Metaheuristics for petrochemical blending problems

by

Lieschen Venter

Thesis presented in partial fulfillment of the requirements for the degree of Masters of Commerce

at

Stellenbosch University

Department of Logistics

Faculty of Economic and Management Sciences

Supervisor: Prof SE Visagie
Date: February 18, 2010

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the authorship owner thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Signature: .....

Date: February 18, 2010

Abstract

The main aim in blending problems is to determine the "best" blend of available ingredients to form a certain quantity of product(s). This product should adhere to strict specifications. In this study the best blend means the least-cost blend of ingredients (input) required to meet a minimum level of product (output) specifications. The most prevalent tools to solve blending problems in the industry are by means of spreadsheets, simulators and mathematical programming. While there may be considerable benefit in using these types of tools to identify potential opportunities and infeasibilities, there is a potentially even greater benefit in searching automitically for alternative solutions that are more economical and efficient. Heuristics and metaheuristics are presented as useful alternative solution approaches.

In this thesis different metaheuristic techniques are developed and applied to three typical blending problems of varied size taken from the petrochemical industry. a fourth instance of real life size is also introduced. Heuristics are developed intuitively, while metaheuristics are adopted from the literature. Random search techniques, such as blind random search and local random search, deliver fair results. Within the class of genetic algorithms the best results for all three problems were obtained using ranked fitness assignment with tournament selection of individuals. Good results are also obtained by means of tabu search approaches - even considering the continuous nature of these problems. A simulated annealing approach also yielded fair results. A comparison of the results of the different approaches shows that the tabu search technique delivers the best result with respect to solution quality and execution time for all three the problems under consideration. Simulated annealing, however, delivers the best result with respect to solution quality and execution time for the introduced real life size problem.

Opsomming

Die hoofdoelwit met die oplos van mengprobleme is om die "beste" mengsel van beskikbare bestandele te bepaal om 'n sekere hoeveelheid produk(te) te vervaardig. Die produk moet aan streng vereistes voldoen. Die beste kombinasie is die goedkoopste kombinasie van bestandele (toevoer) wat aan die minimum produkvereistes (afvoer) voldoen. Die algemeenste benaderings waarmee mengprobleme in die industrie opgelos word, is met behulp van sigblaaie, simulasies en wiskundige programmering. Hierdie metodes is baie nuttig om belowende oplossings of ontoelaatbaarhede te identifiseer, maar dit kan potensieel meer voordelig wees om metodes te gebruik wat sistematies meer ekonomiese en effektiewe oplossings vind. Heuristieke en metaheuristieke word as goeie alternatiewe oplossingsbenaderings aangebied.

In hierdie tesis word verskillende metaheuristiekbenaderings toegepas op drie tipiese mengprobleme van verskillende groottes wat vanuit die petrochemiese industrie spruit. 'n Vierde geval met realistiese (regte wêreld) grootte word ook aangebied. Heuristieke word volgens intuïsie ontwikkel terwyl metaheuristieke aangepas word vanuit die literatuur. Lukrake soektegnieke soos die blinde lukrake soektegniek en die plaaslike lukrake soektegniek lewer redelike resultate. Binne die klas van genetiese algoritmes word die beste resultate gelewer wanneer die algoritme met 'n kombinasie van rangorde fiksheidstoekenning en toernooiseleksie van individue geïmplimenteer word. Goeie resultate word ook verkry met behulp van tabusoektogbenaderings — ten spyte van die kontinue aard van hierdie probleme. Gesimuleerde tempering lewer ook redelike resultate. 'n Vergelyking van die resultate van die verskillende tegnieke toon dat die tabusoektogtegniek die beste resultate met betrekking tot die kwaliteit van die oplossing sowel as uitvoertyd lewer. Gesimuleerde tempering lewer egter die beste resultate met betrekking tot die kwaliteit van die oplossing sowel as uitvoertyd vir die voorgestelde realistiese grootte probleem.

Acknowledgements

I would like to thank:

- the Lord Jesus Christ, for freedom and revelation of what all of this is *really* about;

  > *He is the image of the invisible God, the firstborn over all creation. For by Him all things were created that are in heaven and that are on earth, visible and invisible, whether thrones or dominions or principalities or powers. All things were created through Him and for Him. And He is before all things, and in Him all things consist.* Colossians 1:15-17;

- the Department of Logistics, for the use of their office space and facilities;
- Prof SE Visagie, the supervisor of this thesis, for his unconditional trust, his patient guidance and his loyal friendship;
- Dr Aninda Chakraborty, the Sasol contact of this thesis, for his guidance and provision of industry information;
- my fellow GoreLab natives, for their company and assistance, especially Frank Ortmann, for his unrivaled LaTeX prowess and Darian Raad, for the occasional tango;
- and family, friends, flatmates and familiars for their love, support and interest.

The financial assistance of Sasol Technology towards this research is hereby acknowledged. Any opinions, findings, conclusions or recommendations expressed in this thesis are those of the author and are not necessarily to be attributed to Sasol.

Contents

List of Figures

List of Tables

List of Acronyms

AOA	Average octane approach
BRS	Blind random search
CTS	Continuous tabu search
CTSh	Continuous tabu search by the hypersquare method
CTSz	Continuous tabu search by the immediate zone method
DIA	Differential inventory approach
GA	Genetic algorithm
HPP	Haverly pooling problem
LRS	Local random search
LS	Local search
MBA	Maximized blend approach
MCIA	Minimum closing inventory approach
MIA	Minimum inventory approach
MIPA	Maximum input approach
MMRP	Marco mini-refinery problem
RON	Research octane number
RST	Random search technique
RVI	Reid vapor index
RVP	Reid vapor pressure
SA	Simulated annealing
SSP	Simplified sample problem
TAME	Tertiary amyl methyl ether
TS	Tabu search

List of Reserved Symbols

A number of symbols in this thesis will conform to the following definition:		
$\mathcal{A}$	Symbol denoting a **set**	(Calligraphy capitals)
$\mathbf{a}$	Symbol denoting a solution or portion of a solution	(Boldface lower case letters)
$\mathbf{A}$	Symbol denoting a **matrix**	(Boldface capital letters)
$\underline{\mathrm{a}}$	Symbol denoting a **vector**	(Underlined lower case letters)

Symbol	Meaning
α	Damping constant.
A_{ji}	Process test variable.
b_{it}	Amount in m^3 of blend i that is produced on day t.
β	Tabu region constant.
c_{ij}	Amount in m^3 of component j that is used to produce blend i.
c'_{jkm}	Amount in m^3 of component j from crude k obtained after the crude. has passed through process m.
$\breve{c}_{ij}$	Individual constraint set on the percentage of which blend i may consist. of component j.
$\bar{c}_{jk}$	Amount of component j to go into the pooling mix k.
$\hat{c}_{ij}$	Fraction of blend i which consists of component j.
$\tilde{c}_{ik}$	Amount of pooling mix k required to produce product i.
$C_k^{c'}$	Cost price per barrel of crude k.
C_i^b	Selling price per m^3 of blend i.
C_i^p	Selling price per m^3 of product i.
C_j^c	Cost price per m^3 of component j.
C_m^o	Operating cost per unit crude associated with process m.
$\underline{d}$	Zero mean deviates to be added to the current solution.
d_j	Amount per barrel of domestic product required to form the final product j.
$\boldsymbol{D}$	Maximum change allowed in each decision variable.
E	Thermodynamic energy of the system.
ϵ_f	Tabu list tolerance.
$\underline{\varepsilon}$	A vector of uniform random numbers in the range $(-\sqrt{3}, \sqrt{3})$.
h	Number of submatrices in the data structure.
Γ	Acceptance function.
η	Temperature decrementation constant.
g	Infeasible solution for the penalty function.

I_j^O	Opening inventory level for component j.
I_j^L	Minimum inventory level for component j.
I_j^U	Maximum inventory level for component j.
κ	Any large negative constant [O(10^6)].
K_k	Number of barrels of crude k that is bought as raw material.
$K_{km}^{'}$	Number of barrels of crude k which passes through process m.
$\underline{l}$	Lower bounds of the control variables.
λ	CTSz constant.
L	Markov chain length.
n	Number of candidate solutions generated.
ν	Number of transitions between candidate solutions at a temperature T.
$\mathcal{N}$	Tabu search neighbourhood search space.
ω	Weight associated with the magnitudes of the successful changes made to each control variable.
ν	Information infolding rate
$O_i^{\min}$	Minimum allowable octane rating for blend i.
O_i^*	Octane rating of product i.
O_j	Octane rating of component j.
p	Number of LP model constraints.
p_c	Cross-over probability.
p_m	Mutation probability.
p_t	Selection probability for tournament selection.
ϕ	Reid vapour index constant.
$\boldsymbol{\xi}$	Covariance matrix.
P_i	Number of barrels of product i that is produced.
P_j^*	Reid vapour pressure of component j.
P_i^{u*}	Maximum allowable Reid vapour pressure for blend. i
$P_j^{'}$	Reid vapour index of component j.
$P_i^{u'}$	Maximum allowable Reid vapour index for blend i.
Q	Step size distribution controller.
$\rho_i^{\max}$	Maximum allowable density of product i.
ρ_{jk}	Density of component j obtained from crude k.
r	Random number in the range $(0, 1)$.
R_{jt}	Run down value of component j on day t.
ϱ	Position of an individual in a solution population.
s	Number of LP model constraints that have been satisfied.
$S_i^{\max}$	Maximum allowable sulfur content of product i.
S_j	Sulfur content of component j.
S_{jk}	Sulfur content of component j obtained from crude k.
$\tilde{S}_k$	Sulfur content of the pooled component mix k.
$\mathcal{S}$	Solution search space.
t	Tabu tenure.
θ	Relative accuracy for the tabu search.
$\underline{\tau}$	Random direction vector.
T	Global temperature parameter.
$\mathcal{T}$	Tabu list.

$\underline{\varpi}$	A vector of uniform random numbers in the range [-1,1].
$\underline{u}$	Upper bounds of the control variables.
U_k	Upperbound on the number of barrels of crude k that may be purchased.
$U_m^{'}$	Upperbound on the number of barrels of crude that is present in process m.
Υ	Magnitudes of the successful changes made to each control variable.
$V_i^{\max}$	Maximum allowable vapour pressure of product i.
V_j	Vapour pressure of component j.
$\mathcal{V}$	Solution search space excluding the neighbourhood search space.
Φ	Candidate distribution.
χ	Desired acceptance probability for the initial temperature calculation.
Ψ	Annealing schedule.

Chapter 1

Introduction

In blending problems the aim is to determine the best blend of available ingredients to form a certain quantity of a product under strict specifications. The best blend means the least-cost blend of inputs required to meet a designated level of output or given specifications. Blending problems are especially important in process industries such as petroleum, chemical, and food, as well as in fields where a certain level of service is desired at minimum cost. The decision maker must determine the ingredients to use and in which quantities to use them.

Sasol, originally the Afrikaans acronym for *Suid-Afrikaanse Steenkool en Olie* (South African Coal and Oil), is a South African company engaged in the commercial production and marketing of chemicals and liquid fuels. Headquartered in Johannesburg, it supplies approximately 40% of the national liquid fuel requirements and is the country's largest supplier of industrial gas, explosives, fertilizers, polymers and chemical products.

Sasol continually encounters blending problems during its operations. In the production process numerous product specifications must be met through a number of components that are generally available for each product blend type. Product blends almost never consist of only one type of component and different combinations of the components used to form products have significantly different economic values as result. The quality and amount of each component available for production depends on upstream-process feedstock qualities and on changes in operating conditions. As for the products themselves, some finished product demands are flexible and the optimal volume may change based on economic conditions.

There are an infinite number of blending recipes which will make a product, but there is only one set of feedstock, operating conditions, component yields, qualities and blending recipes that satisfies the inventory constraints and meets all product specifications at the highest economic value. Blend planning methods are intended to identify the optimal operating conditions and identify the feasible and optimal blend recipes. Maximum profit is realized by the planning and implementation of optimal operating conditions and through

implementation of optimal blending strategies.

The most prevalent form of blending tools being used in the industry are spreadsheets and simulators that allow the user to visualize the impact that a given change to a recipe will have [57]. Currently, Sasol's *Market and Process Integration* (MPI) group utilizes spreadsheets to develop blend recipes. These spreadsheets do not optimize blending but are predominately used to manage production and inventory to achieve the blending recipes for the following few weeks. This is no small task as in Sasol, a typical single period refinary LP has approximately 3 000 constraints and 3 000 variables. For multi-period models the variable count can easily increase to 20 000.

However, the optimal solution may be found by means of linear programming and there are several mathematical modeling languages that could be used for formulating and solving LP models such as Lingo [66], GAMS [35] and AMPL [6]. In refinery and petrochemical processing problems it is generally necessary to model not only product flows but the properties of the components as well. When components are combined, nonlinear relationships are often introduced. In a number of blending problems, the qualities of the components contribute to the qualities of the products in a nonlinear and nonconvex manner. *Succesive Linear Programming* (SLP) techniques have been widely used in the industry for over 25 years [31]. SLP algorithms solve nonlinear optimization problems via a sequence of linear programs. Palacios-Gomez *et al.* [74] presents the first such algorithm, the *Method of Approximation Programming* (MAP).

While there is great benefit in using these types of tools to identify potential opportunities and infeasibilities, there is an even greater benefit in searching automatically for alternate recipes that are more economical and efficient. The ideal is to have some general solution method for nonlinear programs (such as for linear programs and integer programs) that always produces the global optimum for any nonlinear program. However, no such solution method exists — a local optimum is produced and it cannot be ensured (in all cases) that a solution is the global optimum [13]. Metaheuristics are useful alternatives in overcoming this problem.

A metaheuristic is a heuristic method for solving a very general class of computational problems by combining user-given procedures — usually heuristics themselves — in the hope of obtaining a more efficient or more robust procedure. The name combines the Greek prefix "meta" (meaning "beyond", here in the sense of "higher level") and "heuristic" (meaning "to find") [101]. Metaheuristics are generally applied to problems for which there is no satisfactory problem-specific algorithm or heuristic or when it is not practical to implement such a method. Most commonly used metaheuristics are targeted to solve combinatorial optimization problems, but of course it can handle any problem that can be recasted in the right form.

1.1 Thesis scope and objectives

As far as could be ascertained, there exist no application of metaheuristic approaches to petrochemical blending problems in the literature. The scope of this thesis is limited to the development of various approaches to three sample problems supplied by Sasol in order to achieve a proof of concept. The main thrust of this thesis is to develop metaheuristic approaches to the three sample problems supplied by Sasol. This is achieved by pursuing five objectives.

Objective I:

a. To *determine* the exact solution for each problem so that the quality of solutions obtained by other approaches may be measured through comparison;

b. To *understand* the nature and characteristics of the problems at hand by means of decomposition;

Objective II:

a. To *examine* the nature of the solution spaces for each problem, *determine* the level of difficulty in finding feasible solutions within them and *find* methods to deal with infeasible solutions;

b. To *develop* data structures for the representation of decision variables and constraints upon these variables for each of the three problems;

Objective III:

a. To *formulate* a solution approach by means of random search techniques and *measure* which technique in particular delivers the best solution for each problem;

b. To *formulate* a solution approach by means of genetic algorithms and *measure* which configuration of algorithm parameters and subalgorithms deliver the best result for each problem;

c. To *examine* the possibility of formulating a tabu search approach despite the continuous nature of the problems at hand and *determine* which method delivers the best result for each problem;

d. To *formulate* a solution approach by means of simulated annealing and *measure* which combination of algorithm parameters delivers the best result for each problem;

Objective IV:

a. To *compare* the performance of each solution approach with respect to average solution quality, stability and execution time;

b. To *investigate* the behavior of the metaheuristic solution approaches for a problem with increased, *i.e.* real life, dimensions.

Objective V: To *pose* open questions and to *suggest* new further development of the approaches to problems of greater size and resemblance to industry type problems.

1.2 Thesis layout and organisation

In Chapter 2 three problems common in the petrochemical industry literature are introduced and described. The first, a simplified sample problem, serves as the base upon which various approaches are developed and upon which initial tests are done before the solutions are applied to a larger problem. The second problem is the Haverly pooling problem and it is used to introduce the concept of the initial combination of certain components into intermediate blends before the combination of them with other components to form the final products. These initial blends have different characteristics than the components which are used to form them. The third problem is the Marco mini-refinery problem. It builds on the simplified sample problem by introducing the crudes from which the components are created as well as the processes used in order to do this.

Chapter 3 contains the formulation of the exact solution to the three problems by means of linear programming. It also contains results obtained from applying a number of so-called "expert" approaches so as to examine the performance of these approaches to solve the different problems. Upperbounds are determined on the quality of solutions obtained by focusing optimisation on only one problem characteristic at a time. In so doing, greater understanding of which problem characteristics are the most important and must receive the most attention during the design of heuristics and metaheuristics, may be obtained.

In Chapter 4 this information is used to determine data structures for each of the three problems. The data structures contain the chosen decision variables and groups them in such a way that the solutions obtained may be handled and manipulated as a single structure.

In Chapter 5 the study of applied heuristics commences and it contains solutions to the problems obtained by applying two random search technique approaches. The application of these techniques reveals information about the nature of the various solution spaces of the problems; it gives an indication of the level of logic required to find good quality solutions by investigating the probability of finding such solutions at random.

Chapter 6 is the first in which the application of metaheuristics is investigated and it contains results obtained from four variations of the genetic algorithm as well as the results of island genetic models. Chapter 7 contains results obtained from the application of two tabu search approaches designed specifically for the continuous nature of the blending problems while Chapter 8 contains results obtained from the application of a simulated annealing approach.

Chapter 9 contains the comparison of each solution approach for the three problems and Chapter 10 contains the results of the metaheuristic approaches applied to an extension of one of the sample problems. Chapter 11, finally, contains the conclusions and closing remarks for the thesis. The chapter closes with a number of ideas with respect to further work.

Chapter 2

Problem Description

To the author's knowledge, there exists no application of metaheuristics to petrochemical blending problems in the literature. The interest therefore lies firstly in achieving a proof of concept whether or not it is indeed possible to solve this class of problems by means of metaheuristics. Three sample problems are supplied by Sasol for the development and testing of metaheuristic approaches.

2.1 The simplified sample problem

When mathematical programming tools are properly integrated with user-friendly interfaces, they turn into effective decision support tools requiring almost no computer programming knowledge. Visual aids and options largely simplify the interpretation process of solutions. Spreadsheets, for example, provide a user-friendly interface for mathematical programs. In particular, Microsoft Excel [71] has become one of the most popular software packages in the business world and have been used by millions of professionals. Ragsdale [80] argues that due to their widespread availability and use in business and engineering community, it is much easier for those with no mathematical programming knowledge to learn and adopt such models when it is interfaced with Microsoft Excel.

The use of spreadsheets for operations research problems is discussed by Leon *et al.* [65] and a spreadsheet application to a production blending problem is given by Al-Shammari and Dawood [2]. Sakalli and Birgoren [86] discuss the development and implementation of spreadsheet-based decision support tools for modeling and solving blending problems in a large-scale brass factory in Turkey. They present a user interface developed in Microsoft Excel, which is linked with the LINGO [66] modeling language and optimizer. Their paper elaborates on difficulties faced in the development and implementation of their solutions as well as the design of the interface.

The first problem considered in this thesis is the *simplified sample problem* (SSP) provided

by KBC Consultants [56]. They developed a spreadsheet management system created in Microsoft Excel for Sasol. The effect of various variables upon each other may be tested by means of this system. The violation of any constraint causes the violating value to be coloured in red. This simplifies violation identification and the testing for feasible solutions.

The SSP considers two petrol blends: Sasol Turbo ULP[1] 93 (Summer Grade) also known as M3S (a 93 octane unleaded grade) and Sasol Turbo ULP 95 (Summer Grade) also known as M5S (a 95 octane unleaded grade). The given selling price for M3S and M5S is R5 300 and R5 430 per cubic meter, respectively. The blends are comprised of five components: *butane* (BUT), *C5 raffinate* (GP1), *unhydrogenated catalytic polypropolene petrol* (GP4), *platformate* (PTF) and *tertiary amyl methyl ether* (TAME).

These components should be blended in such a way as to satisfy the octane rating and vapor pressure specifications given in Table 2.1. The most common type of octane rating worldwide is the *Research Octane Number* (RON). RON is determined by running the fuel in a test engine with a variable compression ratio under controlled conditions, and comparing the results with those for mixtures of iso-octane and n-heptane [102]. *Reid vapor pressure* (RVP) is a common measure of the volatility of petrol. It is defined as the absolute vapor pressure exerted by a liquid at 37.8 řC [103]. TAME is a volatile, low viscosity clear liquid used as an oxygenate to gasoline. It is added both to increase octane enhancement to replace banned tetraethyl lead and to raise the oxygen content in gasoline [104].

Blend	**Price** kR/m^3	**Minimum RON**	**Maximum RVP** KPa	**Maximum TAME %**
M3S	5.30	93	70	15.5
M5S	5.45	95	75	15.5

Table 2.1: *Blend specifications for the SSP.*

Generally octane blending is a nonlinear problem [64], but it is assumed in this problem that octane blends linearly on volume: The sum product of the RON and volume of all the five components in a particular petrol grade is equal to the product of the final volume and RON of the petrol grade. A property that does not blend linearly on volume may be converted to an index which does blend linearly by using

$$\text{property index} = (\text{property value})^{\phi}.$$

RVP does not blend linearly on volume. Therefore, it needs to be converted into a *Reid Vapor Index* (RVI), where $\phi = 1.25$. TAME is high in octane but has a low RVP, which are very good qualities for an additive. However, the high price of TAME restricts addition to a maximum of 15.5% in both petrol grades.

[1] Unleaded Petrol

The properties of each of the five components are given in Tables 2.2 and 2.3. Each of the components (except butane as butane is a domestic product) is subject to inventory constraints which limit the physical amount of component that may be stored. These amounts are influenced by the run down rates. *Run down rates* refer to the replenishment amounts of each component for each day as the components are extracted from crude oils and coal.

The goal is to make an optimal blend recipe that satisfies the inventory and blend specification constraints. A schematic representation of this SSP is supplied in Figure 2.1.

	Opening Inventory m^3	**Minimum Inventory** m^3	**Maximum Inventory** m^3	**Cost** kR/m^3
BUT		Not inventoried		3.00
GP1	1.90	0.60	4.75	4.30
GP4	2.30	0.86	5.75	4.30
PTF	4.74	1.16	11.84	4.80
TAM	2.40	0.40	6.00	5.00

Table 2.2: *Inventory properties of the components which make up each blend for the SSP.*

	Properties		**Run Down Rates**		
	RON	**RVP** KPa	**Day 1** m^3	**Day 2** m^3	**Day 3** m^3
BUT	97.80	350.00	0.36	0.35	0.32
GP1	93.68	106.09	1.03	1.03	1.02
GP4	95.16	55.21	0.86	0.89	1.21
PTF	85.50	43.80	2.29	2.29	2.29
TAM	120.00	18.60	0.92	0.92	0.91

Table 2.3: *Characteristics of the components which make up each blend for the SSP.*

2.2 The Haverly pooling problem

The *Haverly pooling problem* (HPP) is similar to the SSP and is presented in Haverly [44]. It considers two types of final products simply labelled prodX and prodY. These products are formed when 3 components (compA, compB and compC) are combined, but what differentiates the pooling problem from the SSP, is a so-called pooling link. It may exist physically because there is only one tank to store compA and compB in or it may exist because compA and compB must be mixed and transported as a mixture via, for example, a pipeline.

CompA incurs a cost of R6.00 per m^3 purchased while compB and compC incur costs of R16.00 and R10.00 per m^3 purchased, respectively. ProdX is sold at R9.00 per m^3

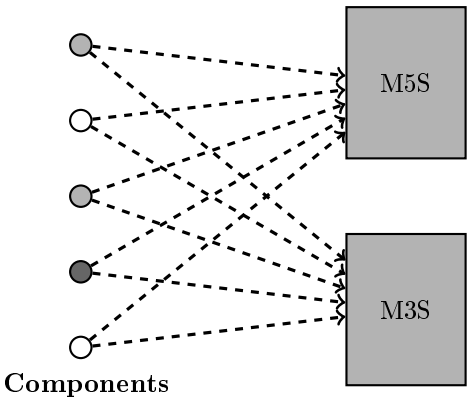

Figure 2.1: *A schematic representation of the SSP.*

while prodY is sold at R15.00 per m^3. A maximum amount of 100 m^3 of prodX may be manufactured while a maximum of 200 m^3 of prodY may be manufactured.

CompA has a sulfur content of 3% per m^3, compB has a sulfur content of 1% per m^3 and compC has a sulfur content of 2% per m^3 while the maximum allowable sulfur content for prodX and prodY is 2.5% and 1.5% per m^3, respectively. When components are pooled together, the sulfur quality of the mix must be estimated according to the quantities of each component in the pool.

From the given information the goal is to make an optimal blend schedule that satisfies the blend specification constraints so that profit is maximized. A schematic representation of the HPP is given in Figure 2.2.

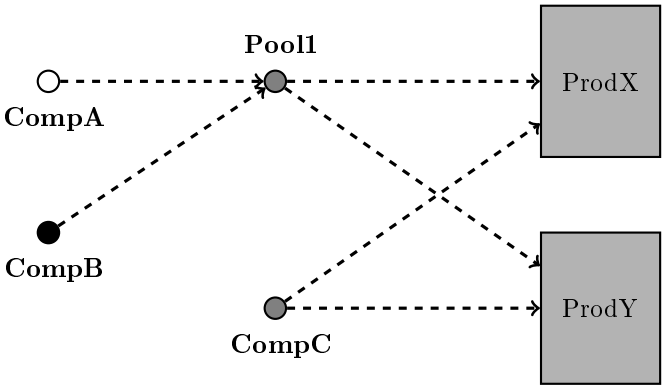

Figure 2.2: *A schematic representation of the HPP.*

2.3 The Marco mini-refinery problem

The *Marco mini-refinery problem* (MMRP) is a generalisation of the SSP discussed in §2.1. It considers five types of final products: Premium grade petrol blends, regular grade petrol blends, distillate, fuel gas and fuel oil. The blends are comprised of eleven components which are obtained from two crude oils (Mid-continent crude and Texas crude). These components are *butane* (but), *fuel gas*, *straight run gasoline* (sr-gas), *straight run naphta* (sr-naphta), *reformed gasoline* (rf-gas), *straight run distillate* (sr-dist), *cracked gasoline* (cc-gas), *cracked gas oil* (cc-gas-oil), *straight run gas oil (sr-gas-oil)*, *straight residuum (sr-res)* and *hydrotreated residuum* (hydro-res).

A maximum of 200 barrels of each type of crude may be purchased each day at a cost of \$60.00 per barrel for both Mid-continent crude and Texas crude. The standard oil barrel of 42 US gallons or 159ℓ is used in the United States as a measure of crude oil and other petroleum products. One standard oil barrel is equal to approximately 1.2 m^3. General attributes for the applicable components to be blended are shown in Table 2.4.

Component	**Octane rating**	**Vapour pressure** (Pa)	**Density** (kg/m^3)	**Sulfur content** (%)
sr-gas	83.50	11.40	-	-
sr-naphta	69.00	9.54	272.00	1.48
rf-gas	110.00	5.57	303.30	-
cc-gas	78.70	9.90	-	-
butane	90.80	22.20	-	-
sr-dist	-	-	292.00	2.86
sr-gas-oil	-	-	295.00	5.05
sr-res	-	-	343.00	11.00
cc-gas-oil	-	-	294.40	1.31
hydro-res	-	-	365.00	6.00

Table 2.4: *General attributes per barrel for the applicable components to be blended for the MMRP. A dash indicates that the component does not have the specific attribute.*

The components are obtained through various chemical processes. Five processes play a role in this problem: *Atmospheric distillation* (a-dist), *naptha reforming* (n-reform), *catalytic cracking of distillates* (cc-dist), *catalytic cracking of gas oil* (cc-gas-oil) and the *hydrotreating of residuum* (hydro). The amount of each component obtained by means of the five processes is given in Tables 2.5 and 2.6. Butane is a domestic product and is manufactured rather than obtained from crudes. Butane has an expense of \$67.50 per barrel to manufacture. The processes have fixed costs as well as costs incurred by each type of component that they produce. The processes are also subject to capacity constraints and operating costs as shown in Table 2.7. All crudes initially pass through a-dist. Therefore all components not obtained through this process must be obtained by running the intermediate streams through the other processes. No new crude is entered into the system to obtain these components. Therefore, the intermediate stream becomes less by one unit for each unit that is run through any of the other processes.

	Process				
Component	**A-dist**	**N-reform**	**Cc-dist**	**Cc-gas-oil**	**Hydro**
sr-gas	0.236	-	-	-	-
sr-naphta	0.233	-1.000	-	-	-
sr-dist	0.087	-	-1.000	-	-
sr-gas-oil	0.111	-	-	-1.000	-
sr-res	0.315	-	-	-	-
rf-gas	-	0.807	-	-	-
fuel-gas	0.029	0.129	0.300	0.310	-
cc-gas	-	-	0.590	0.590	-
cc-gas-oil	-	-	0.210	0.220	-

Table 2.5: *The amount of component per barrel that is obtained from Mid-continent crude oil. A dash indicates that a component is not obtained through that process.*

	Process				
Component	**A-dist**	**N-reform**	**Cc-dist**	**Cc-gas-oil**	**Hydro**
sr-gas	0.180	-	-	-	-
sr-naphta	0.196	-1.000	-	-	-
sr-dist	0.073	-	-1.000	-	-
sr-gas-oil	0.091	-	-	-1.000	-
sr-res	0.443	-	-	-	-1.000
rf-gas	-	0.836	-	-	-
fuel-gas	0.017	0.099	0.360	0.380	-
cc-gas	-	-	0.580	0.600	-
cc-gas-oil	-	-	0.150	0.150	-
hydro-res	-	-	-	-	0.970

Table 2.6: *The amount of component per barrel that is obtained from Texas crude oil. A dash indicates that a component is not obtained through that process.*

Process	**Maximum capacity** (1000 barrels per day)	**Operating cost** (k$ per barrel)
a-dist	100	0.030
n-reform	20	0.045
cc-dist	30	0.240
cc-gas-oil	30	0.024
hydro	-	0.030

Table 2.7: *Process constraints and fixed costs for each of the five processes through which components are obtained from crudes for the MMRP.*

The five types of final products that are produced from the components are subject to certain constraints as shown in Table 2.8. The components which must be combined to produce each type of final product and selling price thereof is contained in Table 2.9.

The objective is to maximise the profit subject to all the above constraints. A schematic

Final product	**Minium RON**	**Maximum RVP** (Pa)	**Maximum density** (kg/m^2)	**Maximum sulfur** (%)
Fuel gas	-	-	-	-
Regular grade petrol	86	12.7	-	-
Premium grade petrol	90	12.7	-	-
Distillate	-	-	306	0.5
Fuel oil	-	-	352	3.5

Table 2.8: *Product constraints for each of the five types of final product. A dash indicates that the constraint is not applicable to the type of final product.*

Product	**Component**	**Selling price** $/barrel
Fuel gas	Fuel gas	15.00
Regular grade petrol	But, sr-gas, rf-gas, cc-gas, sr-naphta	91.00
Premium grade petrol	But, sr-gas, rf-gas, cc-gas, sr-naphta	105.00
Distillate	sr-dist, sr-naphta, sr-gas-oil, cc-gas-oil	77.00
Fuel oil	sr-gass-oil, sr-res, cc-gas-oil, hydro-res	66.50

Table 2.9: *List of components and selling price for each final product for the MMRP.*

representation of the MMRP is given in Figure 2.3.

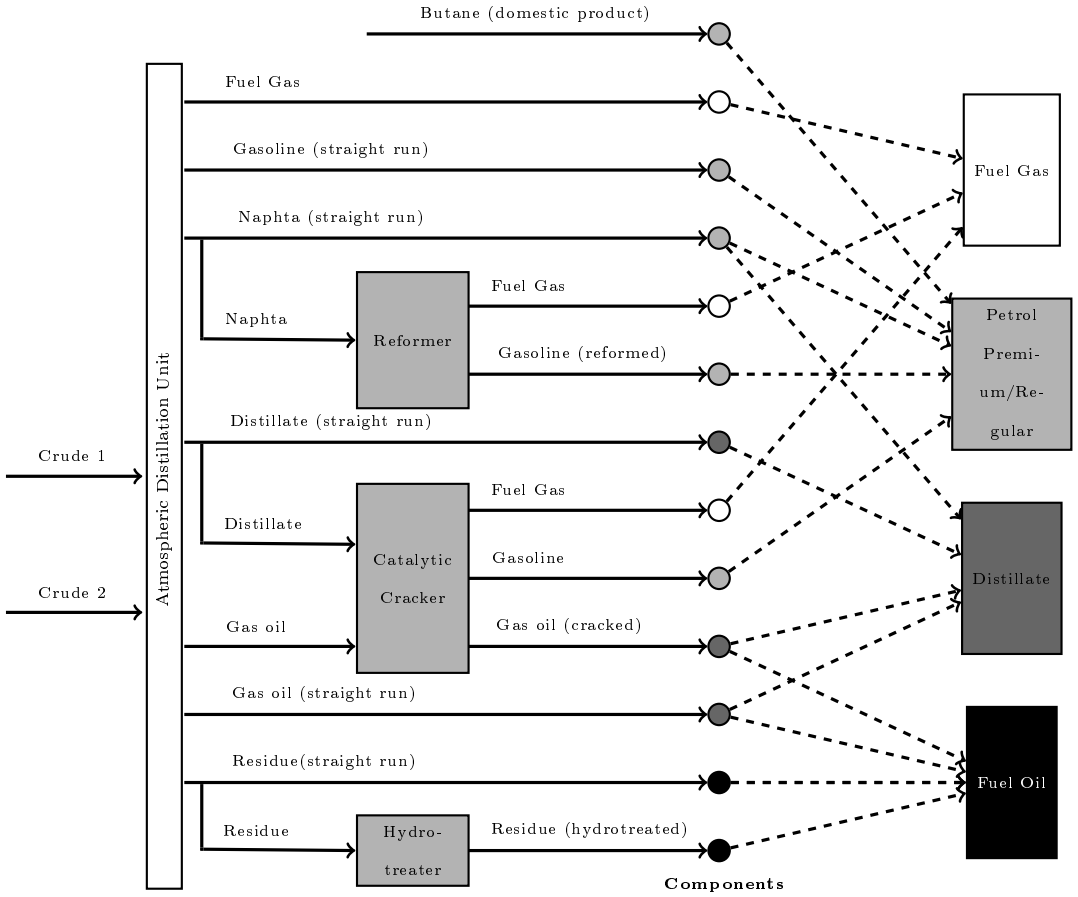

Figure 2.3: *A schematic representation of the MMRP.*

Chapter 3

A linear programming approach

Various solution techniques are explored for the SSP (§2.1), the HPP (§2.2) and the MMRP (§2.3). Exact solutions for the three problems may be obtained by means of linear and non-linear programming. However, the risk of nonlinear programming methods returning merely a local optimum is very high. Thus a need for alternative methods arises. In this chapter, various linear programming (LP) approaches are presented and tested on the problems. In so doing, the inherent characteristics of the problems may be investigated. Greater understanding may be obtained for example, as to which constraints have the greatest effect on moves toward better solutions and which decision variables most influence the objective function values. LPs can also yield more insight into the best possible performance of known greedy heuristics for blending problems.

3.1 An exact solution approach for the SSP

Let C_i^b be the selling price per m^3 of blend i and C_j^c be the cost price per m^3 of component j. Let b_{it} be the amount in m^3 of blend i that is produced on day t and c_{ij} be the amount (in m^3) of component j that is used to produce blend i so that

$$\hat{c}_{ij} = \frac{c_{ij}}{\sum_t b_{it}}$$

represents the fraction of blend i that consists of component j. Let $O_i^{\min}$ be the minimum allowable octane rating for blend i and O_j be the octane rating of component j. Similarly, let P_i^{u*} be the maximum allowable *Reid vapour pressure* (RVP) for blend i and P_j^* be the RVP of component j. After linearization of the pressure as described in §2.1, suppose $P_i^{u'}$ is the maximum allowable *Reid vapour index* (RVI) for blend i and P_j' is the RVI of component j. Furthermore, let I_j^O, I_j^L and I_j^U be the opening, minimum allowable and maximum allowable inventory level, respectively for component j and let R_{jt} be the run

down value of component j on day t. Finally, let $\breve{c}_{ij}$ be the individual constraint set on the percentage of blend i that may consists of component j.

With cost optimization as goal, a total of I blends, a total of J components and a production horizon of total length T, the sample problem is stated as finding a $(T \times I)$ blend solution matrix and a $(M \times J)$ component solution matrix such that $IJ + IT$ nonnegativity constraints, $2I + 2JT + IJ$ inequality constraints and I equality constraints totalling $2I + 2JT + 2IJ + IT$ constraints, will be satisfied.

The objective of the LP is to

$$\text{maximize} \qquad \sum_i \sum_t C_i^b b_{it} - \sum_i \sum_j C_j^c c_{ij}, \tag{3.1}$$

$$\text{subject to} \qquad \sum_j O_j \hat{c}_{ij} \geq O_i^{\min} \quad \text{for all } i, \quad \text{(octane limit)}, \tag{3.2}$$

$$\sum_j P_j' \hat{c}_{ij} \leq P_i^{u'} \quad \text{for all } i, \quad \text{(pressure limit)}, \tag{3.3}$$

$$\sum_j \hat{c}_{ij} = 1 \quad \text{for all } i, \quad \text{(feasibility test)}, \tag{3.4}$$

$$I_j^O + \sum_t R_{jt} - \sum_i \sum_t b_{it} \hat{c}_{ij} \geq I_j^L \quad \text{for all } j, \quad \text{(inventory limit)}, \tag{3.5}$$

$$I_j^O + \sum_t R_{jt} - \sum_i \sum_t b_{it} \hat{c}_{ij} \leq I_j^U \quad \text{for all } j, \quad \text{(inventory limit)}, \tag{3.6}$$

$$\breve{c}_{ij} \leq \bar{c}_{ij} \quad \text{for all } i, j, \tag{3.7}$$

$$c_{ij} \geq 0 \quad \text{for all } i, j, \tag{3.8}$$

$$b_{it} \geq 0 \quad \text{for all } i, t. \tag{3.9}$$

The LP model for the SSP is solved by means of LINGO 11 [66] and the solution is described in Tables 3.1 to 3.3. All LP models contained in this thesis is solved in this manner.

Blend	**Day 1** m^3	**Day 2** m^3	**Day 3** m^3
M3S	4.20	0.00	0.00
M5S	3.80	8.27	7.70

Table 3.1: *Optimal amounts of each blend to be produced as determined by solving the LP in (3.1)–(3.9). A total of 24 m^3 product is produced.*

3.2 LP approaches for the SSP

Several LP formulations to determine upper bounds on heuristic approaches based on intuition are applied to the simplified sample problem. The use of these LP formulations

Component	M3S %	M5S %	Day 1 m^3	Day 2 m^3	Day 3 m^3
BUT	4.81	4.23	Not inventoried		
GP1	14.21	19.13	1.60	1.04	0.60
GP4	4.13	21.41	2.17	1.29	0.86
PTF	61.35	39.72	2.91	1.91	1.16
TAM	15.50	15.50	2.07	1.71	1.43

Table 3.2: *Optimal percentages of each component that make up each blend as well as the resulting inventory amounts as determined by solving the LP in (3.1)–(3.9).*

	Day 1 kR	Day 2 kR	Day 3 kR	Total kR
Product revenue	43.23	45.05	41.70	129.98
Feedstock costs	37.06	37.63	34.83	109.52
Profit margin	6.16	7.42	6.87	**20.46**

Table 3.3: *A summary of optimal economic values as determined by solving the LP in (3.1)–(3.9).*

to investigate these intuitive approaches allow for a greater understanding of how the blend amounts and component percentages effect the various constraints in the model.

3.2.1 The minimum inventory approach

For the *minimum inventory approach* (MIA) a solution is found by drawing the daily inventory down to the minimum and by then creating as much of each blend as possible. The objective function (3.1) is replaced so that the difference between the amount of component j in inventory each day and the minimum allowable amount for component j is minimized, *i.e.*

$$\begin{aligned} &\text{minimise} && I_j^O + \sum_t R_{jt} - \sum_i \sum_t b_{it}\hat{c}_{ij} - I_j^{min} && \text{for all } j, && (3.10)\\ &\text{subject to} && (3.2) - (3.9). \end{aligned}$$

The results for this approach is shown in Tables 3.4 to 3.6.

A feasible solution obtaining a profit of R19 230 indicates that the MIA is an acceptable approach. Although the total amount of product produced has increased when compared to the exact solution, the increased production of the lower price blend and the decreased production of the higher priced blend leads to a lower total profit.

Blend	Day 1 m^3	Day 2 m^3	Day 3 m^3
M3S	8.90	1.02	0.02
M5S	3.70	4.46	5.70

Table 3.4: *The amounts of each blend to be produced as determined by solving the MIA for the SPP. A total of 38 m^3 product is produced.*

Component	M3S %	M5S %	Day 1 m^3	Day 2 m^3	Day 3 m^3
BUT	2.03	4.99	Not inventoried		
GP1	18.57	18.28	0.60	0.62	0.60
GP4	14.74	21.17	1.07	0.86	0.86
PTF	49.17	40.06	1.16	1.16	1.16
TAM	15.50	15.50	1.36	1.43	1.46

Table 3.5: *The percentages of each component that make up each blend as well as the resulting inventory amounts as determined by solving the MIA for the SSP.*

	Day 1 kR	Day 2 kR	Day 3 kR	Total kR
Product revenue	67.38	29.70	31.09	128.17
Feedstock costs	58.04	24.98	25.93	108.95
Profit margin	9.34	4.73	5.17	**19.23**

Table 3.6: *A summary of optimal economic values as obtained by the MIA for the SSP.*

3.2.2 The minimum closing inventory approach

For the *minimum closing inventory approach* (MCIA) a solution is found by drawing the closing inventory down to the minimum and by then creating as much of each blend as possible. Suppose T denotes the last day of the production horizon. The objective function (3.1) is replaced so that the amount of component j in inventory on day T is minimized, *i.e.* to

$$\begin{aligned} &\text{minimise} && \sum_j \left(I_j^O + \sum_{t=1}^{T} R_{jt} - \sum_i \sum_{t=1}^{T} b_{it}\hat{c}_{ij} \right) \\ &\text{subject to} && (3.2) - (3.9). \end{aligned} \tag{3.11}$$

The results for this approach is shown in Tables 3.7 to 3.9

A feasible final cost solution of R19 940 indicates that the MCIA is an acceptable approach. Althought the total amount of product produced has remained constant when compared to the exact solution, the increased production of the lower price blend and the decreased production of the higher priced blend leads to a lower total profit.

Blend	Day 1 m^3	Day 2 m^3	Day 3 m^3
M3S	0.00	2.26	5.42
M5S	9.70	0.00	6.60

Table 3.7: *The amounts of each blend to be produced as determined by solving the MCIA for the SSP. A total of 24 m^3 product is produced.*

Component	M3S %	M5S %	Day 1 m^3	Day 2 m^3	Day 3 m^3
BUT	5.56	3.76	Not inventoried		
GP1	6.73	23.71	0.63	1.50	0.60
GP4	9.52	22.52	0.98	1.65	0.86
PTF	62.71	34.51	3.67	4.54	1.16
TAM	15.50	15.50	1.81	2.38	1.43

Table 3.8: *The percentages of each component that make up each blend as well as the resulting inventory amounts as determined by solving the MCIA for the SSP.*

	Day 1 kR	Day 2 kR	Day 3 kR	Total kR
Product revenue	52.93	11.99	64.54	129.46
Feedstock costs	44.02	10.52	54.99	109.52
Profit margin	8.91	1.47	9.55	**19.94**

Table 3.9: *A summary of the economic values as obtained by the MCIA for the SSP.*

3.2.3 The average octane approach

For the *average octane approach* (AOA) a solution is found by forcing the total amount of octane in each blend to equal the average amount of octane present in all of the components (excluding butane). This approach gives further insight into the amounts of each blend that should be produced each day. Constraint (3.2) is altered for this approach, *i.e.*

$$\begin{aligned} &\text{maximise } (3.1) \\ &\text{subject to } \sum_j O_j \hat{c}_{ij} = \frac{93b_{1t} + 95b_{2t}}{2} \qquad \text{for all } i,t, \qquad (3.12) \\ &\qquad (3.3) - (3.9). \end{aligned}$$

The results for this approach are shown in Tables 3.10 to 3.12.

A feasible final cost solution of R15 150 indicates that the AOA is a poor solution approach. Although production is focussed on the blend that returns the highest revenue, constraint (3.12) limits the total production amount causing a relatively low final profit solution.

Blend	Day 1 m^3	Day 2 m^3	Day 3 m^3
M3S	0.00	0.00	0.00
M5S	5.18	5.18	5.18

Table 3.10: *The amounts of each blend to be produced as determined by solving the AOA for the SSP. A total of 15.5 m^3 product is produced.*

Component	M3S %	M5S %	Day 1 m^3	Day 2 m^3	Day 3 m^3
BUT	6.42	2.37	Not inventoried		
GP1	0.01	28.16	1.47	1.04	0.60
GP4	0.01	28.30	1.70	1.12	0.86
PTF	93.39	41.16	4.89	5.05	5.20
TAM	0.17	0.01	3.32	4.24	5.15

Table 3.11: *The percentages of each component that make up each blend as well as the resulting inventory amounts as determined by solving the AOA for the SSP.*

	Day 1 kR	Day 2 kR	Day 3 kR	Total kR
Product revenue	28.23	28.23	28.23	84.70
Feedstock costs	23.18	23.18	23.18	69.55
Profit margin	5.05	5.05	5.05	**15.15**

Table 3.12: *A summary of the economic values as obtained by the AOA for the SSP.*

3.2.4 The maximized blend approach

In the SSP, the M5S blend sells at almost the same price as the M3S blend but it makes less volume due to its tighter octane constraint. For the *maximized blend approach* (MBA) the solution is the outcome of maximizing the production of M3S as opposed to M5S. Suppose b_{1t} denotes the amount of M3S that is produced on day t. Objective function (3.1) is replaced to maximise the production of this blend. The objective is then to

$$\begin{aligned} \text{maximise} \quad & \sum_t C_1^b b_{1t} \\ \text{subject to} \quad & (3.2) - (3.9). \end{aligned} \tag{3.13}$$

The results for this approach are shown in Tables 3.13 to 3.15.

A feasible final cost solution of R17 140 indicates that the MBA is a fair, but not ideal approach. Although production is focussed on the blend that returns the highest revenue, the constraints on the recipe for this blend do not allow for the maximum use of the components in inventory. Greater economic gain may be achieved by the production of the second blend also as allowed by the reserve components in inventory.

Blend	Day 1 m^3	Day 2 m^3	Day 3 m^3
M3S	9.80	7.49	6.53
M5S	0.00	0.00	0.00

Table 3.13: *The amounts of each blend to be produced as determined by solving the MBA for the SSP. A total of 23.8 m^3 product is produced.*

Component	M3S %	M5S %	Day 1 m^3	Day 2 m^3	Day 3 m^3
BUT	3.74	3.36	Not inventoried		
GP1	18.40	17.43	1.13	0.78	0.60
GP4	18.49	39.06	1.36	0.86	0.86
PTF	43.87	24.65	2.74	1.74	1.16
TAM	15.50	15.50	1.81	1.56	1.46

Table 3.14: *The percentages of each component that make up each blend as well as the resulting inventory amounts as determined by solving the MBA for the SSP.*

	Day 1 kR	Day 2 kR	Day 3 kR	Total kR
Product revenue	51.76	39.71	34.60	126.08
Feedstock costs	44.73	34.31	29.90	108.93
Profit margin	7.04	5.40	4.71	**17.14**

Table 3.15: *A summary of the economic values as obtained by the MBA for the SSP.*

3.2.5 The differential inventory approach

For the *differential inventory approach* (DIA) a solution is obtained by taking the difference between opening and closing inventory as input amounts into each blend. Suppose T denotes the last day of the production horizon.

The heuristic may be formulated by adding constraint 3.14. The formulation is then to

$$\begin{aligned} &\text{minimise } (3.1) \\ &\text{subject to } (3.2) - (3.9), \\ &\sum_{d=1}^{T} \sum ib_{it}\hat{c}_{ij} = \sum_{t=1}^{T} R_{jt} - \sum_{t=1}^{T} \sum_{i} b_{it}\hat{c}_{ij} \text{ for all } j. \end{aligned} \tag{3.14}$$

The results for this approach are shown in Tables 3.16 to 3.18.

A feasible final cost solution of R14 660 indicates that the DIA yields rather poor result, but it provides valuable information on the effect that inventory manipulations have on the objective function. This information may be combined with the results for §3.2.2 to determine a good inventory strategy.

Blend	Day 1 m^3	Day 2 m^3	Day 3 m^3
M3S	0.00	0.00	10.80
M5S	0.30	7.56	1.30

Table 3.16: *The amounts of each blend to be produced as determined by solving the DIA for the SSP. A total of 20.0 m^3 product is produced.*

Component	M3S %	M5S %	Day 1 m^3	Day 2 m^3	Day 3 m^3
BUT	4.81	0.00	Not inventoried		
GP1	11.60	3.11	2.92	3.71	3.44
GP4	6.33	40.37	3.02	0.86	0.86
PTF	61.75	41.02	6.88	6.07	1.16
TAM	15.50	15.50	3.27	3.01	2.05

Table 3.17: *The percentages of each component that make up each blend as well as the resulting inventory amounts as determined by solving the DIA for the SSP.*

	Day 1 kR	Day 2 kR	Day 3 kR	Total kR
Product revenue	1.87	41.20	64.25	107.33
Feedstock costs	1.59	34.88	56.21	92.67
Profit margin	0.29	6.32	8.05	**14.66**

Table 3.18: *A summary of the economic values as determined by the DIA for the SSP.*

From the results for the various approaches applied to the SSP, it may be concluded that heuristics that alter solutions by focussing on the inventory specifications deliver the better solutions.

3.3 An exact solution approach for the HPP

Let C_i^p be the selling price per m^3 of product i and let C_j^c be as defined in §3.1. Also, let P_i be the amount of product i that is produced, let c_{ij} be the amount (in m^3) of component j required to form product i and let $\bar{c}_{jk}$ donate the amount of component j to go into the pooling mix k so that $\tilde{c}_{ik}$ is the amount of pooling mix k required to produce product i. Furthermore, let S_j be the sulfur content of component j, let $\tilde{S}_k$ be the sulfur content of the pooled component mix k and let $S_i^{\max}$ be the maximum allowable sulfur content of product i. Lastly, let $P_i^{\max}$ be the maximum amount of product i that may be produced. Similar to the SSP, let $\hat{c}_{ij}$ represent the fraction of product i that consists of component j.

With cost optimization as goal, a total of I products and a total of J components the HPP is stated as finding a product solution vector of size I and a $(J \times I)$ component

solution matrix such that $IJ + I$ nonnegativity constraints, $3I$ inequality constraints and I equality constraints, totalling $IJ + 5I$ constraints, will be satisfied.

The objective of the LP is to

$$\text{maximise} \quad \sum_i C_i^p P_i - \sum_j C_j^c \left(\sum_i c_{ij} + \sum_k \bar{c}_{jk} \right), \tag{3.15}$$

$$\text{subject to} \quad \sum_j \sum_k \bar{c}_{jk} = \sum_i \sum_k \tilde{c}_{ik} \qquad \text{(pooling balance)}, \tag{3.16}$$

$$\sum_j S_j c_{ij} + \sum_k \tilde{S}_k \tilde{c}_{ik} \leq S_i^{\max} \qquad \text{for all } i, \quad \text{(sulfur limit)}, \tag{3.17}$$

$$\sum_k \tilde{c}_{ik} + \sum_j c_{ij} \geq P_i \qquad \text{for all } i, \quad \text{(balance limit)}, \tag{3.18}$$

$$P_i \leq P_i^{\max} \qquad \text{for all } i, \quad \text{(product limit)}, \tag{3.19}$$

$$c_{ij} \geq 0 \qquad \text{for all } i, j, \tag{3.20}$$

$$P_i \geq 0 \qquad \text{for all } i. \tag{3.21}$$

Before the model can be solved for the problem described in §2.2, correct estimation for the values of $\tilde{S}_k$ must be done. Haverly [44] suggests a recursive method that achieves the solution as described in Tables 3.19 to 3.21. The optimal solution to the Haverly pooling problem delivers an objective function value of R400.00.

Product	**Amount**(m^3)
ProdX	0.00
ProdY	200.00

Table 3.19: *Optimal production amounts as determined in Haverly [44].*

Component	**Pool1** (%)
CompA	0
CompB	100

Table 3.20: *Optimal pool composition percentages as determined in Haverly [44].*

Component	**ProdX**(%)	**ProdY**(%)
Pool1	0	50
CompC	0	50

Table 3.21: *Optimal product composition percentages as determined in Haverly [44].*

3.4 Heuristic solution for the HPP

Haverly [44] suggests a recursive heuristic for the determination of the sulfur content of the pooled mix. The heuristic takes any estimation of $\tilde{S}_k$ as a parameter to the model

described in §3.3. After solving the model, it tests whether the assumed value is correct based on the amount of each component chosen for the pool. If it is not correct, the value is revised and the model is resolved and tested. This recursive heuristic continues until the estimate is close to that calculated from the mixture of the components chosen.

For example, assume Pool1 consist of 50% compA and 50% compB. The assumed $\tilde{S}_1$ will be $\frac{1}{2}(3+1) = 2$. The solution is described by Tables 3.22 to 3.23 and it has an objective function value of R300.00.

Product	**Amount**(m^3)
ProdX	100.00
ProdY	0.00

Table 3.22: *Optimal production amounts determined when* $\tilde{S}_1 = 2$.

Component	**Pool1** (%)
CompA	100
CompB	0

Table 3.23: *Optimal pool composition percentages determined when* $\tilde{S}_1 = 2$.

Component	**ProdX**(%)	**ProdY**(%)
Pool1	100	0
CompC	0	0

Table 3.24: *Optimal product composition percentages determined when* $\tilde{S}_1 = 2$.

This solution is now used to test the estimated value of $\tilde{S}_1$. It is found that $\tilde{S}_1 = (3(100) + 1(0))/100 = 3$ which is in fact not equal to the value of 2 as estimated. The estimation is off and it must be revised. Suppose the estimation is now set so that $\tilde{S}_1 = 3$. Tables 3.25 to 3.26 describe the solution obtained, It has an objective function value of R100.00.

If this solution is now used to test the estimated value of $\tilde{S}_1$, it is found that $\tilde{S}_1 = (3(50) + 1(0))/50 = 3$ which is equal to the estimated value. Therefore, the solution has converged and an optimal solution has been found. The pooling problem is non-linear by nature and there are be a number of local optima of which the optima obtained from the heuristic is only one.

Product	**Amount**(m^3)
ProdX	100.00
ProdY	0.00

Table 3.25: *Optimal production amounts determined when* $\tilde{S}_1 = 3$.

Component	**Pool1** (%)
CompA	100
CompB	0

Table 3.26: *Optimal pool composition percentages determined when* $\tilde{S}_1 = 3$.

3.5 An exact solution approach for the MMRP

Let C_i^p be the selling price per barrel of product i, let $C_k^{c'}$ be the cost price per barrel of crude k and let C_m^o be the operating cost per unit crude associated with process m. Let

Component	**ProdX**(%)	**ProdY**(%)
Pool1	50	0
CompC	50	0

Table 3.27: *Optimal product composition percentages determined when* $\tilde{S}_1 = 3$.

P_i be the number of barrels of product i that is produced and let K_k be the number of barrels of crude k that is bought as raw material. Also let K'_{km} be the number of barrels of crude k that passes through process m.

Let c'_{jkm} be the amount per barrel of component j from crude k obtained after the crude has passed through process m and let c_{ji} be the amount per barrel of component j required to form the final product i. Similar to the SSP and HPP, let $\hat{c}_{ij}$ represent the fraction of product i that consists of component j. Let A_{ji} be a test variable so that

$$A_{ji} = \begin{cases} 1, & \text{if component } j \text{ is obtained from process } i, \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, let d_j be the amount in barrels of domestic product required to form the final product. Let the upperbounds on the number of barrels of crude that is present in process m and the number of barrels of crude k that may be purchased be U'_m and U_k, respectively.

Let O_j and V_j be the octane rating and vapour pressure of component j, respectively and let $O_i^{\min}$ and $V_i^{\max}$ be the minimum allowable octane rating and maximum allowable vapour pressure of product i, respectively. Let ρ_{jk} be the density and S_{jk} the sulfur content of component j obtained from crude k, with $\rho_i^{\max}$ the maximum allowable density and $S_i^{\max}$ the maximum allowable sulfur content of product i.

The objective of the model is to

$$\text{maximise} \quad \sum_i C_i^p P_i - \sum_k C_k^{c'} K_k - \sum_m C_o^m \left(\sum_k K'_{km} \right), \tag{3.22}$$

$$\text{subject to} \quad K_k \leq U_k \quad \text{for all } k, \quad \text{(purchase limit)}, \tag{3.23}$$

$$\sum_m K'_{km} \leq K_k \quad \text{for all } k, \quad \text{(crude balance)}, \tag{3.24}$$

$$\sum_k K'_{km} \leq U'_m \quad \text{for all } m, \quad \text{(process limit)}, \tag{3.25}$$

$$\sum_j O_j c_{ji} \geq O_i^{\min} \quad \text{for all } i, \quad \text{(octane limit)}, \tag{3.26}$$

$$\sum_j V_j c_{ji} \leq V_i^{\max} \quad \text{for all } i, \quad \text{(pressure limit)}, \tag{3.27}$$

$$0 \qquad \sum_j \rho_j \hat{c}_{ji} \leq \rho_i^{\max} \quad \text{for all } i, \quad \text{(density limit)}, \tag{3.28}$$

$$\sum_j S_j c_{ji} \leq S_i^{\max} \quad \text{for all } i, \quad \text{(sulfur limit)}, \tag{3.29}$$

$$\sum_k K'_{km} \leq U'_m \quad \text{for all } m, \quad \text{(capacity)}, \tag{3.30}$$

$$\left[\sum_k \sum_m \left(c'_{jkm} \cdot K'_{km}\right) + d_j\right] \cdot A_{ji} \geq c_{ji} \quad \text{for all } i, j, \tag{3.31}$$

$$\sum_j c_{ji} = P_i \quad \text{for all } i, \tag{3.32}$$

$$c_{ij} \geq 0 \quad \text{for all } i, j. \tag{3.33}$$

Thus the MMRP has a total of I products, a total of J components, a total of K crudes and a total of M processes. The goal is to determine the optimal economic revenue by finding a product solution vector of size M and a $(J \times I)$ component solution matrix such that IJ nonnegativity constraints, $2K + 2M + 4I + IJ$ inequality constraints and M equality constraints, totalling $2K + 3M + 4I + 2IJ$ constraints, will be satisfied.

The LP solution is given in Tables 3.28 and 3.29. The optimal mini-refinery problem has an objective function value of \$4 135.50.

Product	**Barrels produced**
Fuel gas	1.00
Regular petrol	8.00
Premium petrol	7.00
Distillate	2.00
Fuel oil	2.00

Table 3.28: *An optimal number of barrels that should be produced according to the LP in (3.22) – (3.33). A total of 20 barrels of product is produced.*

3.6 LP approaches for the MMRP

As for the SSP, several LP formulations to investigate the heuristic approaches developed by intuition are applied to the MMRP.

Component	Fuel Gas %	Regular petrol %	Premium petrol %	Distillate %	Fuel Oil %
but	0.0	13.0	14.0	0.0	0.0
fuel-gas	100.0	0.0	0.0	0.0	0.0
sr-gas	0.0	46.0	46.0	0.0	0.0
sr-naphta	0.0	2.0	0.0	1.0	0.0
rf-gas	0.0	36.0	40.0	0.0	0.0
sr-dist	0.0	0.0	0.0	68.0	0.0
cc-gas	0.0	3.0	0.0	0.0	0.0
cc-gas-oil	0.0	0.0	0.0	7.0	7.0
sr-gas-oil	0.0	0.0	0.0	15.0	70.0
sr-res	0.0	0.0	0.0	0.0	0.0
hydro-res	0.0	0.0	0.0	0.0	23.0

Table 3.29: *Optimal percentages of each component that make up each product according to the LP in (3.22) – (3.33).*

3.6.1 The minimum closing inventory approach

For the *minimum closing inventory approach* (MCIA) a solution is obtained by drawing the closing inventory down to the minimum by creating as much of each blend as possible. The objective function (3.22) is replaced so that the total amount of each product is maximized. The objective is then to

$$\begin{aligned} \text{maximise} \quad & \sum_i P_i \\ \text{subject to} \quad & (3.23) - (3.33). \end{aligned} \tag{3.34}$$

The results for this approach are shown in Tables 3.30 and 3.31.

Product	Barrels produced
Fuel gas	1.00
Regular petrol	7.00
Premium petrol	6.00
Distillate	2.00
Fuel oil	4.00

Table 3.30: *The number of barrels of product to be produced when using the MCIA to solve the MMRP. A total of 20 barrels of product is produced.*

A feasible solution obtaining a profit of R3 926.78 indicates that the MCIA is an acceptable approach. The decrease in the objective function value from the value obtained by the exact solution is due to the lower production of the products with the highest selling price.

Component	Fuel Gas %	Regular petrol %	Premium petrol %	Distillate %	Fuel Oil %
but	0.0	15.0	17.0	0.0	0.0
fuel-gas	100.0	0.0	0.0	0.0	0.0
sr-gas	0.0	43.0	44.0	0.0	0.0
sr-naphta	0.0	0.0	0.0	0.0	0.0
rf-gas	0.0	33.0	39.0	0.0	0.0
sr-dist	0.0	0.0	0.0	33.0	0.0
cc-gas	0.0	9.0	0.0	0.0	0.0
cc-gas-oil	0.0	0.0	0.0	36.0	18.0
sr-gas-oil	0.0	0.0	0.0	32.0	16.0
sr-res	0.0	0.0	0.0	0.0	0.0
hydro-res	0.0	0.0	0.0	0.0	66.0

Table 3.31: *The percentages of each component that make up each product as determined by the MCIA for the MMRP.*

3.6.2 The maximum input approach

For the *maximum input approach* (MIPA) a solution is determined by maximizing the amount of crude that is purchased and by sending the maximum allowable amount of crude through the production system. To determine the best possible outcome for this heuristic, the objective function (3.22) should be replaced so that the total amount of crude through the system is maximized, *i.e.* to

$$\begin{aligned} \text{maximise} \quad & \sum_k K_k \\ \text{subject to} \quad & (3.23) - (3.33). \end{aligned} \tag{3.35}$$

The results for this approach is shown in Tables 3.32 and 3.33.

Product	Barrels produced
Fuel gas	2.00
Regular petrol	6.00
Premium petrol	5.00
Distillate	2.00
Fuel oil	5.00

Table 3.32: *The number of barrels of each product to be produced when using the MIPA to solve the MMRP. A total of 20 barrels of product is produced.*

A feasible solution obtaining a profit of R3 612.81 indicates that the MIPA is an acceptable approach. Again the decrease in the objective function value from the value obtained by the exact solution is due to the lower production of the products with the highest selling price.

Component	Fuel Gas %	Regular petrol %	Premium petrol %	Distillate %	Fuel Oil %
but	0.0	16.0	20.0	0.0	0.0
fuel-gas	100.0	0.0	0.0	0.0	0.0
sr-gas	0.0	39.0	27.0	0.0	0.0
sr-naphta	0.0	0.0	0.0	15.0	0.0
rf-gas	0.0	33.0	40.0	0.0	0.0
sr-dist	0.0	0.0	0.0	0.0	0.0
cc-gas	0.0	11.0	13.0	0.0	0.0
cc-gas-oil	0.0	0.0	0.0	40.0	16.0
sr-gas-oil	0.0	0.0	0.0	44.0	26.0
sr-res	0.0	0.0	0.0	0.0	0.0
hydro-res	0.0	0.0	0.0	0.0	58.0

Table 3.33: *The percentages of each component that make up each product as determined by the MIPA for the MMRP.*

3.6.3 The average octane approach

For the *average octane approach* (AOA) a solution is obtained by forcing the total octane rating in each product to equal the average octane rating of all of the components (excluding butane). This approach gives insight into the sensitivity of recipes with regards to the quality constraints. To test the effect of alterations to the products' octane ratings, constraint (3.26) is altered to constraint (3.36). The LP then has the objective to

$$\begin{aligned} &\text{maximise } (3.22) \\ &\text{subject to } (3.23)\text{–}(3.18) \\ &\qquad \sum_j O_j = \frac{\sum_i O_i^* P_i}{I} \qquad (3.36) \\ &\qquad (3.27)\text{–}(3.33). \end{aligned}$$

The results for this approach are shown in Tables 3.34 to 3.35.

Product	Barrels produced
Fuel gas	1.00
Regular petrol	4.00
Premium petrol	0.00
Distillate	7.00
Fuel oil	4.00

Table 3.34: *The number of barrels of each product to be produced when using the AOA to solve the MMRP. A total of 16 barrels of product is produced.*

Component	Fuel Gas %	Regular petrol %	Premium petrol %	Distillate %	Fuel Oil %
but	0.0	22.0	0.0	0.0	0.0
fuel-gas	100.0	0.0	0.0	0.0	0.0
sr-gas	0.0	7.0	0.0	0.0	0.0
sr-naphta	0.0	71.0	0.0	47.0	0.0
rf-gas	0.0	0.0	0.0	0.0	0.0
sr-dist	0.0	0.0	0.0	19.0	0.0
cc-gas	0.0	0.0	0.0	0.0	0.0
cc-gas-oil	0.0	0.0	0.0	10.0	15.0
sr-gas-oil	0.0	0.0	0.0	24.0	33.0
sr-res	0.0	0.0	0.0	0.0	0.0
hydro-res	0.0	0.0	0.0	0.0	52.0

Table 3.35: *The percentages of each component that make up each product as determined by the AOA for the MMRP.*

A feasible solution obtaining a profit of \$2 662.70 indicates that the AOA yields rather poor results. The reason for the poor final cost solution is attributed to the nature of this heuristic that specifically limits the production of the products that have the highest selling price.

3.7 Conclusion

Nilson [73] finds that in order to carry out a search by means of a computer program, the search strategy must be defined and the folowing elements of the problem is required:

- A defined problem, often requiring intelligent or complex behaviour to solve.
- A model of the problem and the representation of possible solutions.
- A goal state that defines the ideal solution, where heuristic methods often use an evaluation function to rank the non-goal state solutions.
- Transformation operators that are capable of changing an existing solution into an alternative solution.
- A strategy for searching the space of possible solutions using the representation and transformation operators.

All three the problems at hand meet these requirements. Hence various metaheuristic solution techniques may be explored in an attempt to solve them. The development of effective and efficient metaheuristics for the problems and the measuring of their performance against the known optimal as determined by an LP can give an indication of the quality of metaheuristic solution approaches.

Chapter 4

Data structure

A standard data structure for each problem is presented to facilitate fair comparisons of the performance of various metaheuristics, The generation of solution sets, populations and neighbourhoods is bound to this set structure.

Dealing with constraints is one of the most critical points in creating the data structure. One approach is to design a procedure such that only feasible solutions, that is, only solutions satisfying all the constraints will be accepted. This is obtained either by introducing a routine which solves the set of equations and inequalities that define the constraints and inputting them to the metaheuristic system. Another approach is to create random individuals and to then either reject the infeasible ones or to penalise them using penalty functions. For the first approach, it may be difficult or even impossible to find the solution set for the constraints, especially when dealing with nonlinear equations and inequalities. Hence for all three problems, the penalty function approach is implemented.

4.1 Penalty Functions

Coello [24] shows that the most popular approach to constrained optimisation is by means of penalty functions. Three types of penalty functions exist: *barrier methods* in which no infeasible solution is considered, *partial penalty functions* in which a penalty is applied near the feasibility boundary, and *global penalty functions* which are applied throughout the infeasible region [88]. In combinatorial optimisation, the Lagrangian relaxation method is a variation on the same theme: Temporarily relax the problem's most difficult constraints, using a modified objective function to avoid straying too far from the feasible region [82].

For both problems at hand, the objective function values makes use of a penalty system as suggested by Kuri-Morales and Gutiérrez-García [62] so that infeasible solutions are penalised with an extremely poor fitness value. This ensures that infeasible solutions would be considered to be prematurely dead and that their continuation into later generations

is eliminated. If $\boldsymbol{g}$ is an infeasible solution then the penalty function is defined as

$$p(\boldsymbol{g}) = \begin{cases} \left[\kappa - \sum_{i=1}^{s} \frac{\kappa}{p_i}\right], & s \neq p, \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

where κ is a large negative constant [$\mathrm{O}(10^6)$], p is the number of constraints and s is the number of these which have been satisfied. The only restriction on κ is that it should be large enough to insure that any non-feasible individual is graded much more poorly than any feasible one. Here the penalty function supplies information as to how many constraints have been satisfied but is not otherwise affected by the strategy. The penalty is not subtracted from the objective function $f(\boldsymbol{g})$ but, rather, it replaces $f(\boldsymbol{g})$, *i.e.* $f(\boldsymbol{g}) = p(\boldsymbol{g})$ when any of the constraints is not met.

4.2 Data structure for the SSP

For the SSP, the desired output is both the blend recipe as well as the amount of each blend to be produced for each day of the production horizon. Using the notation of §2.1, the solution may be constructed as an $[(I+J)\times D]$ solution matrix as shown in Figure 4.1 so that the $[I \times D]$ submatrix contains the daily production amounts and the $[J \times D]$ submatrix contains the blend recipes. Should the length of the production horizon be greater than the number of blends to be made, place holders are placed into the recipe matrix so that the set dimensions are maintained.

$$\left[\begin{array}{c} \left[\begin{array}{cccccc} b_{11} & b_{12} & b_{13} & b_{14} & \dots & b_{1d} \\ b_{21} & b_{22} & b_{23} & b_{24} & \dots & b_{2d} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ b_{i1} & b_{i2} & b_{i3} & b_{i4} & \dots & b_{id} \end{array}\right] \\ \\ \left[\begin{array}{cccccc} \hat{c}_{11} & \hat{c}_{12} & \dots & \hat{c}_{1i} & 0 & \dots & 0 \\ \hat{c}_{21} & \hat{c}_{22} & \dots & \hat{c}_{2i} & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ \hat{c}_{j1} & \hat{c}_{j2} & \dots & \hat{c}_{ji} & 0 & \dots & 0 \end{array}\right] \end{array}\right]$$

Figure 4.1: *The data structure for the SSP.*

Due to the various restrictions described in §2.1, the values of the production matrix will depend on the values of the recipe matrix and the recipe matrix is the only one free to be filled with values generated at random. Therefore the fraction of component j used to form each blend is a real number chosen by means of a random number generator. The percentage of each blend which consists of component j has well defined upper and lower

bounds of 0 and 100 respectively (it is impossible for a blend to consist of less than 0% or more than 100% of any component).

The ideal is to maximise the amounts of each blend to be produced each day as this maximises revenue. These maximum amounts are determined by taking into consideration the amount of each component in the opening inventory and the amount of component available through run down values in conjunction with the product fractions. These determined maximum values are used to fill the product amount matrix. Using the relationship between the amount of blend produced and the amount of components in inventory to maximise the choice of production amounts significantly increases the probability that feasible solutions will be generated.

Metaheuristics require an evaluation, objective or fitness function to determine the quality of a solution. The objective function value is typically a single scalar that allows selection methods to distinguish between various levels of individual quality. Multiobjective methods often use a vector to represent fitness.

Rosca and Ballard [84] show that ideally, the data structure and objective function should have the property of strong causality where small changes in the solution cause small changes in the objective function value. They have found, however, that weak causality, the opposite of strong causality, is more common in certain metaheuristics such as genetic programming.

Figure 4.2 contains an example of a feasible solution generated as described. It has an objective function value of R7 929.

$$\left[\begin{array}{l} \left[\begin{array}{ccc} 0.859 & 0.049 & 1.051 \\ 3.762 & 0.599 & 2.296 \end{array}\right] \\ \\ \left[\begin{array}{ccc} 1.308 & 0.107 & 0 \\ 20.080 & 34.379 & 0 \\ 55.023 & 26.678 & 0 \\ 19.823 & 24.560 & 0 \\ 3.766 & 14.276 & 0 \end{array}\right] \end{array}\right]$$

Figure 4.2: *An example of a data structure representing a feasible solution for the SSP.*

4.3 Data structure for the HPP

For the HPP, the desired output is the product and pooling recipes as well as the resultant amount of each product that can be made using these recipes. Using the notation of §3.3, the solution may be constructed as a $[(1+J+J'+K)\times I]$ solution matrix (if $K < I$ and J' is the amount of components that will not form part of a blend) as shown in Figure 4.3 so that the $[1 \times J]$ vector contains the production amounts, the $[J \times K]$ submatrix constains

the pool recipes and the $[(K+J')\times I]$ submatrix contains the product recipes. If $K < I$, the pool recipes sub-matrix is buffered with zero-valued place holders so that the dimensions of the solution matrix is maintained. If $K > I$, the product amounts and recipes submatrices are buffered in the same manner.

$$\begin{bmatrix} \begin{bmatrix} P_1 & P_2 & P_3 & \dots & \dots & P_i \end{bmatrix} \\ \begin{bmatrix} \bar{c}_{11} & \bar{c}_{12} & \dots & \bar{c}_{1k} & 0 & \dots & 0 \\ \bar{c}_{21} & \bar{c}_{22} & \dots & \bar{c}_{2k} & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ \bar{c}_{j1} & \bar{c}_{j2} & \dots & \bar{c}_{jk} & 0 & \dots & 0 \end{bmatrix} \\ \begin{bmatrix} \tilde{c}_{11} & \tilde{c}_{12} & \dots & \dots & \tilde{c}_{1i} \\ \tilde{c}_{21} & \tilde{c}_{22} & \dots & \dots & \tilde{c}_{2i} \\ \vdots & \vdots & & & \vdots \\ \tilde{c}_{k1} & \tilde{c}_{k2} & \dots & \dots & \tilde{c}_{ki} \\ \hat{c}_{11} & \hat{c}_{12} & \dots & \dots & \hat{c}_{1i} \\ \hat{c}_{21} & \hat{c}_{22} & \dots & \dots & \hat{c}_{2i} \\ \vdots & \vdots & & & \vdots \\ \hat{c}_{j'1} & \hat{c}_{j'2} & \dots & \dots & \hat{c}_{j'i} \end{bmatrix} \end{bmatrix}$$

Figure 4.3: *The data structure for the HHP ($k < i$).*

The values of the product amounts vector will depend on the specifications of the recipe matrices, and in turn the values of the components in the pooling recipes matrix will depend on the values of the product recipes matrix. These matrices may be filled by using a random generation of the fraction amounts of the pool and product composition. The fraction of component j used to form each product or each pool is a real number chosen with the use of a random number generator. The fraction of each product which consists of component j or pool k has well defined upper and lower bounds of 0 and 100 respectively (once again it is impossible for a product or pool to consist of less than 0% or more than 100% of any component). The ideal is to maximise the amounts of each blend to be produced each day as this maximises revenue. These maximum amounts are determined by taking into consideration the upper limit on the amount of product that may be produced each day and the amount of each component needed to form the product. These determined values fill the product amount matrix. Using the relationship between the amount of product made and the chosen product recipes to maximise the choice of production amounts significantly increases the probability that feasible solutions will be generated.

4.4 Data structure for the MMRP

For the MMRP, the desired output is the product recipes, the amount of raw material to be purchased as well as the resulting amount of each product that can be produced given the amount of raw material purchased. Using the notation of §3.5, the solution may be constructed as a $[(J+1+1) \times i]$ solution matrix as shown in Figure 4.4. The $[J \times I]$ submatrix contains the product recipes, the first $[1 \times I]$ vector contains the production amounts and the second $[1 \times I]$ vector contains the purchase amount for each raw product. Should the desired amount of products be greater than the amount of raw materials, zero-valued place holders are stored in the purchase amount vector so that the dimensions are maintained.

$$\begin{bmatrix} \begin{bmatrix} \hat{c}_{11} & \hat{c}_{12} & \hat{c}_{13} & \hat{c}_{14} & \dots & \hat{c}_{1i} \\ \hat{c}_{21} & \hat{c}_{22} & \hat{c}_{23} & \hat{c}_{24} & \dots & \hat{c}_{2i} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \hat{c}_{j1} & \hat{c}_{j2} & \hat{c}_{j3} & \hat{c}_{j4} & \dots & \hat{c}_{ji} \end{bmatrix} \\ \begin{bmatrix} P_1 & P_2 & P_3 & P_4 & \dots & P_i \end{bmatrix} \\ \begin{bmatrix} K_1 & \dots & K_k & 0 & \dots & 0 \end{bmatrix} \end{bmatrix}$$

Figure 4.4: *The data structure for the MMRP.*

Similar to the data structure for the SSP, the values of the production matrix will depend on the values of the recipe and purchase matrices so that the recipe matrix is the only one free to populate by means of random numbers. Therefore the fraction of component j used to form each product is a real number chosen with the use of a random number generator. Once again the fraction of each blend which consists of component j has well defined upper and lower bounds of 0 and 100 respectively.

Again the ideal is to maximise the amounts of each blend to be produced each day as this maximises revenue. These maximum amounts are determined by taking into consideration the amount of raw material purchased and the amount of component that can be obtained through the various processes subject to the constraints described in §2.3. The values determined in this way are used in the product matrix.

If we use the relationship between the amount of product made and the amount of raw material purchased to maximise the choice of production amounts significantly increases the chances that feasible solutions will be generated. An example of a feasible solution generated as described above is shown in Figure 4.5. It has an objective function value of R151 800.

$$
\left[
\begin{array}{l}
\left[
\begin{array}{rrrrr}
0.00 & 31.09 & 7.73 & 0.00 & 0.00 \\
1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 16.96 & 8.56 & 0.00 & 0.00 \\
0.00 & 4.66 & 10.74 & 0.56 & 0.00 \\
0.00 & 46.99 & 57.72 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 16.20 & 0.00 \\
0.00 & 0.29 & 15.24 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 46.33 & 50.69 \\
0.00 & 0.00 & 0.00 & 36.91 & 29.89 \\
0.00 & 0.00 & 0.00 & 0.00 & 4.65 \\
0.00 & 0.00 & 0.00 & 0.00 & 14.77
\end{array}
\right] \\
\\
\left[
\begin{array}{rrrrr}
0.00 & 7.00 & 13.00 & 1.00 & 1.00
\end{array}
\right] \\
\\
\left[
\begin{array}{rrrrr}
28.00 & 0.00 & 0 & 0 & 0
\end{array}
\right]
\end{array}
\right]
$$

Figure 4.5: *An example of a data structure representing a feasible solution for the MMRP.*

Chapter 5

Random search techniques

Random search is a simple, but very popular search technique which centers a symmetric probability density function such as the normal distribution about the current best location. The standard N(0, 1) distribution is a popular choice, although the uniform distribution U[-1, 1] is also common [55].

5.1 Overview

Random search techniques (RSTs) were first proposed by Anderson [4] and later by Rastrigin [81] and Karnopp [55]. The development of RSTs for linear programming models begins in 1992 when Kalai [53] explores the use of random pivot rules. Motwani and Raghavan [70] solve LP-models on random subsets of constraints, recursively. The search technique of Dunagan and Vempala [30] makes use of randomly generated solutions which are tested for feasibility. If the solution is not feasible, the search moves in a deterministic (with relation to a random vector already selected) direction to achieve feasibility and ultimately, optimality. For unconstrained NLP-models, Storn and Price [94] develop a heuristic which selects random subsets of solution population vectors upon which addition, subtraction and component swapping is performed to obtain improvements in the objective function value. Solis and Wetts [91] present two general convergence proofs for random search algorithms.

A large class of optimisation problems can be handled effectively by means of RSTs. These methods become attractive in some specific circumstances, for instance when the function characteristics (such as gradients) are difficult to compute, when there is only limited computer memory available or when it is highly desirable to find the global optimum of a function having many local optima. However, its use may also lead to certain problems such as continuous loop executions when no local optimum for the function exists or exhaustive, but inconclusive, examination when the optimum occurs at a point at which the function is singularly discontinuous.

Unfortunately, RSTs do not have well defined stopping conditions. Solis and Wets [91] examine several attempts to define stopping criteria for RSTs but conclude that "...the search for a good stopping criterion seems doomed to fail." In practice, the method is halted after a fixed number of iterations or when the step size becomes smaller than the given threshold.

5.2 Blind random search

Blind random search (BRS) is the simplest random search method in that the choice of candidate solution does not take into account any characteristics of previously considered solutions. That is, this blind search approach does not adapt the current sampling strategy to information that has been gathered in the search process. The approach may be implemented in non-recursive form simply by laying down a number of points in the search space and taking the value of the solution yielding the best objective function value as an estimate of the optimum. The approach can also be implemented in recursive form. Algorithm 1 gives a pseudocode listing of the BRS as presented by Spall [93]. Spall [93] shows that this algorithm converges almost surely to a near optimum solution under very general conditions and when the solution configuration is low dimensional, but that the method is generally a very slow algorithm for even moderately dimensioned solution configurations.

5.3 Local random search

Methods of local search received attention in both theoretical computer science and numerical optimisation. Johnson, Papadimitriou and Yannakakis [50] observe that one of the few general approaches to difficult combinatorial optimisation problems that has met with empirical success is *local* (or *neighbourhood*) search. For example, local search methods have proven very successful for the celebrated travelling salesperson problem [49].

The field of operations research is primarily interested in *local random search* (LRS) methods used for minimizing continuous functions on compact spaces. These methods make use of derivative information like gradients and Hessians and can be distinguished by the highest order derivatives that they use. Algorithms using derivate information of order greater than zero are somewhat more powerful than those which only use function evaluations (order zero derivatives). However, derivative information requires additional calculations, and these algorithms do not always generate good solutions fast enough to compensate for the additional expense [55].

Solis and Wets [91] propose several random local search methods for performing local search on smooth functions without derivative information. Their so-called "Algorithm 1" uses normally distributed steps to generate new solutions in the search space. A new solution is generated by adding zero mean normal variates to every dimension of

the current solution. A variate is a variable for which values occur corresponding to a frequency distribution. A different value for each dimension is chosen at random from a normal distribution so that the new solution resembles the current one, but it does not match it exactly. The algorithm then examines the solution generated by taking a step in the opposite direction from the new direction. If neither solution is better than the current solution, another new solution is generated. This algorithm depends upon parameters that automatically reduce and increase the variance of the normal deviates in response to the rate at which better solutions are found. If new solutions are better more often, the variance is increased to allow the algorithm to take larger steps. If poorer solutions are frequently generated, the variance is decreased to focus the search near the current solution.

Parks [75] suggests that a vector of zero mean normal variates, $\underline{d}$, be added to the current solution and that it should be generated according to

$$\underline{d}_i = \underline{d}_i + \boldsymbol{D}\underline{\varpi} \tag{5.1}$$

where $\underline{\varpi}$ is a vector of uniform random numbers in the range $(-1, 1)$ and $\boldsymbol{D}$ is a diagonal matrix which defines the maximum change allowed in each variable. After a successful trial (*i.e.* after an accepted change in the solution) $\boldsymbol{D}$ is updated, such that

$$\boldsymbol{D}_{i+1} = (1 - \alpha)\boldsymbol{D}_i + \alpha\omega\boldsymbol{\Upsilon}, \tag{5.2}$$

where $\boldsymbol{\Upsilon}$ is a diagonal matrix. The elements of $\boldsymbol{\Upsilon}$ consist of the magnitudes of the successful changes made to each control variable, *i.e.*

$$\Upsilon_{ii} = \|D_{ii}\varpi_i\| \tag{5.3}$$

and the damping constant α controls the rate at which information from $\boldsymbol{\Upsilon}$ is folded into $\boldsymbol{D}_{i+1}$ with weighting ω. This tunes the maximum step size associated with each control variable towards a value giving acceptable changes. Parks concludes that suitable values of α and ω are 0.1 and 2.1, respectively.

Algorithm 2 gives a pseudocode listing of a local random search algorithm as presented by Matyas [68].

5.4 Computational results

Throughout the thesis, computational results are obtained from code written in Python 2.5 [99] executed on an Intel Core2 Duo processor running at 3.00Ghz with 2GB of RAM. Unless stated otherwise, all random numbers are generated according to Python's standard random number generator using a uniform distribution.

Standard deviation results are given throughout to indicate the size of the variance between individual results which make up the average results. A low standard deviation

Algorithm 1: `Blind random search`

Input: An initial solution $\boldsymbol{x}_0$ chosen randomly or deterministically from the search space $\mathcal{S}$. Random initial solutions are obtained according to a probability distribution function.

Output: An approximation of a global optimum solution.

1: Set $k = 0$ and calculate the objective function value $f(\boldsymbol{x}_k)$.
2: Generate a new candidate solution $\hat{\boldsymbol{x}}_{k+1}$ from the search space according to the chosen probability distribution.
3: **if** $f(\hat{\boldsymbol{x}}_{k+1}) > f(\boldsymbol{x}_k)$ **then**
4: $\quad \boldsymbol{x}_{k+1} = \hat{\boldsymbol{x}}_{k+1}$
5: **else**
6: $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k$
7: **end if**
8: $k = k + 1$
9: Go to step 2 and repeat the algorithm until the stopping criterion is reached.

Algorithm 2: `Local random search`

Input: An initial solution $\boldsymbol{x}_0$ chosen randomly or deterministically from the search space $\mathcal{S}$. Random initial solutions are obtained by using the normal distribution. Also take as input a probability distribution for generating a vector $\underline{d}$ that has mean zero and a variance for each component of the solution consistent with the magnitudes of the each of the corresponding elements.

Output: An approximation of a global optimum solution.

1: Set $k = 0$ and calculate the objective function value $f(\boldsymbol{x}_k)$.
2: Generate an independent random vector $\underline{d}_k$ and add it to the current solution $\boldsymbol{x}_k$ to obtain a candidate solution $\hat{\boldsymbol{x}}_{k+1}$.
3: If $\hat{\boldsymbol{x}}_{k+1}$ is not in the search space, generate a new $\underline{d}_k$ and repeat step 2 or, alternatively, choose the nearest valid solution to $\hat{\boldsymbol{x}}_{k+1}$.
4: **if** $f(\hat{\boldsymbol{x}}_{k+1}) > f(\boldsymbol{x}_k)$ **then**
5: $\quad \boldsymbol{x}_{k+1} = \hat{\boldsymbol{x}}_{k+1}$
6: **else**
7: $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k$
8: **end if**
9: $k = k + 1$
10: Go to step 2 and repeat the algorithm until the stopping criterion is reached.

indicates that the data points tend to be very close to the same value (the mean), while a high standard deviation indicates that the data are spread out over a large range of values. Hence the smaller the standard deviation, the more predictably the algorithm will perform.

Throughout, the number of execution runs when average results are obtained, is deter-

mined by means of the half-width confidence interval outlined in Pegden *et al.* [77] with a pilot number of 10 runs and desired confidence level of 90%. The number of 10 is a rule of thumb, but the t-distribution on which the half-width confidence interval method is based is useful for observations less than 30. The use of 10 runs compromises between execution time and the requirements of statistical procedures. If the observations are not normally distributed, the results for the confidence interval hold for 10 replications or more [77]. For the RST approaches, a confidence level of 90% is obtained by calculating the average objective function values for the BRS after 9, 8 335 and 4 046 algorithm runs for the SSP, HPP and MMRP, respectively. The same confidence level is obtained by calculating the average objective function values for the LRS after 102, 1 025 and 10 065 algorithm runs for the SSP, HPP and MMRP, respectively.

For each of the three problems, the initial solution is generated as described in §4.2, §4.3 and §4.4, respectively.

5.4.1 The SSP

The results for the BRS algorithm and the LRS algorithm as applied to the SSP are shown in Figure 5.1. Because the LRS algorithm is computationally less expensive (having only to manipulate a currently feasible solution) than the BRS algorithm (continuously having to find new feasible solutions from the search space), it is possible to have approximately 500 iterations of the LRS execute in the same time as only 100 iterations of the BRS. The comparison of the respective performances of the two approaches are shown in Figure 5.1. Despite the much quicker execution time of the LRS, the BRS is preferred because of the better average result it achieves. Table 5.1 contains a summary of the results obtained after application of the RS approaches for the SSP.

5.4.2 The HPP

The results for the BRS algorithm and the LRS algorithm as applied to the HPP of Chapter 2 are shown in Figure 5.2. Figure 5.2 contains the average objective function values obtained for each application of each RST. The results do not imply that one technique is better than the other by a large margin, yet the BRS technique is preferred due to the better average result it achieves as well as its shorter execution time. Table 5.1 contains a summary of the results obtained after application of the RS approaches for the HPP.

5.4.3 The MMRP

The results for the BRS algorithm and the LRS algorithm as applied to the MMRP of Chapter 2 are shown in Figure 5.3. The average objective function values obtained for each algorithm for 10 execution runs are presented. Even though the LRS technique is

preferred for the MMRP when one looks at the execution time, on average it produces a weaker solution than the BRS technique. This is because the specific combination of constraints for the problem makes it easier to find a feasible solution by generating random recipe matrices than to find a feasible solution by adjusting existing recipes. Table 5.1 contains a summary of the results obtained after application of the RS approaches for the MMRP.

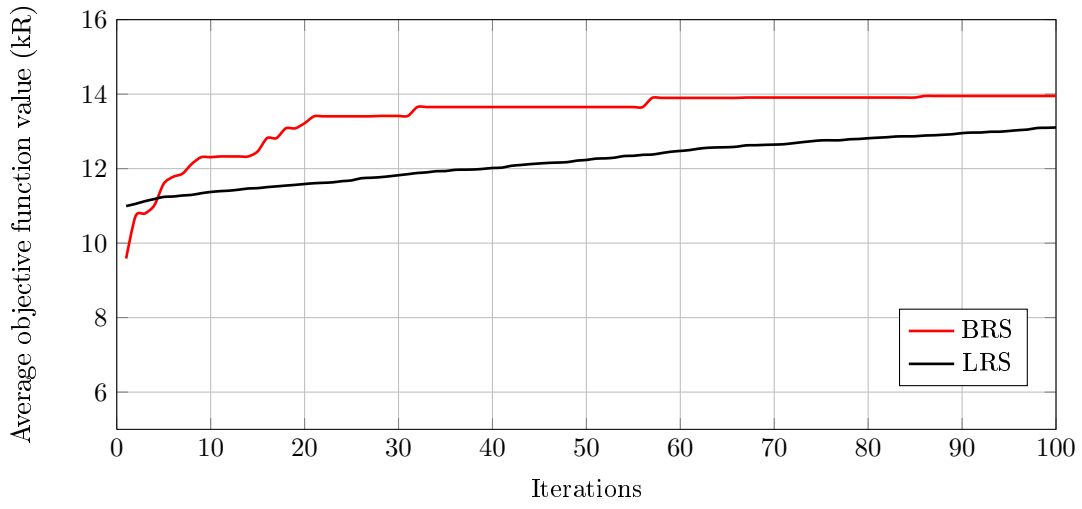

Figure 5.1: *The average objective function values for BRS and LRS, respectively for the SSP.*

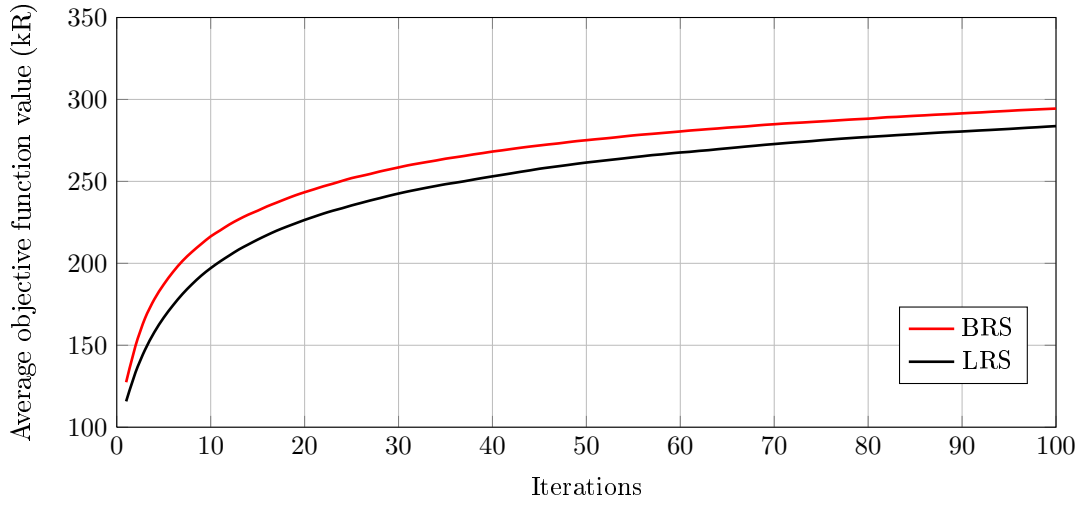

Figure 5.2: *The average objective function values obtained by means of BRS and LRS for the HPP.*

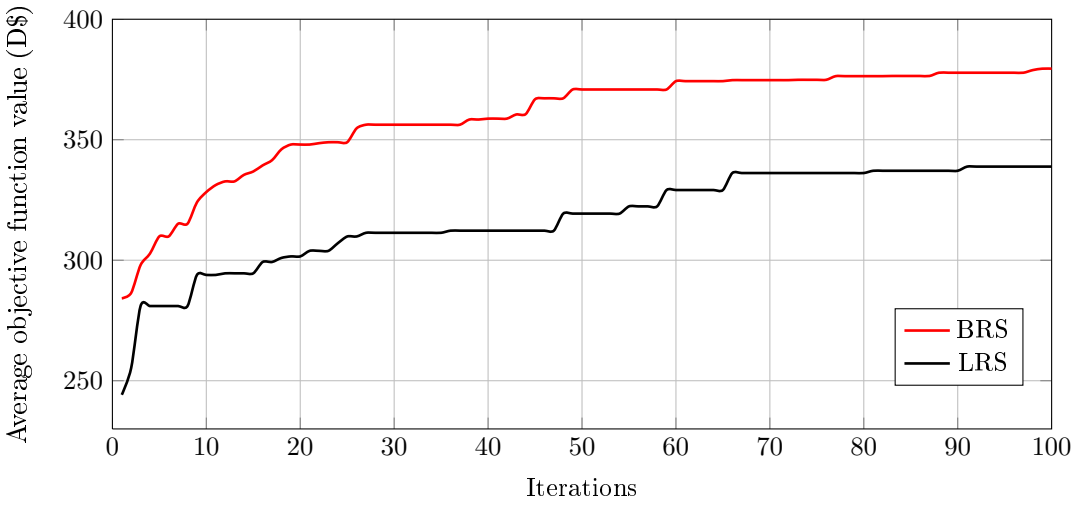

Figure 5.3: *The objective function values obtained by means of BRS and LRS for the MMRP.*

	SSP (kR)		**HPP** (kR)		**MMRP** (D$)	
Result	**BRS**	**LRS**	**BRS**	**LRS**	**BRS**	**LRS**
Best fitness	15.01	17.26	364.33	310.78	413.50	383.65
Average fitness	13.95	13.11	294.98	283.57	379.48	338.85
Standard deviation	0.89	2.91	23.29	25.78	15.43	24.27
Average execution time (sec)	75.27	0.59	0.02	0.11	159.23	2.94

Table 5.1: *Results summary for the random search techniques for the SSP, HPP and MMRP.*

Chapter 6

Genetic algorithm approaches

A genetic algorithm (GA) is a search technique used in computing to find solutions to a wide range of optimization problems. GAs are categorized as global search heuristics. They are a particular class of evolutionary algorithms (also known as evolutionary computation) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection and recombination (also called crossover).

Toklu [95] formulates an aggregate-blending problem as a multi-objective optimization problem and solves it by means of GAs. Toklu shows that all existing formulations of an aggregate-blending problem can be covered and solved by using this technique. It is shown to be quite versatile in tackling multiple objectives including cost minimisation and approaching at best a given target curve.

6.1 Overview

Genetic programming became a popular search technique in the early 1990's due to the work by Koza [59]. Angeline [5] traced the historical foundations of genetic programming back to Friedberg [32] and Friedberg *et al.* [33], where iterative random changes made to computer programs induced better programs. Learning machines were proposed earlier by Turing [96], where an automated process similar to evolution in combination with human interaction was thought to be a possible way for programs to acquire intelligent behaviour. The form of genetic programming used today is most closely related to work done by Cramer [26], where a tree representation of programs was used in conjunction with subtree crossover to evolve a multiplication function. Other background and foundational research of genetic programming and evolutionary algorithms may be found in Banzhaf *et al.* [12], Fogel [34], Bäck *et al.* [8] and Koza [60]. More modern approaches to genetic programming and evolutionary algorithms are described in Langdon and Poli [63], Bäck *et al.* [9] and Koza *et al.* [61].

Genetic algorithms are inspired by Darwin's theory of evolution [28]. The algorithm is started with a set of solutions (represented in analogy by *chromosomes*) called a *population*. A chromosome is actually an object representing the characteristics of a candidate solution to the problem at hand. At early times, these chromosomes were formed by binary numbers only, but later they are generalized so as to be formed by any other entities such as arrays, lists, trees, mathematical operators, or any kind of number depending on the type of problem.

Solutions from one population are taken and used to form a new population. This is motivated by a hope that the new population will be better than the old one. Solutions which are selected to form new solutions (*offspring*) are selected according to their fitness – the more suitable they are the more chances they have to reproduce. This is repeated until some condition (for example number of generations or improvement of the best solution) is satisfied.

A pseudocode listing for the standard GA formulation is provided in Algorithm 3.

Algorithm 3: `Standard Genetic Algorithm`

Input: A combinatorial optimization problem specification including a domain set for each decision variable. An initial configuration $\boldsymbol{x}_1$, a population size N, a probability crossover p_c, and a probability mutation p_m. A genetic code formulation with a function mapping code substrings to a decision variable values. An objective function $f(\cdot)$ to determine individual fitness.

Output: A converged population of solutions containing an approximation of a globally optimal solution to the combinatorial optimization problem.

1: Randomly generate an initial population of N solutions.
2: Calculate the fitness of each individual solution by means of the objective function.
3: Generate a new population using the crossover and mutation operators, applied with probability p_c and p_m respectively. Individuals with higher fitness must have a higher probability of reproducing.
4: Calculate the fitness of the new solutions.
5: Repeat steps 3 to 5 until a termination condition is reached.

In this thesis, a GA is applied to the blending problem with appropriate alterations made to the classical application.

6.2 Genome structure

The solutions are treated as genomes consisting of various chromosomes because the solution structure for the three problems consist of multiple submatrices. In classical genetics, the *genome* of an organism refers to a full set of chromosomes or genes in an organism. Each chromosome is formed by genes. For example, gene (i, d) in the first chromosome

of the SSP solution structure represents the amount of blend i that is manufactured on day d and gene (j, i) on the second chromosome represents the proportion or fraction of component j used to form the blend i. This analogy is illustrated in Figure 6.1 for a solution structure for the SSP. The solution structure for the HPP and MMRP is handled similarly.

$$\text{genome}\left\{\begin{array}{l}\left[\begin{array}{ccccc}\text{gene}_{11} & \text{gene}_{12} & \text{gene}_{13} & \dots & \text{gene}_{1d}\\ \text{gene}_{21} & \text{gene}_{22} & \text{gene}_{23} & \dots & \text{gene}_{2d}\\ \vdots & \vdots & \vdots & & \vdots\\ \text{gene}_{i1} & \text{gene}_{i2} & \text{gene}_{i3} & \dots & \text{gene}_{id}\end{array}\right]\right\}\text{chromosome}\\ \left[\begin{array}{ccccccc}\text{gene}_{11} & \text{gene}_{12} & \dots & \text{gene}_{1i} & 0 & \dots & 0\\ \text{gene}_{21} & \text{gene}_{22} & \dots & \text{gene}_{2i} & 0 & \dots & 0\\ \vdots & \vdots & & \vdots & \vdots & & \vdots\\ \text{gene}_{j1} & \text{gene}_{j2} & \dots & \text{gene}_{ji} & 0 & \dots & 0\end{array}\right]\left\}\begin{array}{l}\\ \\ \\ \\ \text{chromosome}\end{array}\right.\end{array}\right.$$

Figure 6.1: *The genome analogy for the solution structure for the SSP.*

6.3 Fitness determination

To compare and evaluate the fitness values of chromosomes, a proper measure has to be defined. For the blending problem, the objective is to maximise the economic profit by maximising the product revenue while minimising the costs.

Two common fitness assignment methods in genetic algorithms is proportional fitness assignment and rank-base fitness assignment. During *proportional fitness assignment* the fitness assigned to each individual depends on the actual objective value. In *rank-based fitness assignment*, however, the population is sorted according to the objective values. The fitness assigned to each individual depends only on its position in the individual's rank and not on the actual objective value.

Rank-based fitness assignment overcomes the scaling problems of the proportional fitness assignment. It overcomes stagnation in the case where the selective pressure is too small or in the case of premature convergence — it is where selection has caused the search to narrow down too quickly. *Selective pressure* is the probability of the best individual being selected compared to the average probability of selection of all individuals [10]. The reproductive range is limited, so that no individuals generate an excessive number of offspring. Ranking introduces a uniform scaling across the population and provides a simple and effective way of controlling selective pressure. Rank-based fitness assignment behaves in a more robust manner than proportional fitness assignment and, thus, is the method of choice [7].

For linear rank-based fitness assignment, let N be the number of individuals in the population, let ϱ be the position of an individual in this population (the least fit individual

has $\varrho = 1$ and the fittest individual has $\varrho = N$) and let s be the selective pressure. Linear ranking allows values of selective pressure in [1.0, 2.0]. The fitness value for an individual is calculated by means of

$$f(\varrho) = 2 - s + 2(s-1)\frac{\varrho - 1}{N - 1}. \tag{6.1}$$

For the use of higher selective pressures than allowable in the linear ranking method, a new method for ranking using a non-linear distribution is introduced in Polheim [78].

6.4 Genome selection

Before the genetic operators of the algorithm can be applied for the calculation of the new generation, so called "parent" genomes must be selected. This is done by means of *fitness proportionate selection* (also known as *roulette-wheel selection*). The roulette-wheel selection algorithm provides a zero bias but does not guarantee minimum spread. *Bias* is the absolute difference between an individual's normalized fitness and its expected probability of reproduction, while *spread* is the range of possible values for the number of offspring of an individual [10]. The pseudocode listing for this procedure is given in Algorithm 4.

Algorithm 4: `Fitness Proportionate Selection`

Input: A population of N solutions and their fitness values calculated by means of a objective function.

Output: The single solution $\boldsymbol{x}'$ which has the highest probability to reproduce.

1: Normalize the fitness value of each individual in the population so that the sum of all the fitness values equals 1.
2: Sort the population by descending fitness values.
3: Compute the accumulated normalized fitness values. The accumulated fitness value of an individual is the sum of its own fitness value plus the fitness values of all the previous individuals. The accumulated fitness of the last individual should of course be 1.
4: Generate a random number, r, between 0 and 1. The selected individual $\boldsymbol{x}'$, is the first one whose accumulated normalized value is greater than r.

The use of an alternative selection method, namely *tournament selection*, is recommended by Goldberg and Deb [41]. The pseudocode listing for this method is given in Algorithm 5.

Algorithm 5: `Tournament Selection`

Input: A population of N solutions and their fitness values calculated by means of a objective function.

Output: The single solution $\boldsymbol{x}'$ which has the highest probability to reproduce.

1: Generate a random number, r, between 2 and the number of individuals in the population.
2: With probability p_t, choose the best individual from the tournament of r individuals chosen from the population at random.
3: Set $i = i + 1$. Choose the i-th best individual with probability $p_t(1 - p_t)^2$.

6.5 Recombination operator

The modified recombination operator for the sample problem is a combination of discrete recombination and single-point crossover. *Discrete recombination* performs an exchange of variable values between the individuals. For each position the parent who contributes its variable to the offspring is chosen randomly with an equal probability [72]. In *single-point crossover* one crossover position is selected uniformly at random from all the possible points in the genome's make up, and the variables are exchanged between the individuals about this point, so that two new offspring are produced.

As described in Chapter 4, the choice of daily blend production amounts are dependent on the choice of component percentages which are to make up each blend. For this reason the recombination operator is applied only to the second chromosome of the genome after which the new blend amounts for the first chromosome is determined.

A crossover point is chosen at random with the number of blends to be produced set as the upper limit. The resultant child genomes then contain half of the component recipes from the first parent genome and half from the second parent genome and the blend amounts for each child genome is determined. The fitness values of both parent genomes and both child genomes are considered and the genome with the highest fitness is returned. Thus this strategy returns only the strongest individual, regardless of the generation to which it belongs.

6.6 Mutation operator

The mutation operator modifies the second chromosome, *i.e.* the daily blend production amounts only. It recreates the chromosome so that the total of the day's blend production is produced by the blend earning the highest revenue only. This method is inspired by the results obtained in §3.2.4 and does not lose the feasibility of the solution, as it does not require a blend with daily production amounts higher than those set by die inventory constraints.

6.7 Proportions of genetic operators

Four operators are traditionally used for genetic algorithms: recombination, mutation, reproduction and elitism. The first two operators are modified as described in Sections 6.5 and 6.6. The reproduction is applied in such a way that a certain percentage of individuals are chosen to be passed to the next generation without being subject to the applications of the recombination operator. For example, for an instance where the number of individuals is 40 and the proportions are given such that reproduction proportion is 0.30 and mutation probability is 0.01, the next generation will be calculated as follows: 12 best individuals will be chosen as they are, 28 individuals will be obtained through an application of the recombination operator. Then, the mutation operator will be applied to all these offspring so that each individual has a 0.01 probability to form the next generation.

As with all current machine learning problems it is worth fine tuning probability parameters to find reasonable settings for the problem class under consideration. A recombination rate that is too high may lead to premature convergence of the genetic algorithm. A mutation rate that is too high may lead to loss of good solutions unless there is elitist selection. There are theoretical but not yet practical upper and lower bounds for these parameters that can help guide selection. Whitely [105] suggests a recombination rate typically between 0.6 and 1.0.

As for mutation rate, the probability of mutating a variable is inversely proportional to the number of variables it contains (dimensions). The more dimensions one individual has, the smaller the mutation probability should be. Mühlbein [72] reports that a mutation rate of $\frac{1}{n}$ (where n is the number of variables of an individual) produced good results for a wide variety of test instances. That implies that only one variable per individual is mutated when a mutation is performed.

A quicker convergence and a better quality solution is achieved through the use of elitism. Once the population has been sorted according to their individual fitnesses, a certain percentage of the top achieving genomes are passed on to the next generation without alteration. Though these champion genomes are available to be chosen as parents for the recombination operator, the genome itself is only replaced when there are enough new offspring with better fitnesses.

The production and selection of new individuals is carried out in a generational or steady-state algorithm. In a *generational algorithm*, a new population is created from parents in the current population. In a *steady-state algorithm*, one offspring is created and placed into the existing population, either randomly or based on fitness. The GA implementation for the SSP is a generational algorithm.

6.8 Island models

Population diversity affects many aspects of the evolutionary process. The number of unique fitness values in the current population effects selection pressure. Low amounts of genetic differences prematurely imitates a solution that has reached convergence and can lead to lower selection pressure. Without adequate diversity the population may be unable to produce variations to improve solution quality. However, on the other extreme, too much diversity can prevent convergence. Many methods have been proposed and used in evolutionary computation to control diversity, prevent convergence and to distribute individuals over different areas of the search space. The island model is an example of a distributed population model where subpopulations are isolated during selection, breeding and evaluation [25]. Islands focus the evolutionary process within subpopulations before migrating individuals to other islands.

The most common form of island model uses fitness-based probabilistic selection for migration selection and replacement. Breeding and evaluation are typically carried out in isolation on each island. Pettey *et al.* [76] designs a distributed model where for every generation migration sends the best individuals in each population to each neighbour replacing the worst individuals. Whitley *et al.* [106] concludes that for problems where an increase in the population size does not yield better results, island models may yield better results by searching over a larger part of the solution space.

Borovska and Lazarova [18] present a comparative study of five migration policies for parallel genetic algorithms solving the traveling salesman problem. The investigated migration policies utilize one way and two way periodic chromosomes migration and global periodic chromosomes migration. For each migration policy best or random chromosome migration is applied. They recommend a subpopulation size equal to the population size divided by the number of subpopulations and that these subpopulations be filled by random assignment of individuals from the population. They set a migration frequency (the number of migrations per algorithm run) of 10% of the number of generations and determine the number of migrants to be 4% of the population size. Their results show that the two-way circular migration of the best fitness chromosomes gives the best results with respect to solution quality.

6.9 Computational Results

Different genetic algorithm implementations with regard to selection method, fitness assignment and operator parameters are applied to the three problems. The computational setup is the same as described in §5.4.

6.9.1 The SSP

The results summary for the GA applied to the SSP is shown in Tables 6.1 to 6.3. GA1 to GA4 is implemented without the use of elitism, while GA5 to GA8 is implemented with an elitism of 10%. For a population size of 100 individuals, this means that the 10 top performing individuals of the previous generation will be passed on to the next generation without any alteration. For this problem, the results in Figure 6.2 shows that regardless of whether or not elitism is present, the problem is best solved when using tournament selection and ranked assignment. This is in accordance with what was expected from literature.

The same best result is obtained by GA4 to GA8 during the 10 algorithms runs and this may be assumed as the genetic algorithm approach's upper bound for the SSP. This comparison is shown in Figure 6.3. The average execution times of GA1 to GA8 do not differ significantly, but GA8 is found to have a notably quicker execution time than GA7 without delivering an average result that is much poorer than that of GA8. This comparison is shown in 6.4.

The results in Figure 6.5 show that an increase in the population size leads to an improvement in performance for GA1 and GA2 only. The results remain fairly unchanged for GA3 and GA4 and these algorithms are implemented as island models (§6.8) with parameters set at the values given in Table 6.4 to determine whether this approach will yield better solutions. The results of the island model implementation of GA3 and GA4 are shown in Table 6.6. Although a better average fitness solution is obtained by die island GA3 model, the improvement (or the decline as is the case with GA4) is not large enough to justify the much slower convergence.

Parameters	**GA1 – GA4**	**GA5 – GA8**
Population size	100	100
Number of generations	100	100
Elitism proportion	0	0.1
Recombination probability	0.6	0.6
Mutation probability	0.1	0.1

Table 6.1: *The parameters used in GA1 to GA8 for the SSP.*

	Fitness assignment		**Selection method**		**Number of**
	Proportional	**Ranked**	**Roulette wheel**	**Tournament**	**algorithm runs**
GA1 & GA5	x		x		7
GA2 & GA6		x	x		9
GA3 & GA7		x		x	8
GA4 & GA8	x			x	7

Table 6.2: *Combination of fitness assignement and selection methods use in GA1 to GA8 for the SSP as well as the number of algorithm runs required to obtain a confidence interval of 90%.*

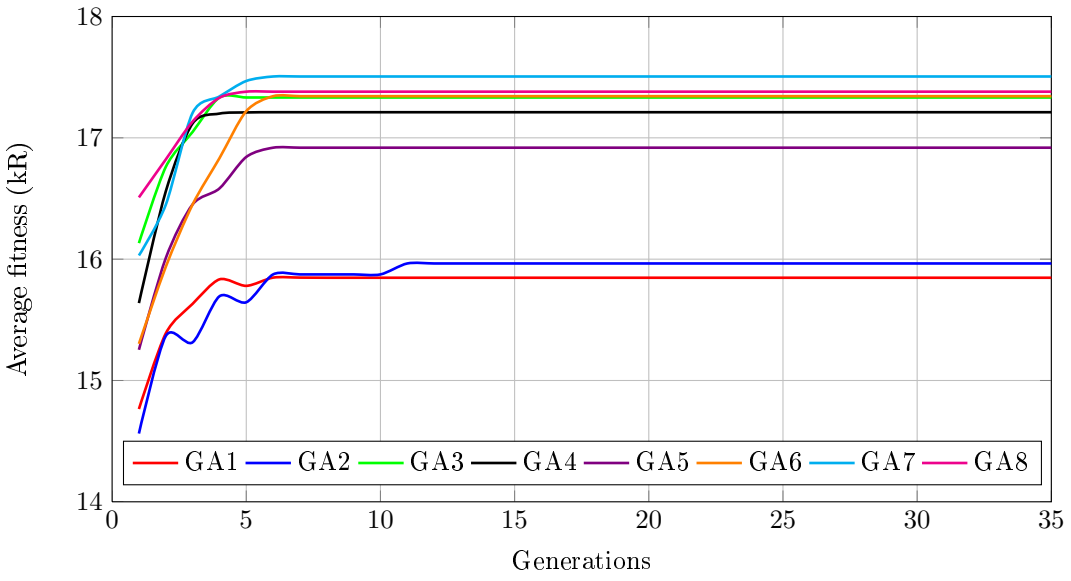

Figure 6.2: *Average fitness results of GA1 to GA8 obtained for the SSP.*

Results	GA1	GA2	GA3	GA4	GA5	GA6	GA7	GA8
Best fitness (kR)	17.07	17.72	18.29	18.52	18.52	18.52	18.52	18.52
Avg fitness (kR)	15.85	15.97	17.33	17.21	16.91	17.34	17.51	17.38
Std deviation (kR)	0.89	0.94	0.59	0.72	60.38	0.71	0.52	0.56
Avg execution time (sec)	60.14	64.56	64.74	62.44	60.14	68.05	69.75	61.59

Table 6.3: *Results obtained after execution of GA1 to GA8, respectively for the SSP.*

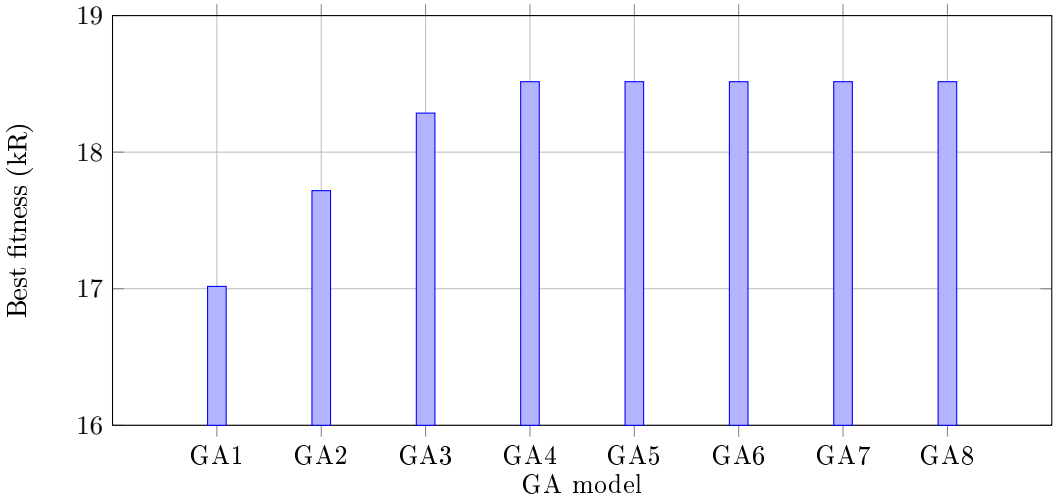

Figure 6.3: *The best fitness comparison of GA1 to GA8.*

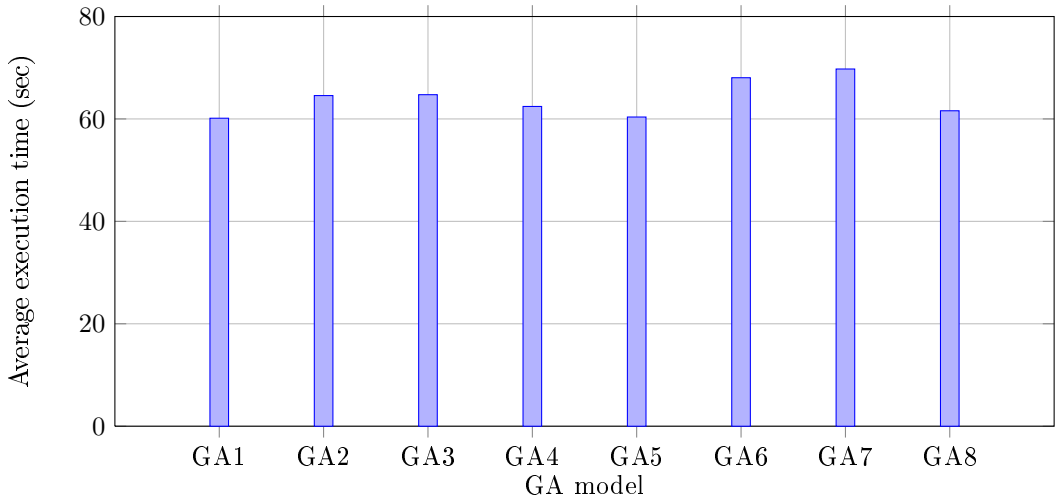

Figure 6.4: *The average execution time comparison per algorithm run for GA1 to GA8.*

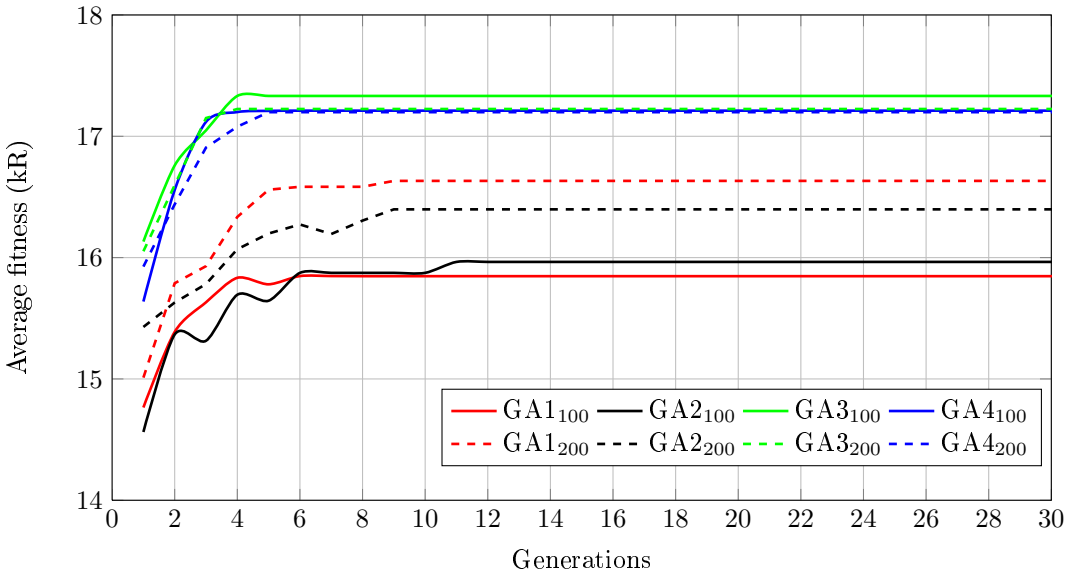

Figure 6.5: *GA1 to GA4 with a population size of 100 versus GA1 to GA4 with a population size of 200 for the SSP.*

6.9.2 The HPP

The results summary for the GA applied to the HPP are shown in Tables 6.5 to 6.7. GA9 to GA12 is implemented without the use of elitism, while GA13 to GA16 is implemented with elitism set at 10%.The results in Figure 6.7 show that regardless of whether or not elitism is present, the problem is best solved when using tournament selection and proportional fitness assignment.

GA12 and GA16 achieve approximately the same best result which may also be assumed

Island Model Parameters	
Population size	100
Number of generations	100
Number of subpopulations	2
Subpopulation size	50
Migration frequency	10
Number of migrants	10

Table 6.4: *The parameters as set for the island model implementation of GA3 and GA4. The best chromosome migration policy is implemented.*

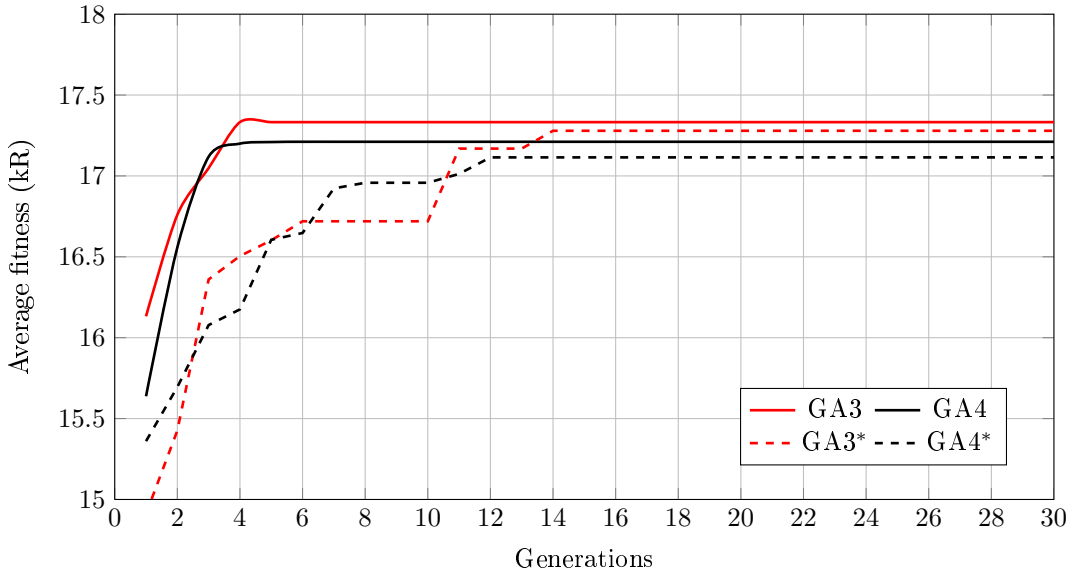

Figure 6.6: *GA3 and GA4 versus GA3* and GA3* which are implemented as an island models.*

to be the genetic algorithm approach's upper bound for the HPP. The best solution comparison is shown in Figure 6.8. The implementation of Roulette Wheel selection causes GA9, GA10, GA13 and GA14 to have the longest average execution time while GA12 and GA16 adds short execution time results to their already favourable best and average solution quality results. The average execution time result is shown in 6.9.

The results in Figure 6.10 shows that an increase in the population size leads an improvement in performance for GA9 and GA11 only. The results remain largely unchanged for GA10 and GA11 and these algorithms are implemented as island models with parameters set as described in Table 6.8 to investigate further improvement. The results of the island model implementation of GA10 and GA11 are shown in Figure 6.11. For both algorithms, the implementation of them as island models produce average solutions of a much lower quality than the results obtained through regular genetic model implementation.

Parameters	GA9 – GA12	GA13 – GA16
Population size	100	100
Number of generations	100	100
Elitism proportion	0	0.1
Recombination probability	0.6	0.6
Mutation probability	0.1	0.1

Table 6.5: *The parameters used in GA9 to GA16 for the HHP.*

	Fitness assignment		Selection method		Number of
	Proportional	Ranked	Roulette wheel	Tournament	algorithm runs
GA9 & GA13	x		x		4325
GA10 & GA14		x	x		1195
GA11 & GA15		x		x	325
GA12 & GA16	x			x	905

Table 6.6: *Combination of fitness assignment and selection methods use in GA9 to GA16 for the HPP as well as the number of algorithm runs required to obtain a confidence interval of 90%.*

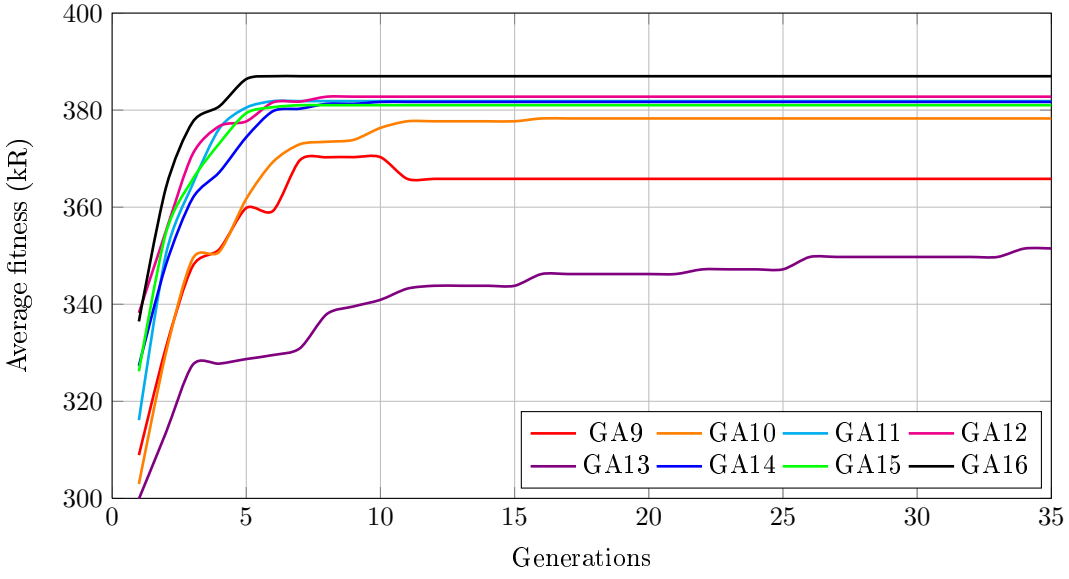

Figure 6.7: *Average fitness results of GA9 to GA16 for the HPP.*

6.9.3 The MMRP

The results summary for the GA applied to the MMRP are shown in Tables 6.9 to 6.11. GA17 to GA20 is implemented without the use of elitism, while GA21 to GA24 is implemented with elitism set at 10%. For the MMRP, the results in Figure 6.12 shows that regardless of whether or not elitism is present, the problem is best solved when using tournament selection and proportional fitness assignment.

Results	GA9	GA10	GA11	GA12	GA13	GA14	GA15	GA16
Best fitness (kR)	386.69	390.63	396.60	398.62	390.64	391.62	390.63	398.54
Avg fitness (kR)	365.84	381.69	381.01	386.98	343.48	356.28	373.85	386.77
Std deviation (kR)	16.77	8.80	14.38	7.66	30.71	16.26	7.48	6.80
Avg execution time (sec)	0.89	0.94	0.33	0.21	0.81	0.91	0.32	0.30

Table 6.7: *Results obtained after execution of GA9 to GA16, respectively for the HPP.*

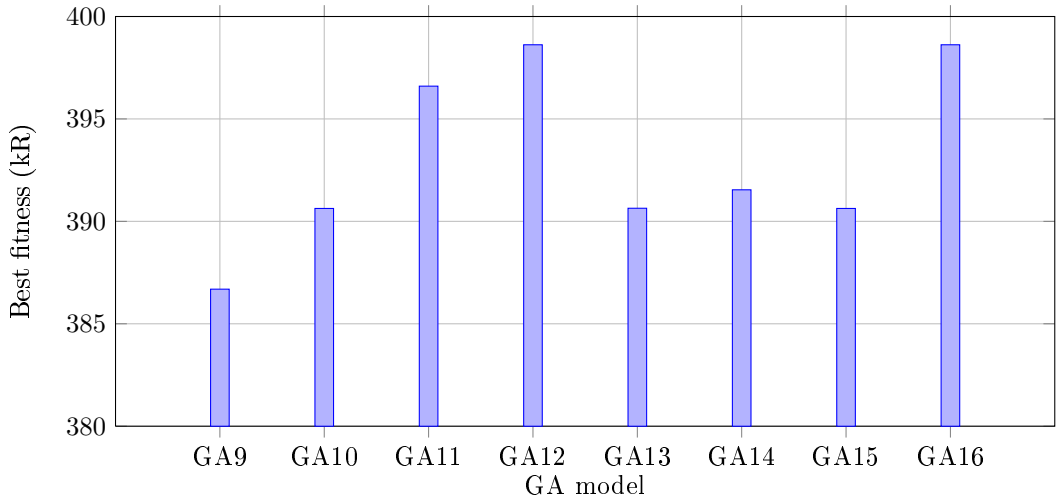

Figure 6.8: *The best fitness comparison of GA9 to GA16.*

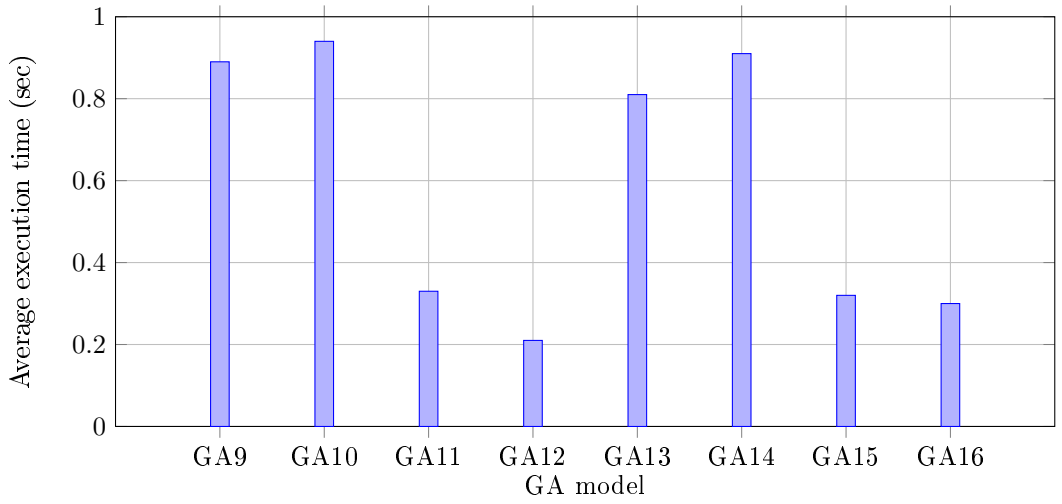

Figure 6.9: *The average execution time comparison per algorithm run for GA9 to GA16.*

GA24 achieves a best solution that is closest to the known optimal for the MMRP. When elitism is used, the use of tournament selection has the greatest positive effect on the performance of the GA approaches. The comparison of the best results obtained by the

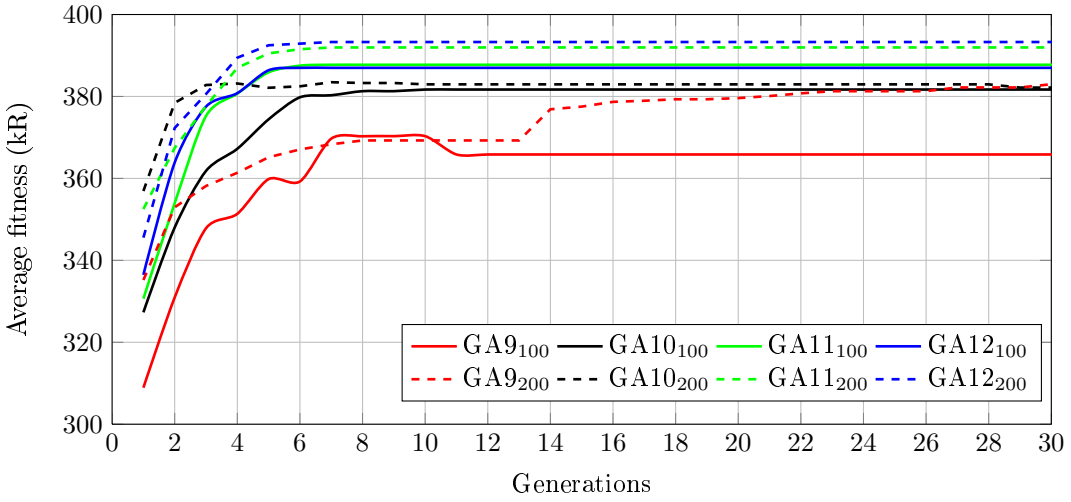

Figure 6.10: *GA9 to GA12 with a population size of 100 versus GA1 to GA4 with a population size of 200 for the SSP.*

Island Model Parameters	
Population size	100
Number of generations	100
Number of subpopulations	2
Subpopulation size	50
Migration frequency	10
Number of migrants	10

Table 6.8: *The parameters as set for the island model implementation of GA10 and GA11 for the HHP. The best chromosome migration policy is implemented.*

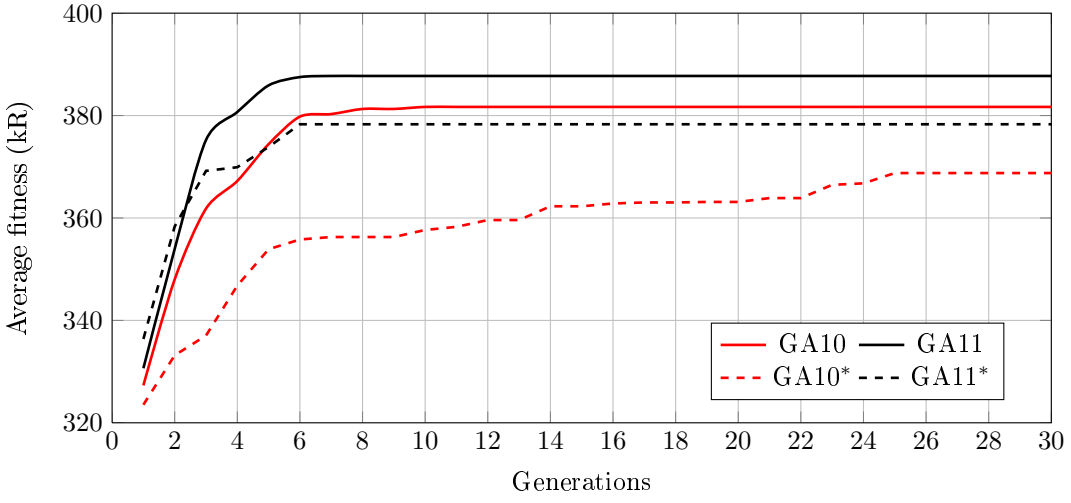

Figure 6.11: *GA10 and GA11 versus GA10* and GA11* which are implemented as an island models.*

various approaches are shown in Figure 6.12. The best performing approach, (*i.e.* GA24) has a fair average execution time associated with it, while GA22 has a good average execution time associated with it without delivering an average result that is much poorer than that of GA24. The average execution time comparison is shown in Figure 6.13.

Parameters	**GA17 – GA20**	**GA21 – GA24**
Population size	100	100
Number of generations	100	100
Elitism proportion	0	0.1
Recombination probability	0.6	0.6
Mutation probability	0.1	0.1

Table 6.9: *The parameters used in GA17 to GA24 for the MMRP.*

	Fitness assignment		**Selection method**		**Number of**
	Proportional	**Ranked**	**Roulette wheel**	**Tournament**	**algorithm runs**
GA17 & GA21	x		x		4845
GA18 & GA22		x	x		2660
GA19 & GA23		x		x	766
GA20 & GA24	x			x	1015

Table 6.10: *Combination of fitness assignement and selection methods use in GA17 to GA24 for the MMRP as well as the number of algorithm runs required to obtain a confidence interval of 90%.*

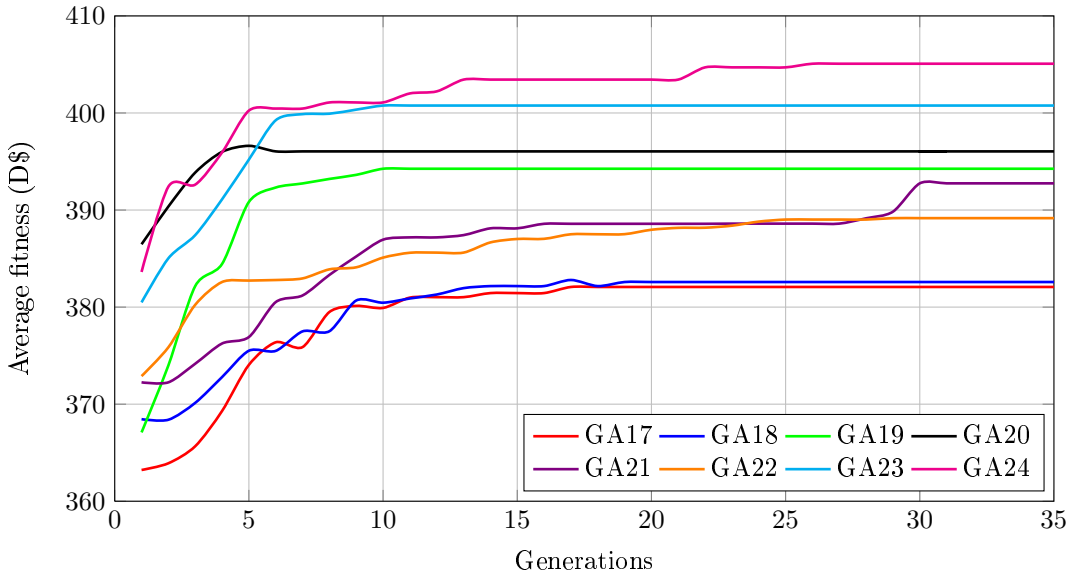

Figure 6.12: *Average fitness results of GA17 to GA24 for the MMRP.*

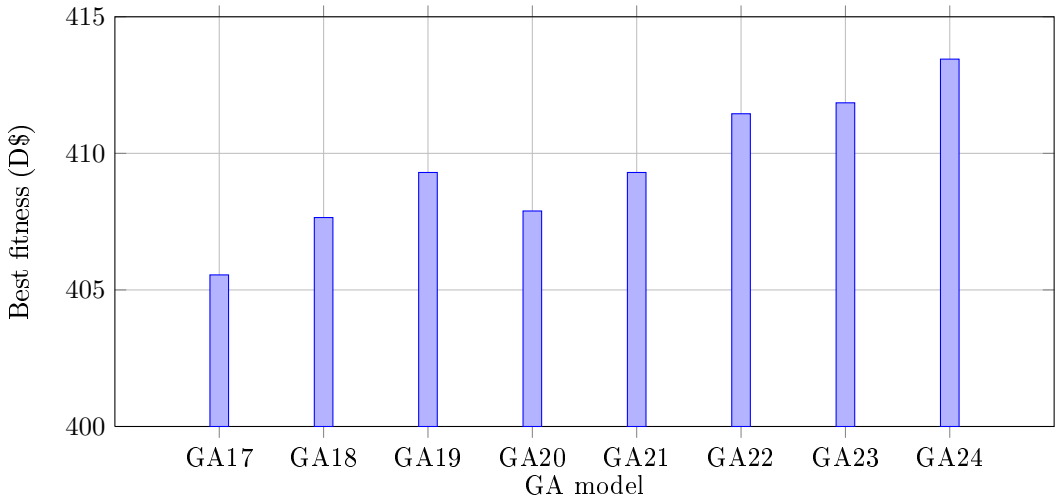

Figure 6.13: *The best fitness comparison of GA17 to GA24.*

Results	GA17	GA18	GA19	GA20	GA21	GA22	GA23	GA24
Best fitness (D$)	405.55	407.65	409.30	407.89	409.30	411.45	411.85	413.45
Avg fitness(D$)	382.08	382.59	394.26	396.043	392.75	389.17	400.77	403.91
Std deviation (D$)	20.46	17.84	13.79	12.34	10.01	17.39	11.93	9.17
Avg execution time (sec)	176.03	157.30	190.09	186.53	176.36	167.70	185.27	171.73

Table 6.11: *Results obtained after execution of GA17 to GA24, respectively for the MMRP.*

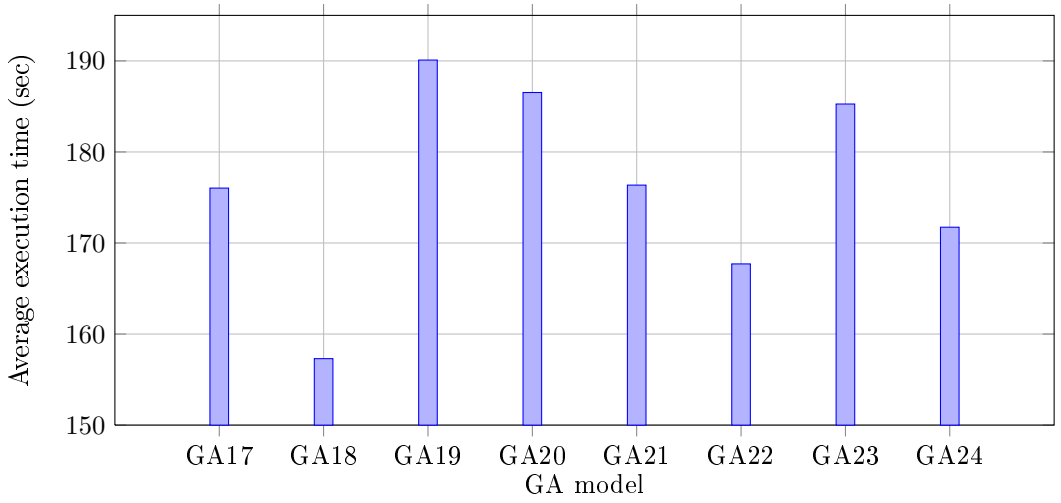

Figure 6.14: *The average execution time per algorithm run comparison of GA17 to GA24 for the MMRP.*

Chapter 7

Tabu search approaches

Tabu search is a metaheuristic optimisation method belonging to the class of local search techniques. Tabu search enhances the performance of a local search method by using memory structures: Once a potential solution has been determined, it is marked as "taboo" so that the algorithm does not visit that possibility repeatedly or converge to a local optimum.

7.1 Overview

Initially the most popular approach to solving hard combinatorial optimisation problems was based on *Local Search* (LS) improvement techniques [46]. LS may be summarized as an iterative search procedure that, starting from an initial feasible solution, progressively improves it by applying a series of local modifications. At each iteration, the search moves to an improving feasible solution that differs only slightly from the current one. The search terminates when it encounters a local optimum with respect to the transformations that it considers. However, unless extremely lucky, this local optimum is often a fairly mediocre solution. In LS the quality of the solution obtained and computing times are usually highly dependent upon the quality of the set of transformations considered at each iteration of the heuristic.

Building upon some of his previous work, Fred Glover [37] proposed a new approach in 1986 which he called *Tabu Search* (TS) to allow LS methods to overcome local optima. Glover did not see TS as a proper heuristic, but rather as a metaheuristic, a term also originally coined by him in 1986. In the same year, Hansen [43] proposed a similar approach which he named the steepest ascent/mildest descent heuristic. Since its introduction TS has enjoyed a number of successes. In a variety of problem settings, it has found solutions superior to the best previously obtained by alternative methods and Glover [40] presents a partial list of these successes. A pseudocode listing for a general TS appears in Algorithm 6 [27].

Algorithm 6: `General tabu search`

Input: A combinatorial optimisation problem specification including a domain set for each decision variable. An initial configuration $\boldsymbol{x}_1$, a memory size $\hat{m}$. An objective function $f(\cdot)$ to determine solution fitness. An adjustment function to reach neighbours.

Output: An approximation of a global optimum solution.

1: Set the current solution $\boldsymbol{x}$ to the initial solution $\boldsymbol{x}_1$. Set the tabu list $\mathcal{T}$ to be the empty set, $\mathcal{T} \rightarrow \emptyset$.
2: Generate a set of neighbours of $\boldsymbol{x}$. If the complete set of neighbours $\mathcal{N}(\boldsymbol{x})$ is too large, apply a reduction or filtering technique to reduce its size. Let the remaining set be $\mathcal{V}(\boldsymbol{x}) \subset \mathcal{N}(\boldsymbol{x})$.
3: Remove all candidates $\boldsymbol{x'} \in \mathcal{V}(\boldsymbol{x})$ for which the move $\boldsymbol{x} \rightarrow \boldsymbol{x'}$ appears in the tabu list.
4: Evaluate the objective function $f(\boldsymbol{y})$ for each remaining $\boldsymbol{y} \in \mathcal{V}(\boldsymbol{x})$. Let $\boldsymbol{y}^*$ be the solution which has the best objective function value amongst these elements.
5: Replace the current $\boldsymbol{x}$ with $\boldsymbol{y}^*$, even if it has a worse objective function value. Insert the reverse move $(\boldsymbol{y}^* \rightarrow \boldsymbol{x})$, into the tabu list $\mathcal{T}$. Remove the oldest element of the tabu list if its size exceeds $\hat{m}$.
6: Go to steps 2 and repeat until a termination condition is reached.

The *search space* of an LS or TS metaheuristic $\mathcal{S}$, is simply the space of all possible solutions that can be considered during the search. Closely linked to the definition of the search space is that of the neighbourhood structure. The *neighbourhood structure* of $\mathcal{S}$, $\mathcal{N}(\mathcal{S})$, is the subset of solutions obtained by applying a single local transformation to $\mathcal{S}$. *Tabu conditions* characterize the forbidden solutions and *aspiration conditions* are algorithmic devices that will allow one to revoke or cancel tabus.

In this study, a TS is applied to the blending problem with appropriate definitions as described in the following sections.

7.1.1 Search space and neighbourhood structure

Gendreau [36] observes that choosing a search space and a neighbourhood structure is by far the most critical step in the design of any TS. In general, for any specific problem at hand, there are many more possible neighbourhood structures than search space definitions. This follows from the fact that there may be several plausible neighbourhood structures for a given definition of the search space.

An important issue in TS is the need for evaluating the neighbourhood structure. At each iteration of a basic TS algorithm, it is normally necessary to identify the best solution in the neighbourhood of the current solution taking into account that the new solution has no tabu attributes or, if it happens to have, the aspiration criterion is satisfied. Usually, the sizes of the neighbourhoods are much larger than can be evaluated by the algorithm,

and thus only the most attractive part of a neighbourhood is actually explored. Having the means of finding efficiently such attractive parts is critical to TS methodology. There are general purpose strategies for reducing neighbourhood size but often these do not work and problem specific strategies have to be developed.

7.1.2 Tabus

Arguably the most important algorithmic device used to determine the solutions admitted to $\mathcal{N}(\mathcal{S})$ is the tabu list. A tabu list is a short-term memory structure which contains the solutions that have been visited in the recent past to help the search move away from previously visited portions of the search space and thus perform more extensive exploration. This past is less than n iterations ago, where n is the number of previous solutions to be stored, also called the *tabu tenure*.

Usually only a fixed and fairly limited quantity of information is recorded in the tabu list. One could record complete solutions, but this requires large storage space and makes it expensive to check whether a potential move is tabu or not. It is therefore seldom used. The most commonly used tabus involve recording the last few transformations performed on the current solution and prohibiting reverse transformations. Another approach is to record only key characteristics of the solutions themselves [38].

Standard tabu lists are usually implemented as circular lists of fixed length. A circular list is a variant of a linked list in which the nominal "tail" is linked to the "head". The entire list may be accessed starting at any item and following links until one comes to the starting item again. The basic role of the tabu list is to prevent *cycling* (repeatedly returning to previously visited solutions). If the length of the list is too short, this role might not be achieved; conversely a too long list creates too many restrictions and it has been observed in [38] and [39] that the mean value of the visited solutions grows with the increase of the tabu list size. Therefore, lists of variable length are usually preferred.

More complex, sophisticated applications require the use of long-term memory. This allows for the re-initialization of the search from a high quality solution set, the redefinition of the neighbourhood structure based on high quality solutions, the redefinition of the objective function to penalize certain attributes of high quality solutions and the adaption of the search strategy based on knowledge acquired during previous searches. An effective way for circumventing this difficulty is to use a tabu list with variable size. Each element of the list belongs to it for a number of iterations that is bounded by given maximal and minimal values. Also, multiple tabu lists can be used simultaneously and are sometimes advisable [39].

7.1.3 Aspiration criteria

While central to TS, tabus are sometimes too limiting. They may prohibit attractive moves, even when there is no danger of cycling, or they may lead to an overall stagnation

of the searching process. It is necessary to use aspiration criteria to cancel a set of tabus. The simplest and most commonly used aspiration criterion consists in allowing a move, even if it falls within the tabu zone, if it results in a solution with an objective value better than that of the current best-known solution (since the new solution has obviously not been previously visited) [36]. This is viable as the number of moves classified tabu will generally be small relative to the number available, and it is assumed that the expense of evaluating a move is not great.

Much more complicated aspiration criteria have been proposed and successfully implemented by, see for instance, Hertz and De Werra [45], but they are rarely used. The key rule in this respect is that if cycling cannot occur, tabus may be disregarded.

7.1.4 Intensification and diversification

In general, *intensification* is based on some intermediate-term memory in which one records the number of consecutive iterations that various solution characteristics have been present in the current solution without interruption. A typical approach to intensification is to restart the search from the best currently known solution and to "freeze" (fix) in it the components that seem more attractive. Another technique that is often used consists in changing the neighbourhood structure to one allowing more powerful or more diverse moves [36]. In probabilistic TS, one could increase the sample size or switch to searching without sampling.

One of the main problems of all methods based on Local Search approaches, including TS, is that they tend to be too local, *i.e.* they tend to spend most of their time in a restricted portion of the search space. The negative consequence of this fact is that, although good solutions may be obtained, one may fail to explore the most favourable parts of the search space and thus end up with solutions that are still fairly far from the optimal ones. *Diversification* is an algorithmic mechanism that attempts to alleviate this problem by forcing the search into previously unexplored parts of the search space. It is usually based on some form of long-term memory of the search in which one records the total number of iterations (since the beginning of the search) that various solution characteristics have been present in the current solution or have been involved in the selected moves.

There are two major diversification techniques. *Restart diversification* involves forcing a few rarely used components in the current solution (or the best known solution) and restarting the search from this point. *Continuous diversification* integrates diversification considerations directly into the regular searching process. This is achieved by biasing the evaluation of possible moves by adding to the objective a small term related to component frequencies. Soriano and Gendreau [92] discusses these two techniques extensively.

7.2 TS for continuous global optimisation

TS is a heuristic optimisation technique developed specifically for combinatorial problems such as graph coloring [29], quadratic assignment [90], electronic circuit design [15] and scheduling [3]. Little attention, however, has been paid to applications to continuous optimisation problems such as the problem at hand. Hu [47] may be the first to present an adaptation of TS to continuous optimisation, but its main principle seems far away from the original TS by Glover.

Siarry and Berihiau present an adaptation of the original simple TS to the optimisation of continuous function [89], but it only deals with low dimension problems efficiently and many concepts such as intensification, diversification, and aspiration level are not included. On the base of this work, Chelouah and Siarry [23] develop an improved TS algorithm where the diversification and intensification strategies are introduced. The neighbourhood space of the current solution is partitioned into a set of concentric balls or hyperrectangles and one point is selected randomly inside each ball or hyperrectangle. To enforce a significant moving away of the current solution, no points inside the central ball or hyperrectangle is permitted in the neighbour selection.

7.2.1 Continuous TS by the hypersquare method

The work of Wang *et al.* [100] provides further improvement to the TS algorithm for continuous problems. Their approach will be referred to as the *hypersquare method* for the *continuous tabu search* (CTSh). An aspiration level is added to a strategy similar to the neighbourhood space partitioning using concentric hyperrectangles used in Chelouah and Siarry [23]. The neighbourhood of the current solution is generated by not only randomly selecting a point inside each concentric hyperrectangle, but also selecting certain points randomly inside the central hyperrectangle, which is inhibited in all previous studies. They find that the extra selection inside the central hyperrectangle can improve the performance of the TS algorithm.

To define the neighbourhood of the current solution $\boldsymbol{x}$, the notion of a hyperrectangle is used. The description of the submatrix dependencies in Chapter 4 allows for the application of the CTSh algorithm to the submatrix which contains the blend recipes only. This is the case for all three the problems at hand. If $\boldsymbol{c} = [\underline{c}_1, \underline{c}_2, \ldots, \underline{c}_i]$ is this submatrix with $\underline{c}_i = [c_{1i}, c_{2i}, \ldots, c_{ji}]$, lowerbounds $\underline{l}_i = [l_{1i}, l_{2i}, \ldots, l_{ji}]$ and upperbounds $\underline{u}_i = [u_{1i}, u_{2i}, \ldots, u_{ji}]$, a *hyperrectangle* $H(\underline{c}_i, \underline{h}_i)$ is defined as a solution space of $\boldsymbol{c}$ with radius $\underline{h}_i = [h_{1i}, h_{2i}, \ldots, h_{ji}]$ such that

$$H(\boldsymbol{c}, \underline{h}_i) = \{\boldsymbol{c}' \big| \, |c'_{ji} - c_{ji}| < h_{ji}, l_{ji} < c_{ji} < u_{ji}, \text{ for all } i, j\}. \tag{7.1}$$

The neighbourhood space of the current solution is partitioned into a set of concentric hyperrectangles $H(\boldsymbol{c}, h_i^{(z-1)}, h_i^z)$ with radii $\underline{h}_i = [h_{i0}, h_{i1}, \ldots, h_i^Z]$ such that

$$H^z(\boldsymbol{c}, \underline{h}_i^{(z-1)}, \underline{h}_i^z) = \{\boldsymbol{c}' \big| h_{ji}^{(z-1)} \leq |c'_{ji} - c_{ji}| < h_{ji}^z, l_{ji} < c'_{ji} < u_{ji}\}, \tag{7.2}$$

for all j, i, and $z = 1, 2, \ldots, Z$ and

$$H^0(\hat{\boldsymbol{x}}, \underline{h}_i^0) = \{\boldsymbol{c}^{'} \big| |c_{ji}^{'} - c_{ji}| < h_{ji}^0, l_{ji} < c_{ji}^{'} < u_{ji}\}, \tag{7.3}$$

where radius $\underline{h}_i^0$ is an independent parameter for all j and i. Figure 7.1 contains a partitioning of a solution space for $z = 4$.

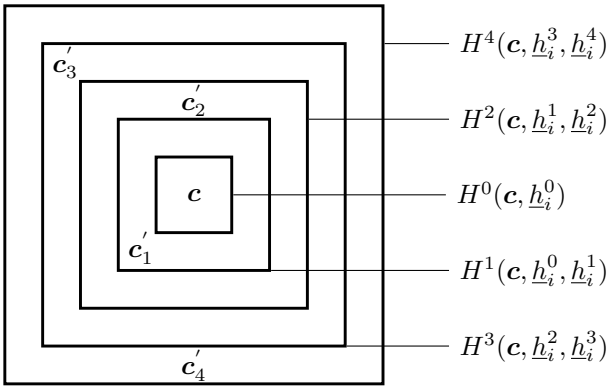

Figure 7.1: *An example of the partitioning of a solution space in two dimensions with $z = 4$.*

Neighbours of $\boldsymbol{c}$ are obtained by randomly (using a uniform distribution) selecting a point inside each hyperrectangle H^z, for $z = 1, \ldots, Z$ and the intensification strategy is conducted by selecting some extra points as neighbours of $\boldsymbol{c}$ inside the hyperrectangle H^0.

The tabu condition is organized by a tabu list which contains all the visited solutions together with the objective function values during the last t iterations (where t is the tabu tenure). In the discrete case, the tabu mechanism relies on an equality test on the configurations. This cannot transpose to interval domains, where two intervals have a near to zero chance to be equal. A tabu configuration must forbid the search not only at a point, but in an area around it. To check if a solution $\boldsymbol{c}$ is tabu, twofold tabu conditions are used with the first condition predetermining the need for the second one. First if $f(\boldsymbol{c})$ is not within a certain tolerance of any function value in the tabu list, solution $\boldsymbol{c}$ is not tabu, otherwise the second tabu condition will be applied. When $f(\boldsymbol{c})$ is within the tolerance for some solution $\boldsymbol{c}'$ in the tabu list, another check is done for the specific $\underline{c}_i$ of $\boldsymbol{c}$ and $\underline{c}_i^{'}$ of $\underline{c}^{'}$. If all values in $\underline{c}_i$ are within a tolerance of $\boldsymbol{c}_i^{'}$, solution $\boldsymbol{c}$ is considered to be tabu.

Making use of the submatrix dependencies described in Chapter 4, a complete solution $\boldsymbol{x}$ may be constructed using $\boldsymbol{c}$. The aspiration level is to compare the objective function value $f(\boldsymbol{x})$ with $f(\boldsymbol{x}^*)$ directly, where $f(\boldsymbol{x}^*)$ is the best solution found so far. If $f(\boldsymbol{x}) > f(\boldsymbol{x}^*)$, the aspiration level is satisfied. A pseudocode listing for the CTSh appears in Algorithm 7.

Algorithm 7: `Continuous TS by the hypersquare method (CTSh)`

Input: An initial solution $\boldsymbol{x}_0 \in \mathcal{S}$ (the search space) as well as its objective function value $f(\boldsymbol{x}_0)$.

Output: An approximation of a global optimum solution.

1: Initialise the current solution $\boldsymbol{x} \leftarrow \boldsymbol{x}_0$ and initialise the best solution $\boldsymbol{x}^* \leftarrow \boldsymbol{x}_0$ with $f(\boldsymbol{x}^*) \leftarrow f(\boldsymbol{x})$.
2: Generate a neighbourhood $\mathcal{N}(\boldsymbol{x})$. Initialize test variable *found* $\leftarrow$ `false` and initialize index $i \leftarrow 1$.
3: **while** *found* = `false` **do**
4: Set $\boldsymbol{x}'$ to the i-th best solution in $\mathcal{N}(\boldsymbol{x})$.
5: **if** $f(\boldsymbol{x}) > f(\boldsymbol{x}^*)$ **then**
6: Accept $\boldsymbol{x}'$ as the new current solution. Enter $\boldsymbol{x}'$ as well as $f(\boldsymbol{x}')$ into the tabu list. Update $\boldsymbol{x}^*$ and $f(\boldsymbol{x}^*)$.
7: **else**
8: **if** $\boldsymbol{x}'$ is not tabu **then**
9: Accept $\boldsymbol{x}'$ the new current solution, even if $f(\boldsymbol{x}') < f(\boldsymbol{x})$ and enter it into the tabu list together with $f(\boldsymbol{x}')$.
10: *found* = `true`.
11: **else**
12: $i = i + 1$
13: **end if**
14: **end if**
15: **end while**
16: Repeat steps 2 to 15 until there is no improvement in $f(\boldsymbol{x}^*)$ after m iterations. Return $\boldsymbol{x}'$

7.2.2 Continuous TS by the immediate zone method

Hajji *et al.* [42] propose a TS method based on the work of Hu [47]. Their approach will be refered to as the *immediate zone method* for the *continuous tabu search* (CTSz). It uses a tabu list that contains all points and a prohibited zone around each point that depends on the value of its objective function. This prohibited zone decreases as the number of iterations increases. Alternation of intensification and diversification phases allows to find the global optimum with a good accuracy.

It is assumed that no information on the location of the optimum is available at the first iteration. Thus the initial solutions in the neighbourhood space are generated in the whole search space using a uniform distribution. For the next iterations, solutions are generated using the normal distribution [47]. The description of the submatrix dependencies in Chapter 4 allows for the application of the CTSz algorithm to the submatrix which contains the blend recipes only. Let this submatrix be $\boldsymbol{c}$ of the solution $\boldsymbol{x}$. The probability

density is defined as

$$p(c_{ji}) = \frac{1}{\sigma_i\sqrt{2\pi}}\left(-\frac{(c_{ji}-c^*_{ji})^2}{2\sigma_i^2}\right) \text{ for all } i,j, \tag{7.4}$$

where σ_i is the standard deviation, and $\boldsymbol{c}^*$ is the submatrix containing the blend recipes of the best solution in the search space at the previous iteration. All candidate solutions $\boldsymbol{c}^{'}$ are generated using a random number, r, given by uniform distribution and the function of $P(x)$ such that

$$\boldsymbol{c}^{'}_i = \boldsymbol{c}^*_i + \sigma_i P^{-1}(r), \qquad \text{for all} i \text{ with} \tag{7.5}$$

$$P(x) = \int_{-\infty}^{x} p(u)\ \mathrm{d}u, \qquad \text{for } 0 \leq r \leq 1. \tag{7.6}$$

The tabu list contains all tabu regions that are hyperrectangles (defined by their center and size) for each solution. A tabu region is associated to every solution already generated. Its center is at the solution. A difficulty arises about the determination of lengths of the sides of these hyper-rectangles and some heuristic rules are proposed. As the probability to find the global optimum near good points is assumed to be higher than near bad points, the side lengths must depend on the value of the objective function. It is assumed that the sum of tabu regions is roughly the same whatever the iteration. This means that the sides lengths of a hyperrectangle $L_i(\boldsymbol{c}^{'}_i, g)$ for all i are computed with

$$L_i(\boldsymbol{c}^{'}_i, g) = \frac{c^u_{ji} - c^l_{ji}}{\lambda}\frac{f(\boldsymbol{x})}{f(\boldsymbol{x}^*_{g-1})}\frac{2}{\sqrt[n]{g}} \tag{7.7}$$

where $\boldsymbol{c}^{'}$ is the center of the hyperrectangle, n is the number of candidate solutions, g is the rank of iterations, $\boldsymbol{c}^u$ and $\boldsymbol{c}^l$ are the respective upper and lower bounds for $\boldsymbol{c}^{'}$, λ is a constant and $f(\boldsymbol{x}^*_{g-1})$ is the objective function value of the best solution from the previous iteration. The tabu list and side lengths are updated at each iteration. No tabu region is removed from the tabu list but the tabu region size decreases with the iteration rank. The process is illustrated in Figure 7.2.

Again, intensification and diversification techniques are applied to improve the effectiveness of the TS. The algorithm proposed here begins with intensification: At the beginning of each iteration, σ_i is set to $\left(c^u_{ji} - c^l_{ji}\right)/10$ because it is assumed that better solutions have a higher probability to be generated close to the current best solution. While using the normal distribution, 68% of generated points are on average between $c^*_{ji} - \sigma_i$ and $c^*_{ji} + \sigma_i$ for all i,j.

As many solutions are generated close to the best one and as the neighbourhood of the best solution becomes tabu, many solutions are rejected because they are inside tabu regions. If the number of rejected solutions is more than 95% of the number of generated solutions, the standard deviation is increased. This way, solutions are generated further from the best one and significantly new configurations can be found. The exploration of never visited regions of the search space is the diversification process.

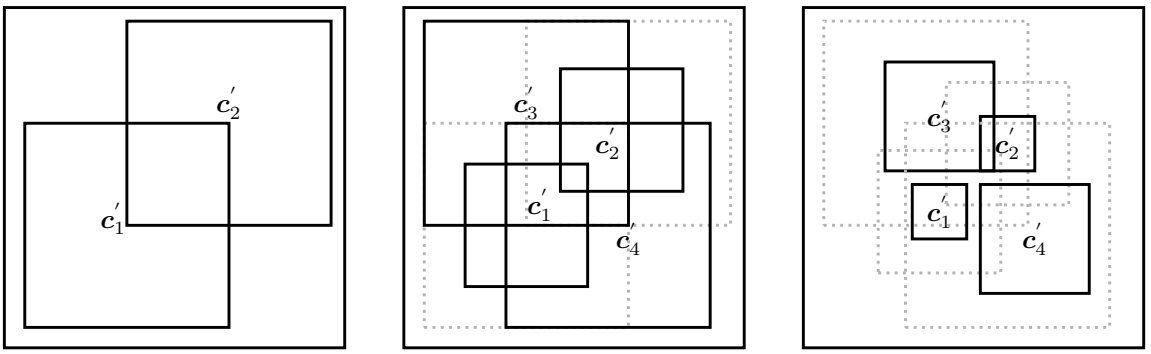

Figure 7.2: *The decrease of each tabu region size with increase in iteration rank for the CTSz. Here $n = 2$ and $G = 3$ where G is the total number of algorithm iterations.*

Four parameters, characterizing the TS algorithm proposed by Hanjii *et al.* have a great influence on its convergence. They are the number of candidate solutions generated at each iteration n, the maximum number of iterations G, the constant c, and the relative accuracy of the optimum location θ. Heuristic rules allows to build relations between them. A relative accuracy

$$\theta = \frac{1}{2}\frac{L_i(\boldsymbol{c}_i^*, g)}{c_{ji}^u - c_{ji}^l} = \frac{1}{\lambda \sqrt[n]{g}} \tag{7.8}$$

is achieved when the algorithm stops. From equation (7.8) the constant

$$\lambda = \frac{1}{\theta \sqrt[n]{g}} \tag{7.9}$$

is computed. The ratio of the total volume of all tabu regions ($\mathcal{T}$) on the volume of the search space ($\mathcal{S}$) is the same for any iteration, namely

$$\frac{|\mathcal{T}|}{|\mathcal{S}|} \approx \beta = \frac{\sum_{j=1}^{n \cdot g} \left[\prod_{i=1}^{n} L_i(\boldsymbol{c}_j, g) \right]}{\prod_{i=1}^{n} (\boldsymbol{c}_i^u - \boldsymbol{c}_i^l)}, \tag{7.10}$$

where β is a constant between 0 and 1. Using equation (7.7) together with equation (7.10) it follows that

$$\beta \approx \frac{2^n \cdot n}{\lambda^n}. \tag{7.11}$$

If $\beta < 1$ then the number of candidate solutions generated at each iteration must fulfill

$$n < \frac{\lambda^n}{2^n}. \tag{7.12}$$

Thus the size of the neighbourhood structure depends on the rank of the current iteration and the relative accuracy of the optimum location.

A pseudocode listing for the CTSz appears in Algorithm 8.

Algorithm 8: `Continuous TS by the immediate zone method (CTSz)`

Input: An initial solution $\boldsymbol{x}_0$ from the search space $\mathcal{S}$ as well as its objective function value $f(\boldsymbol{x}_0)$.

Output: An approximation of a global optimum solution.

1: **while** Generation rank $g <$ total number of iterations G **do**
2: Generate a neighbourhood $\mathcal{N}(\boldsymbol{x})$ with n solutions chosen according to the uniform density probability.
3: Store all solutions in the tabu list and evaluate the size of the hyperrectangles using (7.7).
4: Initialize $k \leftarrow 1$, $\sigma_i \leftarrow k \cdot \frac{(c_i^u - c_i^l)}{10}$ and $\boldsymbol{x}_{g-1}^*$ to the best solution at the previous iteration.
5: **while** Number of rejected solutions $\geq 95\%$ of all generated solutions **do**
6: Generate a neighbourhood $\mathcal{N}(\boldsymbol{x})$ using (7.4) – (7.6). Reject solutions that fall in existing tabu regions and count the number of rejected solutions.
7: **if** Number of rejected solutions $\geq 95\%$ of all generated solutions **then**
8: $k = k + 1$
9: **end if**
10: **end while**
11: **end while**
12: Return $\boldsymbol{x}_G^*$

7.3 Computational results

The two continuous tabu search methods CTSh and CTSz are applied to the three problems. The computational setup is the same as described in §5.4. For the RST approaches, a confidence level of 90% is obtained by calculating the average objective function values for the CTSh after 65, 4 625 and 3 090 algorithm runs for the SSP, HPP and MMRP, respectively. The same confidence level is obtained by calculating the average objective function values for the CTSz after 160, 1 605 and 2 505 algorithm runs for the SSP, HPP and MMRP, respectively.

7.3.1 The CTSh

For the three problems of Chapter 2, $\underline{l} = [0, 0.1, 0.2, \ldots 1]$ and $\underline{u} = [0, 0.1, 0.2, \ldots, 1]$ for every component in each blend. The simplified sample problem has a TAME-component

for which $\underline{l} = [0, 0.15]$ and $\underline{u} = [0, 0.15]$. During each iteration, one component is chosen at random to investigate the effect that an alteration thereof will have on the objective function value. Using the notation of Chapter 3, let this component be c_{ji} for the three problems. It is initialised to be a value in H_0 so that $0 < c_{ji} \leq 0.1$. Values for the remaining components to make up the blend are normalised so that the sum of their values is unity. The neighbourhood for the solution containing c_{ji} is constructed by choosing values for c_{ji} from the remaining 9 hyperrectangles $(H_1(\boldsymbol{c}, 0.1, 0.2), H_2(\boldsymbol{c}, 0.2, 0.3), \ldots, H_9(\boldsymbol{c}, 0.9, 1.0))$ with the normalization of the other components after each choice to maintain feasibility.

The effect of different values for the tabu tenure is shown in Figures 7.3 to 7.5. To ensure that a difference in result is caused by a change in tenure only, the algorithm is fixed so that the same component is altered for all of the 100 iterations for 10 algorithm runs and the same set of initial solutions is used for every tenure under consideration. This causes a poor average objective function value, but facilitates fair judgment of the change caused. The neighbourhood space contains 100 solutions and the tolerance $\epsilon_f = 0.01$. For all three problems, it is concluded that the size of the tenure does not affect the performance of the algorithm with regards to average objective function values obtained and a tenure $t = 5$ is chosen. The choice of this value allows for at least some operation of an element of the TS algorithm that traditionally plays a much greater role in the performance of the algorithm.

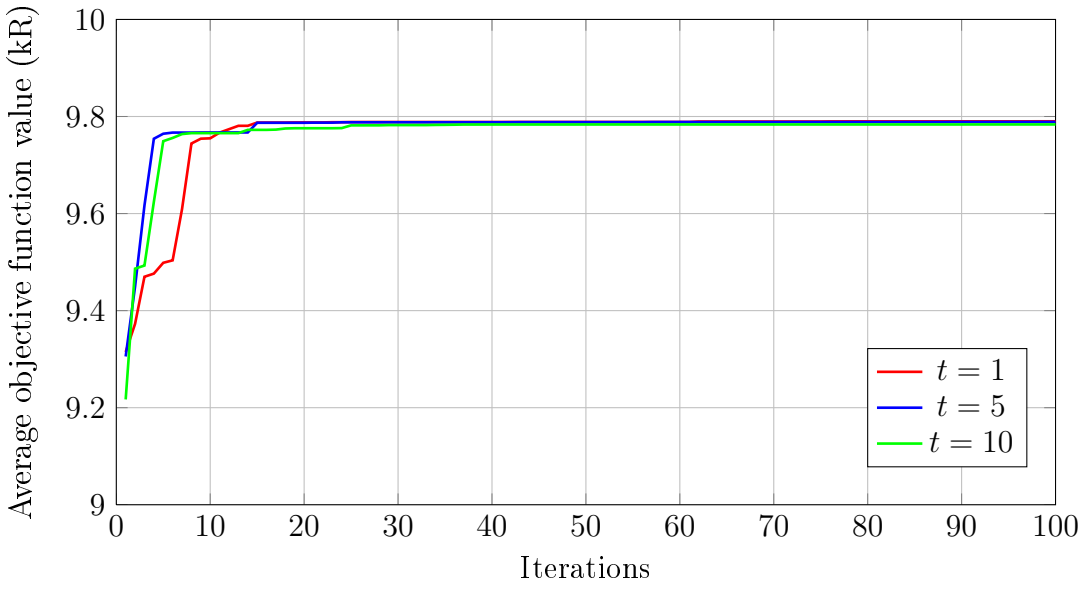

Figure 7.3: *The effect of different tabu tenures on the average objective function value for the CTSh for the SSP.*

The effect of different neighbourhood space sizes ($\mathcal{N}$) on the average objective function value is shown in Figures 7.6 to 7.8. Again the algorithm is fixed so that the same component is altered. For each algorithm run the same initial solution is used for each neighbourhood space size under consideration. The tenure $t = 5$ and the tolerance $\epsilon_f =$

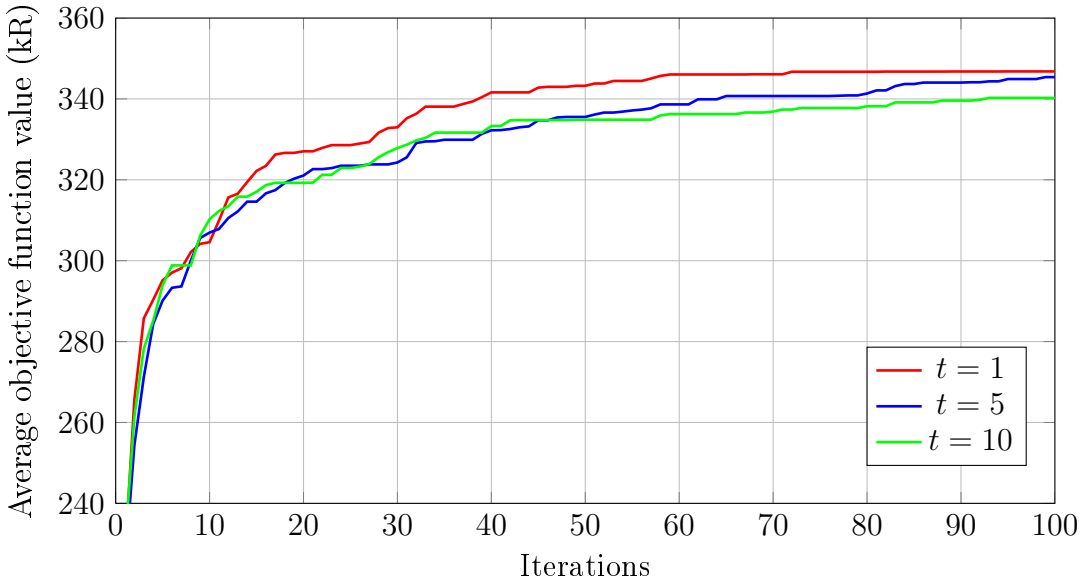

Figure 7.4: *The effect of different tabu tenures on the average objective function value for the CTSh for the HPP.*

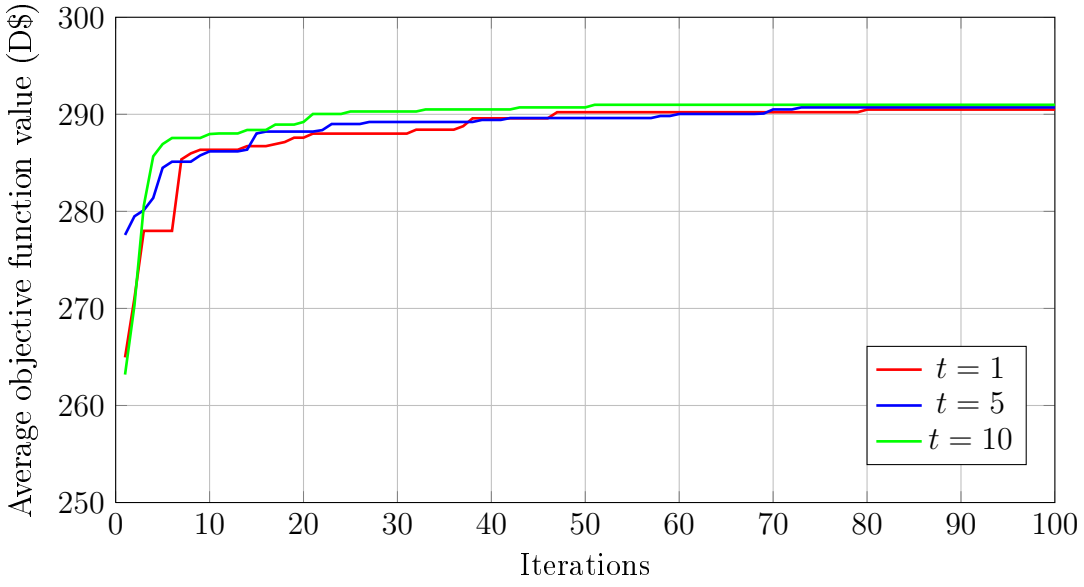

Figure 7.5: *The effect of different tabu tenures on the average objective function value for the CTSh for the MMRP.*

0.01 were used. From the results it is concluded that a change in the neighbourhood space size does not affect the performance of the algorithm in terms of the average objective function value obtained for the SSP and MMRP. Therefore, the smaller neighbourhood space size of 100 solutions is recommended merely because of its quicker execution time. For the HPP, however, a greater neighbourhood space size delivers a better result than a

smaller one.

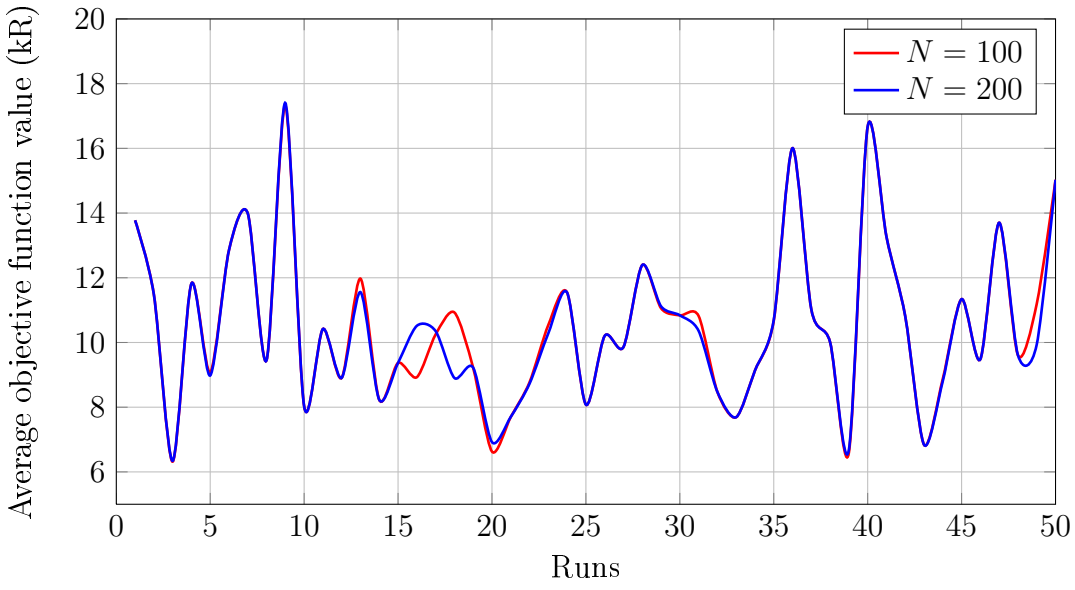

Figure 7.6: *The effect of different neighbourhood space sizes on the average objective function value for the CTSh for the SPP*

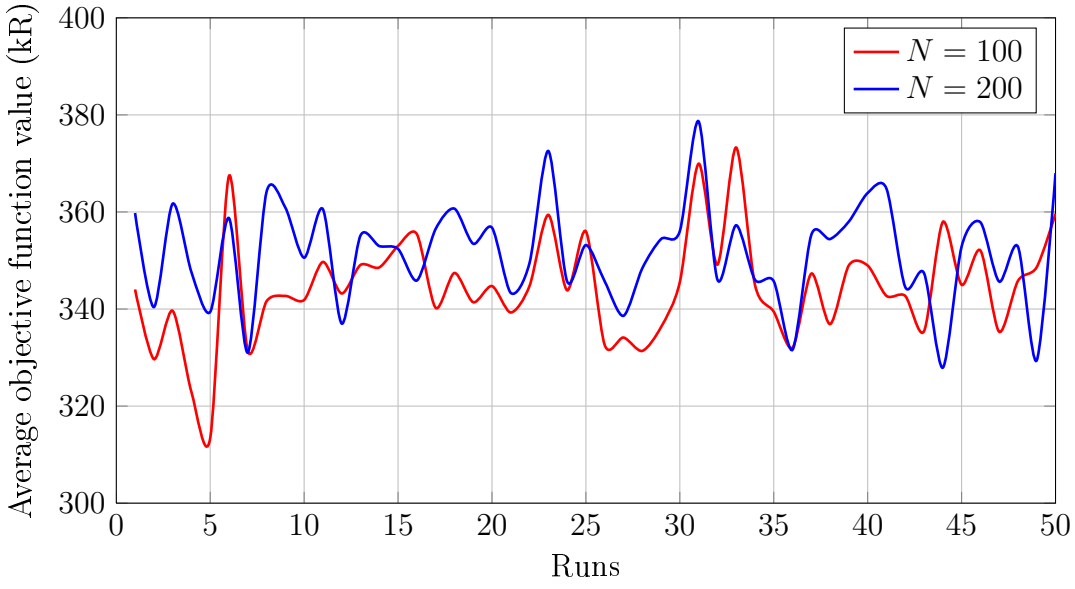

Figure 7.7: *The effect of different neighbourhood space sizes on the average objective function value for the CTSw for the HPP.*

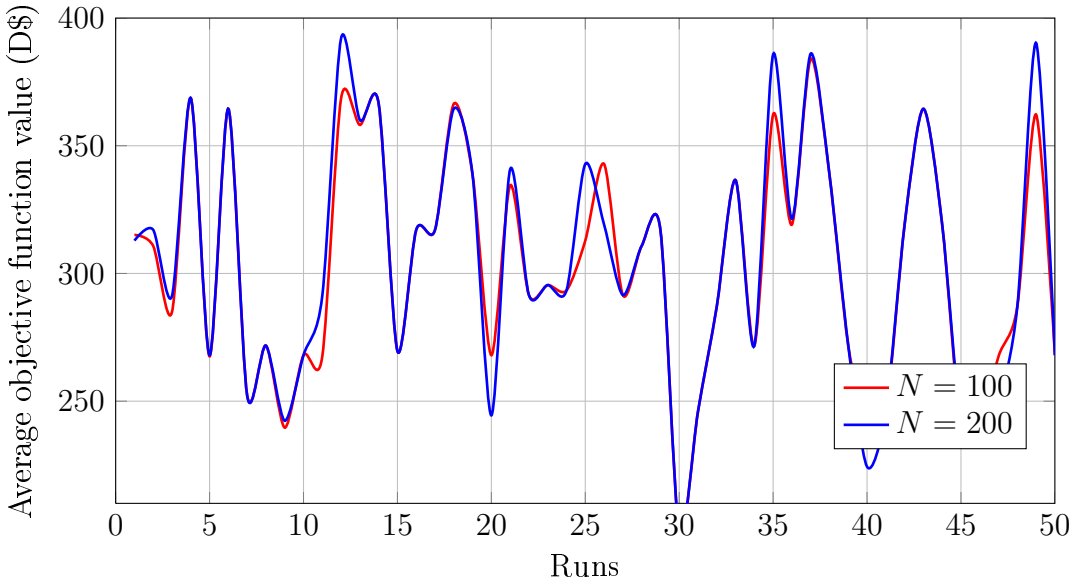

Figure 7.8: *The effect of different neighbourhood space sizes on the average objective function value for the CTSw for the MMRP.*

7.3.2 The CTSz

For the three problems of Chapter 2, $c_{ji}^l = 0$ and $c_{ji}^u = 1$ for every component j in each product i. The simplified sample problem has a TAME-component for which $c_{ji}^u = 0.15$. Also, $n = 10$. During each iteration, one component is chosen at random to investigate the effect their alteration of it will have on the objective function value. Using the notation of Chapter 3, let this component be c_{ji} for all three problems. The neighbourhood structure is constructed by altering the value of c_{ji} as described in §7.2.2 and the components are normalised so that feasibility is maintained.

The effect of different values for the relative accuracy of the optimum location (θ) is shown in Figures 7.9 to 7.11. The value of θ affects the value of the constant c which in turn affects the lenghts of the sides of the hyperrectangles which make up the tabu regions. As in §7.3.1 the algorithm is fixed so that the same component is altered for all of the iterations and the same set of initial solutions is used for every value of θ under consideration to ensure fair judgement of the change caused. The results indicate in Figures 7.9 to 7.11 show that larger values for θ lead to larger objective function values and $\theta = 0.5$ is chosen.

7.3.3 Comparison of methods

The CTSh and CTSz are applied to the SSP, HPP and MMRP with their individual parameters set as determined in §7.3.1 and §7.3.2. A confidence level of 90% is obtained by calculating the average objective function values for the CTSh after 66, 4625 and

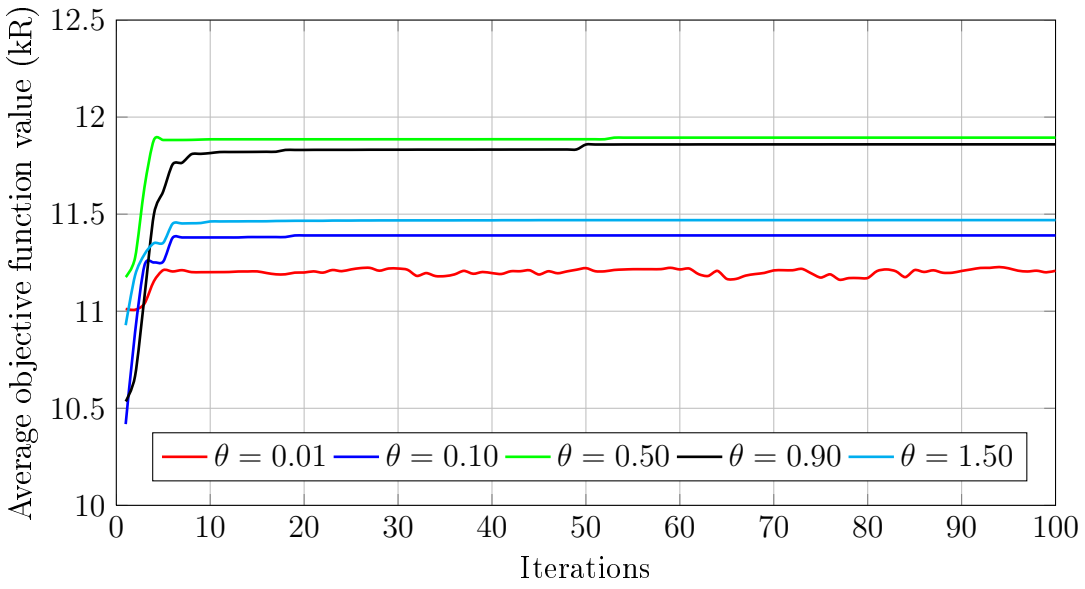

Figure 7.9: *The effect of different θ values on the average objective function value for the CTSz as applied to the SSP.*

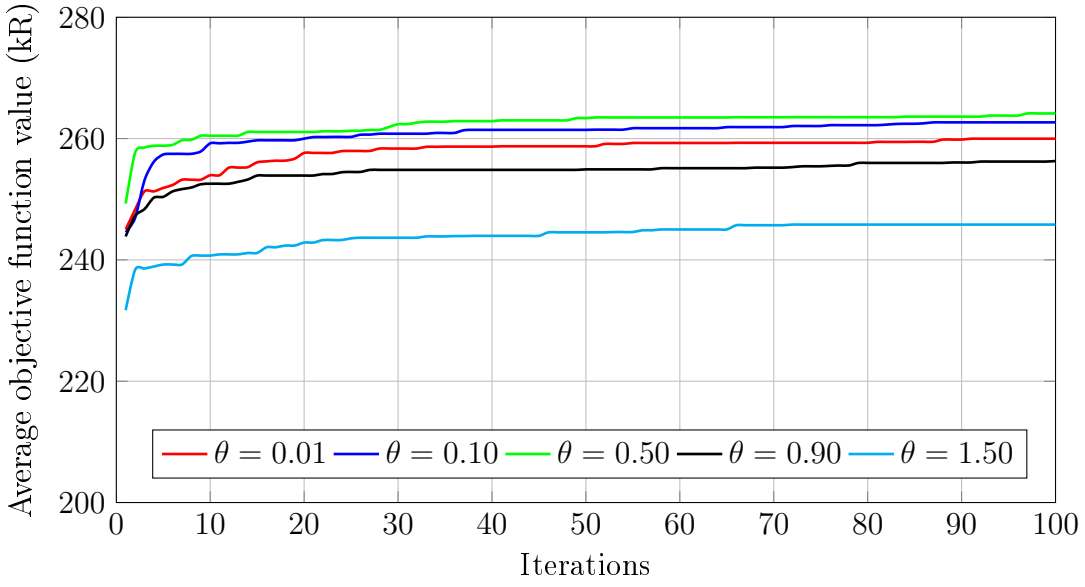

Figure 7.10: *The effect of different θ values on the average objective function value for the CTSz as applied to the HPP.*

2 505 algorithm runs for the SSP, HPP and MMRP, respectively. The same confidence level is obtained by calculating the average objective function values for the CTSz after 165, 1 605 and 3 095 algorithm runs for the SSP, HPP and MMRP, respectively. Figures 7.12 to 7.14 contain the comparative results for the two algorithms when the choice of component to be altered remains constant for all iterations. This fixed value yields a poor

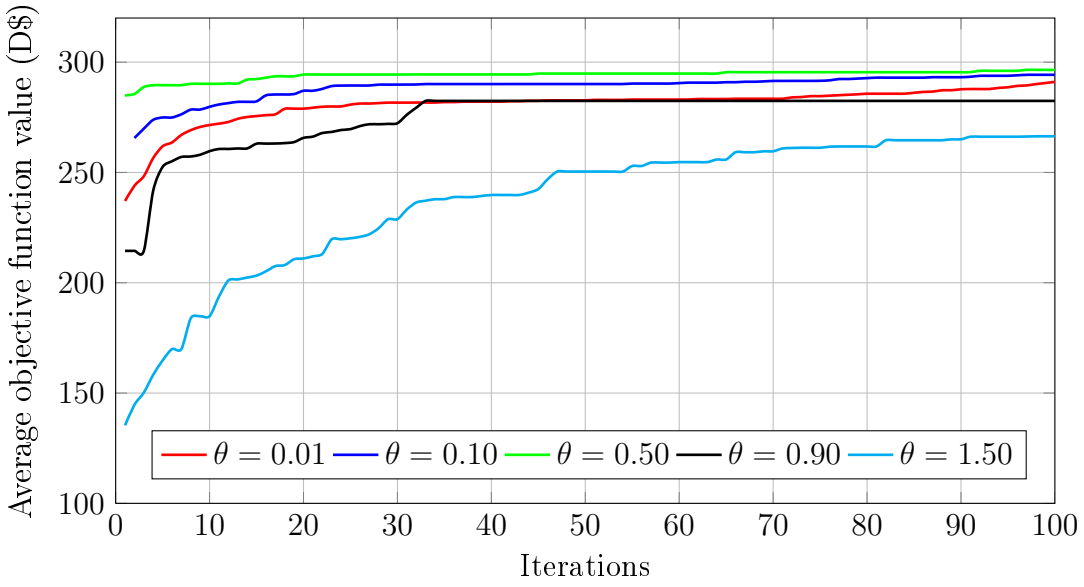

Figure 7.11: *The effect of different θ values on the average objective function value for the CTSz as applied to the MMRP.*

average objective function value, but it facilitates fair judgment of algorithm performance as possible bias from random variables is eliminated. For the three problems, the CTSh delivers a higher average objective function value. Figures 7.12 to 7.14 also contain the comparative results for the two algorithms when the choice of component to be altered is random for all iterations. Again the CTSh delivers a higher average objective function value for the three problems. Table 7.1 contains a summary of the results obtained after application of the CTS approaches for the three problems.

	SSP (kR)		**HPP** (kR)		**MMRP** (D$)	
Result	**CTSh**	**CTSz**	**CTSh**	**CTSz**	**CTSh**	**CTSz**
Best fitness	18.84	19.42	397.44	377.02	412.00	413.95
Average fitness	17.28	16.34	387.25	260.82	411.36	314.23
Standard deviation	1.43	1.70	10.22	54.85	3.66	2.10
Average execution time (sec)	1.14	1.88	0.67	0.32	0.47	3.64

Table 7.1: *Results summary for the TS techniques for the SSP, HPP and MMRP.*

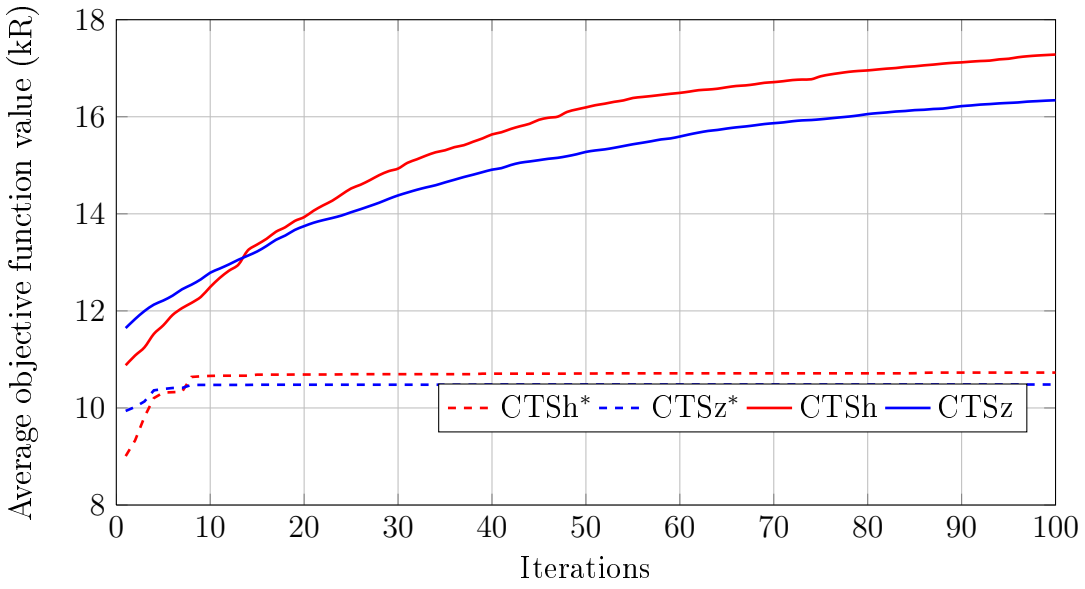

Figure 7.12: *The average objective function values as obtained by the CTSh versus the average objective function values as obtained by the CTSz when the same constant component is chosen for alteration during every iteration (CTSh* and CTSz*) and when any component for alteration is chosen at random during every iteration (CTSh and CTSz) for the SSP.*

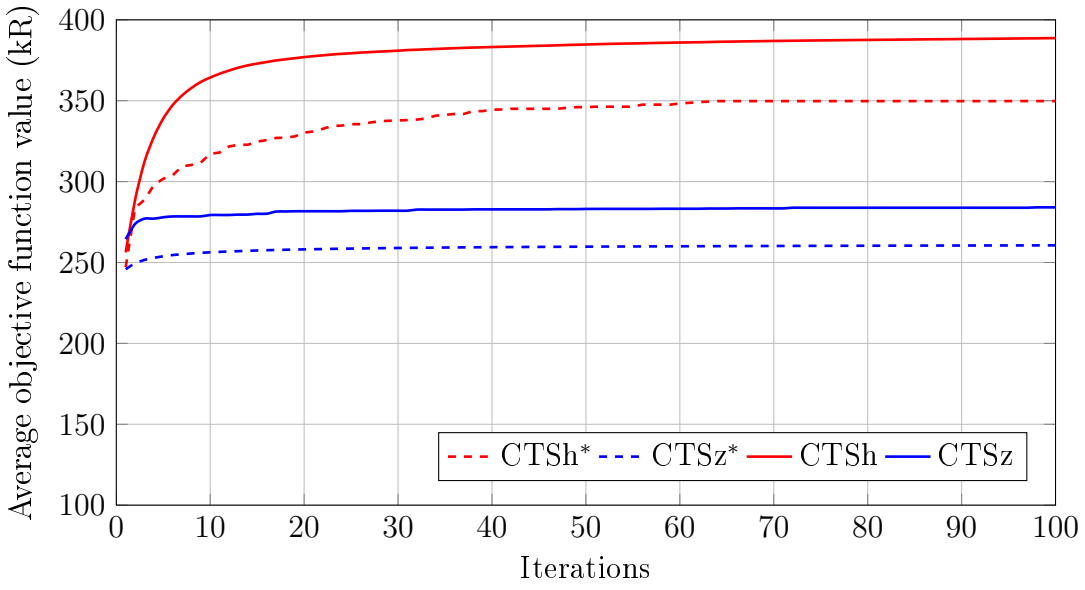

Figure 7.13: *The average objective function values as obtained by the CTSh versus the average objective function values as obtained by the CTSz when the same constant component is chosen for alteration during every iteration (CTSh* and CTSz*) and when any component for alteration is chosen at random during every iteration (CTSh and CTSz) for the HPP.*

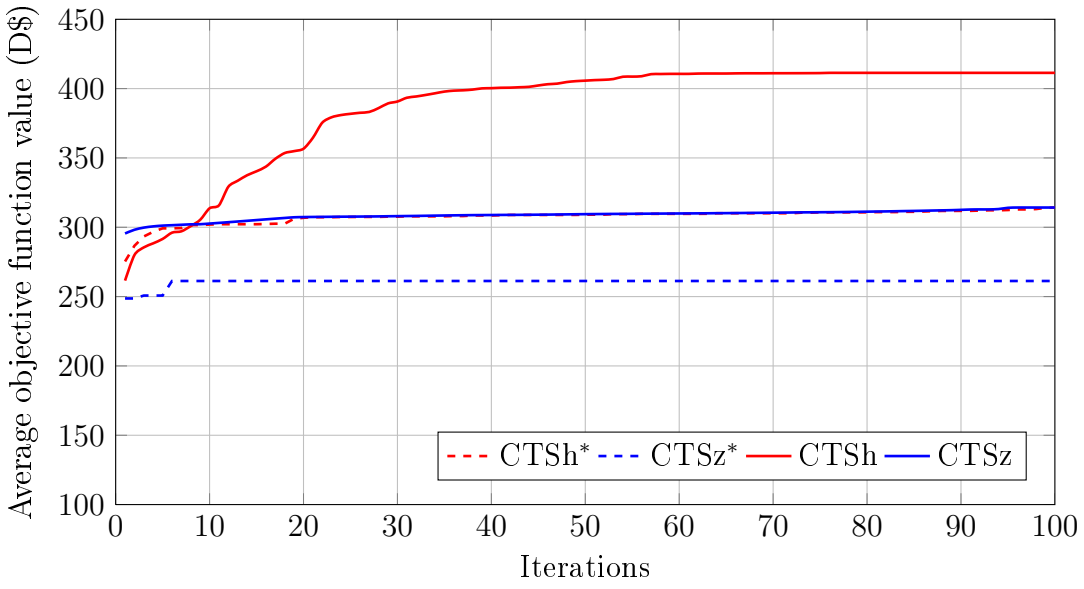

Figure 7.14: *The average objective function values as obtained by the CTSh versus the average objective function values as obtained by the CTSz when the same constant component is chosen for alteration during every iteration (CTSh* and CTSz*) and when any component for alteration is chosen at random during every iteration (CTSh and CTSz) for the MMRP.*

Chapter 8

Simulated annealing approaches

Simulated annealing (SA) is a technique which finds a good solution to an optimization problem by introducing random variations of the current solution. A worse variation is accepted as the new solution with a probability that decreases as the computation proceeds. The slower the rate of this probability decrease, the more likely the algorithm is to find an optimal or near-optimal solution. The search tries to avoid local optima by jumping out of these early in the computation. Toward the end of the computation, when the probability of accepting a worse solution is nearly zero, it seeks the bottom or top of the local optimum. The chance of getting a good solution can be traded off with computation time by slowing down the decrease in probability of accepting worse solutions. The slower the decrease, the higher the chance of finding the optimum solution, but the longer the run time. Thus effective use of this technique depends on determining a decrease rate that determines good enough solutions without taking too much computational time.

8.1 Overview

The technique's name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local optimum of the internal energy) and wander randomly through states of higher energy. The slow cooling gives those more chances of finding configurations with lower internal energy than the initial one.

By analogy with this physical process, each step of the SA algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter T (called the *temperature*), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when T is large, but becomes increasingly better as T goes to zero. The allowance for moves to worse solutions saves

the method from becoming stuck at local optima.

The method is independently described by Kirkpatrick *et al.* [58] and by Cerný [22]. It was first tested on the travelling salesman problem, finding locally optimal solutions for up to 6000 sites. At the time the exact solution had been obtained for up to 318 sites.

SA is an adaptation of the Metropolis-Hastings algorithm, a Monte Carlo method to generate sample states of a thermodynamic system invented by Metropolis *et al.* [69]. The original Metropolis scheme was that an initial state of a thermodynamic system was chosen at energy E and temperature T. Holding T constant, the initial configuration is disturbed and the change in energy, dE, is computed. If the change in energy is negative (or positive for the maximization problem) the new configuration is accepted. If the change in energy is positive (or negative for the maximization problem) it is accepted with a probability given by the Boltzmann factor $e^{-(\frac{dE}{T})}$. This processes is then repeated a sufficient number of times to give good sampling statistics for the current temperature, and then the temperature is decremented and the entire process repeated until a frozen state is achieved at $T \approx 0$.

For SA, the current state of the thermodynamic system is analogous to the current solution to a combinatorial problem, the energy equation for the thermodynamic system is analogous to an objective function and ground state is analogous to a global minimum. The major difficulty during implementation of the algorithm is that there is no obvious analogy for the temperature T with respect to a free parameter in the combinatorial problem. Furthermore, avoidance of being trapped in local optima is dependent on the *annealing schedule*, the choice of initial temperature, the number of iterations that are performed at each temperature and the amount by which the temperature is decremented at each step as cooling proceeds.

Locatelli [67] formulates the SA problem as shown in the pseudocode listing of Algorithm 9. Appropriate choices for the input functions, distribution and parameters are essential in order to guarantee the efficiency of the algorithm and are discussed in the subsequent sections.

8.2 Solution representation

When attempting to solve an optimisation problem using the SA algorithm, the most obvious representation of the control variables is usually appropriate. However, the way in which new solutions are generated may need some thought. The solution generator should introduce small random changes, and allow all possible solutions to be reached.

The SA algorithm merely needs to be supplied with an objective function for each trial solution it generates. Thus, the evaluation of the problem functions is essentially a "black box" operation as far as the optimization algorithm is concerned. Obviously, in the interests of overall computational efficiency, it is important that the objective function value evaluations should be performed efficiently, especially as in many applications these eval-

Algorithm 9: `Simulated annealing algorithm`

Input: A combinatorial optimization problem with a continuous domain X which combined with the continuity of the objective function f, guarantees the existance of the optimum value f^*. An initial configuration $x_0 \in X$, the next candidate distribution Φ, an acceptance function Γ, an initial temperature T_0, an annealing schedule Ψ and a stopping criterion.

Output: An approximation of a global optimum solution.

1: Set $i = 0$ and $T_i = T_0$.
2: Sample a candidate solution $\boldsymbol{x}'_{i+1}$ from the candidate distribution Φ.
3: Sample a uniform random number r in $[0, 1]$ and set

$$\boldsymbol{x}_{i+1} = \begin{cases} \boldsymbol{x}'_{i+1}, & \text{if } r \leq \Gamma(\boldsymbol{x}_i, \boldsymbol{x}'_{i+1}, T_i) \\ \boldsymbol{x}_i, & \text{otherwise.} \end{cases}$$

4: Set $T_{i+1} < T_i$ according to the annealing schedule Ψ.
5: Check the stopping criterion and if it fails, set $i \leftarrow i + 1$ and go back to step 2.

uations are overwhelming the most computationally expensive activity. Some thought needs to be given to the handling of constraints when using the SA algorithm. In many cases the routine can simply be programmed to reject any proposed changes which result in constraint violation, so that a search is executed in feasible space only. However, Busetti [21] notes that there are two important circumstances in which this approach cannot be followed: If there are any equality constraints defined on the system, or if the feasible space defined by the constraints is (or is suspected to be) disjoint, so that it is not possible to move between all feasible solutions without passing through infeasible space. He suggests that in either case the problem be transformed into an unconstrained one by constructing an augmented objective function incorporating any violated constraints as penalty functions.

8.3 Candidate distribution

Bohachevsky *et al.* [17] suggests that the next candidate solution $\boldsymbol{x}'_{i+1}$ be obtained by first generating a random direction vector $\underline{\tau}_i$ with $||\underline{\tau}_i|| = 1$, multiplying it by a fixed step size Δr and, finally, summing the resulting vector to $\boldsymbol{x}_i$, *i.e.*

$$\boldsymbol{x}'_{i+1} = \boldsymbol{x}_i + \Delta r \underline{\tau}_i. \tag{8.1}$$

Locatelli [67] presents an example which shows that it cannot always be assumed that the objective function value behaves the same in every direction. Therefore, the step sizes to define the next candidate solution should not all be equal for all the directions, but different directions should have different step sizes. Vanderbilt and Louie [97] suggest that

new trial solutions can be generated according to the formula

$$\boldsymbol{x}_{i+1}^{'} = \boldsymbol{x}_i + \boldsymbol{Q}\underline{\varepsilon} \tag{8.2}$$

where $\underline{\varepsilon}$ is a vector of uniform random numbers in the range $(-\sqrt{3}, \sqrt{3})$ so that each has zero mean and unit variance, and $\boldsymbol{Q}$ is a matrix that controls the step size distribution. In order to generate random steps with a covariance matrix $\boldsymbol{\xi}$, $\boldsymbol{Q}$ is found by solving

$$\boldsymbol{\xi} = \boldsymbol{Q}\boldsymbol{Q}^T \tag{8.3}$$

by Cholesky decomposition, for example. The matrix $\boldsymbol{\xi}$ should be updated as the search progresses to include information about the local topography, so that

$$\boldsymbol{\xi}_{i+1} = (1-\nu)\boldsymbol{\xi}_i + \nu\omega\mathbf{X} \tag{8.4}$$

where matrix $\boldsymbol{X}$ measures the covariance of the path actually followed and the damping constant ν controls the rate at which information from $\boldsymbol{X}$ is folded into $\boldsymbol{\xi}$ with weighting ω. One drawback of this scheme is that the solution of equation (8.3), which must be done every time $\boldsymbol{\xi}$ is updated, can contribute a substantial computational overhead for problems with high dimensionality.

An alternative strategy suggested by Parks [75] is to generate solutions according to

$$\boldsymbol{x}_{i+1}^{'} = \boldsymbol{x}_i + \boldsymbol{D}\underline{\varpi} \tag{8.5}$$

where $\underline{\varpi}$ is a vector of uniform random numbers in the range $(-1, 1)$ and $\boldsymbol{D}$ is a diagonal matrix which defines the maximum change allowed in each variable. After a successful trial (*i.e.* after an accepted change in the solution) $\boldsymbol{D}$ is updated so that

$$\boldsymbol{D}_{i+1} = (1-\nu)\boldsymbol{D}_i + \alpha\omega\boldsymbol{\Upsilon} \tag{8.6}$$

where $\boldsymbol{\Upsilon}$ is a diagonal matrix. The elements of $\boldsymbol{\Upsilon}$ consist of the magnitudes of the successful changes made to each control variable, *i.e.*

$$\Upsilon_{ii} = \|D_{ii}\varpi_i\| \tag{8.7}$$

and the damping constant α controls the rate at which information from $\boldsymbol{\Upsilon}$ is folded into $\boldsymbol{D}$ with weighting ω. This tunes the maximum step size associated with each control variable towards a value giving acceptable changes. Parks [75] finds that suitable values of ν and ω are 0.1 and 2.1, respectively.

8.4 The acceptance function

In the existing literature about SA algorithms for continuous global optimization very few acceptance functions have been employed. Most applications make use of the so-called Metropolis acceptance function that is given by

$$\Gamma(\boldsymbol{x}, \boldsymbol{x}', T) = \min\left\{1, e^{\left(-\frac{f(\boldsymbol{x}')-f(\boldsymbol{x})}{T}\right)}\right\}. \tag{8.8}$$

This function accepts both steps in which the new candidate solution $\boldsymbol{x}'_{k+1}$ improves or worsens the objective function value with respect to the current solution $\boldsymbol{x}_i$ in order to escape local optimums. The acceptance probability is controlled by the temperature parameter T.

The so-called Barker criterion [11]

$$\Gamma'(\boldsymbol{x}, \boldsymbol{x}', T) = \frac{1}{1 + e^{\left(\frac{f(\boldsymbol{x}') - f(\boldsymbol{x})}{T}\right)}} \tag{8.9}$$

is another possible acceptance function. Initially, this function may reject even steps that improve the function value, in particular those which do not cause noticable improvement. But, as the temperature decreases, steps that improve the function value are more likely to be accepted (unless they are very small ones), while steps that worsen it are more likely to be rejected. Schuur [87] shows that under appropriate assumptions, most other acceptance functions are equivalent to equations (8.8) and (8.9) which explains the restricted choice found in the literature.

8.5 Annealing schedule

The annealing schedule determines the permitted amount of movement towards a worse solution during the search and is critical to the algorithm's performance. The principle underlying the choice of a suitable annealing schedule is stated by Bounds [19] as "The initial temperature should be high enough to melt the system completely and should be reduced towards its freezing point as the search progresses, but choosing an annealing schedule for practical purposes is something of a black art."

The standard implementation of the SA algorithm is one in which homogeneous chains of finite length are generated at decreasing temperatures. It requires that the parameters as defined in the subsequent subsections be defined.

8.5.1 Initial temperature T_0

The choice of a very high initial temperature results in a lot of time being wasted at the beginning stage of the search. However, a very low initial temperature could result in the system being trapped in a local optimum, because the configuration space was not sufficiently explored. Many methods have been proposed in literature to compute the initial temperature T_0. It is suggested in Kirkpatrick *et al.* [58] to take $T_0 = \Delta E_{\max}$, where $\Delta E_{\max}$ is the maximal objective function value difference between any two neighboring solutions.

Another method described by Kirkpatrick *et al.* [58] consists of computing a temperature such that the acceptance ratio is approximately equal to a given value χ_0. First, a large

initial temperature is chosen. Then, a number of candidate solution samplings using this temperature must be performed. The ratio of accepted solutions is compared with χ_0. If it is less than χ_0, then the temperature is multiplied by 2. The procedure continues until the observed acceptance ratio exceeds χ_0. Other variants are proposed to obtain an acceptance ratio which is close to χ_0. It is, for example, possible to divide the temperature by 3 if the acceptance ratio is much higher than χ_0. Using this kind of rules, cycles are avoided and a good estimation of the temperature can be found.

Ben-Ameur [14] proposes a simple algorithm to compute a temperature which is compatible with a given acceptance ratio. The pseudocode listing for it is shown in Algorithm 10.

Algorithm 10: `Computation of the initial temperature`

Input: A solution domain $\mathcal{S}$ as well as an estimated number of solutions, $||\mathcal{S}||$, to be drawn from $\mathcal{S}$ needed to compute the estimated initial temperature $\hat{\chi}(T)$. A desired acceptance probability, $\chi_0 \in [0, 1]$ and a small ($O(10^{-3})$) real number ϵ.

Output: The estimated initial temperature $\hat{\chi}(T)$.

1: Generate and store $||calS||$ random solutions from X.
2: Set T_1 to any strictly positive number and set $n \leftarrow 1$.
3: Compute $\hat{\chi}(T_n) = \frac{\sum_{t \in S} \exp(-f_t^1/T_n)}{\sum_{t \in S} \exp(-f_t^2/T_n)}$ where f^1 and f^2 are the objective values before and after the transition from one solution to the following one in S, respectively.
4: **if** If $|\hat{\chi}(T_n) - \chi_0| \leq \epsilon$ **then**
5: Return T_n
6: **else**
7: $T_{n+1} = T_n \left(\frac{\ln(\hat{\chi}(T_n))}{\ln \chi_0} \right)^{\frac{1}{p}}$.
8: $n = n + 1$.
9: Go to step 3.
10: **end if**

8.5.2 Length of the Markov chains

SA is best described, mathematically, by a Markov chain. Named after the Russian mathematician, A.A. Markov who formalized the theory concerning events whose current condition depends solely on their condition one period before, Markov chains are special processes that also assume finite states exist, constant transition probabilities exist and equal time periods occur [85].

An obvious choice for L_k, the length of the k-th Markov chain, is a value that depends on the size of the problem, so L_k is independent of k. Alternatively it can be argued that a minimum number of transitions $\nu_{\min}$ should be accepted at each temperature. However, as T_k approaches 0, transitions are accepted with decreasing probability so the number

of trials required to achieve $\nu_{\min}$ acceptances approaches ∞. Thus, in practice, an algorithm in which each Markov chain is terminated after L_k transitions or $\nu_{\min}$ acceptances, whichever comes first, is a suitable compromise. Aarts and Van Laarhoven [1] demonstrate that the number of iterations the algorithm should make between temperature changes is an exponential function of problem size.

8.5.3 Temperature decrementation

As for the temperature decrement rule, the simplest and most common method is

$$U(z_i) = \eta T_i \tag{8.10}$$

where η is a constant close to, but smaller than, 1. The exponential cooling scheme (ECS) is first proposed by Kirkpatrick *et al.* [58] with $\eta = 0.95$. Randelman and Grest [79] compares this strategy with a linear cooling scheme (LCS) in which T is reduced every L trials, so that

$$U(z_i) = T_i - \Delta T. \tag{8.11}$$

They find the reductions achieved using the two schemes to be comparable, and also note that the final value of f is, in general, improved with slower cooling rates at the expense, of course, of greater computational effort. Finally, they observe that the algorithm performance depends more on the cooling rate $\frac{\Delta T}{L}$ than on the individual values of ΔT and L. Care must be taken to avoid negative temperatures when using the LCS.

Bohachevsky *et al.* [17] defines the annealing schedule as

$$U(z_i) = \zeta[f(x_i) - f^*]^{g_1}, \tag{8.12}$$

where ζ, $g_1 > 0$ are constants. If the global optimum value f^* is not known, it is suggested to employ an estimate $\hat{f}$ of it, which is decreased when it is too high (*e.g.* each time a function value lower than the estimate is observed), and increased if it is too low. According to equation (8.12), the value of the temperature increases as the function value in the current solution $\boldsymbol{x}_k$ increases. The idea of this annealing schedule is that worsening steps are accepted with a low probability when the current solution $\boldsymbol{x}_k$ has a value close to the global optimum value, thus trapping the algorithm in the promising region around $\boldsymbol{x}_k$. Instead, if the function value of $\boldsymbol{x}_k$ is much greater than the global optimum, the temperature is high and many worsening steps are accepted, which prevents the algorithm from getting trapped in the nonpromising region around $\boldsymbol{x}_k$. Romeijn and Smith [83] motivate the use of (8.12) with $g_1 = 1$ while Brooks and Verdini [20] discuss the critical choice of the value ζ. They find that the value of ζ should be chosen so that the percentage of accepted worsening steps with respect to the total number generated worsening steps is about 60%.

Many researchers have proposed more elaborate annealing schedules, most of which are in some respect adaptive, using statistical measures of the algorithm's current performance to modify its control parameters. These are well reviewed by Van Laarhoven and Aarts [98].

8.6 Stopping criterion

Due to the difficult nature of the problems solved by SA algorithms, it is hard, if not impossible, to define a stopping rule which guarantees to stop when the global optimum has been detected (at least within a given accuracy), or when there is significantly high probability of having detected it. Therefore the stopping rules proposed in the literature are all heuristic in nature.

In Bohachevsky *et al.* [17] and Brooks and Verdini [20] the algorithm is stopped when there has been no acceptance for a fixed number of iterations. Ingber [48] suggests the algorithm stops when the acceptance ratio falls below a prescribed value, while Jones and Forbes [51] terminate their algorithm when the statistics σ^2 falls below a prescribed value. This statistic is a measure of the variance of the function values of the sequence of solutions visited by the algorithm.

Ultimately, these stopping rules are all based on a common and quite natural idea: Stop the algorithm when it does not make significant progress over a number of iterations.

8.7 Computational results

For the three problems described in Chapter 2 the algorithm is initialized with a single randomly generated feasible solution. A candidate solution is generated by means of the method proposed by Parks [75] as described in §8.3.

Figure 8.1 contains the average resultant acceptance values obtained by means of the Metropolis acceptance function as described in §8.4 when the temperature is set from $T_i = 0.1$ to $T_i = 1$ in increments of 0.1 unit. A total of ten algorithm runs per temperature setting with all other variables fixed is performed in order to calculate a reasonable average result. A standard deviation is computed for each set of twenty runs and an average of these deviations is obtained as a measure of the acceptance function's performance. For the Metropolis acceptance function the average standard deviation is computed to be approximately 0.005.

Figure 8.2 contains the average resultant acceptance values obtained by means of the Barker acceptance function as described in §8.4 when the temperature is set from $T_i = 1$ to $T_i = 10$ in increments of 1 unit. For the Barker acceptance function, the average standard deviation is computed to be approximately 0.03. The results show the two methods to be comparable, but the Metropolis function is prefered for the problems at hand because of its smaller average standard deviation.

From the manner with which the acceptance probability decreases as observed in Figure 8.1, it is reasonable to accept the linear cooling scheme $T_{i+1} = T_1 - 0.1$. However, determining the number of iterations between temperature changes is not so obvious.

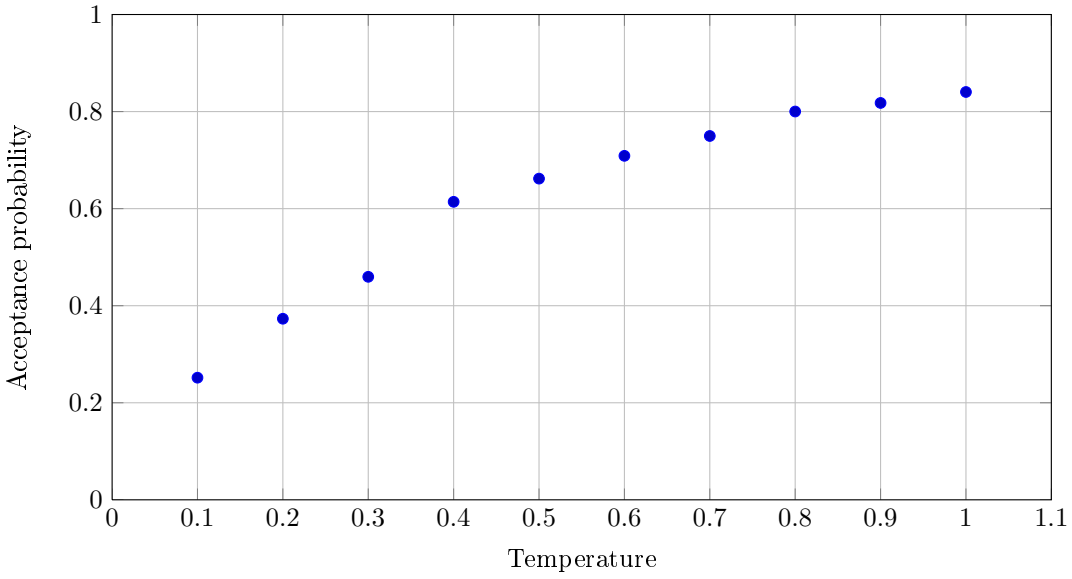

Figure 8.1: *The acceptance probability for each temperature increment for the Metropolis acceptance function.*

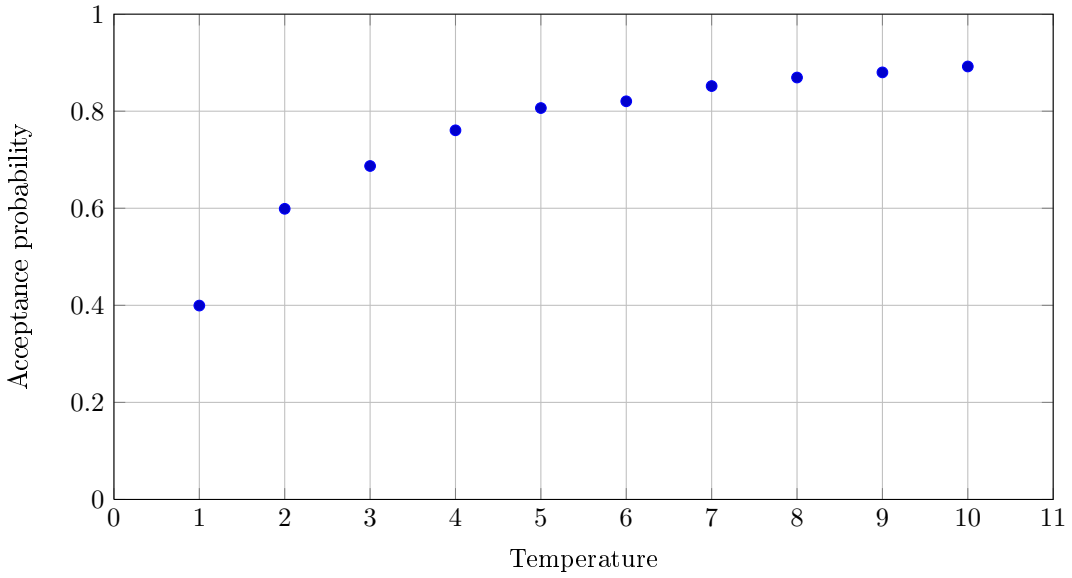

Figure 8.2: *The acceptance probability for each temperature increment for the Barker acceptance function.*

8.7.1 Results for the SSP

Table 8.1 contains a summary of the results of Figures A.1 to A.7 in Appendix A, showing the best performing Markov chain length L for each starting temperature T_0, respectively. When the initial temperature is relatively high, the resulting probability of accepting a worsening solution is high and les iterations at each temperature should be allowed in

order to maintain a good solution. When the initial temperature is relatively low, the resulting probability of accepting a worsening solution is low and more iterations at each temperature may be allowed for a more extensive exploration of the search space.

Initial temperature T_0	**Best chain length L**
0.8	100
0.7	100
0.6	200
0.5	400
0.4	300
0.3	500
0.2	500

Table 8.1: *Summary of the best Markov chain lengths for each starting temperature, respectively, for the SSP.*

Figure 8.3 contains the objective function values obtained by the algorithm for each initial temperature set at its best Markov chain length. For the SSP, a lower starting temperature (0.2 to 0.4) combined with a larger Markov chain length (300 to 500 iterations) delivers the best solutions.

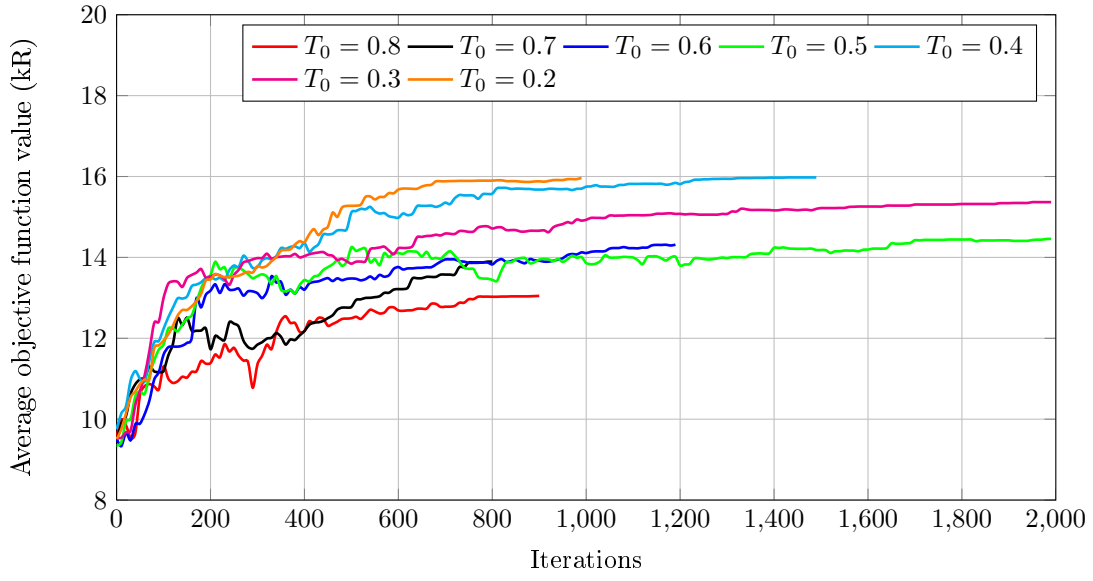

Figure 8.3: *Average objective function values for each initial temperature at its best Markov chain length for the SSP.*

8.7.2 Results for the HPP

Table 8.2 contains a summary of the results of Figures A.8 to A.14 in Appendix A, performing Markov chain length L for each starting temperature T_0, respectively.

Initial temperature T_0	**B**est chain length L
0.8	500
0.7	400
0.6	500
0.5	400
0.4	300
0.3	500
0.2	500

Table 8.2: *Summary of the best Markov chain lengths for each starting temperature, respectively for the HPP.*

Figure 8.4 contains the objective function values obtained by the algorithm for each initial temperature set at its best Markov chain length. For the HPP, a higher starting temperature (0.7 to 0.8) combined with a larger Markov chain length (400 to 500 iterations) delivers the best solutions as it allows for the most extensive solution space exploration.

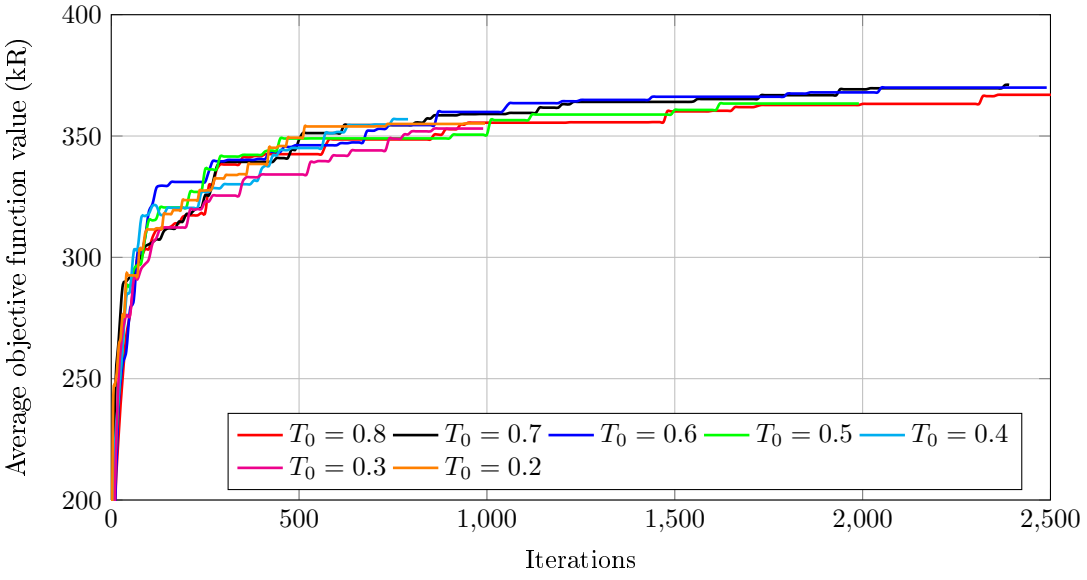

Figure 8.4: *Objective function values for each initial temperature at its best Markov chain length for the HPP.*

8.7.3 Results for the MMRP

Table 8.3 contains a summary of the results of Figures A.15 to A.21 in Appendix A showing the best performing markov chain length L for each starting temperature T_0,

respectively.

Initial temperature T_0	**Best chain length L**
0.8	200
0.7	300
0.6	300
0.5	400
0.4	300
0.3	500
0.2	500

Table 8.3: *Summary of the best Markov chain lengths for each starting temperature, respectively for the MMRP.*

Figure 8.5 contains the objective function values obtained by the algorithm for each initial temperature set at its best Markov chain length. For the MMRP, a mid-range starting temperature (0.4 to 0.7) combined with a mid-range Markov chain length (300 to 400 iterations) delivers the best solutions.

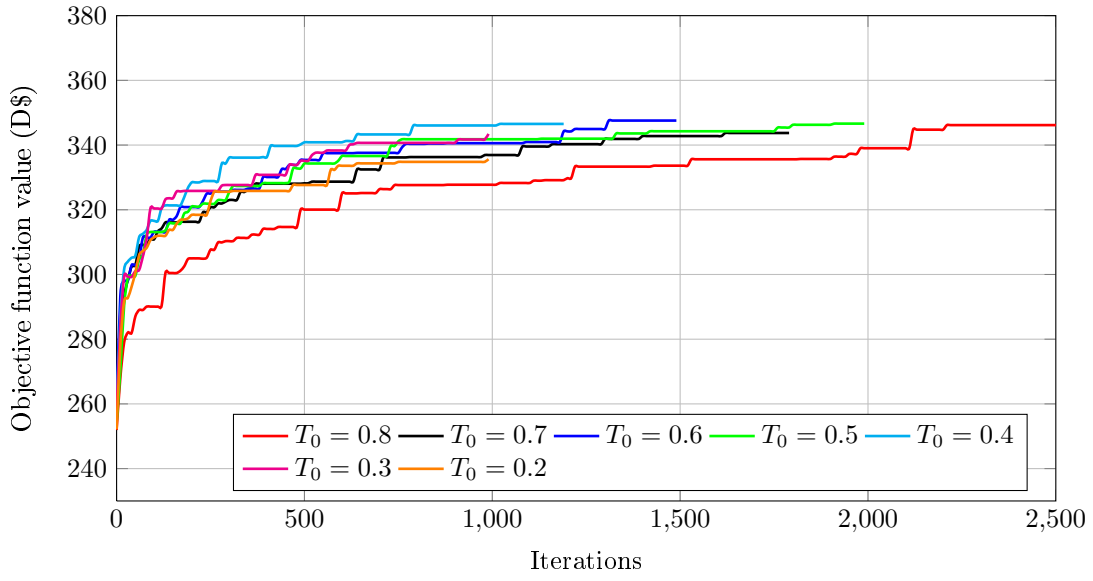

Figure 8.5: *Objective function values for each initial temperature at its best Markov chain length for the MMRP.*

Chapter 9

Solution summary

Chapters 3 to 8 contain the best arrangement for each solution approach with regards to, for example, algorithm design and parameter settings. The results obtained from the approaches set at these determined arrangements are compared in order to determine which approach performs best for the three blending problems at hand.

9.1 The linear programming approach

In this section the results of the various linear programming approaches as described in Chapter 3 are compared.

9.1.1 The SSP

Figure 9.1 contains a comparison of the objective function values obtained after application of the LP approaches for the SSP described in §2.1. By comparing the upperbounds of the LP approaches to the solution obtained by means of the exact approach, it is possible to determine which decisions variables and constraint sets have the greatest influence on the quality of the solution. From Figure 9.1 it can be seen that by concentrating on the optimisation of decision variables that are influenced by inventory constraints, better solutions are obtained than by concentrating, for example, on decision variables that are influenced by component characteristic or blend production limit constraints.

9.1.2 The HPP

The LP approach for the HPP obtaines a final solution of R40 000 while the heuristic approach (described in §2.2) obtains a final solution of R10 000. Though the heuristic presents a good method of ensuring integrity of the component characteristics when they

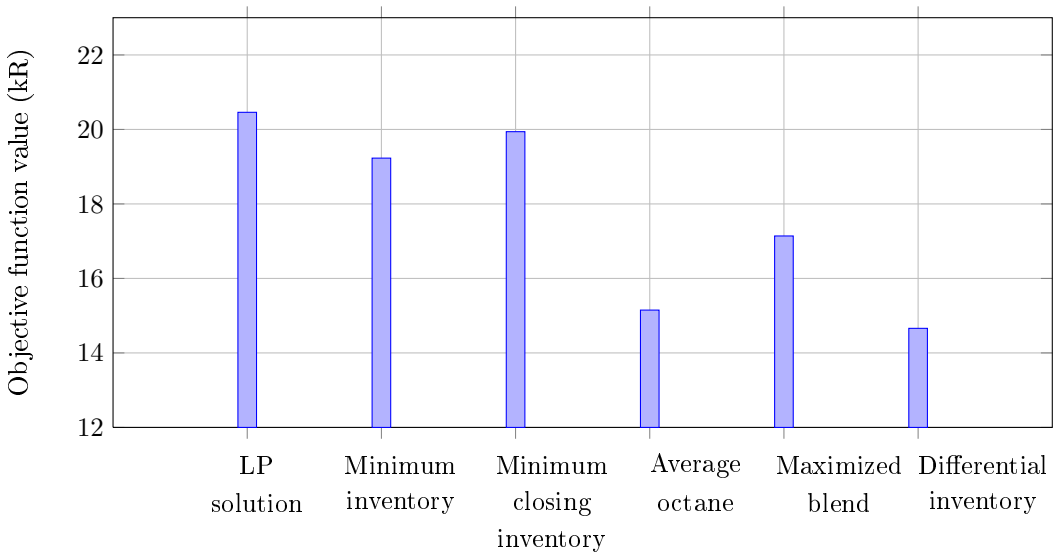

Figure 9.1: *Summary of the results obtained after application of the LP approaches for the SSP.*

are blended in pools, it performs rather poorly when compared to the solution obtained by the exact solution approach.

9.1.3 The MMRP

Figure 9.2 contains a comparison of the objective function values obtained after application of the LP approaches to the MMRP described in §2.3. Similar to the results obtained for the SSP, better solutions are obtained when optimisation is concentrated on the decision variables that are influenced by inventory constraints than when it is concentrated on decision variables that are influenced by component characteristic and input constraints.

9.2 Metaheuristic solution summary

In this section the results of the various metaheuristic approaches described in Chapters 5 to 8 are compared with each other.

9.2.1 The SSP

Figure 9.3 contains the best and average results, respectively, obtained during 100 independent algorithm runs. The genetic algorithm achieves a best result, *i.e.* closest to the LP solution, with the best result of the tabu search as a close second. On average, the tabu search obtaines a result closest to the exact solution with the genetic algorithm as a close second. However, the error bars on the graph show that the average objective

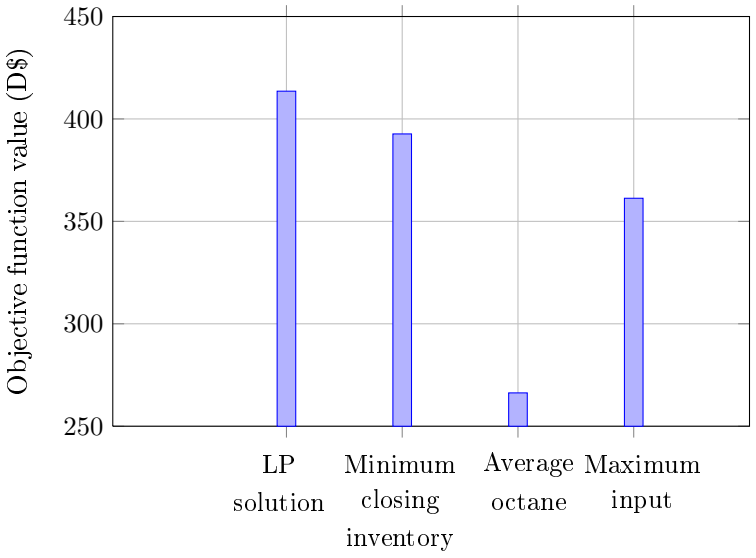

Figure 9.2: *Summary of the results obtained after application of the LP approaches for the MMRP.*

function value of the tabu search is subject to a greater standard deviation than the average objective function value obtained by the genetic algorithm approach. Regardless, it is understood that a larger deviation around a better solution is prefered to a smaller deviation around a weaker solution.

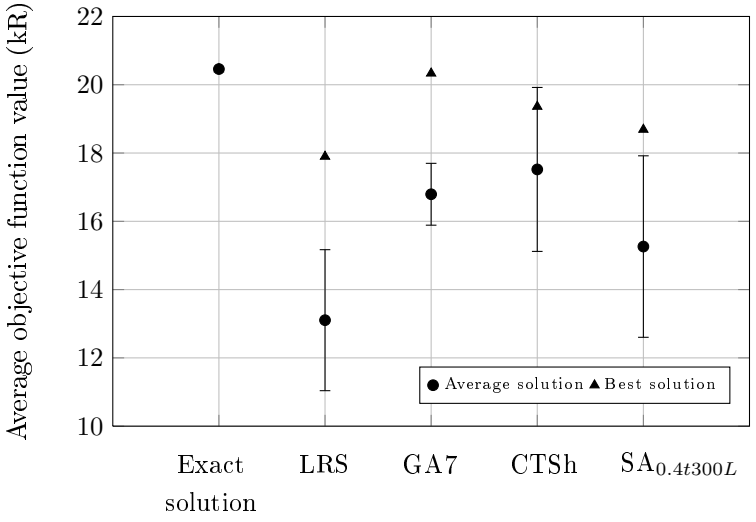

Figure 9.3: *The best and average solution for each metaheuristic obtained after 100 independent runs for the SSP. The error bars indicate the standard deviation associated with each result.*

Figure 9.4 contains the average execution time of a single run for each metaheuristic, respectively. The time it takes for a single run of the genetic algorithm is so much greater than the time it takes for a single run of any of the other metaheuristics because the

genetic algorithm requires an entire population of feasible solutions to be found in the search space as opposed to only one as in the case of the other metaheuristics.

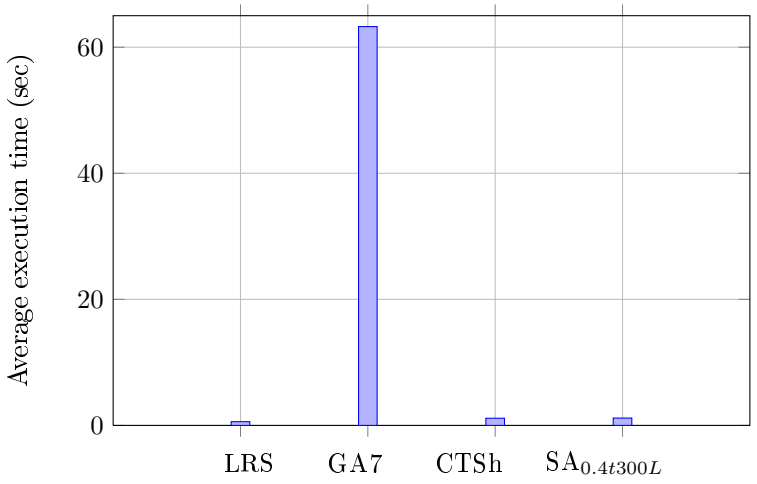

Figure 9.4: *The average execution time per run for each metaheuristic for the SSP.*

9.2.2 The HPP

Figure 9.5 contains the best and average results obtained for each metaheuristic approach obtained during 100 independent algorithm runs. Best results are obtained after application of the blind random search, genetic algorithm and tabu search, with the simulated annealing approach does not performing as well. Similar to the results obtained for the sample problem, again on average, the tabu search obtains a result closest to the exact solution with the genetic algorithm as a close second. The standard deviations indicated by the error bars show that the difference between results is least in the average best performing approach, *i.e.*, the tabu search.

Figure 9.6 contains the average execution time of a single run for each metaheuristic, respectively. Again the time it takes for a single run of the genetic algorithm or blind random search is so much greater than the time it takes for a single run of any of the other metaheuristics because both techniques require that an entire population of feasible solutions be found in the search space as oppose to only one as in the case of the other metaheuristics.

9.2.3 The MMRP

Figure 9.7 contains the best and average results obtained for each metaheuristic approach during 100 independent algorithm runs. The best results are obtained after application of the blind random search, genetic algorithm and tabu search, with the simulated annealing approach not performing as well. Similar to the results obtained for the SSP, on average,

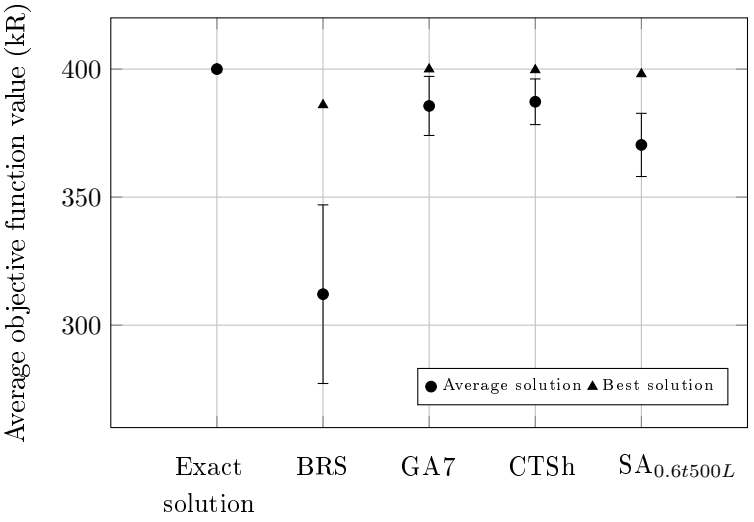

Figure 9.5: *The best and average solution for each metaheuristic obtained after 100 independent runs for the HPP.*

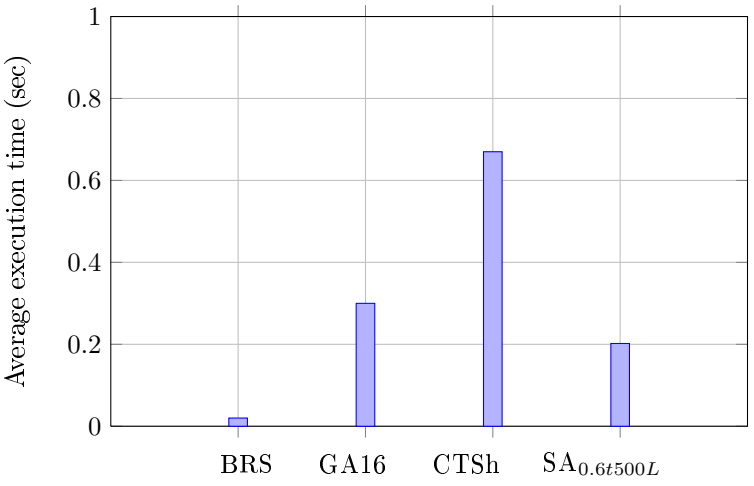

Figure 9.6: *The average execution time per run for each metaheuristic for the HPP.*

the tabu search obtains a result closest to the exact solution with the genetic algorithm as a close second. All four approaches are are quite stable and the small standard deviations associated with each approach is illustrated by the error bars in the figure.

Figure 9.8 contains the average execution time of a single run for each metaheuristic approach, respectively. Again the time it takes for a single run of the genetic algorithm or blind random search is so much longer than the time it takes for a single run of any of the other metaheuristics because both techniques require that an entire population of feasible solutions be found in the search space as opposed to only one as in the case of the other approaches.

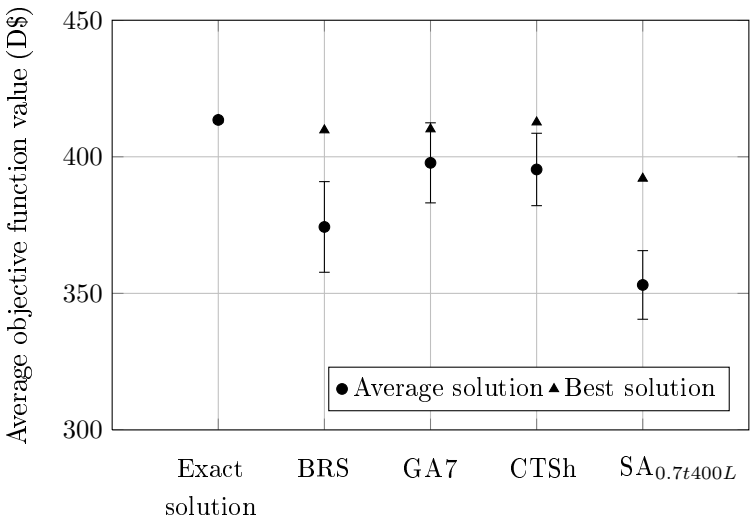

Figure 9.7: *The best and average solution for each metaheuristic obtained after 100 independent runs for the MMRP.*

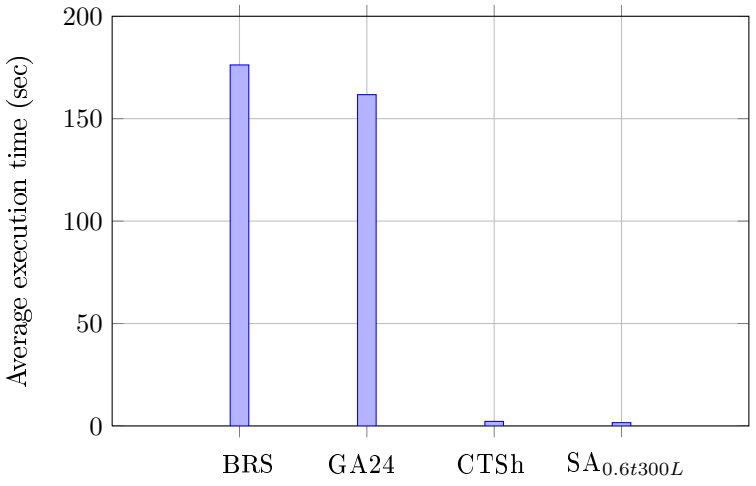

Figure 9.8: *The average execution time per run for each metaheuristic for the MMRP.*

Chapter 10

The extended MMRP

Certain characteristics of the four metaheuristic approaches have been identified and illustrated in the literature. RSTs may be assumed to perform better on relatively small problems while more intelligent configuration search techniques such as the GA, TS and SA approaches may be assumed to perform better on relatively large problems. This behavior is illustrated by Judson *et al.* [52] where RST, GA and SA approaches are applied to a scalable model problem to measure relative performance over a range of molecule sizes. They find that both GA and SA approaches perform progressively better relative to the RST approach as the molecule size increases.

The MMRP is the most complex of the three problems and is the closest to a problem likely to be found in reality. Therefore performance of the metaheuristic approaches to this problem when the size of the problem is greatly increased, is investigated. The parameters are set so that the problem now considers the production of 100 types of final products as opposed to the original 5. These products are formed by blending from a selection of 1000 components as opposed to the original 11 and these components are obtained from 100 types of crude oil (as opposed to the original number of 2). The components are obtained by refining the crude with 100 types processes as opposed to the original number of 5. All other problem parameters are maintained as described in §2.3.

Candidate solutions for the initial solutions and populations are created by choosing random values within estimated bounds. For the product characteristic constraints, the minimum allowable RON limit is a value in $[0, 130]$ while the maximum allowable RVP, density and sulfur limits are values in $[0, 20]$, $[300, 350]$ and $[0, 5]$, respectively. The price for each product is a value in $[0, 20]$. For the process constraints, the maximum number of barrels of crude that may pass through each process is an integer in $[0, 100]$ while the cost per barrel that passes through each process is a value in $[0, 1]$. For the crude constraints, the maximum allowable number of barrels that may be purchased for each crude is an integer in $[0, 500]$ while the cost per barrel for each crude is a value in $[0, 10]$. Lastly, for the component constraints, the RON, RVP, density and sulfur present in each is a value in $[0, 130]$, $[0, 20]$, $[0, 350]$ and $[0, 5]$, respectively.

With these bounds, 100 initial solutions were generated. These solutions were feasible in terms of the four main characteristic constraints, *i.e.* octane rating, vapour pressure, density and sulfur content as well as in terms of production constraints for example, the limit on the amount of crude that might pass through each process. The four metaheuristic approaches were all applied to this same set of initial solutions to facilitate fair comparison of their performance.

Figure 10.1 contains the best and average results obtained for each metaheuristic approach 100 independent algorithm runs. The best results are obtained after application of the simulated annealing and tabu search approaches. The SA approach obtains the best average result, although it has the greatest standard deviation associated with its final average solution. The GA approach delivers the most stable average result as is illustrated by the error bars on the figure.

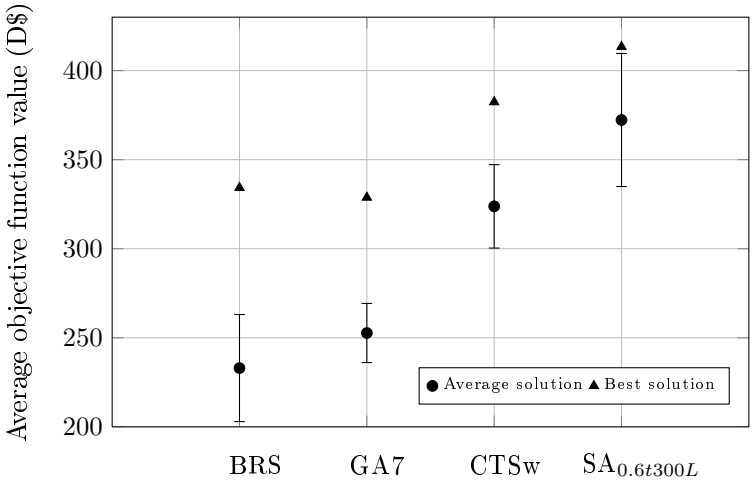

Figure 10.1: *The best and average solution for each metaheuristic obtained after 100 independent runs for the extended MMRP.*

Figure 9.8 contains the average execution time of a single run of each metaheuristic approach for the extended MMRP, respectively.

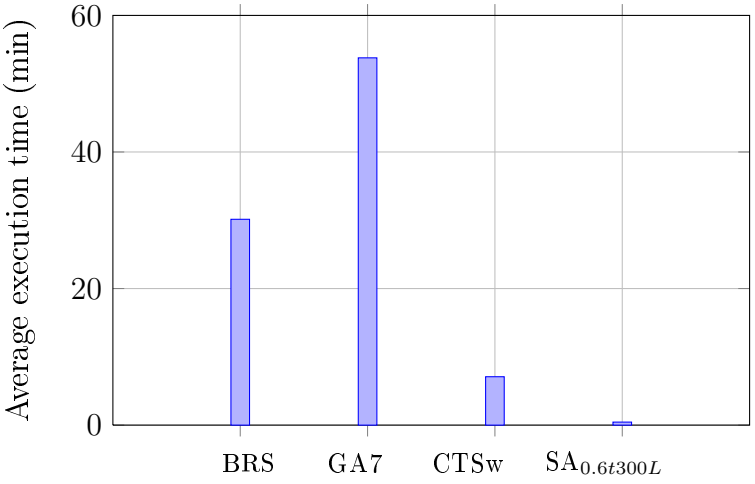

Figure 10.2: *The average execution time per run for each metaheuristic for the extended MMRP.*

Chapter 11

Conclusion

A brief summary of the work contained in this thesis is presented followed by a description of various ideas to be explored in future research.

11.1 Thesis summary

This thesis opens by defining petrochemical blending problems and by giving a description of Sasol and its operations. An introduction into the current blending solutions used by Sasol and the need for metaheuristic approaches is given. Objectives set out for this thesis also forms part of the introductory chapter.

The purpose of Chapter 2 is to provide a description of the three problems that form the basis of this thesis and upon which the design and application of the metaheuristic approaches is based. The input data and constraints for the SSP, HPP and MMRP are presented in §2.1, §2.2 and §2.3, respectively.

The three problems at hand are then modeled mathematically and the exact solution for each is obtained in §3.1, §3.3 and §3.5, respectively by means of linear programming. To understand the nature and characteristics of the problems at hand, various linear programming solution approaches are applied to the problems in §3.2, §3.4 and §3.6, respectively. This is done in fulfillment of Thesis Objective I, as stated in §1.1. Constraints placed on decision variables which describe component inventories prove to have a greater influence on the solution quality than constraints placed on decisions variables which describe the inherent characteristics of the product components. OK In Chapter 4 data structures are introduced to represent the decision variables as well as the constraints sets for each problem in §4.2, §4.3 and §4.4, respectively. Dependencies between the values of the variables are discovered and methods for generating feasible solutions are developed. A method for dealing with infeasible solutions is decided upon in §4.1. This is done in fulfillment of Thesis Objective II, as stated in §1.1.

A solution approach for the three problems at hand by means of random search techniques is investigated in Chapter 5 in fulfillment of Thesis Objective III(a), as stated in §1.1. From §5.4.1 it is concluded that local random search outperforms blind random search on average for the SSP, albeit by a small margin. From §5.4.2 it is concluded that blind random search outperforms local random search on average for the HPP and in §5.4.3, this result also holds for the MMRP.

The application of metaheuristic approaches to the petrochemical blending problems commences in Chapter 6 in fulfillment of Thesis Objective III(b), as stated in §1.1. An algorithm configuration that combines the use of elitism, ranked fitness assignment and tournament selection of solution candidates gives the best average performance for all three problems. The method of fitness assignment, however, has the greatest influence on the average performance of the algorithm. In §6.9.1 the implementation of the genetic algorithm by using island models is investigated, but it has little impact on the quality of the solutions obtained.

A tabu search approach for the three problems at hand is given in Chapter 7 in fulfillment of Thesis Objective III(c), as stated in §1.1. Despite the continuous nature of the problems, two methods are successfully applied. In conclusion (§7.3.1) neither the size of the tabu tenure nor the size of the neighbourhood significantly influence the performance of the tabu search algorithm for the three problems. In §7.3.2 it is concluded that a relative accuracy value of 0.5 delivers a better average result and the hypersquare method outperforms the immediate zone method with respect to the average objective function values obtained for the three problems.

The last metaheuristic approach investigated is the simulated annealing approach in Chapter 8 in fulfillment of Thesis Objective III(d), as stated in §1.1. Two candidate solution acceptance function values are presented, but the Metropolis acceptance function is preferred above the Barker acceptance function because of its smaller standard deviation. From §8.7.1, §8.7.2 and §8.7.3 it is concluded that a higher starting temperature in combination with a shorter markov chain length delivers the best solution for the three problems.

Chapter 9 contains the conclusion of this thesis as a summary of the performance of the various solution approaches in fulfillment of Thesis Objective IV(a), as stated in §1.1. For the SSP, the minimum inventory approach and the minimum closing inventory approach obtain solutions closest to the LP solution. For the MMRP the minimum closing inventory approach again obtains a solution closes to the LP solution. In §9.2.1 it is concluded that on average, the tabu search approach delivers the best average result with regards to objective function value and execution time. In §9.2.2 it is concluded that both the tabu search and the genetic algorithm approaches deliver the best average objective function value result, but that the tabu search performs better in terms of its execution time. In §9.2.3 this result again is evident. Furthermore, in fulfillment of Thesis Objective IV(b), Chapter 10 contains results of the metaheuristic approaches after application of them to an extended version of the MMRP. The simulated annealing approach is found to deliver

the best average performance for this problem.

11.2 Possible future work

Three suggestions are made with respect to possible future research emanating from the work presented in this thesis (in fulfillment of Thesis Objective V).

11.2.1 Size and complexity extension of the problems

In this thesis, four sample problems were presented in order to achieve a proof of concept for the successful application of metaheuristic solution approaches. As can be seen from the results for the extended MMRP in Chapter 10, the behaviour of the approaches are subject to a change in the size of the problems. The developed approaches should be applied to real life problems. Realistic datasets describing actual refinery conditions and constraints should be used instead of randomly generated values. All effort was made to obtain a real life data set from the industry, but all attempts were unsuccessful.

11.2.2 Metaheuristic configurations

Even more configurations for each of the four metaheuristic approaches should be investigated. These further configurations may include different numbers of algorithm runs for the RST approaches, different recombination and mutation rates for the GA approaches, different bounds on the tabu regions for the TS approaches and combinations and different markov chain lengths for the SA approaches. The application of self adjusting or adaptive metaheuristics may also be investigated in an effort to optimise the parameter settings [27].

11.2.3 Sensitivity

Once real life data sets have been obtained, sensitivity analysis of the various metaheuristic approaches may be conducted. The true nature of some metaheuristic approaches may only be revealed when they are applied to larger problems so that a more accurate verdict may be made as to the sensitivity of the parameters.

References

[1] Aarts EHL & Van Laarhoven PJM, 1985, *Statistical cooling: A general approach to combinatorial optimization problems*, Philips Journal of Research, **40**, pp. 193–226.

[2] Al-Shammari M & Dawood I, 1997, *Linear programming applied to a production blending problem: A spreadsheet modeling approach*, Production and Inventory Management Journal, **38(1)**, pp. 1–7.

[3] Amellal S & Kaminska B, 1993, *Scheduling algorithm in data path synthesis using the tabu search technique*, Proceedings of the EDAC-EUROASIC 1993 Conference, Paris, pp. 398–402.

[4] Anderson, RL, 1953, *Recent advances in finding best operating conditions*, Journal of the American Statistical Association, **48**, pp. 789–798.

[5] Angeline PJ, 1998, *A historical perspective on the evolution of executable structures*, Informaticae, **36(1–4)** pp. 179–195.

[6] AMPL, 2008, *Bell Laboratory's index page*, [Online], [Cited November 10th, 2008], Available from `http://www.ampl.com/`

[7] Bäck T & Hoffmeister F, 1991, *Extended selection mechanisms in genetic algorithms*, Proceedings of the Fourth International Conference on Genetic Algorithms, San Mateo (CA), pp. 92–99.

[8] Bäck T, Fogel D & Michalewicz Z, 2000, *Evolutionary computation 1: Basic algorithms and operators*. Institute of Physics Publishing, Bristol.

[9] Bäck T, Fogel DB & Michalewicz Z, 2000, *Evolutionary computation 2: Advanced algorithms and operators*, Institute of Physics Publishing, Bristol.

[10] Baker JE, 1987, *Reducing bias and inefficiency in the selection algorithm*, Proceedings of the Second International Conference on Genetic Algorithms and their Application, Hillsdale (NJ), pp. 14–21.

[11] Barker A, 1989, *Neural networks for data fusion*, Masters Thesis, University of Virginia, Charlottesville (VA).

[12] Banzhaf W, Nordin P, Keller RE & Francone FD, 1998, *Genetic programming: An introduction*, Morgan Kaufmann, San Francisco (CA).

[13] Beasly JE, 1996, *Advances in linear and integer programming*, Oxford University Press, New York (NY).

[14] Ben-Ameur W, 2004, *Computing the intial temperature of simulated annealing*, Computational Optimization and Applications, **29**, pp. 369–385.

[15] Bland A & Dawson GP, 1991, *Tabu search and design optimization*, IEEE Transactions on Computer Aided Design, **23**, pp. 195–201.

[16] Blickle T & Thiele L, 1995, *A comparison of selection schemes used in genetic algorithms* (2nd edition), TIK Report No. 11, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zürich.

[17] Bohachevsky IO, Johnson ME & Stein ML, 1986, *Generalized simulated annealing for function optimization*, Technometrics, **28**, pp. 209–217.

[18] Borovska P & Lazarova M, 2007, *Migration policies for island genetic models on a multicomputer platform*, IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems, Technology and Applications, pp.143–148.

[19] Bounds DG, 1987, *New optimization methods from physics and biology*, Nature **329**, pp. 215–218.

[20] Brooks DG & Verdini WA, 1988, *Computational experience with generalized simulated annealing over continuous variables*, American Journal of Mathematical and Management Sciences, **8**, pp. 425–449.

[21] Busetti F, 2003, *Simulated annealing overview*, [Online], [Cited April 2nd, 2009], Available from `http://www.geocities.com/francorbusetti/saweb.pdf`.

[22] Cerný V, 1985, *A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm*, Journal of Optimization Theory and Applications, **45**, pp. 41–51.

[23] Chelouah R & Siarry P 2000, *Tabu search applied to global optimization*, European Journal of Operational Research, **123**, pp 256–170.

[24] Coello C, 2001, *Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art*, Computer Methods in Applied Mechanics and Engineering, **191(11–12)**, pp. 1245–1287.

[25] Cohoon J, Hegde S, Martin W & Richards D, 1987, *Punctuated equilibria: A parallel genetic algorithm*, Proceedings of the Second International Conference on Genetic Algorithms, Hillsdale (NJ), pp. 148–154.

[26] Cramer N, 1985, *A representation for the adaptive generation of simple sequential programs*, Proceedings of an International Conference on Genetic Algorithms and the Applications, Carnegie-Mellon University, Pittsburgh (PA), pp. 183–187.

[27] Dreo J, Petrowski A, Siarry P & Talliard E, 2006, *Metaheuristics for hard optimization: Methods and case studies*, Springer-Verlag, Berlin.

[28] Darwin C, 1859, *The Origin of species by means of natural selection*, Mentor Reprint, New York (NY).

[29] Dubois N & De Werra D, 1993, *Epcot: An efficient procedure for coloring optimally with tabu search*, Computers and Mathematics with Applications, **25**, pp. 35–45.

[30] *Dunagan J & Vempala S*, 2008, *A polynomial-time rescaling algorithm for solving linear programs*, Mathematical Programming, **114(1)**, pp 101–114.

[31] Floudas CA & Pardalos PM, 1990, *A collection of test problems for constrained global optimization algorithms*, Springer Lecture Notes In Computer Science, **455**, pp. 58 – 80.

[32] Friedberg R, 1958, *A learning machine: Part I*, IBM Journal of Research and Development, **2**, pp. 2–13.

[33] Friedberg R, Dunham B & North J, 1959, *A learning machine: Part II*, IBM Journal of Research and Development, **3**, pp. 282–287.

[34] Fogel D, 1998, *Evolutionary computation: The fossil record*, IEEE Press, Piscataway, New Jersey (NY).

[35] General algebraic modeling system (GAMS), 2008, *GAMS' index page*, [Online], [Cited on November 10th, 2008], Available from `http://www.gams.com/`.

[36] Gendreau M, 2003, *An introduction to tabu search*, Handbook of Metaheuristics. Kluwer Academic Publishers, Boston (MA).

[37] Glover F, 1986, *Future paths for integer programming and links to artificial intelligence*, Computers and Operations Research, **13**, pp. 533–549.

[38] Glover F, 1989, *Tabu Search - Part I*, ORSA Journal on Computing, **1**, pp. 190–206.

[39] Glover F, 1990, *Tabu Search - Part II*, ORSA Journal on Computing, **2**, pp. 4–32.

[40] Glover F, 1990, *Tabu Search - A Tutorial*, Interfaces, **20**, pp. 74–94.

[41] Goldberg DE & Deb K, 1991, *A comparative analysis of selection schemes used in genetic algorithms*, Foundations of Genetic Algorithms, **1**, pp. 69–93.

[42] Hajji O, Brisset S & Brochet P, 2004, *A new tabu search method for optimization with continuous parameters*, IEEE Transactions on Magnetics, **40(2)**, pp. 1184–1187.

[43] Hansen P, 1986, *The steepest ascent mildest descent heuristic for combinatorial programming*, Congress on Numerical Methods in Combinatorial Optimization, Capri.

[44] Haverly CA, 1978, *Studies of the behavior of recursion for the pooling problem*, ACM SIGMAP Bulletin **25**, pp. 19–28.

[45] Hertz A & De Werra D, 1991, *The tabu search metaheuristic: How we used it*, Annals of Mathematics and Artificial Intelligence, **1**, pp. 111–121.

[46] Hoos HH & Stutzle T, 2005, *Stochastic local search: Foundations & applications*, Morgan Kaufmann Publishers, San Francisco (CA).

[47] Hu N, 1992, *Tabu search method with random moves for globally optimal design*, International Journal for Numerical Methods in Engineering, **35**, pp 1055–1070.

[48] Ingber, L, 1993, *Simulated Annealing: Practice versus theory*, Mathematical and Computer Modelling, **18(11)**, pp. 29–56.

[49] Johnson DS, 1990, *Local optimization and the travelling salesman problem*, Proceedings of the Automata, Languages and Programming - 17^{th} International Colloquium, Springer Verlag, New York (NY), **443**, pp. 446-461.

[50] Johnson DS, Papadimitriou CH & Yannakakis M, 1988, *How easy is local search?*, Journal of Computer and System Sciences, **37(1)**, pp. 79 – 100.

[51] Jones AEW & Forbes GW, 1995, *An adaptive simulated annealing algorithm for global optimization over continuous variables*, Journal of Global Optimization, **6**, pp. 1–37.

[52] Judson RS, Colvin ME, Meza JC, Huffer A & Gutierrez D, 1992, *Do intelligent configuration search techniques outperform random search for large molecules?*, Sandia Report SAND91-8740, Sandia National Laboratories, Center for Computational Engineering, Livermore (CA).

[53] Kalai G, 1992, *A subexponential randomized simplex algorithm*, Proceedings of the 24^{th} ACM Symposium on Theory of Computing (STOC 1992), pp. 475–482.

[54] Kallrath J, 2004, *Modeling languages and mathematical optimization*, Kluver Academic Publishers, Boston (MA).

[55] Karnopp DC, 1963, *Random search techniques for optimization problems*, Automatica, **1**, pp. 111-121.

[56] KBC Consultants, 2008, KBC Advanced Technologies, KBC House, Surrey, UK, [Personal correspondence], Contactable at tel. +44 (0) 1932 242424.

[57] Kelly JD, 2003, *Next-generation refinery scheduling technology*, Proceedings of the September, 2003 NPRA Plant Automation and Decision Support Conference, San Antonio (TX).

[58] Kirkpatrick S, Gelatt CD & Vecchi MP, 1983, *Optimization by simulated annealing*, Science **220 (4598)**, pp. 671–680.

[59] Koza J, 1992, *Genetic programming: On the programming of computers by means of natural selection*, MIT Press, Cambridge (MA).

[60] Koza J, 1994, *Genetic programming II: Automatic discovery of reusable programs*, MIT Press, Cambridge (MA).

[61] Koza J, Keane M, Streeter M, Mydlowec W, Yu J & Lanza G, 2003, *Genetic programming IV: Routine human-competitive machine intelligence*, Kluwer Academic Publishers, Boston (MA).

[62] Kuri-Morales AF & Gutíerrez-Garcia J, 2001, *Penalty functions methods for constrained optimization with genetic algorithms: A statistical analysis*, Proceedings of the 2nd Mexican International Conference on Artificial Intelligence, Heidelberg.

[63] Langdon W, 1999, *Scaling of program fitness spaces*, Evolutionary Computation, **7(4)**, pp. 399–428.

[64] Langly D & Coetzer R, 2008, *Computer experiments: Synergies between linear programming, experimental design and statistical metamodels*, Proceedings of the International Federation of Operations Research Societies (IFORS) Conference, Sandton, South Africa.

[65] Leon L, Przasnyski Z & Seal KC, 1996, *Spreadsheet and OR/MS models: An end-user perspective*, Interfaces, **26(2)**, pp. 92–104

[66] Lingo, 2008, *Lindo systems' index page*, [Online], [Cited November 10, 2008], Available from `http://www.lingo.com/`.

[67] Locatelli M, 2000, *Convergence of a simulated annealing algorithm for continuous global optimization*, Journal of Global Optimization, **18(3)**, pp.219–233.

[68] Matyas J, 1965, *Random Optimization*, Automation and Remote Control, **26**, pp. 244–251.

[69] Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH & Teller E, 1953, *Equations of state calculations by fast computing machines*, Journal of Chemical Physics, **21(6)**, pp. 1087–1092.

[70] Motwani R & Raghavan P, 1996, *Randomized algorithms*, ACM Computing Surveys, **28**, pp. 33–38.

[71] Microsoft, 2003, *Microsoft Office 2003 Excel*, [Online], [Cited November 10, 2008], Available from `http://office.microsoft.com/en--us/excel/FX100487621033.aspx`.

[72] Mühlenbein H & Schlierkamp-Voosen D, 1993, *Predictive models for the breeder genetic algorithm: Continuous parameter optimization*, Evolutionary Computation, **1(1)**, pp. 25–49.

[73] Nilsson N, 1971, *Problem-solving methods in artificial intelligence*, McGraw-Hill, New York (NY).

[74] Palacios-Gomez F, Lasdon L, Engquist M, 1982, *Nonlinear optimization by successive linear programming*, Management Science, **28(10)**, pp. 1106–1120.

[75] Parks GT, 1990, *An intelligent stochastic optimization routine for nuclear fuel cycle design*, Nuclear Technology, **89**, pp. 233–246.

[76] Pettey C, Leuze M & Grefenstette J, 1987, *A parallel genetic algorithm*, Proceedings of the Second International Conference on Genetic Algorithms and Their Applications, Hillsdale (NJ).

[77] Pegden D, Shannon RE & Sadowski RP, 1995, *Introduction to simulation using SIMAN*, 2nd Edition, McGraw-Hill, New York (NY).

[78] Pohlheim H, 1997, *Advanced techniques for the visualization of evolutionary algorithms*, Proceedings of the 42nd International Scientific Colloquium, Ilmenau, **3**, pp. 60–68.

[79] Randelman RE & Grest GS, 1986, *N-City traveling salesman problem — Optimization by simulated annealings*, Journal of Statistics and Physics, **45**, pp. 885–890.

[80] Ragsdale CT, 2007, *Spreadsheet modeling and decision analysis* 5th edition, South-Western College Publishing, Columbus (OH).

[81] Rastrigin LA, 1963, *The convergence of the random search method in the extremal control of a many-parameter system*, Automation and Remote Control, **24**, pp. 1337–1342.

[82] Reeves CR, 1993, *Modern heuristic techniques for combinatorial problems*, John Wiley & Sons, New York (NY).

[83] Romeijn HE & Smith RL, 1994, *Simulated annealing for constrained global optimization*, Journal of Global Optimization, **5**, pp. 101–126.

[84] Rosca J & Ballard D, 1999, *Rooted-tree schemata in genetic programming*, Advances in Genetic Programming, **3(11)**, pp 243–271.

[85] Roscoe Davis K & McKeown PG, 1984, *Quantitative models for management*, Kent Publishing Company, Boston (MA).

[86] Sakalli US & Birgoren B, 2008, *A spreadsheet-based decision support tool for blending problems in brass casting industry*, Computers & Industrial Engineering, **10**, pp. 1016–1028.

[87] Schuur PC, 1997, *Classification of acceptance criteria for the simulated annealing algorithm*, Mathematics of Operations Research, **22(2)**, pp. 266–275.

[88] Schwefel HP, 1995, *Evolution and optimum seeking*, John Wiley & Sons, New York (NY).

[89] Siarry P & Berthiau G, 1997, *Fitting of tabu search to optimize functions of continuous variables*, International Journal for Numerical Methods in Engineering, **40**, pp. 2449–2457.

[90] Skorin-Kapov J, 1990, *Tabu search applied to the quadratic assignment problem*, ORSA Journal on Computing, **2**, pp. 33–41.

[91] Solis FJ & Wets RBJ, 1981, *Minimization by random search techniques*, Mathematics of Operations Research, **6(1)**, pp. 19–30.

[92] Soriano P & Gendreau M, 1996, *Diversification strategies in tabu search algorithms for the maximum clique problems*, Annals of Operations Research, **63**, pp. 189–207.

[93] Spall JC, 2003, *Introduction to stochastic search and optimization: Estimation, simulation and control*, Wiley, Hoboken (NJ).

[94] Storn R & Price K, 1997, *Differential evolution — A simple and efficient heuristic for global optimizaiton over continuous spaces* Journal of Global Optimization, **11**, pp. 341–359.

[95] Toklu YC, 2005, *Aggregate blending using genetic algorithms*, Computer-Aided Civil and Infrastructure Engineering, **20**, pp. 450–460.

[96] Turing A, 1950, *Computing machinery and intelligence*, Mind, **59**, pp. 433–460.

[97] Vanderbilt D & Louie SG, 1984, *A Monte Carlo simulated annealing approach to optimization over continuous variables*, Journal of Computational Physics, **56**, pp. 259–271.

[98] Van Laarhoven PJM & Aarts EHL, 1987, *Simulated annealing: Theory and applications*, Reidel, Dordrecht.

[99] Van Rossum G, 2008, *Python Language Website*, [Online], [Cited on November 10, 2008], Available from `http://www.python.org/`.

[100] Wang M, Chen X & Qian J, 2004, *An improvement of continuous tabu search for global optimization*, Intelligent Control and Automation, **1**, pp. 375–377.

[101] Wikipedia, 2001, *Metaheuristic - Wikipedia, the free encyclopedia*, [Online], [Cited March 5th, 2009], Available from `http://en.wikipedia.org/wiki/Metaheuristic`.

[102] Wikipedia, 2001, *Octane rating - Wikipedia, the free encyclopedia*, [Online], [Cited March 5th, 2009], Available from `http://en.wikipedia.org/wiki/Research_Octane_Number`.

[103] Wikipedia, 2001, *Reid Vapour Pressure - Wikipedia, the free encyclopedia*, [Online], [Cited March 5th, 2009], Available from `http://en.wikipedia.org/wiki/Reid_Vapor_Pressure`.

[104] Wikipedia, 2001, *Tertiary amyl methyl ether - Wikipedia, the free encyclopedia*, [Online], [Cited March 5th, 2009], Available from `http://en.wikipedia.org/wiki/Tertiary_amyl_methyl_ether`.

[105] Whitely D, 1993, *A genetic algorithm tutorial*, Technical Report CS-93-103, Colorado State University, Fort Collins (CO).

[106] Whitley D, Rana S & Heckendorn R 1997, *Island model genetic algorithms and linearly separable problems*, Proceedings of AISB Workshop on Evolutionary Computation, Manchester, pp. 109–125.

Appendix A

Additional SA results

Figures A.1 to A.7, A.8 to A.14 and A.15 to A.21 contain the average results for the SSP, HPP and MMRP, respectively when the lengths of each Markov chain is set equally over the range $L = 100$ to $L = 500$ in increments of 100 iterations for initial temperatures over the range $T_0 = 0.8$ to $T_0 = 0.2$. For every initial temperature and for every Markov chain length the algorithm was run the number of times as shown in Tables A.1, A.2 and A.3 for the SSP, HPP and MMRP, respectively in order to determine its average performance. The average for each Markov chain length is compared in order to determine which length compliments which initial temperature best.

Chain	**Number of algorithm runs**						
length (L)	$T_0 = 0.8$	$T_0 = 0.7$	$T_0 = 0.6$	$T_0 = 0.5$	$T_0 = 0.4$	$T_0 = 0.3$	$T_0 = 0.2$
100	95	95	160	95	120	100	100
200	130	210	95	70	105	180	145
300	215	235	205	235	85	80	115
400	236	240	100	180	115	145	100
500	229	285	230	185	60	160	131

Table A.1: *The number of algorithm runs required to obtain a 90% confidence level for the SSP.*

Chain	**Number of algorithm runs**						
length (L)	$T_0 = 0.8$	$T_0 = 0.7$	$T_0 = 0.6$	$T_0 = 0.5$	$T_0 = 0.4$	$T_0 = 0.3$	$T_0 = 0.2$
100	2 240	7 419	7 989	6 845	7 190	9 315	9 661
200	2 460	6 165	1 490	6 160	6 300	9 715	9 510
300	4 740	3 965	4 300	3 305	2 930	6 040	4 765
400	2 705	1 585	1 089	4 220	6 015	2 602	2 820
500	2 009	2 168	1 250	4 428	3 650	9 480	8 040

Table A.2: *The number of algorithm runs required to obtain a 90% confidence level for the HPP.*

Chain length (L)	Number of algorithm runs						
	$T_0 = 0.8$	$T_0 = 0.7$	$T_0 = 0.6$	$T_0 = 0.5$	$T_0 = 0.4$	$T_0 = 0.3$	$T_0 = 0.2$
100	1 639	400	5 126	5 005	4 686	6 676	3 296
200	2 335	2 566	2 013	3 781	6 942	4 264	9 31
300	2 260	1 784	1 637	2 427	2 416	1 668	3 076
400	518	4 015	2 229	3 162	1 065	4 671	1 426
500	2 830	1 911	3 007	1 259	2 282	3 298	4 371

Table A.3: *The number of algorithm runs required to obtain a 90% confidence level for the MMRP.*

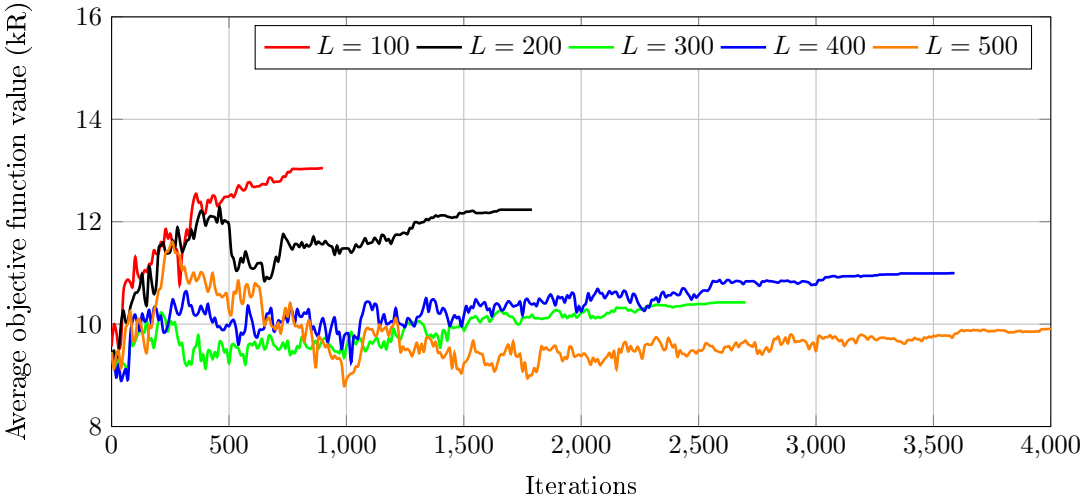

Figure A.1: *The average objective function values for $T_0 = 0.8$ in combination with a Markov chain length of L for the SSP.*

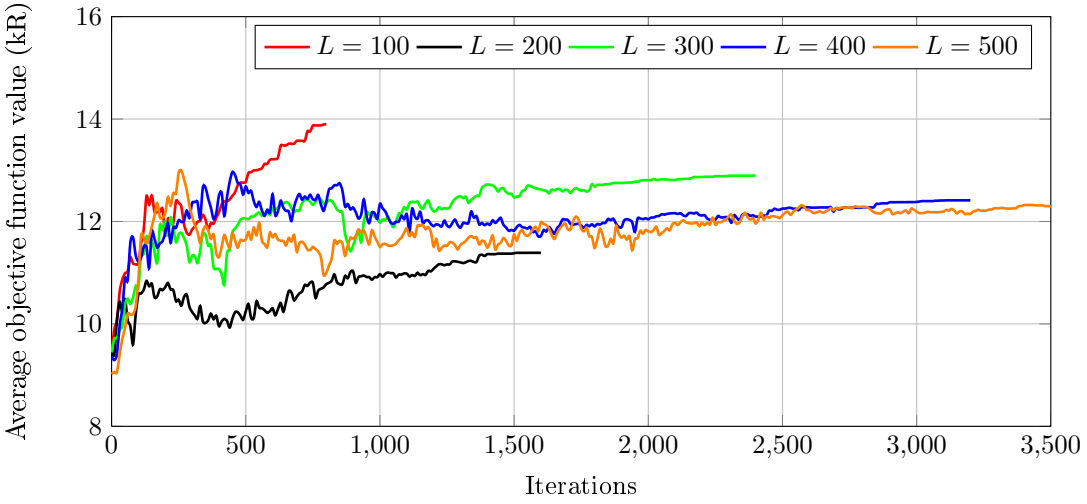

Figure A.2: *The average objective function values for $T_0 = 0.7$ in combination with a Markov chain length of L for the SSP.*

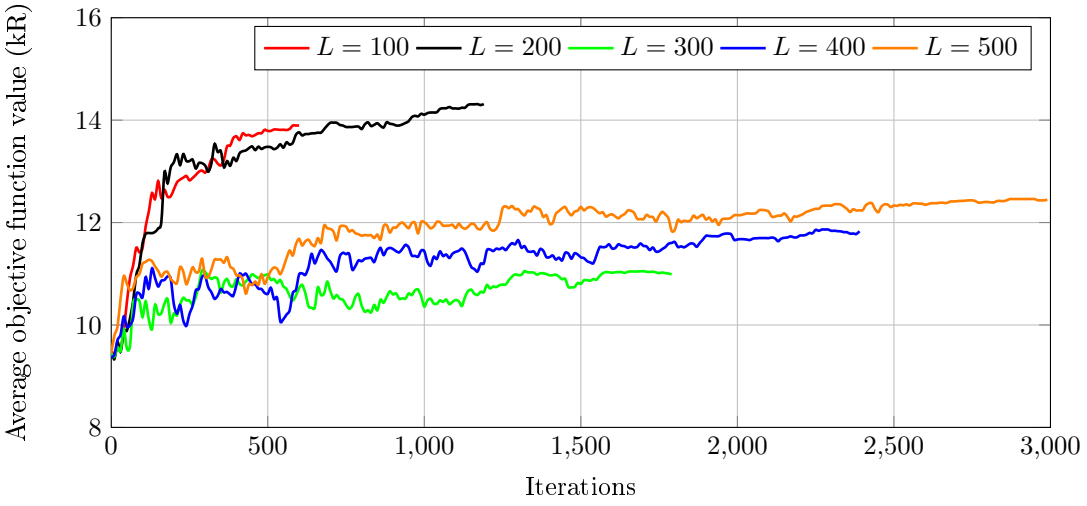

Figure A.3: *The average objective function values for $T_0 = 0.6$ in combination with a Markov chain length of L for the SSP.*

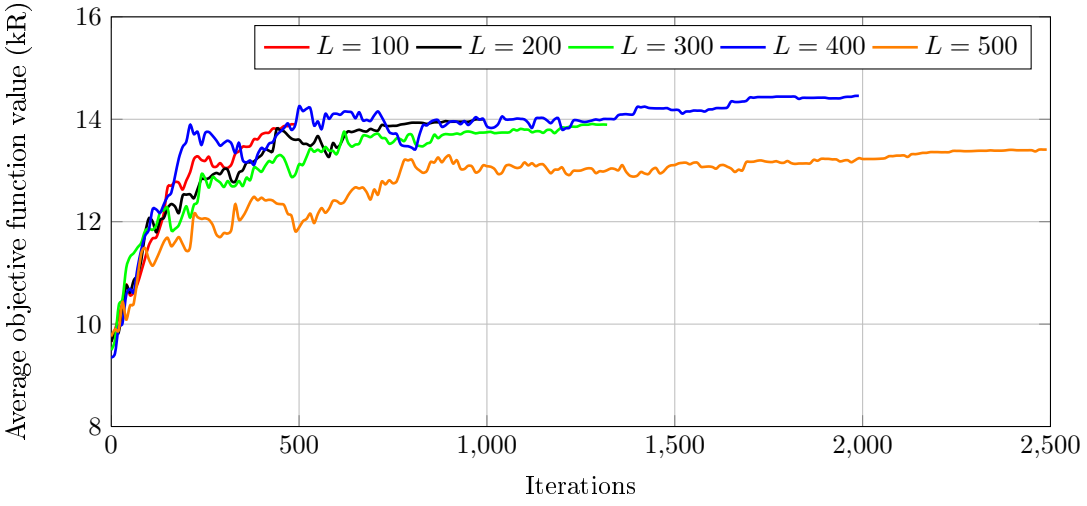

Figure A.4: *The average objective function values for $T_0 = 0.5$ in combination with a Markov chain length of L for the SSP.*

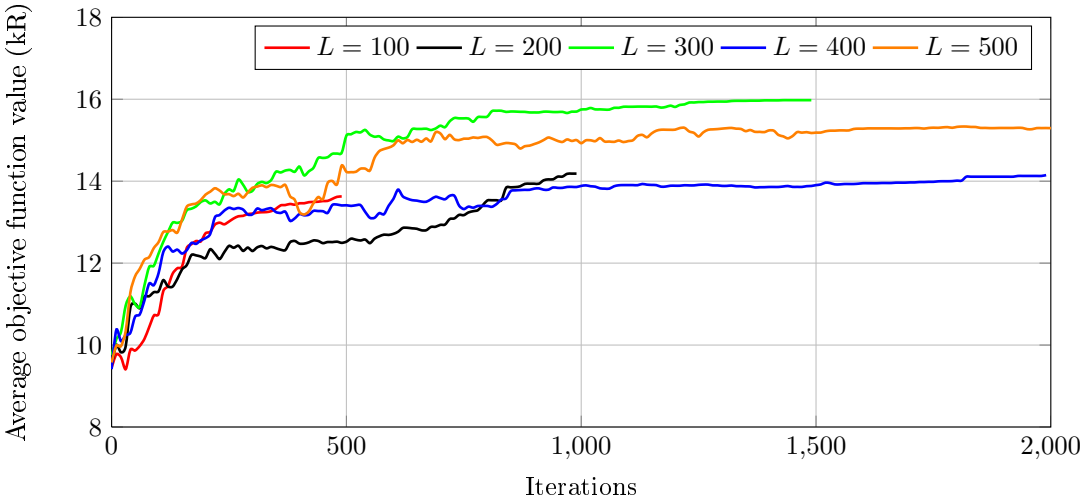

Figure A.5: *The average objective function values for $T_0 = 0.4$ in combination with a Markov chain length of L for the SSP.*

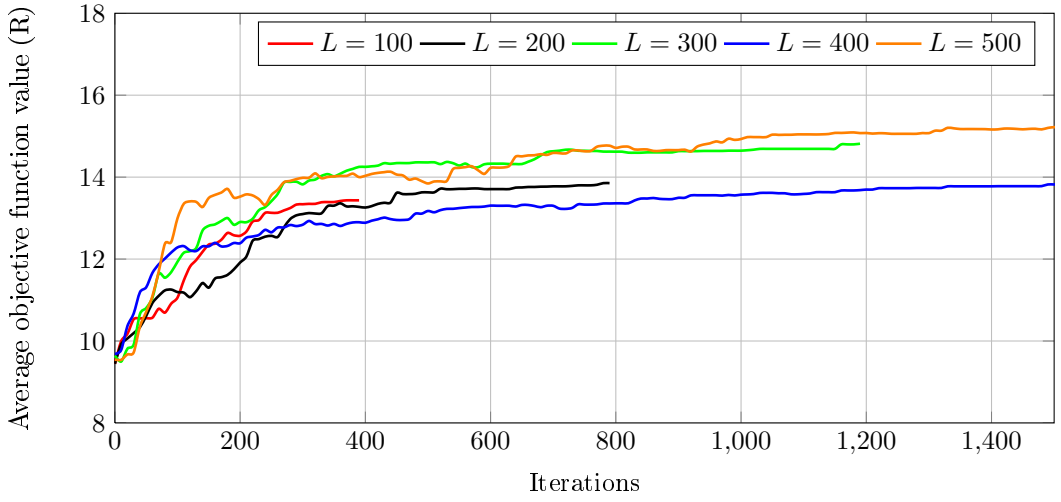

Figure A.6: *The average objective function values for $T_0 = 0.3$ in combination with a Markov chain length of L for the SSP.*

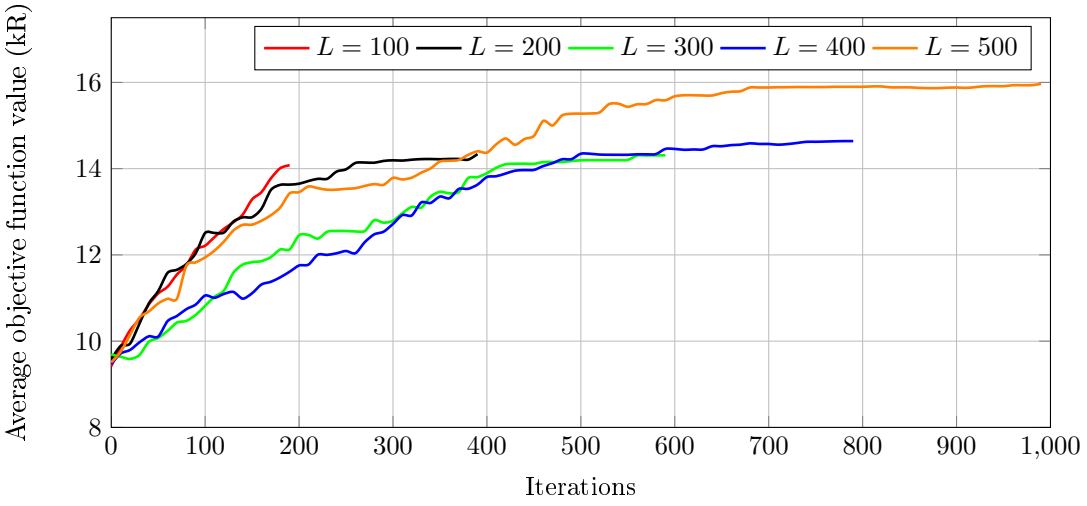

Figure A.7: *The average objective function values for $T_0 = 0.2$ in combination with a Markov chain length of L for the SSP.*

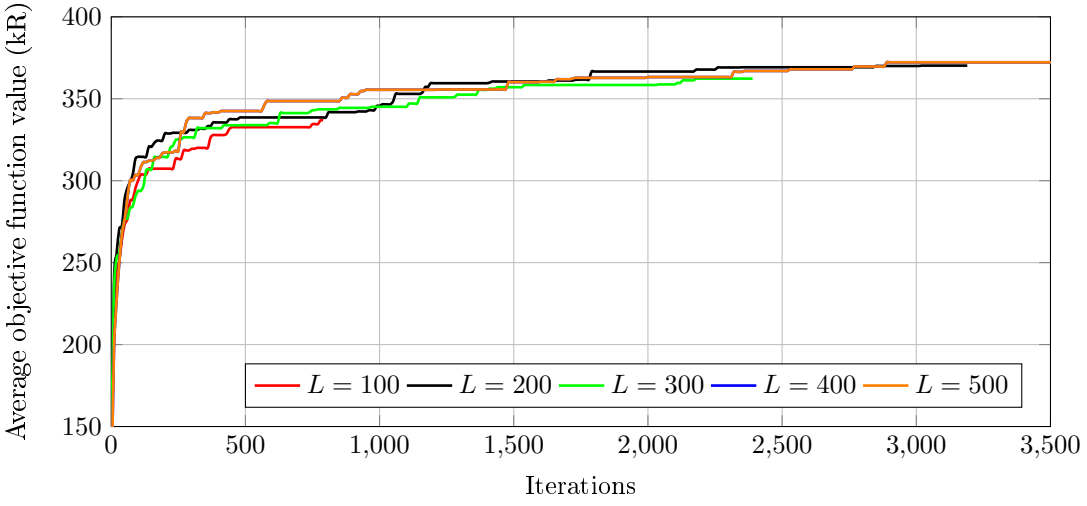

Figure A.8: *The average objective function values for $T_0 = 0.8$ in combination with a Markov chain length of L for the HPP.*

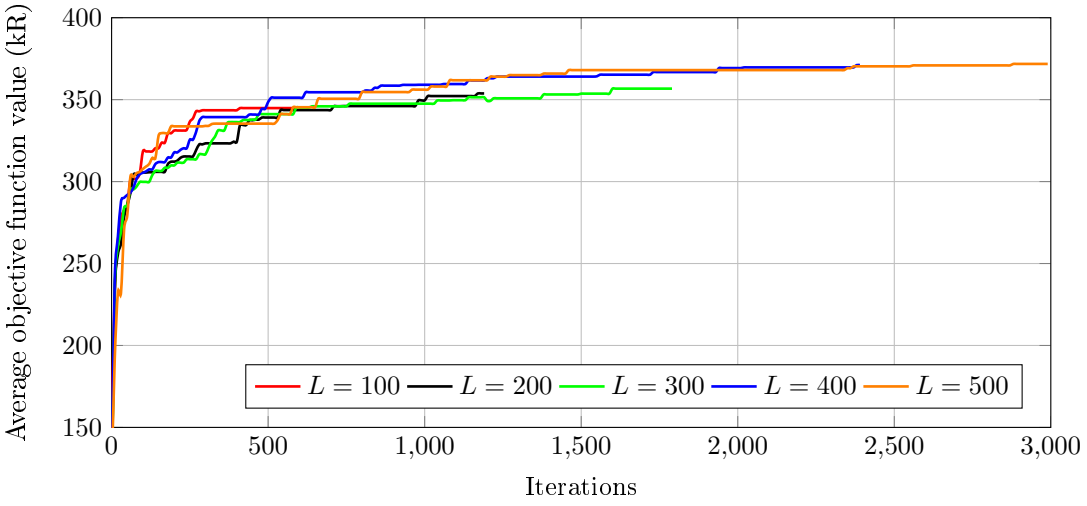

Figure A.9: *The average objective function values for $T_0 = 0.7$ in combination with a Markov chain length of L for the HPP.*

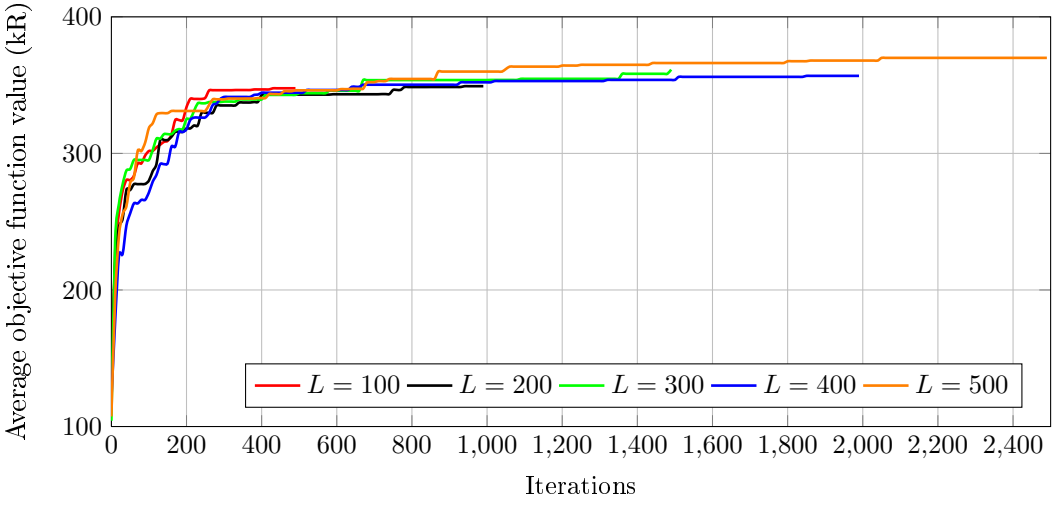

Figure A.10: *The average objective function values for $T_0 = 0.6$ in combination with a Markov chain length L for the HPP.*

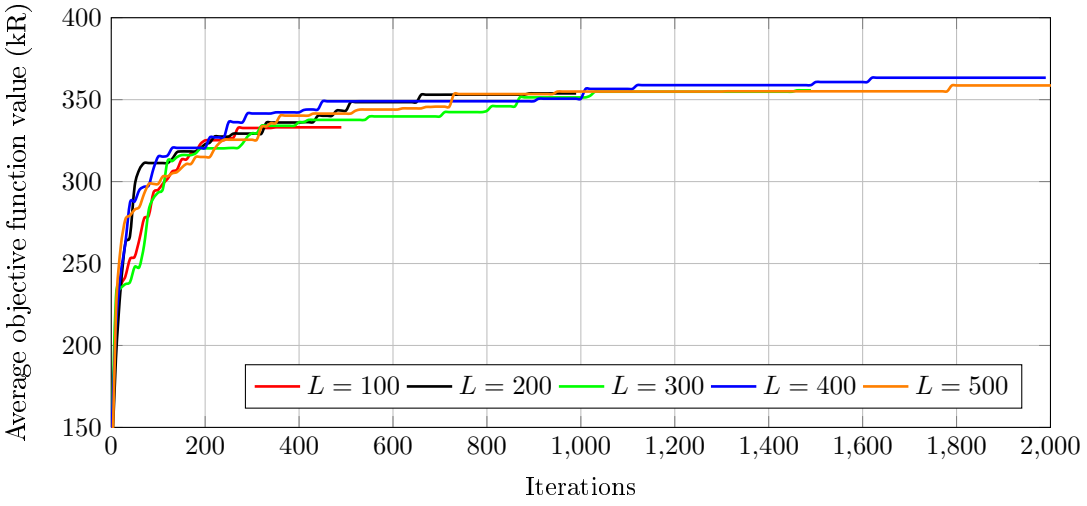

Figure A.11: *The average objective function values for $T_0 = 0.5$ in combination with a Markov chain length L for the HPP.*

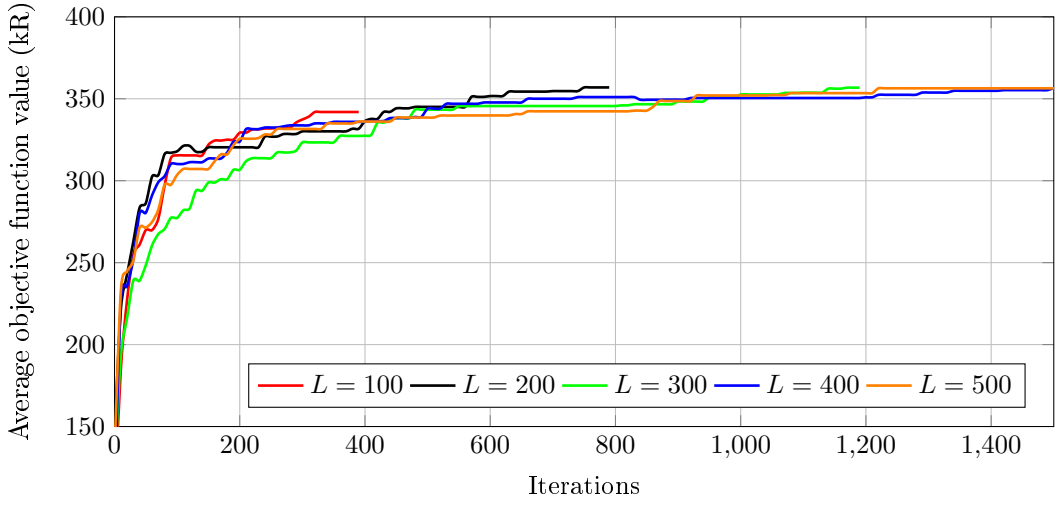

Figure A.12: *The average objective function values for $T_0 = 0.4$ in combination with a Markov chain length L for the HPP.*

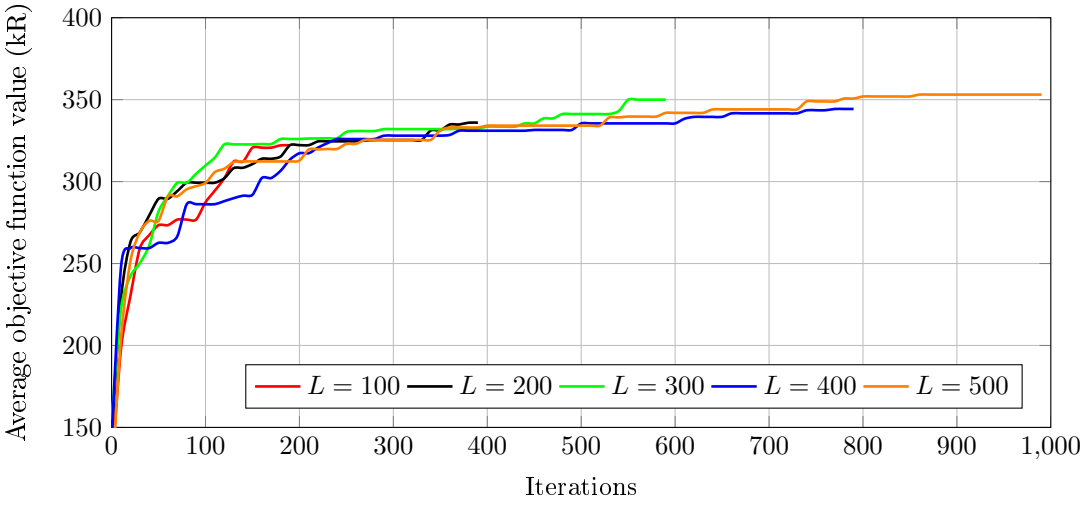

Figure A.13: *The average objective function values for $T_0 = 0.3$ in combination with a Markov chain length L for the HPP.*

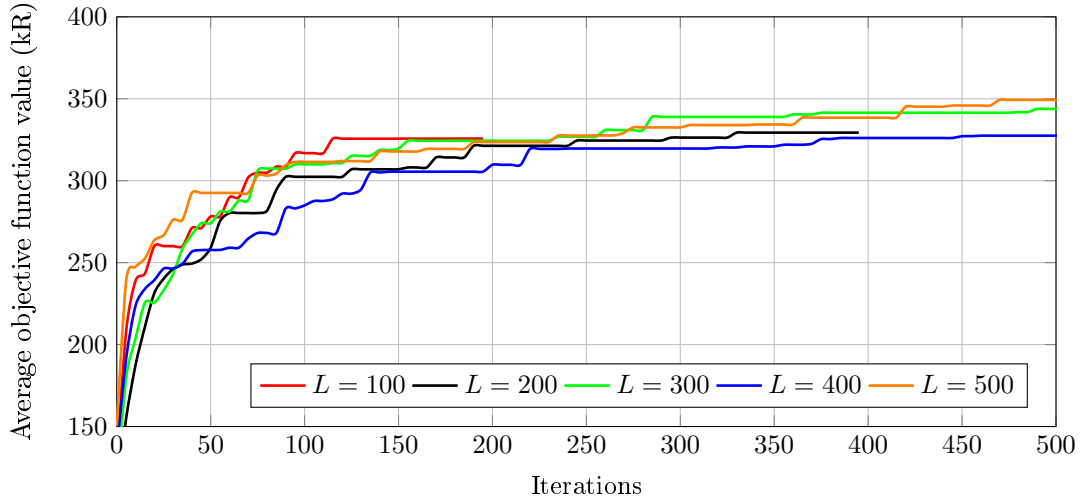

Figure A.14: *The average objective function values for $T_0 = 0.2$ in combination with a Markov chain length L for the HPP.*

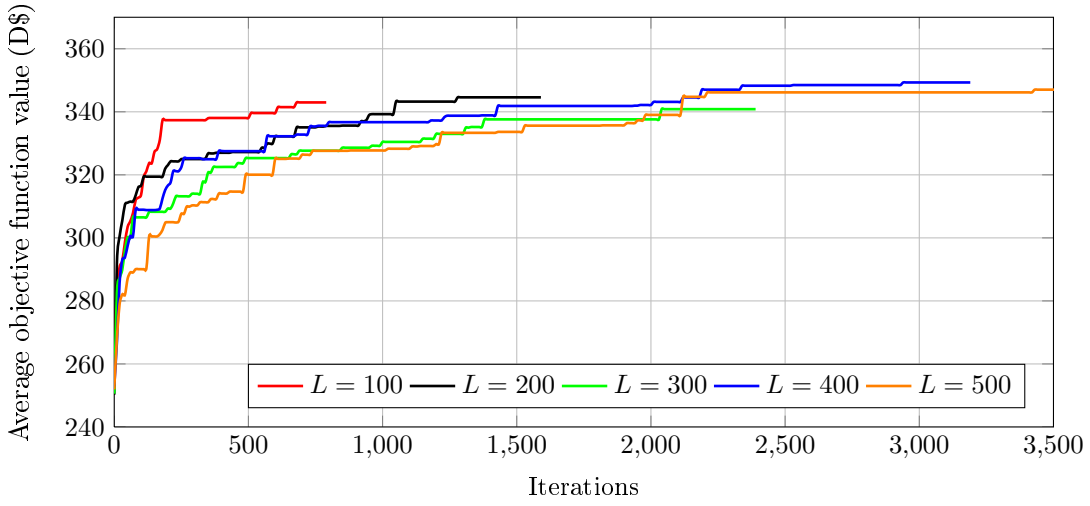

Figure A.15: *The average objective function values for $T_0 = 0.8$ in combination with a Markov chain length L for the MMRP.*

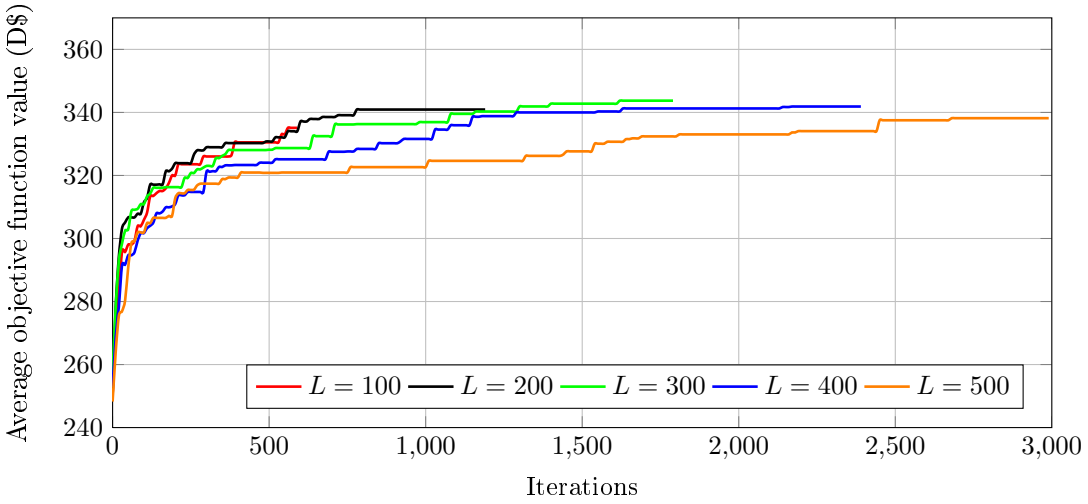

Figure A.16: *The average objective function values for $T_0 = 0.7$ in combination with a Markov chain length L for the MMRP.*

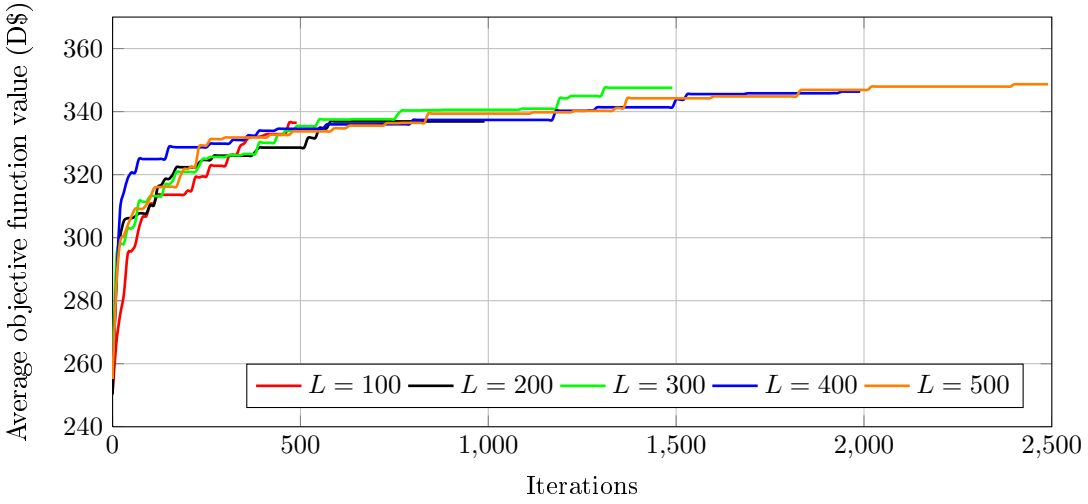

Figure A.17: *The average objective function values for $T_0 = 0.6$ in combination with a Markov chain length L for the MMRP.*

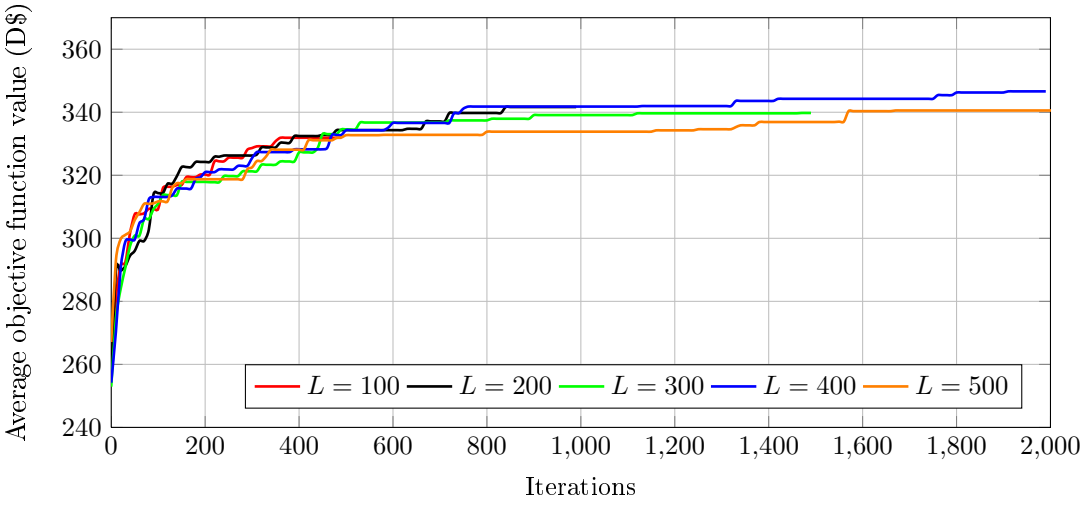

Figure A.18: *The average objective function values for $T_0 = 0.5$ in combination with a Markov chain length L for the MMRP.*

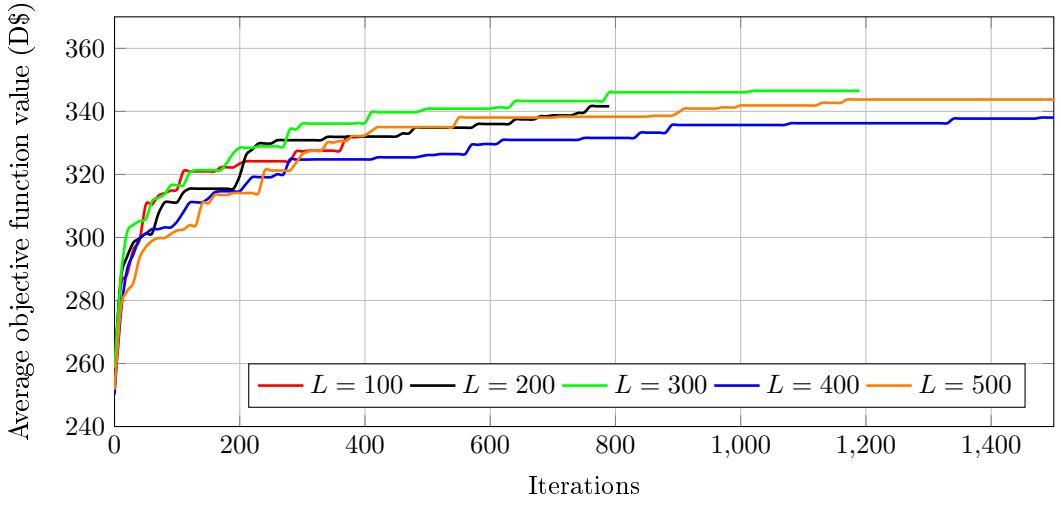

Figure A.19: *The average objective function values for $T_0 = 0.4$ in combination with a Markov chain length L for the MMRP.*

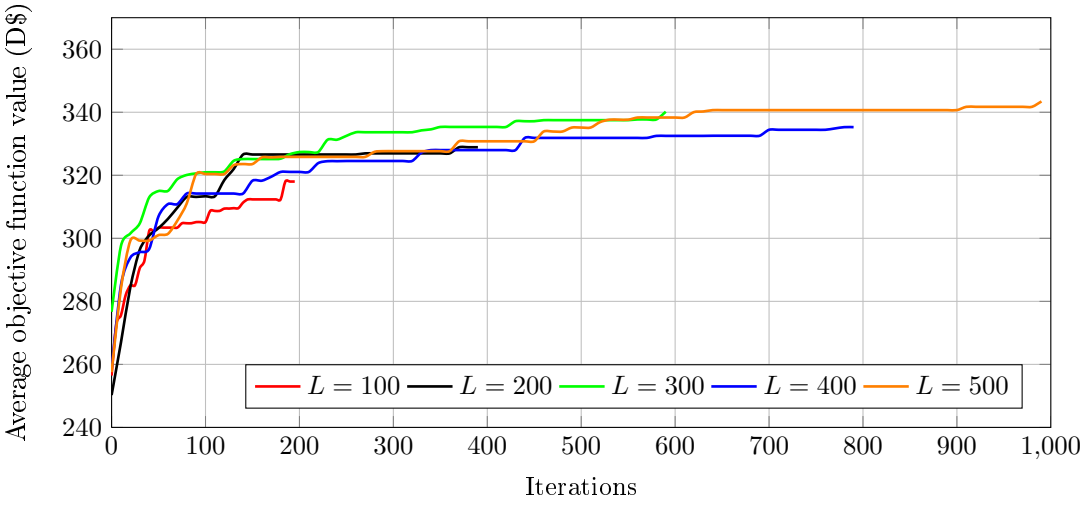

Figure A.20: *The average objective function values for $T_0 = 0.3$ in combination with a Markov chain length L for the MMRP.*

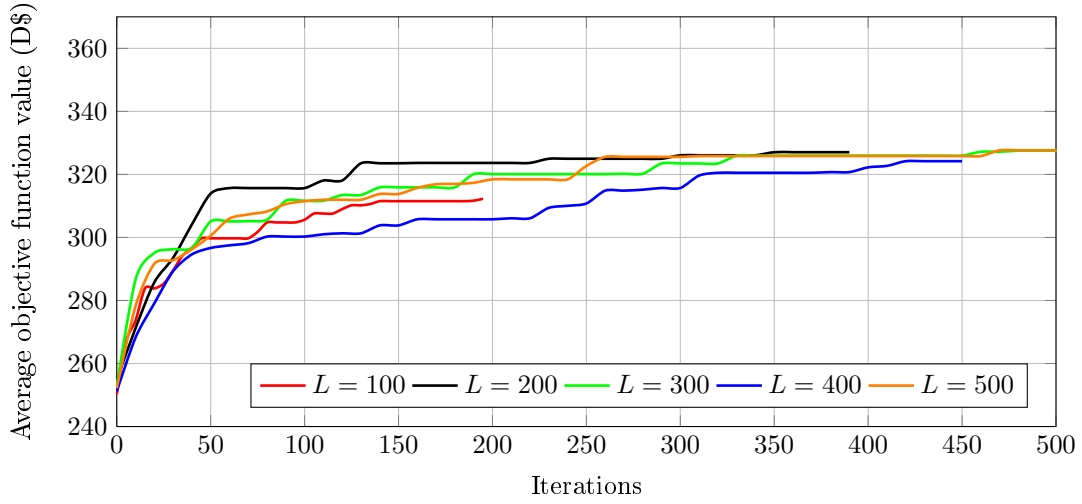

Figure A.21: *The average objective function values for $T_0 = 0.2$ in combination with a Markov chain length for the MMRP.*