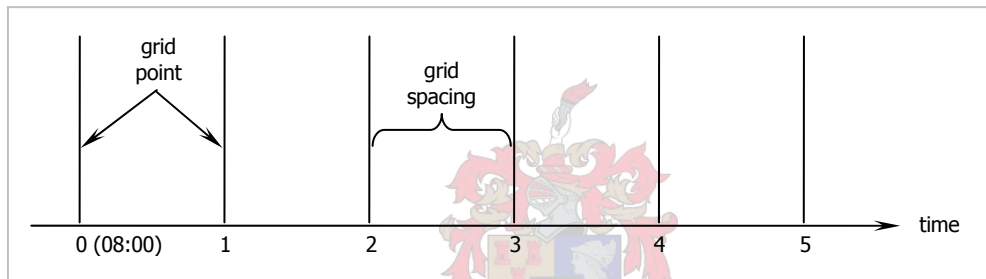


Figure 3-1: A time grid mapping

Suppose further that the ten hours in the day is divided by a grid spacing of 120 minutes (2 hours). There will now be five equally spaced grid spacings. The time 10:00 will now be mapped on grid point 1, 12:00 on grid point 2, and so forth.

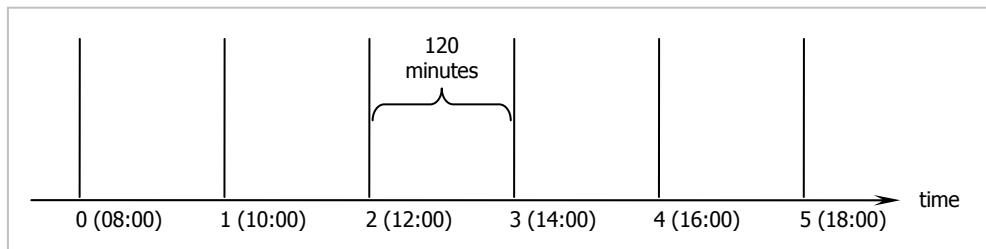


Figure 3-2: Mapping time on the time grid

3.4 Travel, Loading, Unloading & Waiting Times

The time it takes for a vehicle to travel from one SP to another is termed the travel time. Consider four stocking points as in Figure 3-3. The figure shows how the travel time matrix is constructed from the graph.

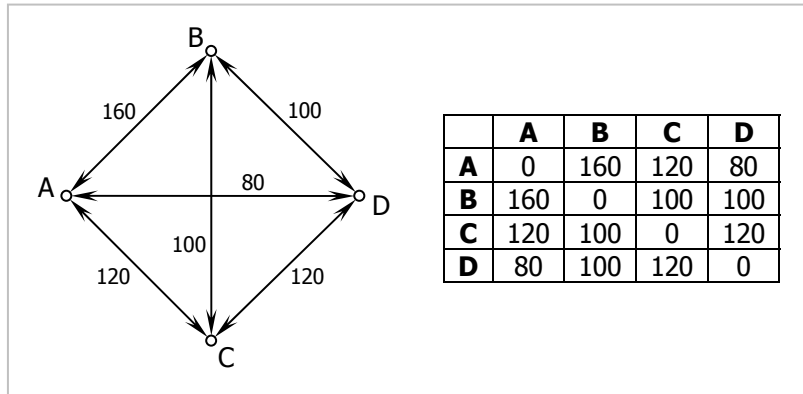


Figure 3-3: Travel times between stocking points

If these stocking points were in some metropolitan area, it would be very unlikely for the travel times to be the same throughout the day. Traffic patterns will have a definite impact on the travel times. In addition, it is also very improbable that a travel time and the return time will be the same. It might be that the vehicle must take a whole other set of highways to drive from B to A than it would to get from A to B. For these reasons, the time matrix need not necessarily be symmetric. As depicted in Figure 3-4, a different time matrix may be specified for each grid time to incorporate the effects of traffic etc.

08:00	A	B	C	D
A	0	60	50	35
B	40	0	30	80
C	50	20	0	65
D	40	90	65	0

10:00	A	B	C	D
A	0	40	50	35
B	40	0	20	80
C	50	20	0	65
D	35	80	65	0

12:00	A	B	C	D
A	0	60	50	35
B	40	0	30	80
C	50	30	0	65
D	45	80	65	0

14:00	A	B	C	D
A	0	40	50	35
B	40	0	20	80
C	50	20	0	65
D	35	80	65	0

16:00	A	B	C	D
A	0	40	50	45
B	60	0	20	90
C	50	30	0	65
D	35	80	65	0

Figure 3-4: A time matrix for every part of the day

Another way to incorporate the effects of traffic patterns etc. into the travel time matrix, is to use a single matrix of which the elements is functions of time. Figure 3-5 shows that the travel time from A to D is subject to some function of time where t is the time of the day for which the travel time is wanted, and a is the total length of the day so that the fraction t/a evaluates to a value between zero and one. The third term in the function represents the statistical beta distribution. This distribution is superimposed on the cosine function. Figure 3-6 shows the function presented in Figure 3-5.

	A	B	C	D
A	$20 + \cos(\pi t/a) + \text{beta}(t/a, 1, 5)$
B
C
D

Figure 3-5 Travel times expressed by functions of time

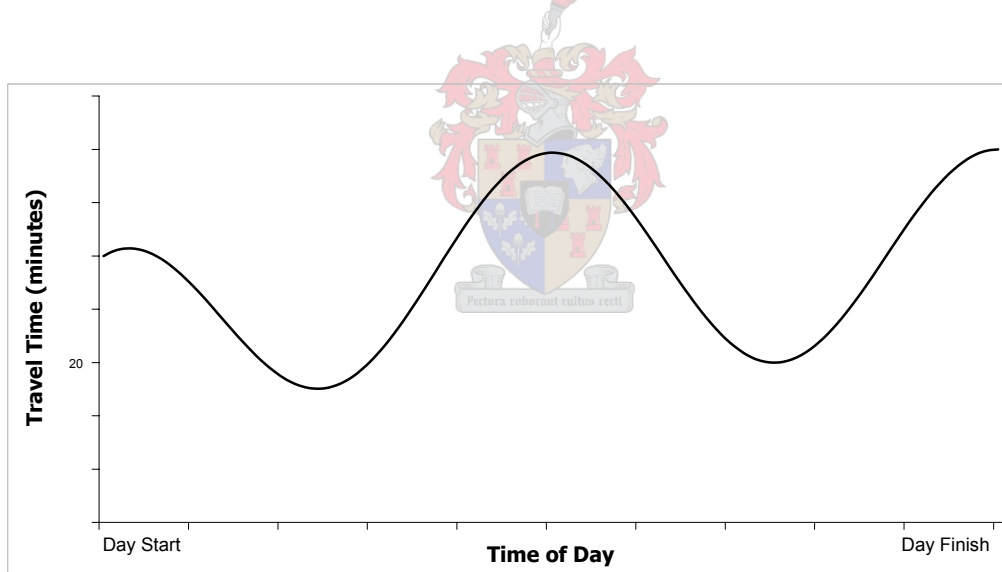


Figure 3-6 Travel times between two stocking points as a function of the time of day

For the sake of simplicity, assume from this point forward that in all explanations and discussions, travel times stay constant during the whole day. If a first vehicle now has the assigned route $A \rightarrow B \rightarrow A \rightarrow C$, and a second the route $C \rightarrow D \rightarrow B \rightarrow C \rightarrow D \rightarrow A$, with the travel times as indicated in Figure 3-3, the visitation of the two vehicles to the four stocking points during the day may be portrayed as in Figure 3-7. From this figure, the

concept of synchronization may become clearer. It actually means that a vehicle may not leave a stocking point before the time of the next grid point.

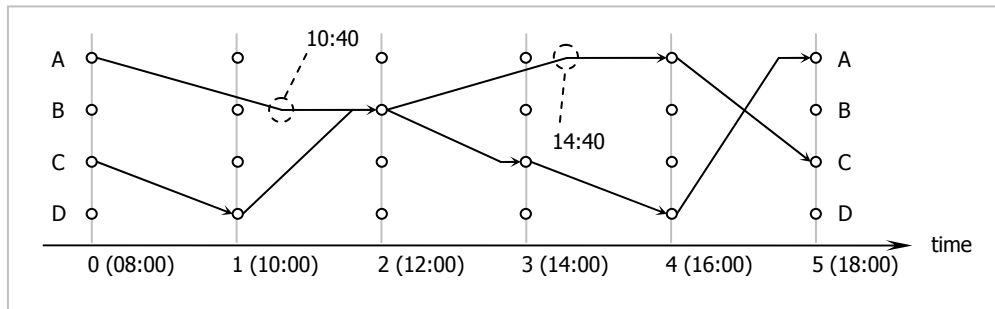


Figure 3-7: Visitation of vehicles to stocking points

If vehicle one travels from point A to point B, it will take 160 minutes. Since the next grid point is only at 240 minutes after the vehicle's departure from A, the vehicle will have to wait for 80 minutes at B before leaving for the next stocking point. These waiting times at stocking points are obviously counting as dead time and are affecting the utilization of the vehicles very badly. These waiting times should be minimized, and is later discussed in section 3.7.

During the first vehicle's waiting at B, the second vehicle will also arrive at B from D. These two vehicles are now synchronized at B and an opportunity arises for packages to be transferred directly between the vehicles. Suppose that at 09:00 an order is placed by A for a spare part only held in stock at D. The earliest that vehicle one arrives at A after 09:00 is at 14:40 (12:00 plus 160 minutes of travel time). It will be sub-optimal just to let the second vehicle pick the part up at 10:00 at D and transport it to A, although there will be no package handling involved. In this way, the part will arrive at A at 17:20. By transferring the part from the second to the first vehicle while they are synchronized at B, the first vehicle will be able to deliver it at 14:40. The synchronization of the two vehicles is shown on a timeline in Figure 3-8.

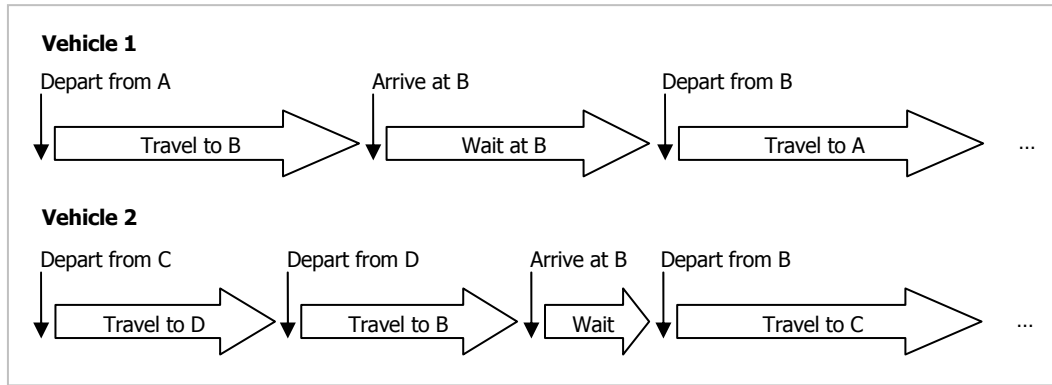


Figure 3-8 Events on two vehicles' routes

The loading and unloading of packages to and from the vehicles also take time and have to be taken into account, since it will affect the departure times of the vehicles and eventually also the waiting times. The easiest way to incorporate loading and unloading times into the model is by simply adding a constant expected value to the travel times*. This expected value may be some percentile from a distribution of loading and unloading times. The average could also be used. Another approach, which may be a tedious one but represent a more realistic case, is to sample the loading and unloading times randomly from a certain distribution at the time of constructing the vehicle routes.

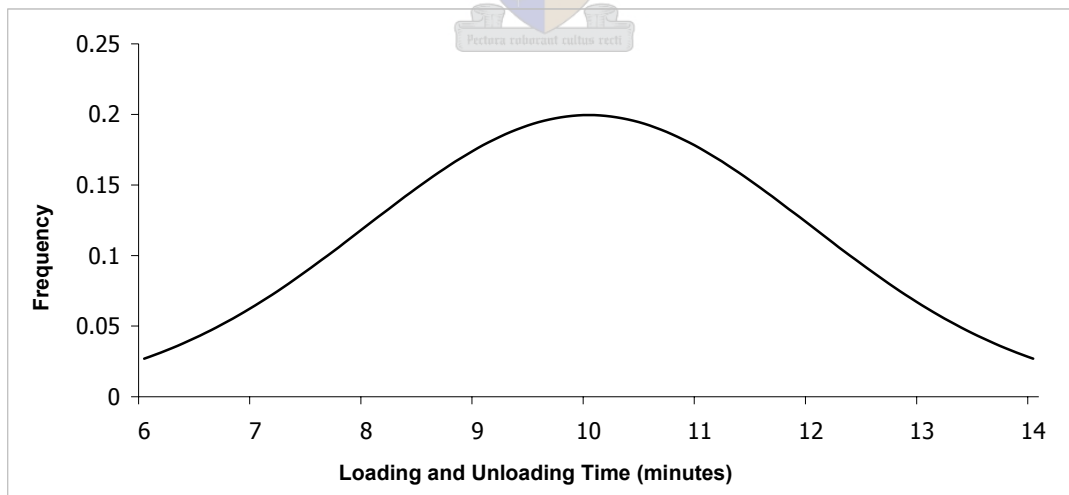


Figure 3-9: Loading/Unloading times distributed normal with a mean of 10 minutes and standard deviation of 2 minutes

* Unless stated otherwise, the travel times indicated in this text already include loading and unloading times.

When loading and unloading times can be seen as coming from *i.e.* a normal distribution as in Figure 3-9, then each time a travel time between two stocking points is requested, a random loading and unloading time can be sampled from the distribution and added to the original travel time.

In the problem under study, the packages that are to be delivered are usually small engine parts that are delivered by small pick-up vans. Therefore, the handling of packages should have a small effect on the total travel time of the package since it can be transferred from one vehicle to another by hand.

3.5 Route Cycling

When a set of routes are planned and assigned to the fleet of vehicles available, the aim is to have these vehicles traverse the same routes every day. So, by definition, a vehicle is assigned only one route per day. When the vehicle has finished its route, it is done for the day. It would be a waste of time and probably other valuable resources to plan and reassign different routes each day. In order to have vehicles follow the same set of routes for many days, a constraint (following in the discussion after Figure 3-10) is introduced with regard to the construction of a set of routes.

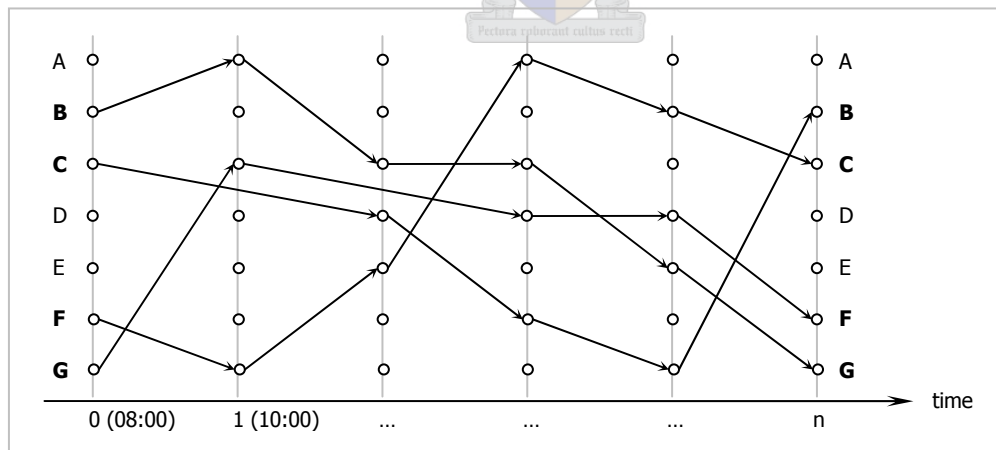


Figure 3-10: The constraining of route start and end points

Figure 3-10 demonstrates the conditions for a set of routes that allow the cycling of the routes from one day to the next. If there is a fleet of k vehicles (which implies that the set of routes consist of k routes), then there exists a collection of stocking points representing

the initial stocking points of each route in the set of k routes. For the set of routes in Figure 3-10, this collection is [B, C, F, G] (shown in bold typeface in the latter figure). To allow the cycling of these routes, each stocking point contained in the first collection, must also be present in the collection of SPs representing the final SPs of each route. This collection is [G, B, C, F]. Note that the order of the SPs may differ from the first to the second collection, meaning that a vehicle does not have to finish the day at the SP where it has started. As long as some other vehicle finishes at that SP, it will traverse the route starting there the following day. If this is not the case, and one route ends at a SP where no route starts, then that vehicle will have to travel back to a SP where some route starts, in reality then changing the route it follows. This travelling back will incur extra travel time, which in the case of a 24-hour working day will cause the following day's route to be late.

3.6 Package Delivery Times

Suppose that four vehicles traverse the routes as presented in Figure 3-10. If an order is placed by C at 09:15 for a spare part held at A, the decision based on everything discussed so far would be to let the vehicle coming from B pick the part up and deliver it, since there is no synchronization with other vehicles in this set of routes. The part will then only be delivered early the next day. However, if the package is allowed to wait, and not be picked up by that vehicle, then the vehicle passing through E can pick it up later. This way, it will be delivered at the end of the same day. This delaying of the pickup of a package is illustrated in Figure 3-11. All of the added edges together with the vehicle routes form a graph network from which a package may now follow a shortest path to its destination stocking point. If a package is to follow one of the broken lines, it means that it will actually be waiting at the stocking point for a vehicle to pick it up later.

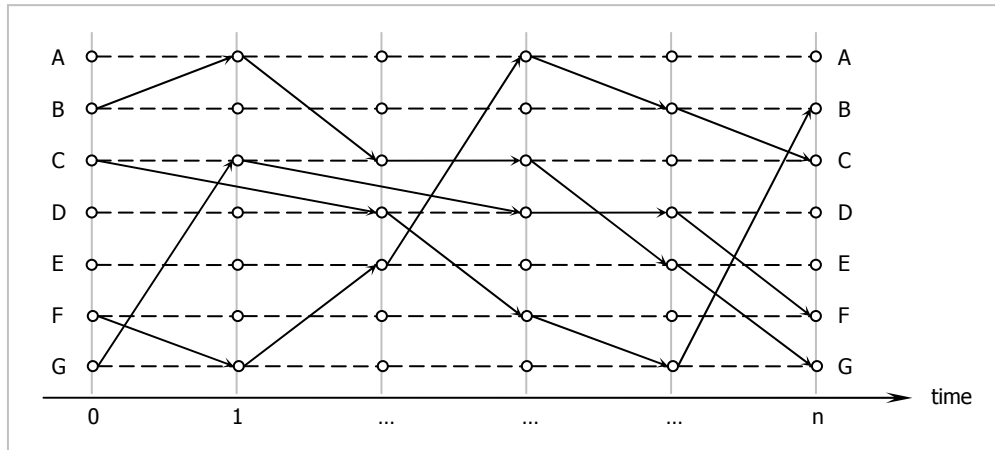


Figure 3-11: Possible paths for a package to follow

3.7 Variability of the Grid Spacing

As shown earlier, the size of the grid spacing has an immediate effect on the waiting times of vehicles. The bigger the grid spacing, the longer a vehicle will have to wait at a SP before departing for the next SP on its route. Initially the answer to this is simply to minimize the grid spacing as to minimize the total waiting time, but small sized grid spacings comes at a price. In Chapter 5, the effect of the grid spacing on the PBIL algorithm running time will be discussed.

To resolve the problem of grid spacing, a method is proposed to choose a sufficient size for the grid spacings. Given is a number of n stocking points along with the travel times t_{ij} between them for $i = [1, n]$, $j = [1, n]$, $i \neq j$. The waiting time of any vehicle at any stocking point may then be calculated as

$$w_{ij} = g \times \left\lceil \frac{t_{ij}}{g} \right\rceil - t_{ij} \quad \dots(3.1)$$

where g is the size of the grid spacings and $\left\lceil \frac{a}{b} \right\rceil$ means that $\frac{a}{b}$ is rounded up to the nearest integer. The average of all these waiting times is then given as

$$W = \frac{\sum_{i=1}^n \sum_{j=1}^n w_{ij(i \neq j)}}{n(n-1)} = \frac{\sum_{i=1}^n \sum_{j=1}^n \left(g \times \left\lceil \frac{t_{ij(i \neq j)}}{g} \right\rceil - t_{ij(i \neq j)} \right)}{n(n-1)} \quad \dots(3.2)$$

The objective is to minimize the average waiting time while at the same time maximize the grid spacing. For this reason, the objective function in (3.3) is introduced. This function has to be minimized to get a value for the grid spacing.

$$f = \frac{W}{g} \quad \dots(3.3)$$

This objective function has a trivial answer of $g = 1^*$. When $g = 1$, all of the waiting times will equal zero. In this case, the ideal situation prevails, which means that this study is redundant. So, one would specify that $g \neq 1$. In section 5.3, the effect of g on the algorithm's running time will be shown, also indicating that small values of g is undesirable. However, for some g where $g_{\min} \leq g \leq \max\{t_{ij}\}$ the objective function as represented in (3.3) will have a definite minimum value. The value for g at this minimum has proven to be a good value for the grid spacing. However, the size of the grid spacings in itself does not have a dominating effect. The combination of the length of the working day and the size of the grid spacings as shown in the relation in (3.4) has a dominant effect.

$$\text{Number of grid spacings} = \frac{\text{length of working day}}{\text{size of grid spacing}} \quad \dots(3.4)$$

Section 5.3 covers some tests that will show that in a worst-case scenario, where the length of a working day is 24 hours, a grid spacing of one hour is quite acceptable, although it might not be acceptable if there are more than 10 or 11 stocking points.

* Note that g is expressed in the same units as the travel times are.

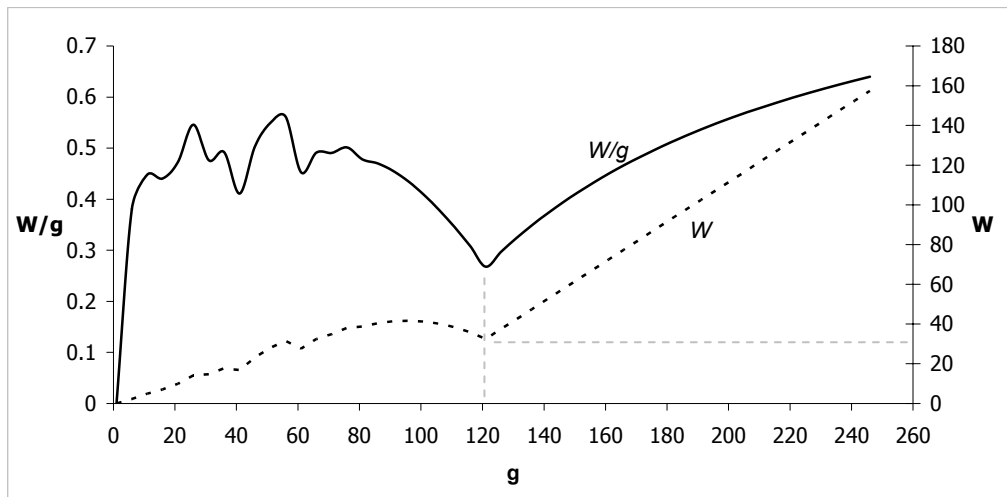


Figure 3-12: A graph showing the optimizing of the grid spacing

In Figure 3-12, the objective function is drawn for a number of travel times. Clearly for $g > 11$ the objective function reaches a minimum value where $g = 120$. For $g = 120$, the average waiting time is around 32 minutes.

If one would choose to make the grid spacing equal to zero *i.e.* the grid is ‘turned off’, one would expect a representation as shown in Figure 3-13. The way the PBIL algorithm is implemented to solve this problem unfortunately does not allow the grid spacing to be zero. A further explanation of this will follow in Chapter 4.

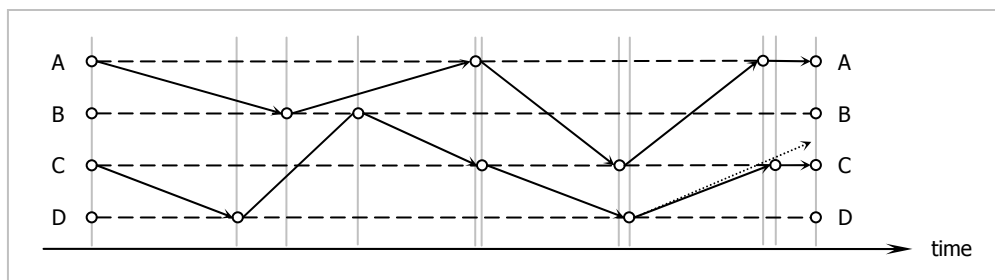


Figure 3-13: Vehicle routes with the time grid turned off

3.8 A Mathematical Model

What follows is a mathematical representation of the problem discussed in this text. Any resemblance between symbols used in this section and in other sections are purely incidental and should be ignored.

Let v_i be the set of n stocking points for $i = 1, 2, \dots, n$. Then t_{ij} is defined as the travel time between SP i and SP j for $i, j \in [1, n]$. Define $t_{ij} = 0$ for $i = j$. Travelling between these n stocking points is a fleet of k delivery vehicles, each on its own assigned route. A route is a sequential list of stocking points in which a specific SP may be repeated.

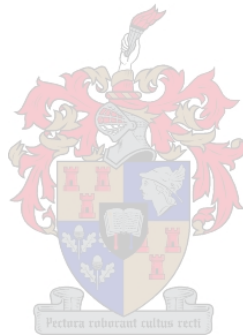
Let T be the length of a normal working day, and G the size of the grid spacings so that $\Omega = T/G$ evaluates to a positive integral number. Then $g = 0, 1, \dots, \Omega$ is the grid points of which $g = 0$ are where the routes start at the beginning of the day and $g = \Omega$ at which the routes end for the day. This also implies that there are Ω grid spacings in a day and orders for goods may be placed in any of these spacings. A package will be picked up at the earliest on the first grid point following the time of the placed order. Since there are n possible stocking points that can place an order, there will be a total of $n(n-1)\Omega$ possible shortest paths for packages to follow from their origins to their destinations. Let s_{ijg} be the length of the shortest time in which a package can be delivered from i to j ($i, j \in [1, n]$), if j placed an order on i in the grid spacing just before the grid point g ($g \in [0, \Omega]$). In the case where $g = 0$, it means that the order was placed in the last grid spacing of the previous day. The average shortest delivery time of a package can then be calculated as shown in (3.5).

$$S = \frac{\sum_{g=0}^{\Omega-1} \sum_{i=1}^n \sum_{j=1}^n s_{ijg(i \neq j)}}{n(n-1)\Omega} \quad \dots(3.5)$$

Van Wijck (2004) proposed that this value found in (3.5) should be used as a measurement by which the given set of k routes may be evaluated. In other words, (3.5) will be used as the objective function in finding an optimal set of routes. It might make more sense to use, say, the 90th percentile in the distribution of shortest paths as will be discussed later in section 4.3.4. Formally, a solution to the problem under study will be a set of k routes for the k vehicles that minimizes the value of S .

3.9 Conclusion

The problem under study as represented in this chapter forms a part of a wide variety of distribution and logistics problems. It draws many of its characteristics from general PDPs and also VRPs. It is, however, a unique problem and therefore needs a unique implementation of some algorithm to find an optimal answer for it. In the next chapter, a tailored version of the PBIL algorithm will be discussed that may be used to optimize the problem.

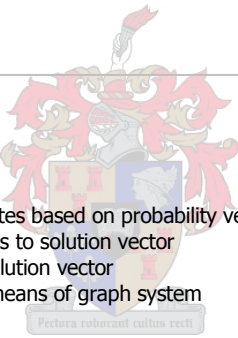


THE TAILORED PBIL ALGORITHM

4.1 Introduction

The PBIL algorithm as discussed in section 2.4 is easy to implement for the optimization of some function or combination of functions. To implement the PBIL algorithm on the problem under study, some adjustments have to be made in order to address the specific needs of the problem. This section covers the different aspects of the PBIL algorithm and highlights the differences in this tailored version.

Figure 4-1 shows the PBIL algorithm in pseudo-code as adjusted for the specific problem. Terms that are more relevant to the problem is used to describe the different sections of the algorithm.



```

1 Start Algorithm
2 Initialization
3 While ( $i < \text{NumberOfIterations}$ )
4   For  $j = 1$  To  $\text{PopulationSize}$ 
5     Generation of a set of vehicle routes based on probability vector
6     Translation of set of vehicle routes to solution vector
7     Creation of graph system from solution vector
8     Evaluation of solution vector by means of graph system
9   Next  $j$ 
10    Find the best solution vector  $B$ 
11    Update probability vector  $P$  according to  $B$ 
12    For  $j = 1$  To  $\text{NumberOfElementsOfP}$ 
13      If ( $\text{Random}(0,1) < \text{MutationProbability}$ ) Then
14        Mutate  $P(j)$ 
15      End If
16    Next  $j$ 
17     $i = i + 1$ 
18    If Converged Then
19      End Algorithm
20    End If
21  End While
22 End Algorithm

```

Figure 4-1: The tailored PBIL algorithm

4.2 The Three-Dimensional Probability Vector

In function optimization, normally the probability vector of the PBIL algorithm serves as some binary representation of a decimal solution. Also very often, it is used to represent

the permutation of some entities. In fact, with some initiative the probability vector can be used to represent almost anything within the boundaries of optimization.

The final solution to the problem under study will be a set of routes forming a network between the stocking points that the fleet of vehicles may follow. To represent this set of routes by means of the probability vector, a three-dimensional probability vector is introduced as in Figure 4-2. Remember that the presentation of a single solution in a population of solutions will have exactly the same form as the probability vector, with the exception that the probability vector contains probabilities and the solution vector contains either zeros and ones (in the case of a binary representation) or some other values depending on the solution it represents.

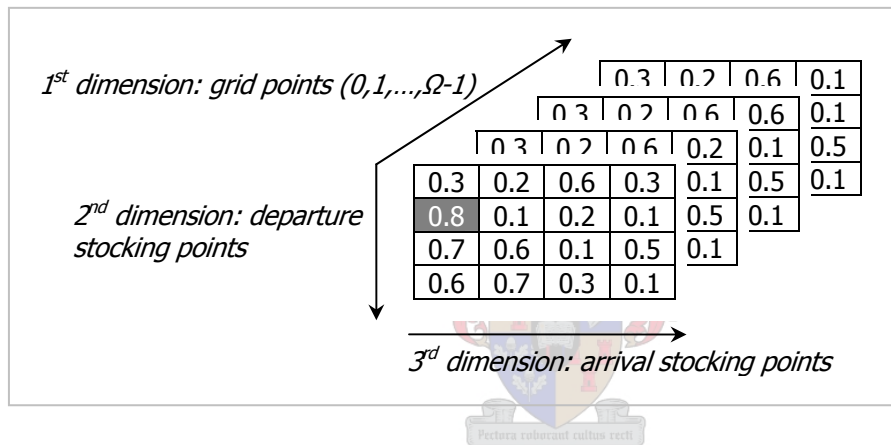


Figure 4-2: The three-dimensional probability vector

Let p_{kij} be the probability vector value in the position where the first, second and third dimensions are k , i and j respectively. This p_{kij} is the probability that there will be a “1” in that position of a solution vector. In the normal PBIL algorithm, solution vectors is constructed directly from the probability vector. In this implementation of the PBIL, a set of vehicle routes is constructed from the probability vector, and the set of vehicle routes are then translated to a solution vector. So, a “1” in a solution vector position where the first, second and third dimensions are k , i and j respectively, means that at least one route will go from SP i to SP j departing from SP i at the k^{th} grid point. For example, in Figure 4-2 there is an 80% chance that there will be a “1” in the position (0, SP2, SP1) in a solution

vector, which will then translates to the fact that at least one vehicle will be departing from SP 2 at the start of the working day (grid point 0) to travel to SP 1.

For a day divided up in four grid spacings, where two vehicles travel between four SPs on their routes, the solution vector will look like that in Figure 4-3.

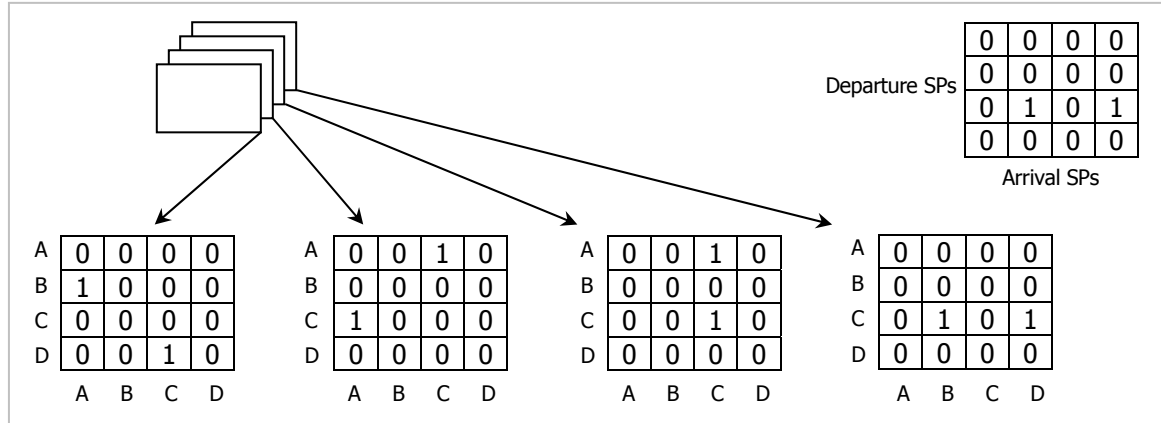


Figure 4-3: A three-dimensional solution vector

Assume that all travel times between stocking points in Figure 4-3 are less than the size of the grid spacing. Since two vehicles are travelling between the four SPs, one of them will start the day at B and the other at D as the first dimension of the solution vector shows. The first vehicle will follow the route $B \rightarrow A \rightarrow C \rightarrow C \rightarrow D$. The second vehicle will follow the route $D \rightarrow C \rightarrow A \rightarrow C \rightarrow B$. Note that the last SPs in these two routes can be swapped without the solution vector changing. In effect, two different solutions are here represented by one solution vector (see Figure 4-3). However, the evaluation of these two different solutions will amount to exactly the same fitness because of the method proposed by Van Wijck (2004).

4.3 Solution Vectors Population

The first part of a single iteration in the PBIL algorithm is to form the population of solution vectors, and evaluate them. This process (listed in lines 4 through 9 in Figure 4-1) is shown in Figure 4-4. The generation of one solution in the population then consists of the random construction of routes for vehicles, the translation thereof into a solution vector, the preparation of a graph system based on the solution vector and the evaluation of the solution vector (which is in effect the initially constructed routes).

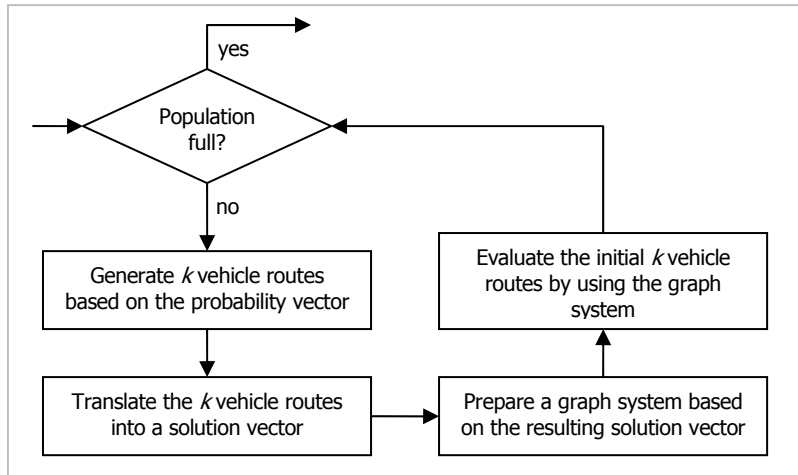


Figure 4-4 The forming of a population of solution vectors

The four parts as depicted in Figure 4-4 will be discussed in the next four subsections.

4.3.1 Route Construction

In the random generation of vehicle routes, two main constraints need to hold. These are:

1. The constraint on the starting and ending SPs of a set of routes as described in section 3.5.
2. The length (in time) of any route in the set of routes may not exceed the length of the working day.

The use of a random insertion method to generate routes provides a good way to manage both these constraints. The first constraint specifies that each SP that is specified as the starting SP for a route in the set must also be specified as an ending SP for some route in the set. These two collections of k SPs each, both just permutations of the same list of k SPs, can be user-specified in the initialization phase of the algorithm or it can be generated randomly. Whichever way is used to obtain these collections of SPs, as depicted in Table 4-1 as an example, they are used as the source in randomly generating routes.

Vehicle	Starting SP	Ending SP	Route	Route Length
1	B	A	B → A	160
2	C	B	C → B	180
3	A	C	A → C	170

Table 4-1: The collections of starting and ending stocking points for three routes

The random insertion method continually inserts a stocking point before the last SP in the route as long as the length of the route is shorter than the length of a working day. Insertions into a route are based on the probability vector, which is along with the travel times between SPs given in Figure 4-5. Let the day be 540 minutes and the grid spacing 180 minutes.

Travel times				Probability vector									Grid points		
	A	B	C	0	A	B	C	1	A	B	C	2	A	B	C
A	0	200	170	A	0.6	0.3	0.4	A	0.5	0.1	0.6	A	0.4	0.8	0.1
B	160	0	210	B	0.2	0.6	0.5	B	0.2	0.3	0.4	B	0.3	0.4	0.8
C	170	180	0	C	0.4	0.3	0.5	C	0.7	0.9	0.3	C	0.5	0.6	0.6

Figure 4-5: Travel times between three stocking points and the probability vector

To illustrate the working of the insertion method, take the first route in Table 4-1 as an example. If a SP is inserted before the last SP in the route, it means that the vehicle will now visit that SP before it goes on and finishes the route. The vehicle will arrive at the inserted SP after coming from the SP that is just before the inserted SP, in this case being B. To insert a SP into the route, let a random number generator generate a number between zero and the sum of the probabilities in the B row of the grid point 0. Suppose the random number generator returns the number 1.169 as shown in Figure 4-6.

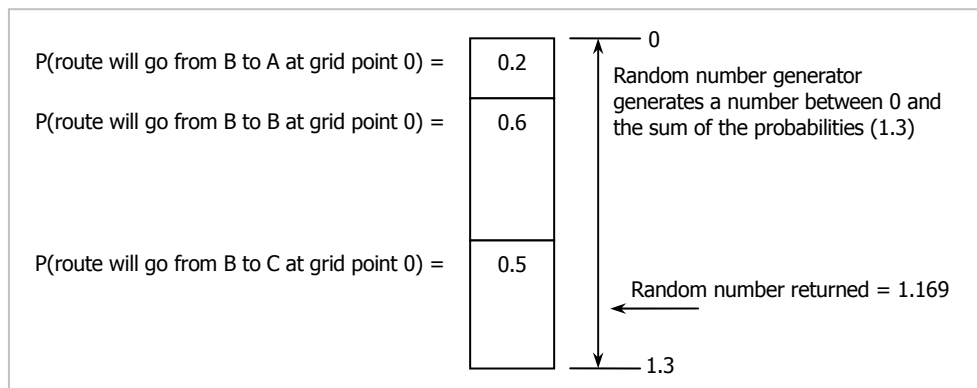


Figure 4-6 Choosing the correct SP to insert based on the probability vector

Based on this number, stocking point C is chosen (see Figure 4-6) to be inserted into the route to become B → C → A. To determine if this insertion into the route can be allowed, the route length with the inserted SP should now be determined to see if the length of the

route does not exceed the length of the day. This is done by the formula given in (4.1), where g_α is the number of the grid point where the vehicle will leave the second last SP (α) in the route and $t_{\alpha\beta}$ is the travel time from α to β , β being the last SP in the route.

$$\text{Route length after inserting } \alpha = (g_\alpha \times \text{gridspacing size}) + t_{\alpha\beta} \quad \dots(4.1)$$

The original length of the route was 160 minutes. By looking at the timeline in Figure 4-7, one can see how the length of the new route is calculated. Note that a SP is always inserted just before the last SP in the route. So, the second last SP in a route is always the inserted SP.

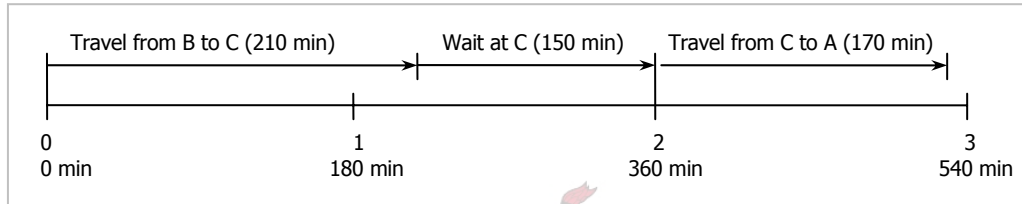


Figure 4-7 The length of a route determined on a timeline

If the new length of the route is still shorter than the length of the day, the insertion made is allowed. In the route $B \rightarrow C \rightarrow A$, the number of the grid point where the vehicle will leave the second last (or inserted) SP in the route (C) is 2, since after travelling for 210 minutes from B to C (passing grid point 1) the vehicle has to wait until the next grid point (grid point 2) before departing for A. This means that the length of the route after inserting C will be 530 minutes (210 minutes travel time + 150 minutes waiting time + 170 minutes travel time), which is still less than the length of the day (540 minutes). The random insertion of stocking points, based on the probability vector as discussed in section 4.2, is now repeated until no more stocking points can be inserted into the route for which the new length will be shorter than the length of the day.

When such an insertion procedure is finished, exactly one route will have been constructed. The insertion procedure is repeated k times, so that a set of k routes for the fleet of vehicles has been constructed. This set of routes represents one solution to the problem, and it is this set of routes that is translated into a solution vector.

4.3.2 Route Translation

After a set of routes have been constructed, they are translated into the solution vector as illustrated in Figure 4-8. Each route in a set of routes is taken at a time and the appropriate values are filled into the solution vector. For example, the route $B \rightarrow C \rightarrow A$ has a starting leg of $B \rightarrow C$ that will start at grid point 0, so a “1” is filled in the solution vector at the position (grid point 0, SP B, SP C). This leg of the route takes more than 180 minutes (the grid spacing) to complete, so the vehicle will only depart from C for the second leg $C \rightarrow A$ on grid point 2, as shown in Figure 4-7. Therefore, a “1” is filled in the solution vector at (grid point 2, SP C, SP A). Figure 4-8 shows how the solution vector will look after the routes are translated.

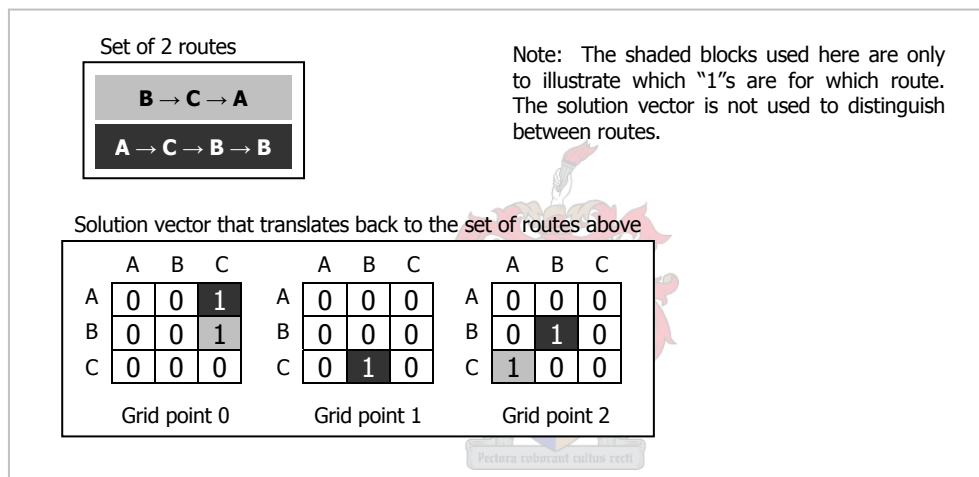


Figure 4-8: Routes translated into a solution vector

4.3.3 The Solution Graph System

The evaluation of a set of routes will be no simple task if the set of routes or the translated solution vector is used. Therefore, the solution vector is used to prepare a graph system on which the routes evaluation method proposed by Van Wijck (2004) can be applied. Note that the graph system actually represents the possible paths that a package may travel between SPs, and so one would expect that, in the absence of vehicle routes, these paths should be only those shown in Figure 4-9. As expected, there is no way how a package can get from one SP to another, since these paths only enables a package to stay at one SP during the whole day.

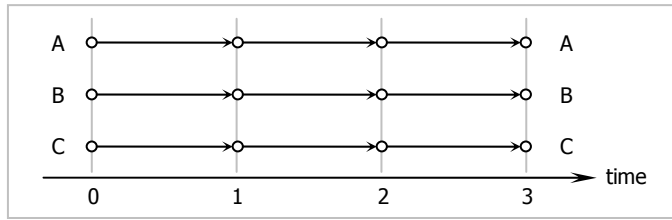


Figure 4-9: Package paths in the absence of vehicle routes

Adding a set of vehicle routes as in Figure 4-8 to the graph system would give a package a wide variety of paths to follow, enabling it to go from one SP to another. The complete graph system is shown in Figure 4-10.

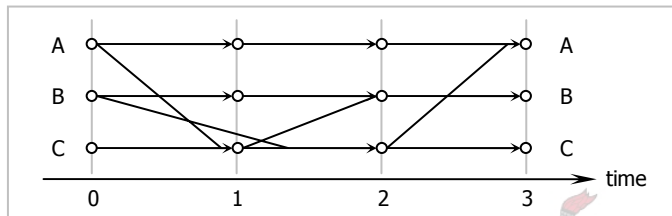


Figure 4-10: The complete graph system

4.3.4 Solution Evaluation

The original set of routes constructed, which translates into a solution vector, which are in turn used to prepare a graph system, must be evaluated. The reason why the set of routes are translated into a solution vector is that the solution vector is in the same format as the probability vector and may therefore be used to update the probability vector. The reason why a graph system is prepared from the solution vector is that the graph system provides a means by which the set of routes may be evaluated using the objective function as discussed in section 3.8.

During the day, many orders may be placed by SPs on other SPs for spare parts. In one particular grid spacing in a day, orders may occur at random. Suppose one would model the occurrence of new orders in one particular grid spacing by some statistical distribution. There is an equal probability that an order can occur at any time within the grid spacing, so the uniform distribution may be used to model this order occurrences. The distribution of orders is shown in Figure 4-11.

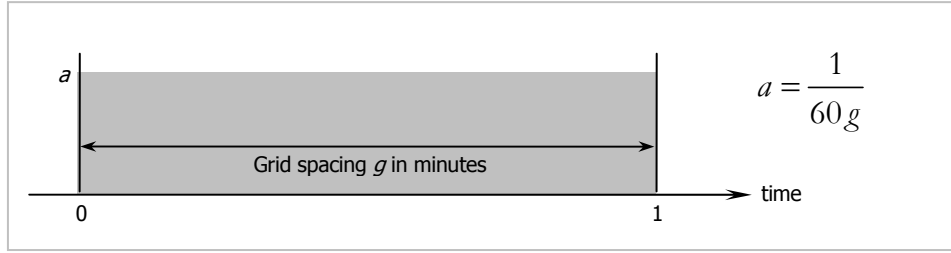


Figure 4-11: The probabilistic distribution of order arrivals

When orders are modelled by the uniform distribution, one can say that orders are occurring on average $\frac{g}{2}$ minutes before a grid point. This in turn means that any package will stay static at its holding SP for $\frac{g}{2}$ minutes before the vehicle will depart for the next SP on its route. In the case where no vehicle visits the SP in the same grid spacing as the order was placed in, the package will stay static until some vehicle passes. This extra static time of a package contributes to the overall speed with which a package is delivered and, again, is affected by the size of the grid spacings.

In a graph system such as in Figure 4-10, there are many possible paths that a package may follow from its holding SP to its destination. The evaluation method proposed by Van Wijck (2004) and mentioned earlier in section 3.8 states that the shortest paths between all possible SPs should be used to determine the fitness of the solution. Formula (3.5) from section 3.8 is repeated here for clarity.

$$S = \frac{\sum_{g=0}^{\Omega-1} \sum_{i=1}^n \sum_{j=1}^n s_{ijg(i \neq j)}}{n(n-1)\Omega} \quad \dots(4.2)$$

The value S is the average shortest path length. If a histogram is drawn from all possible s_{ijg} as in (4.2), one could expect some distribution of shortest path lengths as in Figure 4-12. The shortest path between two given nodes in the graph system is obtained by applying Dijkstra's algorithm. Since there are $n(n-1)\Omega$ (see section 3.8) possible shortest paths in a graph system, Dijkstra's algorithm will be applied $n(n-1)\Omega$ times, giving the evaluation of one set of routes a worst case time complexity of $O(n^4)$.

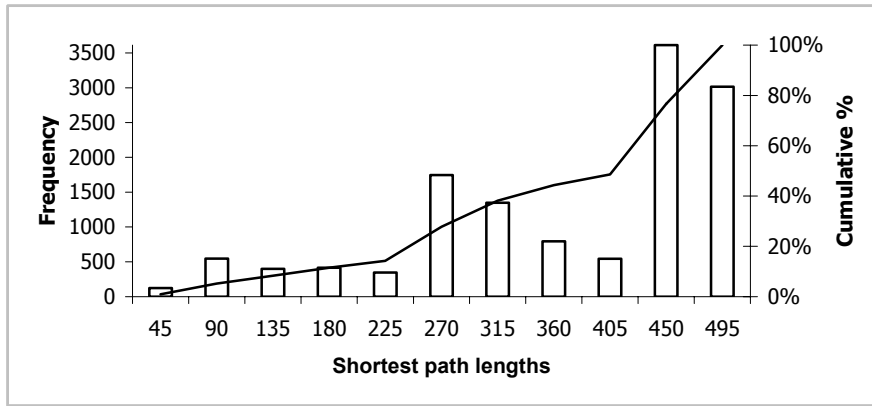


Figure 4-12: A histogram of shortest path lengths

The distribution in Figure 4-12 will shift to the right if the average static time of $\frac{s}{2}$ minutes of a package is now also added to each of the possible shortest paths. Therefore, when comparing two solutions with each other, the average static time of packages will not have any effect. It will therefore only be included for the sake of completeness.

As mentioned earlier, this statistical representation of the shortest path lengths lends the possibility that any percentile can be used in the comparison of solutions. For the sake of simplicity in the explanations, the average is used here since it is the easiest to calculate.

After the graph system of each set of routes is used to evaluate all the sets of routes in the population, the solution vector of the highest evaluating set of routes is used to update the probability vector. The updating or teaching of the PV is discussed in the next section.

4.4 Probability Vector Update and Mutation

The update and mutation of the probability vector is done in exactly the same manner as explained in sections 2.4.5 and 2.4.7. If $probability_{kij}$ (which is the same as the probability p_{kij} as on page 44) is the probability that at least one route is present between SP i and SP j starting at the grid time k , then the probability vector is updated with the learning rate and the highest evaluating solution found from the population as shown in (4.3). LR is the learning rate.

$$probability_{kij}^{new} = (probability_{kij}^{old} \times (1.0 - LR)) + (LR \times vector_{kij}) \quad \dots(4.3)$$

The probability vector is mutated based on a mutation probability (MP) by as much as the mutation shift specifies as shown in (4.4).

$$probability_{kij}^{new} = (probability_{kij}^{old} \times (1.0 - MS)) + (random(0,1) \times MS) \quad \dots(4.4)$$

MS is the mutation shift, the amount with which the probability vector is mutated.

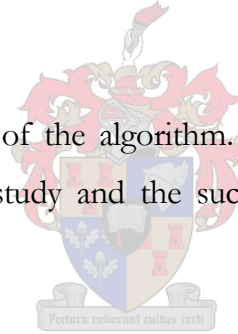
4.5 Termination

Termination takes place as explained in section 2.4.8. Either the algorithm terminates because the number of iterations have reached a predefined limit, or the inequalities in (4.5) have been satisfied.

$$probability_{kij} \geq l_u \text{ or } probability_{kij} \leq l_l \quad \dots(4.5)$$

In (4.5), l_u and l_l are the upper and lower convergence limits respectively specified before the start of the algorithm.

This concludes the discussion of the algorithm. In the next chapter, the algorithm as applied to the problem under study and the success of the underlying theories will be verified and validated by tests.



VALIDATION AND VERIFICATION

5.1 Introduction

This chapter covers the validation and verification of the model and the algorithm. This is done by setting up some trivial and some complex enough problems, and testing the algorithm on these problems. After the actual validation and verification, the effects of the different input variables are studied to determine which of the variables have the greatest effect on the complexity of the algorithm. The algorithm is programmed in Visual Basic on the .NET Framework and the code is shown in Appendix B.

5.2 Test Problems Setup and Execution

5.2.1 First Test Problem

Let there be two stocking points and two vehicles. Let the two vehicles' routes each start at alternative stocking points, and let them end at the same stocking point they started. The travel time in both directions between the two SPs is 80 minutes. The length of the day is nine hours (540 minutes), and the size of the grid spacings is chosen to be 90 minutes. This evaluates to six grid spacings, which will enable the routes to end at the same stocking point that they started. The expected optimal solution to this problem is shown in Figure 5-1. By simply continuing to visit alternating stocking points, this zigzag pattern evolves and creates an opportunity at each grid time for a package to follow a path to the other stocking point.

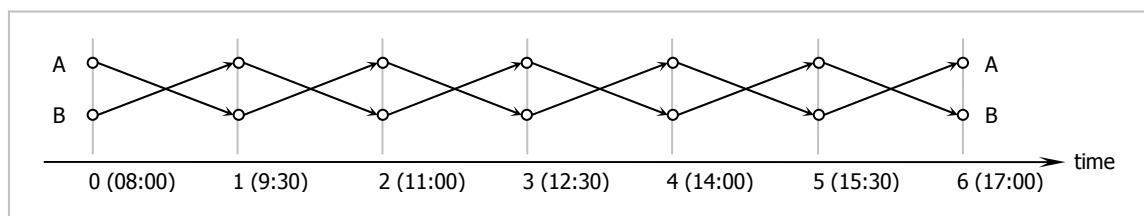


Figure 5-1: Expected solution of first test problem

The algorithm is executed on the problem using a population size of ten. A mutation probability and mutation shift of 0.02 and 0.05 respectively, a learning rate of 0.1 and

convergence limits of 0.02 and 0.98 are used. The algorithm was run ten times and the expected solution was returned as the optimal solution all ten times. The convergence output is shown in Figure 5-2. The solution vector associated with the expected optimal solution as returned by the algorithm can be found in Appendix A.

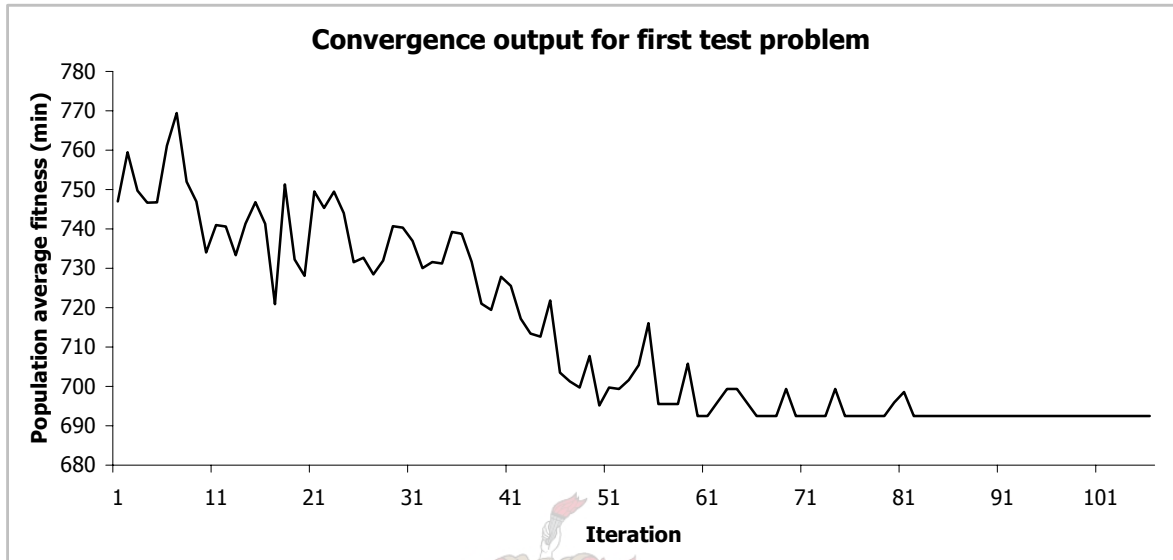


Figure 5-2: Convergence output for first test problem

5.2.2 Second Test Problem

The first test problem is repeated, but with a grid spacing size of 108 minutes, which will evaluate to five grid spacings. Now the vehicles do not have a way to end at the SP that they started if they visit alternating SPs. Each vehicle will have to, at some grid point, stay at the current SP. Possible solutions are shown in Figure 5-3.

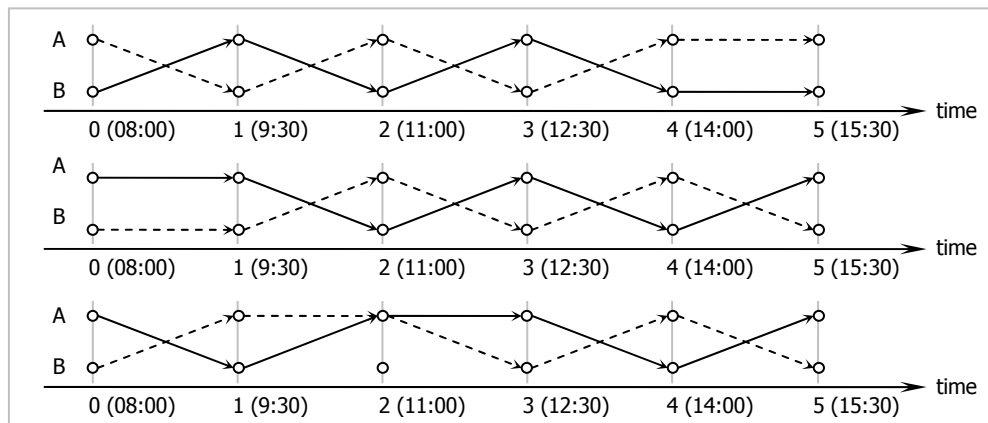


Figure 5-3 Expected solutions of second test problem

Note that the top two solutions in Figure 5-3 is different, but will counted equal if evaluated. Since there are one SP in the third solution in Figure 5-3 that will not be visited at 11:00, that type of solution will be sub-optimal if compared with the top two.

The algorithm is executed on the problem using a population size of ten and a mutation probability and mutation shift of 0.02 and 0.05 respectively. A learning rate of 0.1 and convergence limits of 0.02 and 0.98 is used. The algorithm was run ten times, and in every case, a solution equivalent to the top two solutions in Figure 5-3 was returned. The convergence output is shown in Figure 5-4 and the solution vector associated with the returned optimal solution can be found in Appendix A.

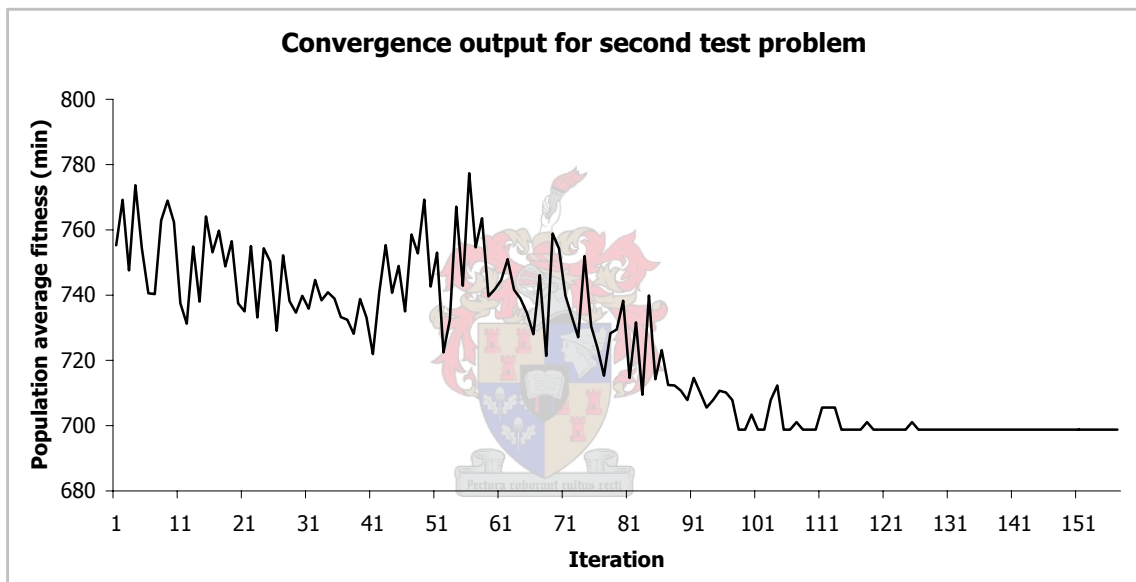


Figure 5-4 Convergence output for second test problem

The solution of which the solution vector is shown in Appendix A is shown in Figure 5-5.

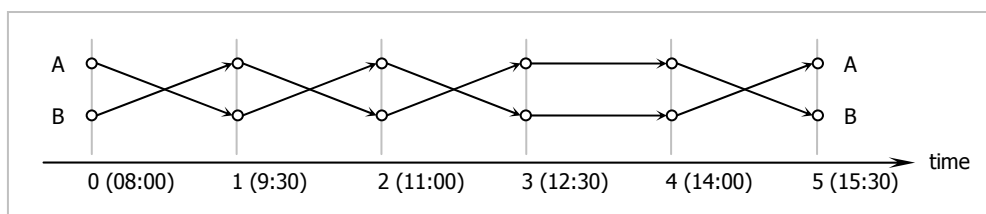


Figure 5-5 Optimal solution to second test problem

5.2.3 Third Test Problem

There are six stocking points that are divided into two groups of three each. SPs in one group are located close to each other, while there are no two SPs from different groups that are close to each other, except for one pair. Figure 5-6 gives a graphical representation of this stocking point topography, with the travel times between possible pairs of stocking points given in Table 5-1.

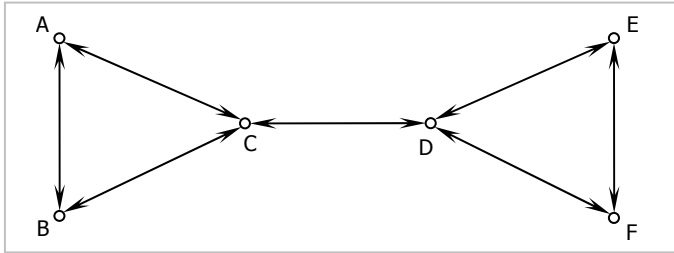


Figure 5-6: Graph representation of third test problem

Travel times						
	A	B	C	D	E	F
A	0	80	70	65	55	-
B	80	0	60	-	-	-
C	70	60	0	85	-	-
D	65	-	85	0	60	80
E	55	-	-	60	0	70
F	-	-	-	80	70	0

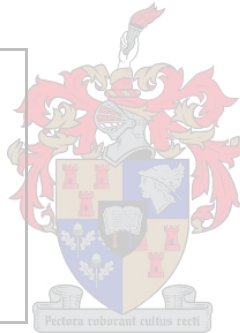


Table 5-1: Travel times between the six stocking points of the third test problem

Let there be three vehicles with start and end SPs as shown in Table 5-2. One would not want vehicles to have routes that lead between SPs that are far from each other in this case, so the expected optimal solution to this problem is that as shown in Figure 5-7. One vehicle is left to travel alternating between C and D, while each of the other vehicles is supposed to cycle the two groups of SPs.

Vehicle	Start	End
1	A	A
2	C	C
3	F	F

Table 5-2: Start and end stocking points for third test problem

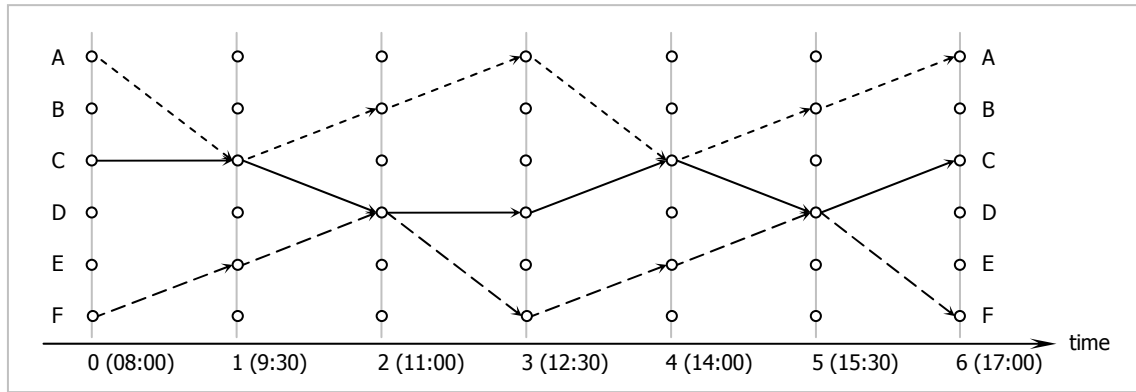


Figure 5-7: Expected solution of third test problem

The algorithm was run with a population size of 20 and the same input variables as in the first test problem. The expected solution was returned seven out of the ten times as the optimal solution. The convergence output of one of the seven runs is shown in Figure 5-8.

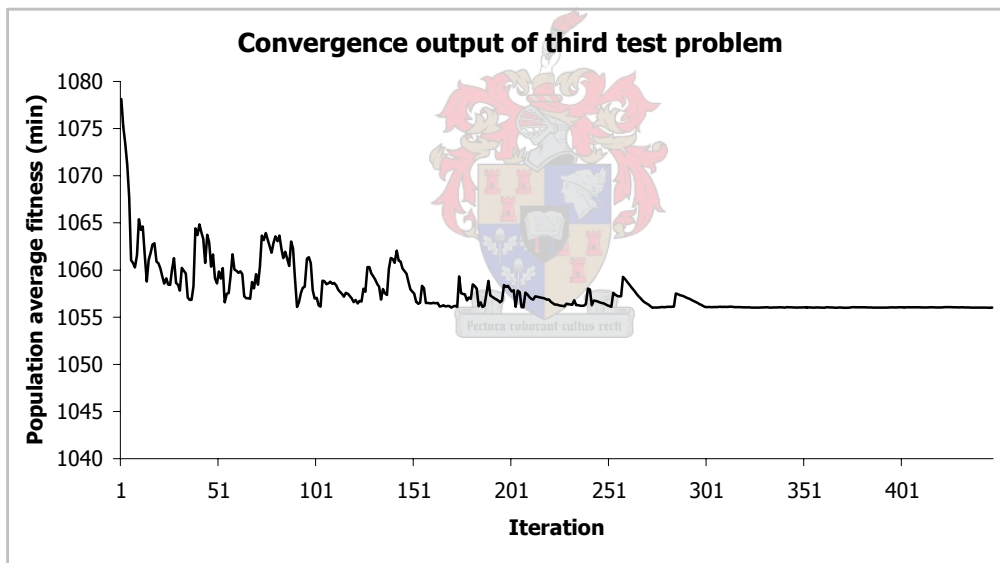


Figure 5-8: Convergence output for third test problem

It should be mentioned that the other three returned solutions were in essence the same as the expected solution. The only difference was that the two vehicles cycling the two groups of SPs did it in different directions. The solution returned in these three runs is shown in Figure 5-9.

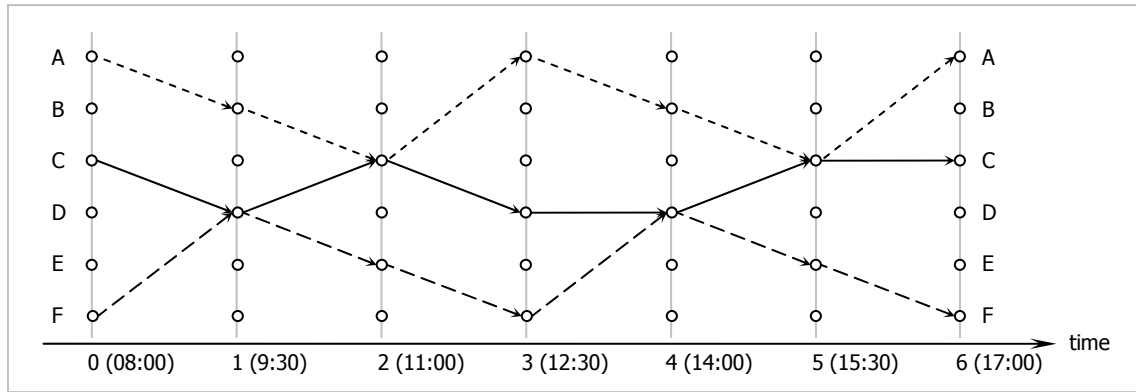


Figure 5-9 Alternate solution of third test problem

5.2.4 Fourth Test Problem

The fourth test problem is in essence the same as the third problem, with the exception that the travel matrix is now not a sparse matrix anymore. Travel times are now indicated for all possible traversals in .

Travel times						
	A	B	C	D	E	F
A	0	80	70	65	55	-
B	80	0	60	-	-	-
C	70	60	0	85	-	-
D	65	-	85	0	60	80
E	55	-	-	60	0	70
F	-	-	-	80	70	0

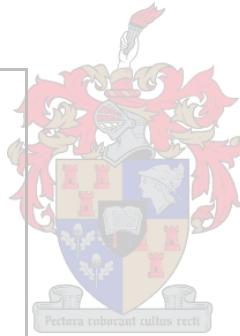


Table 5-3: Travel times between the six stocking points of the fourth test problem

The algorithm was run ten times with the same parameters as in the third test problem. In seven of the ten runs, the same solution as with the third test problem was returned. The solutions returned in the other three cases was worse than this solution. The convergence output for one of the seven runs is shown in Figure 5-10.

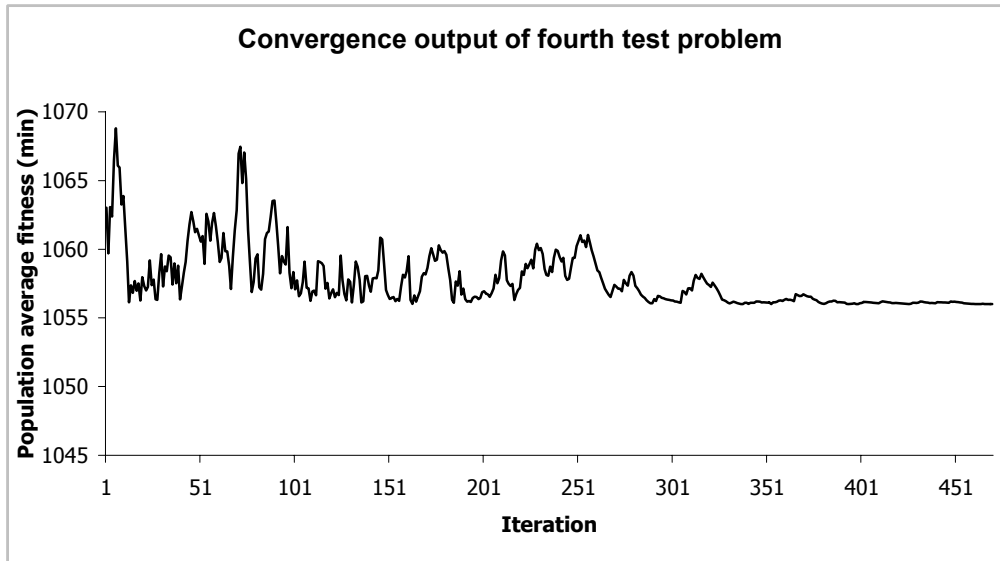


Figure 5-10: Convergence output for fourth test problem

5.2.5 Fifth Test Problem

The fifth test problem is a more complex problem, typically one that would be encountered in real life. For this problem, it would be very difficult to obtain an expected optimal solution, so the algorithm will be used here to get the optimal solution. Different from the first four test problems, will the measure of success of the algorithm here not be to compare the returned solution with something else. The goal is to see if the algorithm does find a solution here. Let there be twelve SPs connected to each other as shown in Figure 5-11. The travel times in minutes between possible SPs is shown in **Error!**
Reference source not found..

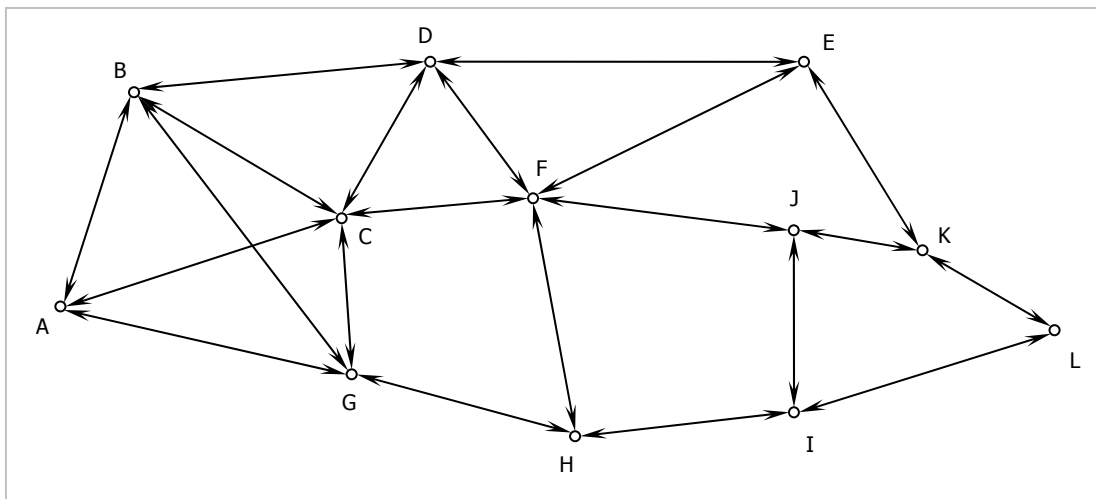


Figure 5-11 Graph representation of fifth test problem

Travel times												
	A	B	C	D	E	F	G	H	I	J	K	L
A	0	55	50	-	-	-	40	-	-	-	-	-
B	55	0	40	80	-	-	70	-	-	-	-	-
C	50	40	0	65	-	55	45	-	-	-	-	-
D	-	80	65	0	75	45	-	-	-	-	-	-
E	-	-	-	75	0	75	-	-	-	-	100	-
F	-	-	55	45	75	0	-	90	-	50	-	-
G	40	70	45	-	-	-	0	75	-	-	-	-
H	-	-	-	-	-	90	75	0	45	-	-	-
I	-	-	-	-	-	-	-	45	0	60	-	55
J	-	-	-	-	-	50	-	-	60	0	30	-
K	-	-	-	-	100	-	-	-	-	30	0	45
L	-	-	-	-	-	-	-	-	55	-	45	0

Table 5-4: Travel times between SPs of fifth test problem

Suppose that some company's six delivery vehicles have to start the day at 09:00 in the morning and finishes at 17:00 in the evening. The day is then eight hours long and can be divided into eight grid spacings of 60 minutes each. The algorithm is executed on this problem with a population size of ten. A mutation probability and mutation shift of 0.02 and 0.05 respectively, a learning rate of 0.1 and convergence limits of 0.02 and 0.98 are used. The convergence output of the algorithm is shown in Figure 5-12.

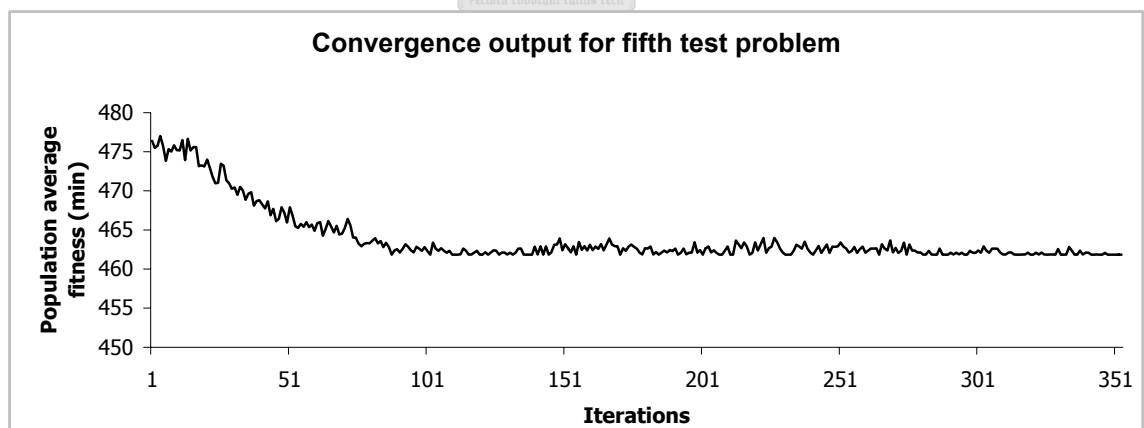


Figure 5-12 Convergence output for fifth test problem

The algorithm returned the routes as shown in Figure 5-13. Unfortunately, there is no way in which one can tell if this solution is the real optimal solution to the problem. What

one can say is that the algorithm definitely converged to this solution, making it one of the better possible solutions.

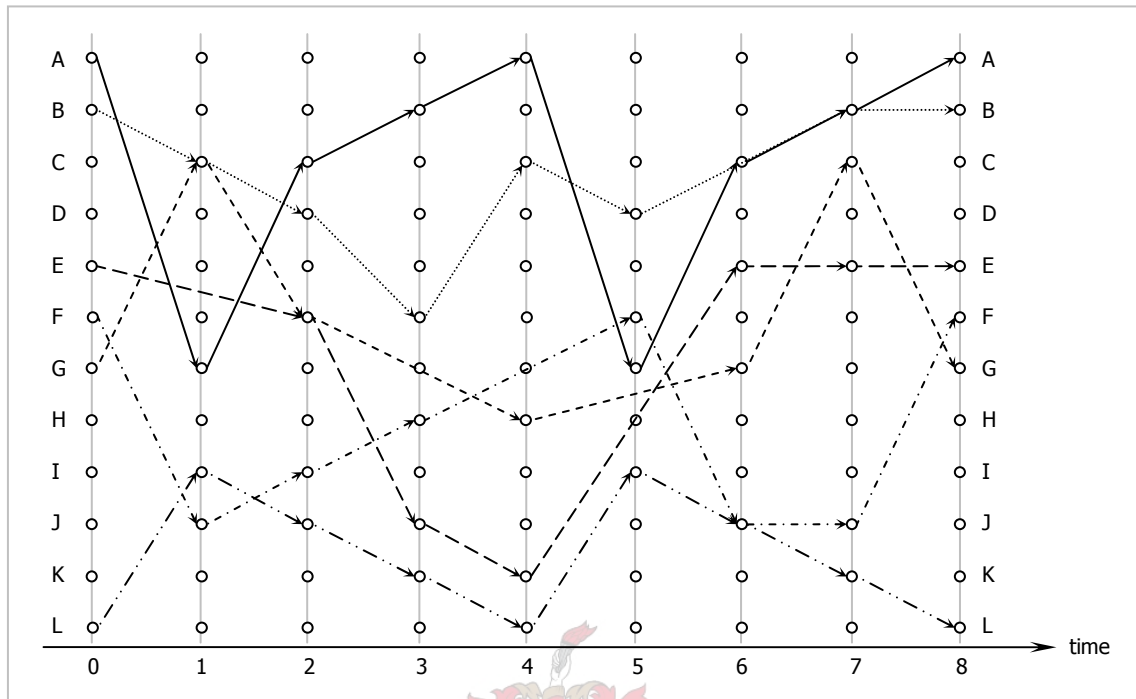


Figure 5-13 Returned optimal solution to fifth test problem

5.2.6 Sixth Test Problem

The sixth test problem is set up to investigate the sensitivity of the algorithm when loading and unloading times are not included in the travel times, but randomly sampled from a normal distribution. A mean of 10 minutes and a standard deviation of 2 minutes is used in the normal distribution, and a random sample is drawn from the distribution only when a certain travel time between two stocking points is requested.

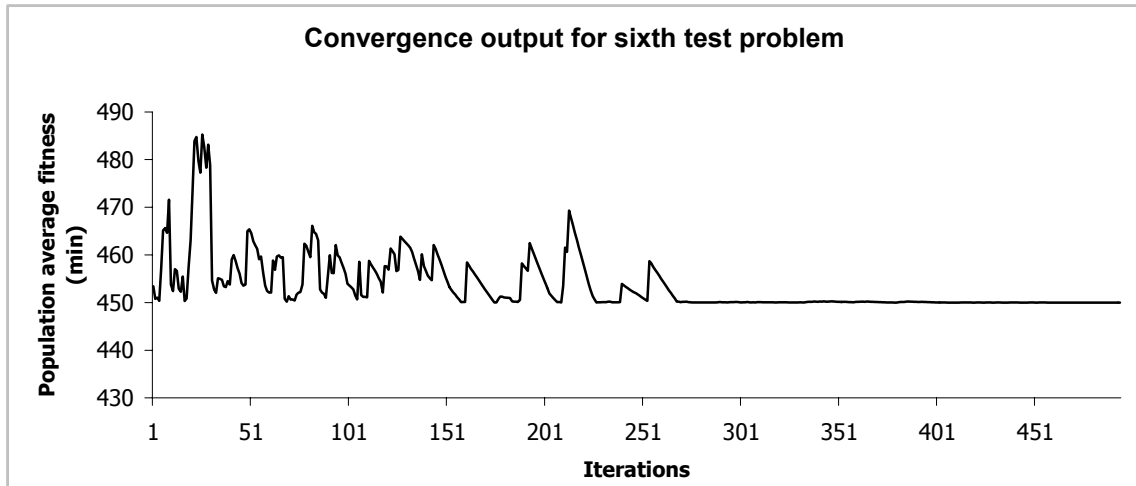


Figure 5-14: Convergence output for sixth test problem

The algorithm was run with the same parameters as the fifth test problem. Although the algorithm had difficulty in the first stages of its search to converge, it found a solution slightly better than the one found in the fifth test problem. The convergence output is shown in Figure 5-14.

5.2.7 Conclusion

In all test problems formulated, the PBIL algorithm returned the expected optimal solutions (where one could be formulated) as the final solutions to the problems. Since the expected solutions was generated from what is expected to happen in practise, one can conclude that the model represents the problem in real life accurately and that the meta-heuristic returns solutions that can be implemented in real life situations.

5.3 The Effects of Input Variables

In testing the effects of different input variables on the running time of the algorithm, the algorithmic complexity can be determined by inspection. The running time of different parts of the algorithm were measured, to determine what the effect of the variables is on each part. These parts are the construction of routes (construction), the conversion of the routes to a graph system (conversion), the evaluation of the graph system (evaluation), the update of the probability vector (update) and the mutation of the probability vector (mutate). Table 5-5 shows the 22 combinations of input variable values that were used in conducting the experiment. The input variables were chosen so that the experiment will

cover a wide spectrum of different size problems. Note that the complexities given to parts of the algorithm in this section are based on the specific variable under study (other variables are held constant), and cannot be seen as the complexity of the problem as a whole.

Day	Grid Spacing	# of Grid Spacings	SPs	Vehicles
1440	480	3	12	6
1440	360	4	12	6
1440	288	5	12	6
1440	240	6	12	6
1440	180	8	12	6
1440	160	9	12	6
1440	144	10	12	6
1440	120	12	12	6
1440	96	15	12	6
1440	90	16	12	6
1440	80	18	12	6
1440	360	4	3	2
1440	360	4	5	2
1440	360	4	9	2
1440	360	4	12	2
1440	360	4	7	2
1440	360	4	7	4
1440	360	4	7	6
1440	360	4	7	8
1440	360	4	7	10
1440	360	4	7	12
1440	360	4	7	14

Table 5-5: Input variable values for the effects experiment

5.3.1 Number of Stocking Points

Figure 5-15 shows the effects of the number of stocking points on the running time of the different parts. The evaluation of the graph system has a time complexity of at least the fourth order ($O(n^4)$ where n is the number of SPs, see also section 4.3.4), since Dijkstra's algorithm is executed in this part. All other parts have at most linear time complexity ($O(n)$).

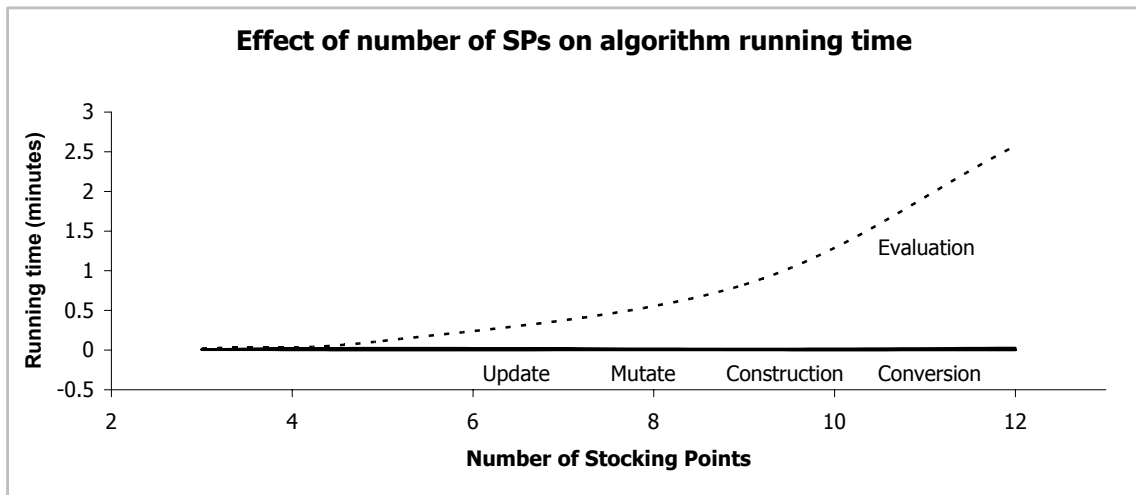


Figure 5-15: Effect of number of stocking points on algorithm running time

5.3.2 Vehicle Fleet Size

The effect of the vehicle fleet size on the algorithm running time is shown in Figure 5-16.

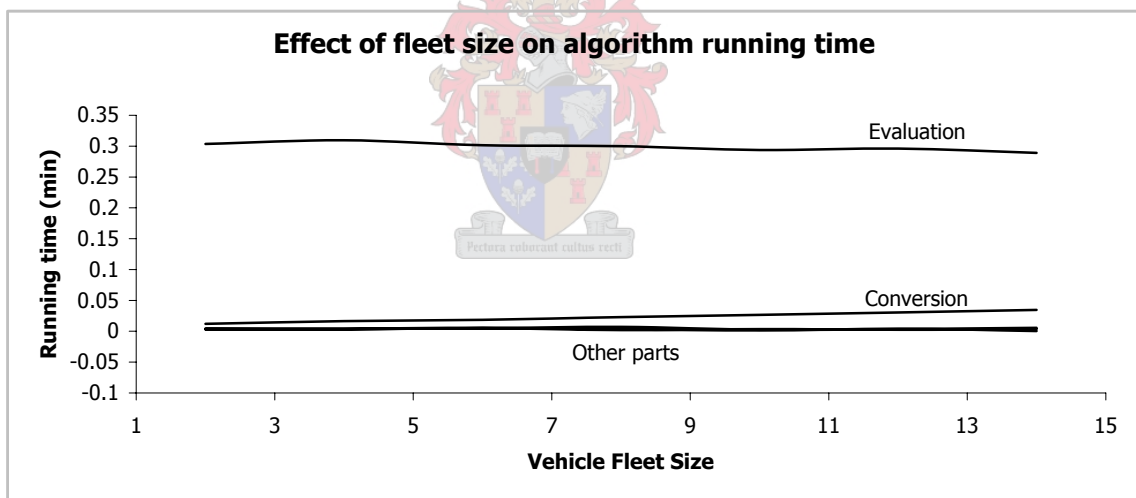


Figure 5-16: Effect of vehicle fleet size on algorithm running time

Again, the evaluation part stands alone from the rest of the parts, although an increase in the fleet size does not seem to affect the running time of the algorithm very much. In this case, the complexity of all the parts is at most linear ($O(n)$ where n is the number of vehicles), although it is probably only the conversion part that is $O(n)$.

5.3.3 Grid Spacing

In the execution of this part of the experiment, the worst-case scenario of a 24-hour working day was taken. As discussed in section 3.7, it is actually the number of grid spacings the day is divided into that makes the big difference. Figure 5-17 shows the effect of the number of grid spacings.

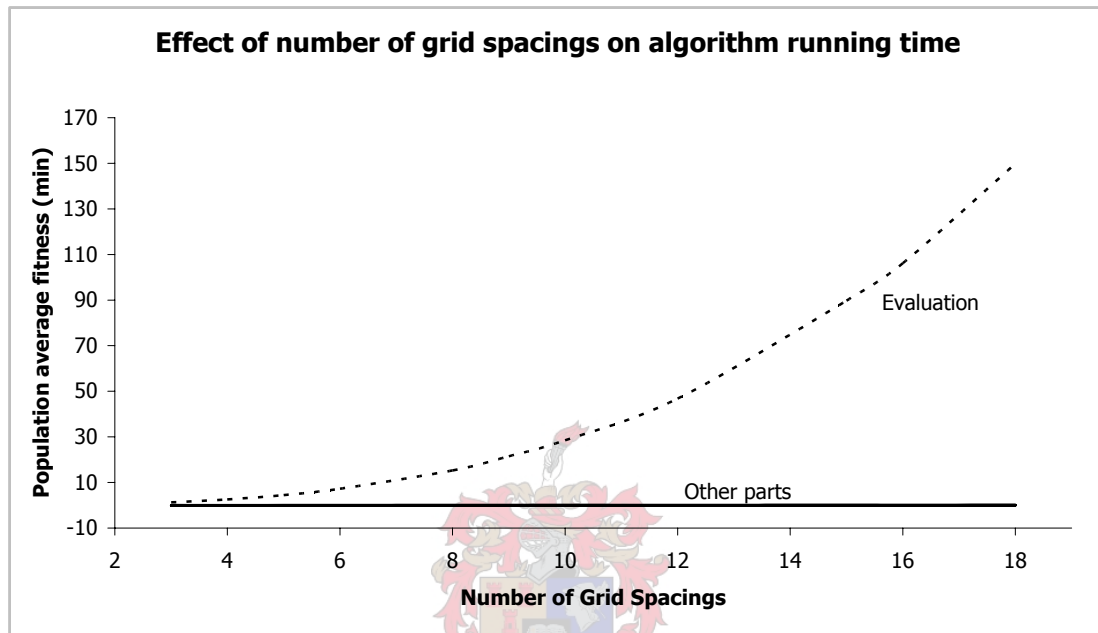


Figure 5-17: Effect of grid spacing on algorithm running time

The increasing number of grid spacings affects the evaluation part greatly. In this case, it is clear that all other parts have constant time complexity and the evaluation part has at least a time complexity of the third order.

5.3.4 Conclusion

The outcome of this experiment is as follows: the evaluation part of the algorithm, where a solution is evaluated before it can be compared, can be isolated as the one part of the algorithm that limits it in terms of the size of the grid spacing that can be used. All other parts have constant time complexities or at most linear time complexities. In the first place, the number of grid spacings in a day has the most notable effect on the running time of the algorithm, while the number of stocking points is in second place.

RECOMMENDATIONS

6.1 Alternative Evaluation Technique

Although the proposed method for evaluating a solution by Van Wijck (2004) is an ingenious approach, the method unfortunately does limit the functionality of this algorithm if implemented here. The utilization of some other objective function might improve the performance in terms of running time. It is recommended that an alternative method is seek to evaluate solutions, preferably one that will have no more than a quadratic time complexity to contribute to the algorithm's complexity. One should remember that any method that involves the finding of a path between two stocking points would implement a path-finding algorithm in some way and the best path-finding algorithms all have a time complexity of $O(n^2)$.

6.2 Automation of Choosing Grid Spacings Size

The choice of what the size of the grid spacings should be is one that can receive some more attention. The method to determine the optimal grid spacing size discussed in section 3.7 is somewhat limiting. Human judgement still plays a big role in the choice of what the grid size should be (not that one would ever want to eliminate human judgement). Another function could be introduced that would make the choosing of the grid spacing size a more automated process.

6.3 Turning Off the Time Grid

The time grid used in this text provides a unique opportunity to implement the Population-based Incremental Learning algorithm. As discussed in the text, it is, however, possible to leave out the time grid and still have a graph system where packages may follow paths from and to stocking points. If this is done, either the PBIL algorithm will have to be replaced by some other meta-heuristic, or it should be adapted to accommodate the graph system without a time grid.

6.4 Further Research

The three topics mentioned in this chapter provide some opportunities for further research to enhance the work in this text. Some other extensions to this work are listed below.

- ◆ The fleet size has been a given in this project. The minimization of the fleet size will certainly add an extra dimension to this work.
- ◆ Historical data may give an indication of what routes carry many or less orders. This data may then be used to link weights to different routes, which uncovers another variable that might be minimized: vehicle utilization.



CONCLUSIONS**7.1 Success of the Project**

The outcome of the project is according to the set standards a success. It successfully solves a problem with twelve SPs, where the day is divided up into 18 grid spacings, for any number of vehicles. This means that for a working day consisting of 9 hours, the minimum grid spacing size recommended is 30 minutes, which is far smaller than the requirement, since there are not many SPs that is situated within 30 minutes from each other. Thus, the problem experienced by the automotive company mentioned in the introductory chapter will quickly be solved in terms of what routes to follow to minimize the delivery time of spare parts.

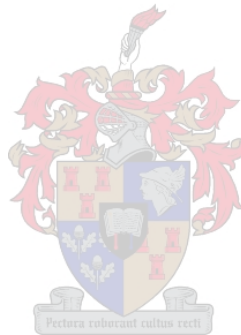
The approach taken in solving a problem in this project is one that focuses almost only on customer satisfaction. Nowhere are travelling costs, overhead costs, overnight facilities for vehicles, utilization of vehicles etc. taken into account. Therefore, a complete solution to the company's problem will probably not be solved by merely applying the method proposed by this project, but a more encompassing approach should be taken to address the problem.

REFERENCES

- [1] Baluja, S., 1994, Article on Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning, School of Computer Science, Carnegie Mellon University.
- [2] Bekker, J., Daronnat, L., 2004, Meta-heuristics for Simulation Optimization in Manufacturing, Proceedings of the International Conference on Competitive Manufacturing (COMA '04).
- [3] Bläzer, M., 2003, Complexity Theory and Algorithms, Internet Document, Institute for Theoretical Computer Science, ETH Zurich.
- [4] Chartrand, C., Oellermann, O., 1993, Applied and Algorithmic Graph Theory, McGraw-Hill, New York.
- [5] Clark, G., Wright, J., 1964, Scheduling of Vehicles from a Central Depot to a Number of Delivery Points, Operations Research 12.
- [6] Dantzig, G.B., Ramser, R.H., 1959, The Truck Dispatching Problem, Management Science, 6th Edition.
- [7] De Marco, G., Codenotti, B., Leoncini, M., Montangero, M., Santini, M., 2003, Article on Approximation Algorithms for a Hierarchically Structured Bin Packing Problem, Department of Computer Science, University of Iowa.
- [8] Fogel, D.B., Stayton, L.C., 1993, On the Effectiveness of Crossover in Simulated Evolutionary Optimization, BioSystems Journal.
- [9] Gambardella, L.M., Di Caro, G., Dorigo, M., 1999, Ant Algorithms for Discrete Optimization, Artificial Life 5.
- [10] Gendreau, M., Guertin, F., Badeau, P., Taillard, E., 1997, A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows, Transportation Science 31.
- [11] Goodrich, M.T., Tamassia, R., 2001, Data Structures and Algorithms in Java, 2nd Edition, John Wiley & Sons, New York.

- [12] Kelly, J., Xu, J., 1996, A Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem, *Transportation Science* 30.
- [13] Kindervater, G.A.P., Savelsbergh, M.W.P., 1997, *Vehicle Routing: Handling Edge Exchanges*, E.H. Aarts, J.K. Lenstra (editors), *Local Search in Combinatorial Optimization*, John Wiley & Sons, Chichester.
- [14] Michalewicz, Z., Fogel, D.B., 2002, *How to Solve It: Modern Heuristics*, 3rd Edition, Springer-Verlag, Berlin.
- [15] Ralphs, T.K., Kopman, L., Pulleyblank, W.R., Trotter, L.E., 2002, On the Capacitated Vehicle Routing Problem, *Mathematical Programming Manuscript*, Department of Industrial and Systems Engineering, Lehigh University, Bethlehem.
- [16] Rinaldi, G., Naddef, D., Corberán, A., Benavent, E., Belenguer, J.M., Augerat, P., 1995, Computational Results with a Branch and Cut Code for the Capacitated Vehicle Routing Problem, *Research Report 949-M*, University Joseph Fourier, Grenoble, France.
- [17] Savelsbergh, M.W.P., Sol, M., 1995, The General Pickup and Delivery Problem, *Transportation Science* 29.
- [18] Shaw, P., 1998, Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems, *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming*, M. Maher and J.F. Puget (editors), Springer-Verlag, Berlin.
- [19] Spears, W.M., 1992, Crossover or Mutation?, In Whitley (editor), *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, CA.
- [20] Taillard, E., Agazzi, G., Gambardella, L.M., 1999, A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows, In D. Corne, M. Dorigo and F. Glover (editors), *New Ideas In Optimization*, McGraw-Hill, London.
- [21] Thangiah, S.R., Salhi, S., 2001, Genetic Clustering: An Adaptive Heuristic for the Multi Depot Vehicle Routing Problem, *Applied Artificial Intelligence*, Computer Science Department, Slippery Rock University.

- [22] Van Wijck, W., 2004, An Efficient and Robust Method For Evaluating the Effectiveness of Scheduling Alternatives for a Common Distribution Logistics Problem, Unpublished Article, Department of Industrial Engineering, University of Stellenbosch.
- [23] Wikipedia Internet Encyclopaedia: <http://www.wikipedia.com>.
- [24] Winston, W.L., 1994, Operations Research: Applications and Algorithms, 3rd Edition, Wadsworth Publishing Company, Belmont, California.



BIBLIOGRAPHY

- [1] Ackley, D.H., 1987, *A Connectionist Machine for Genetic Hillclimbing*, Kluwer Academic Publishers, Boston, MA.
- [2] Baeck, T., 1993, *Optimal Mutation Rates in Genetic Search*. In Forrest (editor), *Proceedings of the Fifth International Conference on Genetic Algorithms 2-8*, Morgan Kaufmann Publishers.
- [3] Baluja, S., 1994, *Article on Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning*, School of Computer Science, Carnegie Mellon University.
- [4] Bekker, J., Daronnat, L., 2004, *Meta-heuristics for Simulation Optimization in Manufacturing*, *Proceedings of the International Conference on Competitive Manufacturing (COMA '04)*.
- [5] Bläzer, M., 2003, *Complexity Theory and Algorithms*, PDF Document, Institute for Theoretical Computer Science, ETH Zurich.
- [6] Chartrand, C., Oellermann, O., 1993, *Applied and Algorithmic Graph Theory*, McGraw-Hill, New York.
- [7] Clark, G., Wright, J., 1964, *Scheduling of Vehicles from a Central Depot to a Number of Delivery Points*, *Operations Research* 12.
- [8] Dantzig, G.B., Ramser, R.H., 1959, *The Truck Dispatching Problem*, *Management Science*, 6th Edition.
- [9] De Jong, K., 1975, *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*, PhD Thesis, University of Michigan.
- [10] De Marco, G., Codenotti, B., Leoncini, M., Montangero, M., Santini, M., 2003, *Article on Approximation Algorithms for a Hierarchically Structured Bin Packing Problem*, Department of Computer Science, University of Iowa.
- [11] Fisher, M.L., Jaikumur, R., 1981, *A Generalized Assignment Heuristic for Vehicle Routing*, *Network* 11.

- [12] Fisher, M.L., 1994, Optimal Solution of Vehicle Routing Problems Using Minimum K-trees, Operations Research, Volume 42.
- [13] Fogel, D.B., Stayton, L.C., 1993, On the Effectiveness of Crossover in Simulated Evolutionary Optimization, BioSystems Journal.
- [14] Gambardella, L.M., Di Caro, G., Dorigo, M., 1999, Ant Algorithms for Discrete Optimization, Artificial Life 5.
- [15] Garey, M., Johnson, D., 1979, Computers and Intractability: A Guide to the Theory of NP-completeness, W.H. Freeman & Co.
- [16] Gendreau, M., Guertin, F., Badeau, P., Taillard, E., 1997, A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows, Transportation Science 31.
- [17] Goodrich, M.T., Tamassia, R., 2001, Data Structures and Algorithms in Java, 2nd Edition, John Wiley & Sons, New York.
- [18] Holland, J., 1992, Adaptation in Natural and Artificial Systems, MIT Press, Cambridge, MA.
- [19] Janikow, C.Z., 1993, A Knowledge Intensive GA for Supervised Learning, Machine Learning V.13 2-3.
- [20] Kelly, J., Xu, J., 1996, A Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem, Transportation Science 30.
- [21] Kindervater, G.A.P., Savelsbergh, M.W.P., 1997, Vehicle Routing: Handling Edge Exchanges, E.H. Aarts, J.K. Lenstra (editors), Local Search in Combinatorial Optimization, John Wiley & Sons, Chichester.
- [22] Michalewicz, Z., Fogel, D.B., 2002, How to Solve It: Modern Heuristics, 3rd Edition, Springer-Verlag, Berlin.
- [23] Michalewicz, Z., 1994, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, Berlin.
- [24] Psaraftis, H., 1988, Dynamic Vehicle Routing Problems, In Vehicle Routing: Methods and Studies, B.L. Golden and A.A. Assad (editors).

- [25] Ralphs, T.K., Kopman, L., Pulleyblank, W.R., Trotter, L.E., 2002, On the Capacitated Vehicle Routing Problem, Mathematical Programming Manuscript, Department of Industrial and Systems Engineering, Lehigh University, Bethlehem.
- [26] Rinaldi, G., Naddef, D., Corberán, A., Benavent, E., Belenguer, J.M., Augerat, P., 1995, Computational Results with a Branch and Cut Code for the Capacitated Vehicle Routing Problem, Research Report 949-M, University Joseph Fourier, Grenoble, France.
- [27] Savelsbergh, M.W.P., Sol, M., 1995, The General Pickup and Delivery Problem, *Transportation Science* 29.
- [28] Shaw, P., 1998, Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems, *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming*, M. Maher and J.F. Puget (editors), Springer-Verlag, Berlin.
- [29] Spears, W.M., 1992, Crossover or Mutation?, In Whitley (editor), *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, CA.
- [30] Taillard, E., Agazzi, G., Gambardella, L.M., 1999, A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows, In D. Corne, M. Dorigo and F. Glover (editors), *New Ideas In Optimization*, McGraw-Hill, London.
- [31] Thangiah, S.R., Salhi, S., 2001, Genetic Clustering: An Adaptive Heuristic for the Multi Depot Vehicle Routing Problem, *Applied Artificial Intelligence*, Computer Science Department, Slippery Rock University.
- [32] Van Wijck, W., 2004, An Efficient and Robust Method For Evaluating the Effectiveness of Scheduling Alternatives for a Common Distribution Logistics Problem, Unpublished Article, Department of Industrial Engineering, University of Stellenbosch.
- [33] Winston, W.L., 1994, *Operations Research: Applications and Algorithms*, 3rd Edition, Wadsworth Publishing Company, Belmont, California.

SOLUTION VECTORS FOR TEST PROBLEMS

Solution Vector for First Test Problem

Grid point: 0
0 1
1 0

Grid point: 1
0 1
1 0

Grid point: 2
0 1
1 0

Grid point: 3
0 1
1 0

Grid point: 4
0 1
1 0

Grid point: 5
0 1
1 0

**Solution Vector for Second Test Problem**

Grid point: 0
0 1
1 0

Grid point: 1
0 1
1 0

Grid point: 2
0 1
1 0

Grid point: 3
1 0
0 1

Grid point: 4
0 1
1 0

Solution Vector for Third Test Problem

Grid point: 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 1 0

Grid point: 1
0 0 0 0 0 0
0 0 0 0 0 0
0 1 0 1 0 0
0 0 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 0

Grid point: 2
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 1
0 0 0 0 0 0
0 0 0 0 0 0

Grid point: 3
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0
0 0 0 0 1 0

Grid point: 4
0 0 0 0 0 0
0 0 0 0 0 0
0 1 0 1 0 0
0 0 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 0

Grid point: 5
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 0 0 0
0 0 1 0 0 1
0 0 0 0 0 0
0 0 0 0 0 0



VISUAL BASIC CODE

Declaration Section

```

Private Structure NodePoint
    Dim name As Integer
    Dim time As Integer
End Structure
Private Structure TreeNode
    Dim CurrNode As Integer
    Dim NextNode() As Integer
    Dim Weight() As Double
    Dim VisitNumber As Integer
    Dim Distance As Double
    Dim TmpVar As Double
End Structure

Dim nodelist(,) As NodePoint
Dim treenodelist(,) As TreeNode
Dim times(,), prob(,), converge, learn, mutate, shift As Double
Dim day, grid, slots, vehicles, pop As Integer
Dim vector(,,,), testsol(,), dtestsol(,), dvector(,,,), globalbest(,,)
Dim cbestsol As Integer
Dim initroutes(,), nodes, repnodes As Integer
Dim iteration, iterations, solution, routenr As Integer
Dim routes(,), tabu As ArrayList
Dim dataloaded, converged As Boolean
Dim reached(,), cbestfit, gbestfit, popfit As Double

```

Algorithm Starting Sub Procedure

```

LoadTimesAndRouteData()
If Not dataloaded = True Then Exit Sub
Initiate()
iteration = 0 : gbestfit = Double.PositiveInfinity
Do While iteration < iterations
    ConstructRoutesPopulation()
    ConvertRoutesPopulationToGraphs()
    EvaluateGraphs()
    UpdateProbabilities()
    If converged = True Then Exit Do
    MutateProbabilities()
    iteration += 1
Loop

```

Initialization Sub Procedure

```

Private Sub Initiate()
    Dim i, j, t, p As Integer
    day = txtDay.Text : grid = txtGrid.Text : slots = day / grid
    repnodes = (slots * cycles + 1) * nodes
    pop = txtPop.Text
    converge = txtConv.Text : iterations = txtIter.Text
    learn = txtLR.Text : mutate = txtMutate.Text

```

```

shift = txtShift.Text
ReDim prob(slots * cycles - 1, nodes - 1, nodes)
ReDim vector(pop - 1, slots * cycles - 1, nodes - 1, nodes - 1)
ReDim testsol(slots - 1, nodes - 1, nodes - 1)
ReDim dtestsol(slots - 1, nodes - 1, nodes - 1)
ReDim globalbest(slots - 1, nodes - 1, nodes - 1)
ReDim dvector(pop - 1, slots - 1, nodes - 1, nodes - 1)
ReDim routes(pop - 1, vehicles - 1)
ReDim nodelist(pop - 1, renodes - 1)
ReDim treenodelist(pop - 1, renodes - 1)
ReDim reached(pop - 1, slots * nodes - 1, nodes - 1)
For t = 0 To slots * cycles - 1
  For i = 0 To nodes - 1
    For j = 0 To nodes - 1
      prob(t, i, j) = 0.5
      If times(i, j) = -1 Then times(i, j) = day * vehicles
    Next j
    prob(t, i, nodes) = nodes * 0.5
  Next i
Next t
End Sub

```

Data Loading Sub Procedure

```

Private Sub LoadTimesAndRouteData()
  Dim fd As OpenFileDialog
  Dim temp As Double
  Dim i, j, n As Integer
  dataloaded = False
  fd = New OpenFileDialog()
  With fd
    .CheckFileExists = True
    .CheckPathExists = True
    .Filter = "Text Files|*.txt"
    .Multiselect = False
  End With
  If .ShowDialog() = DialogResult.OK Then
    FileOpen(1, .FileName, OpenMode.Input)
    i = 0
    Do While Not EOF(1)
      Input(1, temp)
      i += 1
    Loop
    FileClose(1)
    n = Sqrt(i)
    nodes = n
    ReDim times(n - 1, n - 1)
    FileOpen(1, .FileName, OpenMode.Input)
    For i = 0 To n - 1
      For j = 0 To n - 1
        Input(1, times(i, j))
      Next j
    Next i
    FileClose(1)
    dataloaded = True
  End If
End Sub
With fd
  If Not dataloaded = True Then Exit Sub
  If .ShowDialog() = DialogResult.OK Then
    FileOpen(1, .FileName, OpenMode.Input)
    i = 0

```

```

Do While Not EOF(1)
  Input(1, temp)
  i += 1
Loop
FileClose(1)
n = i / 2
vehicles = n
ReDim initroutes(n - 1, 1)
FileOpen(1, .FileName, OpenMode.Input)
For i = 0 To n - 1
  For j = 0 To 1
    Input(1, initroutes(i, j))
  Next j
Next i
FileClose(1)
Else
  dataloaded = False
End If
End With
End Sub

```

Routes Construction Sub Procedure

```

Private Sub ConstructRoutesPopulation()
  Dim length, lose, ins, tmp As Integer
  Dim curr, finl As SP
  For solution = 0 To pop - 1
    For routenr = 0 To vehicles - 1
      length = 0
      tabu = New ArrayList()
      routes(solution, routenr) = New ArrayList()
      routes(solution, routenr).Add(New SP(initroutes(routenr, 0), 0))
      routes(solution, routenr).Add(New SP(initroutes(routenr, 1), slots))
      curr = routes(solution, routenr).Item(0)
      finl = routes(solution, routenr).Item(routes(solution, routenr).Count
- 1)
      Do
        ins = GetRandomSP(curr.Name, curr.Time)
        If Not tabu.Contains(ins) Then
          lose = Int(times(curr.Name, ins) / grid) + Int(times(ins,
finl.Name) / grid) + 2
          If lose + length <= slots Then
            tmp = curr.Time + Int(times(curr.Name, ins) / grid) + 1
            routes(solution, routenr).Insert(routes(solution, routenr).Count
- 1, New SP(ins, tmp))
            length = length + Int(times(curr.Name, ins) / grid) + 1
            curr = routes(solution, routenr).Item(routes(solution,
routenr).Count - 2)
            finl = routes(solution, routenr).Item(routes(solution,
routenr).Count - 1)
            If (length = slots - 1) Then Exit Do
          Else
            tabu.Add(ins)
            If (tabu.Count = nodes) Then Exit Do
          End If
        End If
      Loop
    Next routenr
  Next solution
End Sub

```


Function Returning Random SP

```

Private Function GetRandomSP(ByVal naam, ByVal tyd)
    Dim co As Integer
    Dim x, sum As Double
    x = prob(tyd, naam, nodes) * Rnd()
    sum = 0
    For co = 0 To nodes - 1
        sum += prob(tyd, naam, co)
        If x < sum Then
            Return co
        End If
    Next co
End Function

```

Route Conversion Sub Procedure

```

Private Sub ConvertRoutesPopulationToGraphs()
    Dim po, ct, cn, nn, nt, ti, fr, tmp, r, i, j, k As Integer
    Dim found As Boolean
    For solution = 0 To pop - 1
        For i = 0 To slots - 1
            For j = 0 To nodes - 1
                For k = 0 To nodes - 1
                    vector(solution, i, j, k) = 0
                Next k
            Next j
        Next i
        For routenr = 0 To vehicles - 1
            po = 0
            Do While po < routes(solution, routenr).Count - 1
                ct = routes(solution, routenr).Item(po).Time
                cn = routes(solution, routenr).Item(po).Name
                nn = routes(solution, routenr).Item(po + 1).Name
                vector(solution, ct, cn, nn) = 1
                For ti = 0 To slots * cycles - 1 Step slots
                    vector(solution, ct + ti, cn, nn) = 1
                Next ti
                po += 1
            Loop
        Next routenr
        dvector = vector.Clone
        tmp = 0
        For ti = 0 To slots * cycles
            For fr = 0 To nodes - 1
                If ti < slots * cycles Then vector(solution, ti, fr, fr) = 1
                nodelist(solution, tmp).name = fr
                nodelist(solution, tmp).time = ti
                tmp += 1
            Next fr
        Next ti
        r = 0
        For k = 0 To slots * cycles
            For i = 0 To nodes - 1
                treenodelist(solution, r).CurrNode = r
                For j = 0 To nodes - 1
                    If k = slots * cycles Then
                        ReDim Preserve treenodelist(solution, r).NextNode(j)
                        treenodelist(solution, r).NextNode(j) = -1
                    Else

```

```

    If dvector(solution, k, i, j) = 1 Then
        found = False
        fr = 0
        Do While found = False
            nt = nodelist(solution, r).time +
Int(times(nodelist(solution, r).name, j) / grid) + 1
            If ((nodelist(solution, fr).name = j) And
(nodelist(solution, fr).time = nt)) Then
                ReDim Preserve treenodelist(solution, r).NextNode(j)
                treenodelist(solution, r).NextNode(j) = fr
                found = True
            End If
            fr += 1
        Loop
        Else
            ReDim Preserve treenodelist(solution, r).NextNode(j)
            treenodelist(solution, r).NextNode(j) = -1
        End If
    End If
Next j
r = r + 1
Next i
Next k
For i = 0 To repnodes - 1
    For j = 0 To nodes - 1
        If Not (treenodelist(solution, i).NextNode(j) = -1) Then
            If i = j Then
                ReDim Preserve treenodelist(solution, i).Weight(j)
                treenodelist(solution, i).Weight(j) = grid
            Else
                ReDim Preserve treenodelist(solution, i).Weight(j)
                treenodelist(solution, i).Weight(j) = times(nodelist(solution,
treenodelist(solution, i).CurrNode).name, nodelist(solution,
treenodelist(solution, i).NextNode(j)).name)
            End If
        End If
    Next j
Next i
Next solution
End Sub

```

Graph System Evaluation Sub Procedure

```

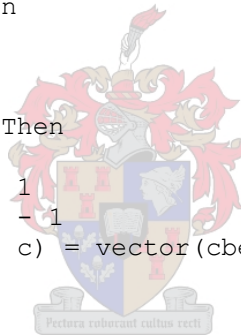
Private Sub EvaluateGraphs()
    Dim fro, vis, count, p, i, j, a, b, c As Integer
    Dim d, total As Double
    popfit = 0 : cbestsol = -1 : cbestfit = Double.PositiveInfinity
    For p = 0 To pop - 1
        For i = 0 To slots * nodes - 1
            For j = 0 To nodes - 1
                If nodelist(p, i).name = j Then
                    reached(p, i, j) = 0
                Else
                    reached(p, i, j) = Double.PositiveInfinity
                End If
            Next j
        Next i
    Next p
    For solution = 0 To pop - 1
        For fro = 0 To slots * nodes - 1
            For vis = fro + 1 To repnodes - 1

```

```

    If reached(solution, fro, nodelist(solution, vis).name) =
Double.PositiveInfinity Then
        If Not (nodelist(solution, fro).name = nodelist(solution,
vis).name) Then
            d = DijkstraPathFinder(solution, fro, vis)
            If d <= day Then reached(solution, fro, nodelist(solution,
vis).name) = d
        End If
    End If
Next vis
Next fro
count = 0 : total = 0
For fro = 0 To slots * nodes - 1
    For vis = 0 To nodes - 1
        If (reached(solution, fro, vis) < Double.PositiveInfinity) And
(reached(solution, fro, vis) > 0) Then
            count = count + 1
            total += reached(solution, fro, vis)
        Else
            count = count + 1
            total += (day * vehicles)
        End If
    Next vis
Next fro
total /= count
If total <= cbestfit Then
    cbestfit = total
    cbestsol = solution
End If
If cbestfit <= gbestfit Then
    For a = 0 To slots - 1
        For b = 0 To nodes - 1
            For c = 0 To nodes - 1
                globalbest(a, b, c) = vector(cbestsol, a, b, c)
            Next c
        Next b
    Next a
    gbestfit = cbestfit
End If
popfit += total
Next solution
popfit /= pop
End Sub

```



Dijkstra Path-Finding Function

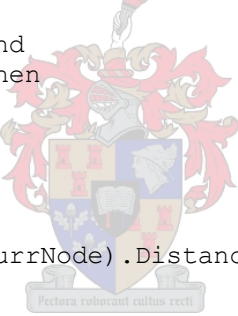
```

Private Function DijkstraPathFinder(ByVal Sol, ByVal NodeSrc, ByVal
NodeDest)
    Dim i As Integer
    Dim bRunning As Boolean
    Dim CurrentVisitNumber As Integer
    Dim CurrNode As Integer
    Dim LowestNodeFound As Integer
    Dim LowestValFound As Double
    If NodeSrc = NodeDest Then
        Return 0
    End If
    For i = 0 To repnodes - 1
        treenodelist(Sol, i).VisitNumber = -1
        treenodelist(Sol, i).Distance = -1
        treenodelist(Sol, i).TmpVar = day * vehicles
    
```

```

Next i
treenodelist(Sol, NodeSrc).VisitNumber = 0
CurrentVisitNumber = 0
CurrNode = NodeSrc
treenodelist(Sol, NodeSrc).Distance = 0
treenodelist(Sol, NodeSrc).TmpVar = 0
Do While bRunning = False
  For i = 0 To nodes - 1
    If Not (treenodelist(Sol, CurrNode).NextNode(i) = -1) Then
treenodelist(Sol, treenodelist(Sol, CurrNode).NextNode(i)).TmpVar =
Min(treenodelist(Sol, CurrNode).Weight(i) + (nodelist(Sol, treenodelist(Sol,
CurrNode).CurrNode).time * grid), treenodelist(Sol, treenodelist(Sol,
CurrNode).NextNode(i)).TmpVar)
    Next i
    LowestValFound = day * vehicles
    For i = 0 To repnodes - 1
      If ((treenodelist(Sol, i).TmpVar <= LowestValFound) And
(treenodelist(Sol, i).TmpVar >= 0) And (treenodelist(Sol, i).VisitNumber <
0)) Then
        LowestValFound = treenodelist(Sol, i).TmpVar
        LowestNodeFound = i
      End If
    Next i
    CurrentVisitNumber = CurrentVisitNumber + 1
    treenodelist(Sol, LowestNodeFound).VisitNumber = CurrentVisitNumber
    treenodelist(Sol, LowestNodeFound).Distance = treenodelist(Sol,
LowestNodeFound).TmpVar
    CurrNode = LowestNodeFound
    If CurrNode = NodeDest Then
      bRunning = True
    Else
      bRunning = False
    End If
  Loop
  Return treenodelist(Sol, CurrNode).Distance
End Function

```



Probability Vector Update Sub Procedure

```

Private Sub UpdateProbabilities()
  Dim t, i, j As Integer
  Dim sum As Double
  converged = True
  For t = 0 To slots * cycles - 1
    For i = 0 To nodes - 1
      sum = 0
      For j = 0 To nodes - 1
        prob(t, i, j) = (prob(t, i, j) * (1 - learn)) + (learn *
vector(cbestsol, t, i, j))
        sum += prob(t, i, j)
        If (converge <= prob(t, i, j)) And (prob(t, i, j) <= 1 - converge)
Then
          converged = False
        End If
      Next j
      prob(t, i, nodes) = sum
    Next i
  Next t
End Sub

```

Probability Vector Mutate Sub Procedure

```

Private Sub MutateProbabilities()
  Dim mut, sum, rndzeroone As Double
  Dim t, i, j As Integer
  For t = 0 To slots * cycles - 1
    For i = 0 To nodes - 1
      sum = 0
      For j = 0 To nodes - 1
        mut = Rnd()
        If mut < mutate Then
          rndzeroone = Int(Rnd(2))
          prob(t, i, j) = (prob(t, i, j) * (1 - shift)) + (rndzeroone *
shift)
        End If
        sum += prob(t, i, j)
      Next j
      prob(t, i, nodes) = sum
    Next i
  Next t
End Sub

```

