Volume 32 (2), pp. 79–104
http://orion.journals.ac.za

ORiON
ISSN 0259–191X (print)
ISSN 2224–0004 (online)

On the solution of petrochemical blending problems with classical metaheuristics

L Venter* SE Visagie†

Received: 10 June 2014; Revised: 26 September 2014; Accepted: 12 March 2015

Abstract

In this paper a comparison of classical metaheuristic techniques over different sizes of petrochemical blending problems is presented. Three problems are taken from the literature and used for initial comparisons and parameter setting. A fourth instance of real world size is then introduced and the best performing algorithm of each type is then applied to it. Random search techniques, such as blind random search and local random search, deliver fair results for the smaller instances. Within the class of genetic algorithms the best results for all three problems were obtained using ranked fitness assignment with tournament selection. Good results are also obtained by means of continuous tabu search approaches. A simulated annealing approach also yielded fair results. Comparisons of the results for the different approaches shows that the tabu search technique delivers the best results with respect to solution quality and execution time for all of the three smaller problems under consideration. However, simulated annealing delivers the best result with respect to solution quality and execution time for the introduced real world size problem.

Key words: Petrochemical blending problems, random search, genetic algorithm, continuous tabu search, simulated annealing.

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. The decision maker must determine the ingredients to use as well as the quantities thereof.

*Department of Logistics, University of Stellenbosch, South Africa
†Corresponding author: Department of Logistics, University of Stellenbosch, South Africa, email: svisagie@sun.ac.za

http://dx.doi.org/10.5784/32-2-520

In general there are an infinite number of blending recipes which will make a product, but there is usually 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. The aim with blend planning methods is to find optimal operating conditions and to identify 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.

In refinery and petrochemical processing problems it is generally necessary to model product flows in addition to the properties of the components. When components are combined, nonlinear relationships are often introduced. This results in a problem where the qualities of the components contribute to the qualities of the products in a nonlinear and nonconvex manner. Although successive linear programming techniques [3] and approximation programming [18] may be used to address this problem, simulation and spreadsheets are often used in industry [12]. On the other hand, metaheuristics are useful alternative methods to search for alternative efficient recipes.

The objective of this study is to present a comparison of the performance of the different types of classical metaheuristics when applied to typical petrochemical blending problems. To the best of the authors' knowledge, there exists no application or comparison of any metaheuristic approaches to petrochemical blending problems in the academic literature. The comparison of these metaheuristics was chosen because a petrochemical company wanted to know how classical metaheuristic approaches compare with each other and their current methods. Unfortunately, the company is unwilling to disclose the working or performance of their current method.

The remainder of the paper is structured as follows. The three sample problems from literature are discussed in §2. In the following four sections, random search algorithms (§3), genetic algorithms (§4), tabu search algorithms (§5) and simulated annealing algorithms (§6) are introduced, and their respective solution results presented and discussed. A summary of the main results is supplied in §7. A new real world size instance is presented in §8. The best performing metaheuristics from the previous sections are then applied to this instance and the results discussed. The paper is concluded in §9.

2 Sample problems

Three known instances were used for the development, testing and comparison of the metaheuristic approaches. For fitness comparison purposes, all three the small instances were modelled and solved to optimality [26].

2.1 The simplified sample problem

The first problem is the *simplified sample problem* (SSP) that was introduced by KBC Consultants. It considers two petrol blends, namely ULP 93 (Summer Grade), also known as M3S (a 93 octane unleaded grade), and ULP 95 (Summer Grade), also known as M5S (a 95 octane unleaded grade). 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 octane rating, vapor pressure and TAME specifications.

Generally octane blending may be a nonlinear problem, but it is assumed in this problem that octane blends linearly by volume: The sum product of the *Research Octane Number* (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. *Reid vapor pressure* (RVP) does not blend linearly on volume. Therefore, it needs to be converted into a *Reid Vapor Index* (RVI) where $\text{RVI} = \text{RVP}^{1.25}$. TAME is high in octane and low in 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.

The properties of each of the five components and two products may be sourced from Venter & Visagie [27]. Each of the components (except butane) 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 determine an optimal blend recipe that satisfies the inventory and blend specification constraints. A schematic representation of the SSP is supplied in Figure 1.

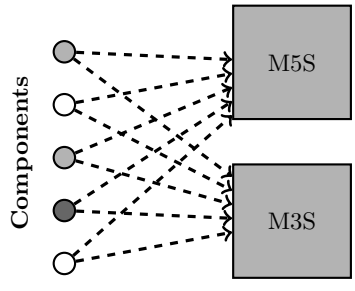


Figure 1: *A schematic representation of the SSP.*

2.2 The Haverly pooling problem

The *Haverly pooling problem* (HPP) is similar to the SSP and is presented in Haverly [7]. 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.

The specifications for the component and product characteristics may be sourced from Venter & Visagie [27]. From the given information, the goal is to determine 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.

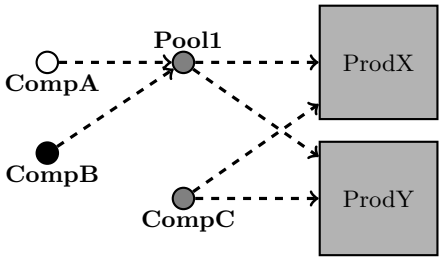


Figure 2: *A schematic representation of the HPP.*

2.3 The Marco mini-refinery problem

The *Marco mini-refinery problem* (MMRP) [4] 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 US$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 may be sourced from Venter & Visagie [27].

The components are obtained through various chemical processes. Five processes play a role in this problem: *Atmospheric distillation*, *naptha reforming*, *catalytic cracking of distillates*, *catalytic cracking of gas oil* and the *hydrotreating of residuum*. Butane is a domestic product and is manufactured rather than obtained from crudes. All crudes initially pass through atmospheric distillation. 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. Specific attributes and values for the processes, components, crudes and products may be sourced from Venter & Visagie [27]. The objective is to maximise the profit subject to all the above constraints. A schematic representation of the MMRP is given in Figure 3.

3 Random search techniques

Random search is a simple search technique which sample candidate solutions around the current solution. Usually candidate solutions are sampled from a hypersphere centred around the current solution.

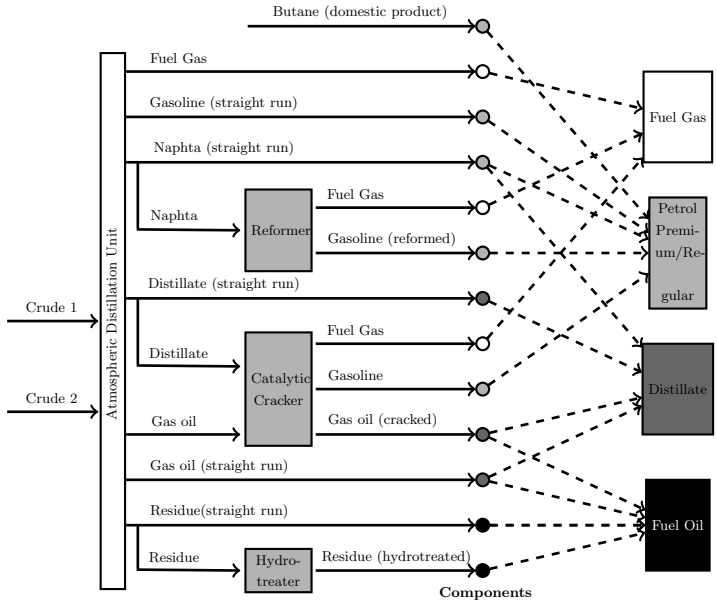


Figure 3: *A schematic representation of the MMRP.*

3.1 Blind random search

Blind random search (BRS) is the simplest random search method because the choice of candidate solution does not take into account any characteristics of previously considered solutions. That is, the 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 an optimum. The approach can also be implemented in recursive form. Algorithm 1 gives a pseudocode listing of the BRS as presented by Spall [23]. Spall [23] 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.

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.

3.2 Local random search

Methods of local search received attention in both theoretical computer science and numerical optimisation. Johnson *et al.* [10] observed that one of the few general approaches to difficult combinatorial optimisation problems that has achieved empirical success is *local* (or *neighbourhood*) search. For example, local search methods have proven very successful for the celebrated travelling salesperson problem [9].

Solis & Wets [22] propose several local random 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 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 [19] suggests that a vector of zero mean normal variates, $\boldsymbol{d}^k$, be added to the k-th solution, and that it should be generated according to

$$\boldsymbol{d}_i^{k+1} = \boldsymbol{d}_i^k + \boldsymbol{D}^k \boldsymbol{\varpi}, \tag{1}$$

where $\boldsymbol{\varpi}$ is a vector of uniform random numbers in the range $(-1, 1)$ and $\boldsymbol{D}^k$ 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) $\boldsymbol{D}^k$ is updated, such that

$$\boldsymbol{D}^{k+1} = (1 - \alpha)\boldsymbol{D}^k + \alpha\omega\boldsymbol{\Upsilon}^k, \tag{2}$$

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

$$\Upsilon_{ii}^k = \|D_{ii}^k \varpi_i\|, \tag{3}$$

where D_{ii}^k is the element in the i-th row and i-th column of $\boldsymbol{D}$ during the k-th iteration and Υ_{ii}^k carries the same meaning for $\boldsymbol{\Upsilon}$. 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 [16].

3.3 Computational results

Throughout the paper, computational results were obtained from code written in Python 2.5 [25] executed on an Intel Core2 Duo processor running at 3.00Ghz with 2GB of

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 $\boldsymbol{d}^k$ 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 $\boldsymbol{d}^k$ 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 $\boldsymbol{d}^k$ 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.

RAM. Unless stated otherwise, all random numbers are generated according to Python's standard random number generator using a uniform distribution.

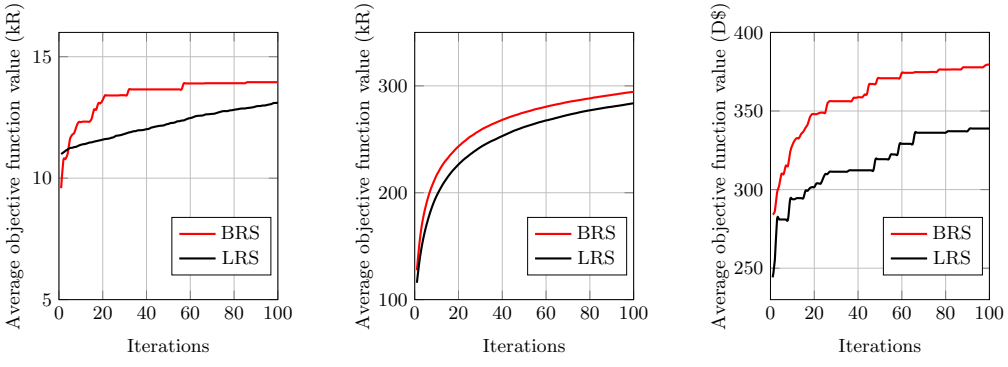
The standard deviation of results was given throughout to indicate the size of the variance between individual runs which made up the average results. Throughout, the number of execution runs was determined by means of the half-width confidence interval oulined in Pegden *et al.* [20] with a pilot number of 10 runs and desired confidence level of 90%. The number of 10 was a rule of thumb, but the t-distribution on which the half-width confidence interval method was based was useful for observations less than 30. The use of 10 runs compromised between execution time and the requirements of statistical procedures. If the observations were not normally distributed, the results for the confidence interval held for 10 replications or more [20]. A confidence level of 90% was 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 was 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.

The results for the BRS algorithm and the LRS algorithm as applied to the SSP are shown in Figure 4(a). Because the LRS algorithm was 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 was possible to have approximately 500 iterations of the LRS execute in the same time as only 100 iterations of the BRS. To make the results comparable, however, the stopping criterion for both algorithms were set at 100 iterations. The comparison of the respective performances of the two approaches are shown in Figure 4(a). Despite the much quicker execution time of the LRS, the BRS was preferred because of the better average result it achieved. Table 1 contains a summary of the results obtained after application of the RS approaches for the SSP.

The results for the BRS algorithm and the LRS algorithm are shown in Figure 4(b). The

results did not imply that one technique was better than the other by a large margin, yet the BRS technique was preferred due to the better average result it achieved as well as its shorter execution time. Table 1 contains a summary of the results obtained after application of the RS approaches for the HPP.

The results for the BRS algorithm and the LRS algorithm are shown in Figure 4(c). Even though the LRS technique was preferred for the MMRP when one looks at the execution time, on average it produced a weaker solution than the BRS technique. This is because the specific combination of constraints for the problem made it easier to find a feasible solution by generating random recipe matrices than finding a feasible solution by adjusting existing recipes. Table 1 contains a summary of the results obtained after application of the RS approaches for the MMRP.



(a) Results for the SSP. (b) Results for the HPP. (c) Results for the MMRP.

Figure 4: *The average objective function values obtained by means of BRS and LRS for the SSP, HPP and MMRP. The units are in thousands of South African rands (kR) and tens of US$ (D$). [Figure can be viewed in colour in the electronic version, available at* `http://orion.journals.ac.za`*.]*

Result	SSP (kR)		HPP (kR)		MMRP (D$)	
	BRS	**LRS**	**BRS**	**LRS**	**BRS**	**LRS**
Best objective function value	15.01	17.26	364.33	310.78	413.50	383.65
Average objective function value	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 1: *Results summary for the random search techniques for the SSP, HPP and MMRP. The units are in thousands of South African rands (kR) and tens of US$ (D$).*

4 Genetic algorithm approaches

Genetic algorithms (GAs) are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology, such as inheritance, mutation, selection and recombination.

4.1 Genetic algorithms for blending problems

Toklu [24] 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 means of this technique. It is shown to be quite versatile when applied to multiple objectives, including cost minimisation and approaching a given target curve. A pseudocode listing for a 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 of 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.

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. The solution structure for the HPP and MMRP is handled similarly.

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 are proportional fitness assignment and rank-base fitness assignment. 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 [1]. *Tournament selection*, as recommended by Goldberg & Deb [5], was also implemented and tested.

The recombination and mutation operators modify the chromosome containing the daily blend production amounts only. The recombination operator 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 [17]. 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.

Mutation occurs by means of a repair operator, as it generates the chromosome so that the total of the day's blend production is produced by the blend earning the highest revenue only. This method does not lose the feasibility of the solution, as it does not require a blend with daily production amounts higher than those set by the inventory constraints.

4.2 Computational results

The methods used for fitness assignment and selection in all the genetic algorithms are presented in Table 2. GA1 to GA4 were applied to the SSP, GA5 to GA8 were applied to the HPP, while GA9 to GA12 were applied to the MMRP. The results summary for all the GA approaches applied to the three problems is shown in Table 3. The following parameter settings were used for all the GA approaches as they achieved the best results. The population size and the number of generations, were set to 100 (*i.e.*, the stopping criterion was set at 100 generations). The elitism proportion was set to 0.1 of the population size, the recombination probability was set to 0.6 and the mutation probability was set to 0.1.

	Fitness assignment		**Selection method**	
	Proportional	**Ranked**	**Roulette wheel**	**Tournament**
GA1 & GA5 & GA9	x		x	
GA2 & GA6 & GA10		x	x	
GA3 & GA7 & GA11		x		x
GA4 & GA8 & GA12	x			x

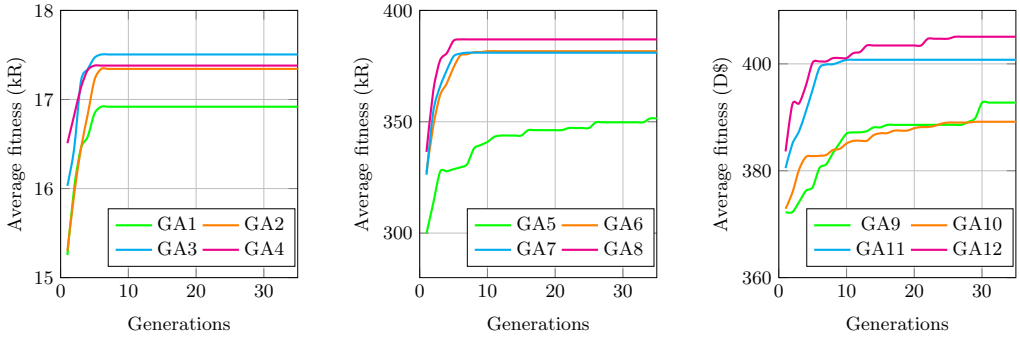
Table 2: *Combination of fitness assignment and selection methods use in GA1 to GA12 for the SSP, HPP and MMRP.*

For the SSP, the results in Figure 5(a) show that the problem was best solved when using tournament selection and ranked assignment. This was in accordance with what was expected from literature. The same best result was obtained by GA1 to GA4 during all the algorithms runs. The average execution times of GA1 to GA4 did not differ significantly, but GA4 was found to have a notably quicker execution time than GA3 without delivering an average result that was much poorer than that of GA3.

For the HPP, the results in Figure 5(b) show that the problem is best solved when using tournament selection and proportional fitness assignment. GA5 and GA8 achieved approximately the same best result during all algorithm runs. The implementation of roulette wheel selection caused GA5 and GA6 to have the longest average execution time while GA7 and GA8 added short execution time results to their already favourable best and average solution quality results.

For the MMRP, the results in Figure 5(c) show that the problem is best solved when using tournament selection and proportional fitness assignment. GA12 achieved a best solution that was closest to the known optimal for the MMRP. The use of tournament selection had the greatest positive effect on the performanance of the GA approaches. The best

performing approach, (*i.e.* GA12) had a fair average execution time associated with it, while GA10 had a good average execution time associated with it without delivering an average result that was much poorer than that of GA12.



(a) Results for SSP. (b) Results for HPP. (c) Results for the MMRP

Figure 5: *Average fitness results of GA1 to GA8 obtained for the SSP, HPP and MMRP. The units are in thousands of South African rands (kR) and tens of US$ (D$). [Figure can be viewed in colour in the electronic version, available at* `http://orion.journals.ac.za`*.]*

	Algorithm	Best fitness	Average fitness	Standard deviation	Average execution time (sec)	Number of runs
SSP (kR)	GA1	18.52	16.91	0.38	60.14	7
	GA2	18.52	17.34	0.71	68.05	9
	GA3	18.52	17.51	0.52	69.75	8
	GA4	18.52	17.38	0.56	61.59	7
HPP (kR)	GA5	390.64	343.48	30.71	0.81	4325
	GA6	391.62	356.28	16.26	0.91	1195
	GA7	390.63	373.85	7.48	0.32	325
	GA8	398.54	386.77	6.80	0.30	905
MMRP (D$)	GA9	409.30	392.75	10.01	176.36	4845
	GA10	411.45	389.17	17.39	167.70	2660
	GA11	411.85	400.77	11.93	185.27	766
	GA12	413.45	403.91	9.17	171.73	1015

Table 3: *Results obtained after execution of GA1 to GA12, respectively for the SSP, HPP and MMRP as well as the number of algorithm runs required to obtain a confidence interval of 90%. The units for the fitness are in thousands of South African rands (kR) for SSP and HHPP, and in tens of US$ (D$) for the MMRP.*

5 Tabu search approaches

Tabu search (TS) 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. Relatively little attention has been paid to apply TS algorithms to continuous optimisation problems such as the problem considered here. Two approaches of the TS algorithm for continuous problems are presented.

5.1 The hypersquare method

The work of Wang *et al.* [28] provides a 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 & Siarry [2]. 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. In all the problems considered here a solution $\boldsymbol{X}$ is a $J \times I$ matrix containing the blend recipe of components to products. The element x_{ji} of $\boldsymbol{X}$ contains the amount (in m^3) of component j blended into product i. Let $\boldsymbol{L}$ and $\boldsymbol{U}$ be the matrices containing the lower and upper bounds on the blend recipe. Define the hyperrectange $H(\boldsymbol{X}, \boldsymbol{H})$, centred around a solution $\boldsymbol{X}$, with radius $\boldsymbol{H}$ to be

$$H(\boldsymbol{X}, \boldsymbol{H}) = \{\boldsymbol{X}' \big| \ |x'_{ji} - x_{ji}| < h_{ji}, l_{ji} < x'_{ji} < u_{ji}\}. \tag{4}$$

The set of concentric hyperrectangles may then be defined as

$$H^z(\boldsymbol{X}, \boldsymbol{H}^{(z-1)}, \boldsymbol{H}^z) = \{\boldsymbol{X}' \big| h_{ji}^{(z-1)} \leq |x'_{ji} - x_{ji}| < h_{ji}^z, l_{ji} < x'_{ji} < u_{ji}\}, \tag{5}$$

for $z = 1, 2, \ldots, z_\tau$, and

$$H^0(\boldsymbol{X}, \boldsymbol{H}^0) = \{\boldsymbol{X}' \big| |x'_{ji} - c_{ji}| < h_{ji}^0, l_{ji} < c'_{ji} < u_{ji}\}, \tag{6}$$

where the radius $\boldsymbol{H}^0$ is an independent parameter. Figure 6 contains a partitioning of a solution space for $z_\tau = 3$.

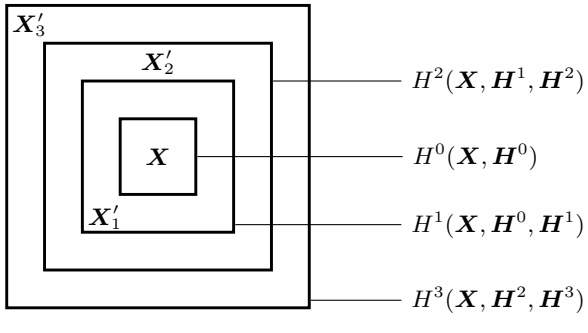


Figure 6: *An example of the partitioning of a solution space in two dimensions with* $z_\tau = 3$.

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

The tabu list 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 $\boldsymbol{X}$ is tabu, two tabu conditions are applied. The first condition is applied to the objective function, while the second considers the solution itself. The first condition predetermine the applicability of the second one. The first condition determines whether $f(\boldsymbol{X})$ is within a certain tolerance of any objective function value in the tabu list. If $f(\boldsymbol{X})$ is within this tolerance, the second condition is applied, else it is not tabu. The second check is done for all the x_{ij} of $\boldsymbol{X}$ and x^b_{ij} of $\boldsymbol{X}^b$. If all values in x_{ij} are within a specified tolerance of x'_{ij}, solution $\boldsymbol{X}$ is considered to be tabu.

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. If $f(\boldsymbol{X}) > f(\boldsymbol{X}^*)$, the aspiration level is satisfied. A pseudocode listing for the CTSh appears in Algorithm 4.

Algorithm 4: `Hypersquare algorithm (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 $k \leftarrow 1$.
3: **while** *found* = `false` **do**
4: Set $\boldsymbol{X}'$ to the k-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: $k = k + 1$
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 $\boldsymbol{X}^*$

5.2 The immediate zone method

Hajji *et al.* [6] propose a TS method based on the work of Hu [8]. The algorithm presented here is based on their approach and is refered to as the *immediate zone method* for the

continuous tabu search (CTSz). It uses a tabu list that contains points in the solution space, and a prohibited zone around each or these points that depends on the value of its objective function value. This prohibited zone decreases in size as the number of iterations increases. Alternation of intensification and diversification phases allows the finding of the global optimum with a good accuracy.

It is assumed that no information on the location of an 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 [8]. Let the solution $\boldsymbol{X}$ be a matrix with elements x_{ji}, containing the blend recipe — *i.e.* the amount (in m^3) of component j blended into product i. The probability density is defined as

$$p(x_{ji}) = \frac{1}{\sigma_{ji}\sqrt{2\pi}} \exp\left(-\frac{(x_{ji} - x^*_{ji})^2}{2\sigma^2_{ji}}\right) \text{ for } i = 1, 2, \ldots I, \quad j = 1, 2, \ldots J, \tag{7}$$

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

$$x'_{ji} = c^*_{ji} + \sigma_{ji} P^{-1}(r), \qquad \text{for } i = 1, 2, \ldots I, \quad j = 1, 2, \ldots J, \tag{8}$$

with

$$P(x) = \int_{-\infty}^{x} p(u)\ \mathrm{d}u, \qquad \text{for } 0 \leq r \leq 1. \tag{9}$$

The tabu list contains all tabu regions. These regions are hyperrectangles that are defined by their centers and sizes. Every solution generated has a tabu region associated with it. Its center is the solution it is associated with. It is assumed that the probability to find the global optimum near good points are higher than near bad points. The side lengths of the hyperrectangles must therefore 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 side lengths of a hyperrectangle at the g^{th} iteration $L_{ji}(\boldsymbol{X'}, g)$ are computed as

$$L_{ji}(\boldsymbol{X'}, g) = \frac{x^u_{ji} - x^l_{ji}}{\lambda} \frac{f(\boldsymbol{X'})}{f(\boldsymbol{X}^*_{g-1})} \frac{2}{\sqrt[n]{g}} \quad \text{for } i = 1, 2, \ldots I, \quad j = 1, 2, \ldots J \tag{10}$$

where $\boldsymbol{X'}$ is the center of the hyperrectangle, $n - I \cdot J$, g is the number of iterations, $\boldsymbol{X}^u$ and $\boldsymbol{X}^l$ are the respective upper and lower bounds of $\boldsymbol{X'}$, λ is a constant and $f(\boldsymbol{X}^*_{g-1})$ is the objective function value of the best solution from the previous iteration. During each iteration the tabu list and side lengths are updated. Tabu regions are not removed from the tabu list, but the tabu region size decreases with an increase in the number of iterations. The process is illustrated in Figure 7.

Again, intensification and diversification techniques are applied to improve the effectiveness of the TS. Algorithm 5 begins with intensification. It is assumed that better solutions

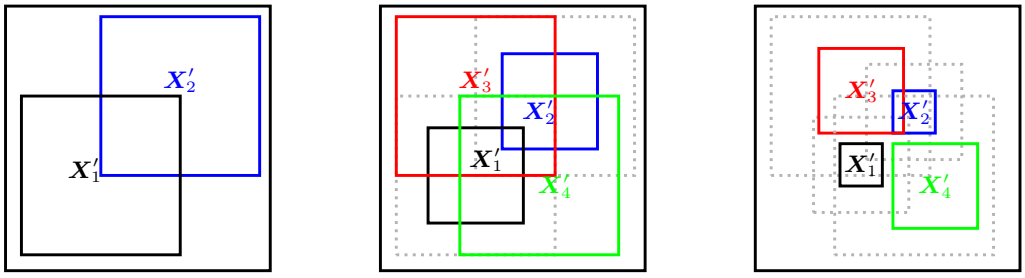


Figure 7: *The decrease of each tabu region size with increase in iteration rank for the CTSz. Here $n = 2$ and $G = 3$ where G is the total number of algorithm iterations. The dashed line corresponds with the tabu regions in iteration $g - 1$. [Figure can be viewed in colour in the electronic version, available at* `http://orion.journals.ac.za`*.]*

have a higher probability to be generated close to the current best solution and thus σ_{ji} is set to $\left(x^u_{ji} - x^l_{ji}\right)/10$ at the start of each iteration. This intensifies the search around the best solution found during the previous iteration. The normal distribution is used and thus 68% of generated points are on average within $x^*_{ji} \pm \sigma_{ji}$ for all i, j.

The majority of new solutions are generated close to the best solution found. As the solution space around the best solution becomes tabu, an increasing number of new solutions are rejected because they are inside existing tabu regions. To diversify the search the standard deviation is increased when 95% of the generated solutions are rejected. An increase in the standard deviation ensures that new solutions are generated further away from the best solution.

The TS algorithm, proposed by Hajji *et al.* [6], has four parameters that influence its convergence. The four parameters are p – the number of candidate solutions generated at each iteration, G – the maximum number of iterations, λ – a constant, and θ – the relative accuracy of an optimum location.

A relative accuracy of

$$\theta = \frac{1}{2}\frac{L_{ij}(\boldsymbol{X}^*_i, g)}{x^u_{ji} - x^l_{ji}} = \frac{1}{\lambda \sqrt[n]{g}} \tag{11}$$

is achieved when the algorithm stops. From equation (11) the constant

$$\lambda = \frac{1}{\theta \sqrt[n]{g}} \tag{12}$$

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

$$\frac{|\mathcal{T}|}{|\mathcal{S}|} \approx \beta = \frac{\sum_{\ell=1}^{p \cdot g} \left[\prod_{i=1}^{I} \prod_{j=1}^{J} L_{ij}(\boldsymbol{X}_\ell, g) \right]}{\prod_{i=1}^{I} \prod_{j=1}^{J} (x^u_{ij} - x^l_{ij})}, \tag{13}$$

where β is a constant between 0 and 1. Using equation (10) together with equation (13) it follows that

$$\beta \approx \frac{2^n \cdot p}{\lambda^n}. \tag{14}$$

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

$$p < \frac{\lambda^n}{2^n}. \tag{15}$$

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

A pseudocode listing for the CTSz appears in Algorithm 5.

Algorithm 5: `Immediate zone algorithm (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.

Generate a neighbourhood $\mathcal{N}(\boldsymbol{X})$ with p solutions chosen according to the uniform density probability.
Initialize $k \leftarrow 1$, $g \leftarrow 1$.
while $g < G$ **do**
  Store all solutions in the tabu list. Evaluate the size of the hyperrectangles using (10).
  $\sigma_{ji} \leftarrow k \cdot \frac{(x^u_{ji} - x^l_{ji})}{10}$ and set $\boldsymbol{X}^*_{g-1}$ to the best solution at the previous iteration.
  Generate a neighbourhood $\mathcal{N}(\boldsymbol{X})$ with p solutions using (7) – (9).
  Reject solutions that fall in existing tabu regions and count the number of rejected solutions.
  if Number of rejected solutions $\geq$ 95% of all generated solutions **then**
    $k = k + 1$
  else
    $g = g + 1$ and $k \leftarrow 1$.
  end if
end while
Return $\boldsymbol{X}^*_G$

5.3 Computational results

The two continuous tabu search methods CTSh and CTSz were applied to the SSP, HPP and MMRP.

5.3.1 The CTSh

The effect of different parameter settings were investigated on all the problems, and the parameter values that yielded the best results were used. For all three problems, it was concluded that the size of the tenure does not significantly affect the performance of the algorithm with regard to average objective function value. A tenure of $t = 5$ was chosen. The effect of different neighbourhood space sizes on the average objective function value was also investigated. It was 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 was recommended merely because of its quicker execution time. For the

HPP, however, a larger neighbourhood space size delivered a better result than a smaller one. The stopping criterion was set to 100 iterations.

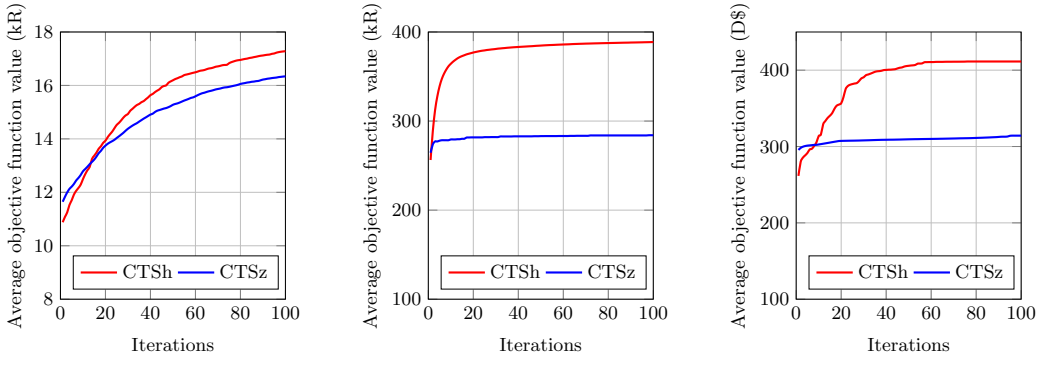
5.3.2 The CTSz

During each iteration, one component was chosen at random to investigate the effect its alteration had on the objective function value. This neighbourhood structure was constructed by altering the value of $\hat{x}_{ji}$ as described in §5.2 and the components were normalised so that feasibility was maintained.

The effect of different values for the relative accuracy, θ, of an solution was investigated. The value of θ affected the value of the constant, λ, which in turn affected the lenghts of the sides of the hyperrectangles which made up the tabu regions. It was concluded that the best objective function value was achieved with $\theta = 0.5$. The stopping criterion was set to 100 iterations.

5.3.3 Comparison of methods

The CTSh and CTSz were applied to the SSP, HPP and MMRP with their individual parameters set to optimize their results. Figures 8(a) to 8(c) contain the comparative results for the two algorithms. The CTSh delivered a higher average objective function value for all three the problems. Table 4 contains a summary of the results obtained after applying the CTSh and the CTSz approaches on all three of the problems.



(a) Results for the SSP. (b) Results for the HPP (c) Results for the MMRP

Figure 8: *The average objective function values as obtained by the CTSh* versus *the average objective function values as obtained by the CTSz. The units for the objective function values are in thousands of South African rands (kR) for SSP and HHPP, and in tens of US$ (D$) for the MMRP. [Figure can be viewed in colour in the electronic version, available at* `http://orion.journals.ac.za`*.]*

Result	SSP (kR)		HPP (kR)		MMRP (D$)	
	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
Number of runs	65	160	4625	1605	300	2505

Table 4: *Results summary for the TS techniques for the SSP, HPP and MMRP as well as the number of algorithm runs required to obtain a confidence interval of 90%. The units for the objective function values are in thousands of South African rands (kR) for SSP and HHPP, and in tens of US$ (D$) for the MMRP.*

6 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 attempts to avoid/escape local optima by moving away from them early in the computation when the probability of accepting worse solutions is still high. Towards 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 an 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. Locatelli [14] formulates the SA problem as shown in the pseudocode listing of Algorithm 6.

Algorithm 6: `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 existence of an optimum value f^*. An initial configuration $\boldsymbol{X}_0 \in \mathcal{S}$, 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 $\ell = 0$ and $T_\ell = T_0$.
2: Sample a candidate solution $\boldsymbol{X}'_{\ell+1}$ from the candidate distribution Φ.
3: Sample a uniform random number r in $[0, 1]$ and set

$$\boldsymbol{X}_{\ell+1} = \begin{cases} \boldsymbol{X}'_{\ell+1}, & \text{if } r \leq \Gamma(\boldsymbol{X}_\ell, \boldsymbol{X}'_{\ell+1}, T_\ell) \\ \boldsymbol{X}_\ell, & \text{otherwise.} \end{cases}$$

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

For the three problems the algorithm was initialized with a single randomly generated feasible solution. A candidate solution was generated by means of the method proposed

by Parks [19] as described in §3.2. The Metropolis acceptance function [21] was used. A total of ten algorithm runs per temperature setting with all other variables fixed was performed in order to calculate a reasonable average result. A standard deviation was computed for each set of twenty runs and an average of these deviations was obtained as a measure of the acceptance function's performance. The average standard deviation was computed to be approximately 0.005 for the Metropolis acceptance function.

Table 5 shows the best performing number of algorithm iterations for each starting temperature T_0 for the SSP, HHP and MMRP. When the initial temperature was relatively high, the resulting probability of accepting a worsening solution was high and fewer iterations at each temperature should be allowed in order to obtain a good solution. When the initial temperature was relatively low, the resulting probability of accepting a worsening solution was low and more iterations at each temperature may be allowed for a more extensive exploration of the search space.

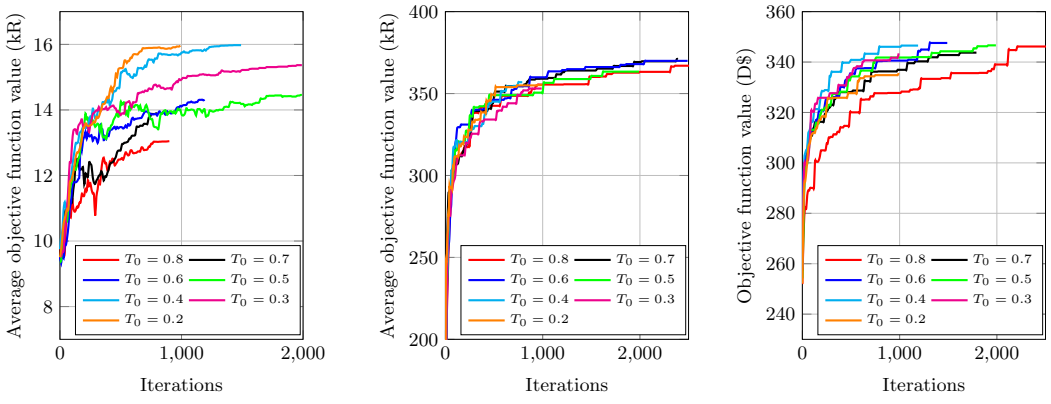
Initial temperature T_0	**Number of iterations**		
	SSP	**HPP**	**MMRP**
0.8	100	500	200
0.7	100	400	300
0.6	200	500	300
0.5	400	400	400
0.4	300	300	300
0.3	500	500	500
0.2	500	500	500

Table 5: *A summary of the best number of iterations for each starting temperature for the SSP, HPP and MMRP.*

Figure 9 contains the objective function values obtained by the algorithm for each problem for each initial temperature set at its best number of iterations. For the SSP, a lower starting temperature (0.2 to 0.4) combined with a mid-range number of iterations (300 to 500 iterations) delivered the best solutions. For the HPP, a higher starting temperature (0.7 to 0.8) combined with a larger number of iterations (400 to 500 iterations) delivered the best solutions as it allows for the most extensive solution space exploration. For the MMRP, a mid-range starting temperature (0.4 to 0.7) combined with a mid-range number of iterations (300 to 400 iterations) delivered the best solutions.

7 Solution summary for SSP, HPP and MMRP

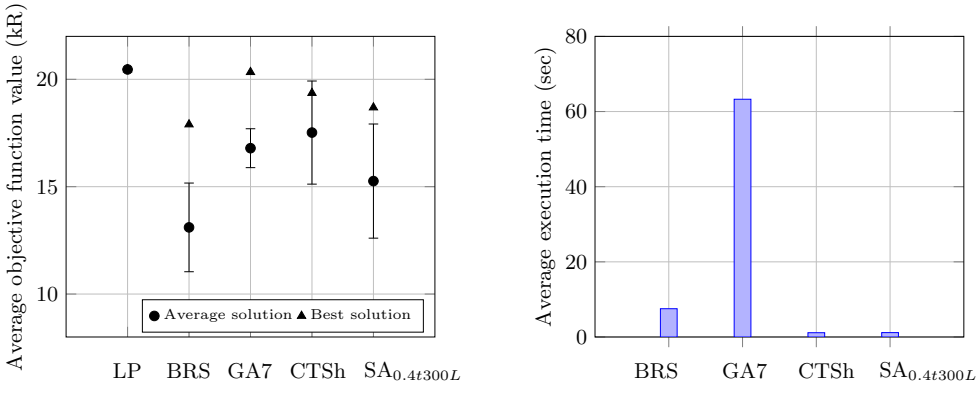
Figure 10(a) contains the best and average results obtained over 100 independent algorithm runs for the SSP. The genetic algorithm achieves a best result, *i.e.* closest to the LP solution, with the best result of the tabu search as a close second. On average, the tabu search obtains a result closest to the exact solution, with the genetic algorithm as a close second. However, the error bars on the graph show that the average objective function value of the tabu search is subject to a greater standard deviation than the average objective function value obtained by the genetic algorithm. Figure 10(b) contains the average execution time of a single run for each metaheuristic. The time it takes for a



(a) Results for the SPP (b) Results for the HPP. (c) Results for the MMRP.

Figure 9: *Average objective function values for each initial temperature at its best number of iterations. [Figure can be viewed in colour in the electronic version, available at* `http://orion.journals.ac.za`*.]*

single run of the GA is much longer than the time it takes for a single run of any of the other metaheuristics. The genetic algorithm requires an entire population of solutions to be found in the search space as opposed to only one as in the case of the other metaheuristics.



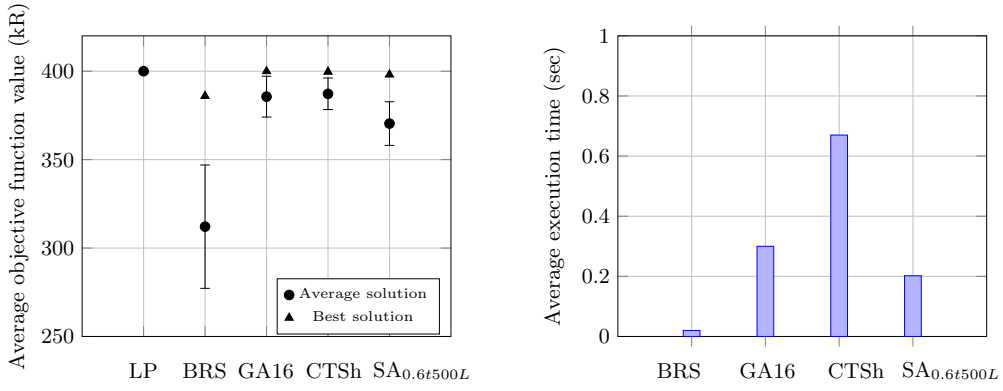
(a) Best and average solutions of the metaheuristic approaches. The error bars indicate one standard deviation around the average objective function value.

(b) Average execution times for each metaheuristic

Figure 10: *A summary of the results for the SSP.*

Figure 11(a) contains the best and average results obtained during 100 independent algorithm runs for the HPP. 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 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 11(b) contains the average execution time of a single run for each metaheuristic. 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.



(a) Best and average solutions of the metaheuristic approaches. The error bars indicate one standard deviation around the average objective function value.

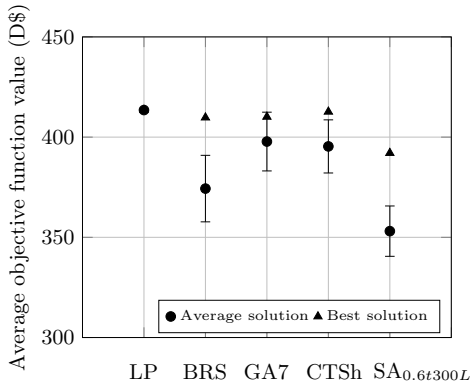
(b) Average execution times for each metaheuristic

Figure 11: *A summary of the results for the HPP.*

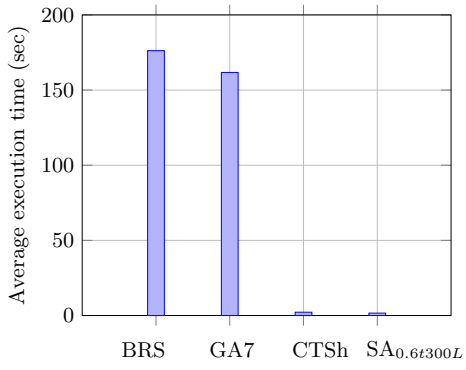
Figure 12(a) contains the best and average results obtained during 100 independent algorithm runs for the MMRP. The best results are obtained after application of the blind random search, genetic algorithm and tabu search, with the simulated annealing approach not performing as well. Similar to the results obtained for the SSP, on average, the tabu search obtains a result closest to the exact solution with the genetic algorithm as a close second. All four approaches are quite stable and the small standard deviations associated with each approach is illustrated by the error bars in Figure 12(a). Figure 12(b) contains the average execution time of a single run for each metaheuristic approach. Again the time it takes for a single run of the genetic algorithm or blind random search is 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.

8 A real world size instance

Certain characteristics of the four metaheuristic approaches have been identified and illustrated in the literature. Random search techniques may perform better on relatively small problems while more intelligent configuration search techniques such as the GA, TS and SA approaches may perform better on larger problems. This behavior is illustrated by Judson *et al.* [11] where random search techniques, GA and SA approaches are applied to



(a) Best and average solutions of the metaheuristic approaches. The error bars indicate one standard deviation around the average objective function value.



(b) Average execution times for each metaheuristic

Figure 12: *A summary of the results for the MMRP.*

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 random search techniques as the problem size increases.

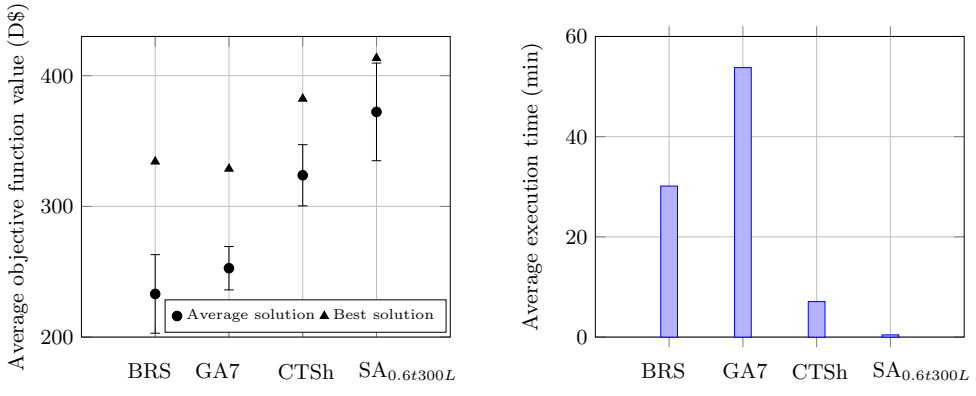
The MMRP is the most complex of the three problems and is the closest to a problem likely to be found in real life. Therefore performance of the metaheuristic approaches to this problem is investigated when the size of the problem is increased. The structure of the MMRP is used to generate a larger problem, called the extended MMRP (EMMRP). The EMMRP is comparable to a real world instance of petrochemical blending problems. 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 processes as opposed to the original number of 5. All other problem parameters are maintained as described in §2.3. The data for this problem is available form Venter & Visagie [27].

Candidate solutions for the initial solutions and populations are generated 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 assumes 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 $[0, 10]$. Finally, for the component constraints, the RON, RVP, density and sulfur present in each is generated as a value in $[0, 130]$, $[0, 20]$, $[0, 350]$ and $[0, 5]$, respectively.

A hundred 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 (*e.g.*, the limit on the amount of crude that might pass through each process).

The best performing algorithm of each type from the previous sections were used to solve the EMMRP. The commercial software package Lingo [13] was not able to load the input data for this problem due to size of the dataset and thus the results could not be compared to an optimal solution. Figure 13(a) contains the best and average results obtained for each metaheuristic approach using 100 independent algorithm runs. The best results are obtained from the simulated annealing and tabu search approaches. The SA approach obtains the best average result, although it has the largest standard deviation associated with its average solution. The GA approach delivers the most stable result as is illustrated by the error bars in Figure 13(a). Figure 13(a) contains the average execution time of a single run of each metaheuristic approach for the EMMRP. In conclusion the SA is preferred in terms of solution quality and execution time.



(a) Best and average solutions of the best four metaheuristic approaches. The error bars indicate one standard deviation around the average objective function value.

(b) Average execution times for each of the four best metaheuristic approaches.

Figure 13: *A summary of the results for the EMMRP.*

9 Conclusion

This paper starts out with a brief introduction to petrochemical blending problems. In §2 a description of the three instances from literature, namely the SSP, HPP and the MMRP, that are used to test the metaheuristics is presented.

A solution approach for this three problem instances by means of random search techniques is investigated in §3. It is concluded that local random search outperforms blind random search on average for the SSP, albeit by a small margin, blind random search outperforms local random search on average for the HPP and this result also holds for the MMRP.

The discussion of the application of metaheuristic approaches to the petrochemical blending problems commences in §4. 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.

Tabu search approaches follow in §5. Despite the continuous nature of the problems, two methods are successfully applied. 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. 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 final metaheuristic, simulated annealing, is presented in §6. It is concluded that a higher initial temperature in combination with a smaller number of algorithm runs per temperature level delivers the best solutions to all three problems. Section 7 contains a summary of the performance of the various solution approaches on the three small instances. It is concluded that, on average, the tabu search approach delivers the best result with regards to objective function value and execution time for the SSP. Both the tabu search and the genetic algorithm approaches deliver the best average objective function value, but the tabu search performs better in terms of its execution time for the HPP. For the MMRP, this result again is evident.

Section 8 contains results of the metaheuristic approaches when applied to an extended version of the MMRP. The simulated annealing approach is found to deliver the best average performance for this problem. The SA approach performs better relative to the other approaches as the size of the problem increases due to the fact that it requires fewer computations per iteration relative to the other approaches as the size to the problem increases. This trend (of simulated annealing performing better as the problem size increases) was also found by Low *et al.* [15] and Judson *et al.* [11]. The recommendation is thus that simulated annealing should be the preferred approach amongst classical metaheuristics for larger instances of blending problems.

The objective of this study was to compare the performance of classical metaheuristics over a range of blending problems. It was beyond the scope of this study to find the best performing metaheuristic to solve blending problems in general. In future research improvements on the classical metaheuristics, such as hybrid methods or variable neighbourhood search algorithms may be considered in the quest to find the best algorithm(s) to solve large blending problems.

References

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

[2] Chelouah R & Siarry P, 2000, *Tabu search applied to global optimization*, European Journal of Operational Research, **123(2)**, pp. 256–270.

[3] Floudas CA & Pardalos PM, 1990, *A collection of test problems for constrained global optimization algorithms*, volume 55, Springer Lecture Notes In Computer Science.

[4] General algebraic modeling system (GAMS), 2013, *Marco mini-refinery*, [Online], [Cited on June 27th, 2013, Available from http://www.gams.com/modlib/libhtml/marco.htm.

[5] Goldberg D & Deb K, 1993, *A comparative analysis of selection schemes used in genetic algorithms*, Journal of the Society of Instrument and Control Engineers, **32(1)**, pp. 10–16.

[6] 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.

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

[8] Hu N, 1992, *Tabu search method with random moves for globally optimal design*, International Journal for Numerical Methods in Engineering, **35(2)**, pp. 1055–1070.

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

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

[11] Judson RS, Colvin ME, Meza JC, Huffer A & Gutierrez D, 1992, (Unpublished) , Technical report, Sandia National Laboratories, Center for Computational Engineering.

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

[13] Lingo, 2012, *Lindo systems' index page*, [Online], [Cited November 10, 2012], Available from http://www.lingo.com/.

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

[15] Low C, Yeh YJ & Huang KI, 2004, *A robust simulated annealing heuristic for flow shop scheduling problems*, International Journal of Advanced Manufacturing Technology, **23(9-10)**, pp. 762–767.

[16] Matyas J, 1965, *Random optimization*, Automation and Remote Control, **26(2)**, pp. 224–251.

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

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

[19] Parks GT, 1990, *An intelligent stochastic optimization routine for nuclear fuel cycle design*, Nuclear Technology, **89(2)**, pp. 233–246.

[20] Pegden D, Shannon RE & Sadowski RP, 1995, *Introduction to simulation using SIMAN*, 2nd Edition, McGraw-Hill.

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

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

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

[24] Toklu YC, 2005, *Aggregate blending using genetic algorithms*, Computer-Aided Civil and Infrastructure Engineering, **20(6)**, pp. 450–460.

[25] Van Rossum G, 2008, *Python language website*, [Online], [Cited on November 10th, 2008], Available from http://www.python.org/.

[26] Venter L, 2009, *Metaheuristics for petrochemical blending problems*, Masters Thesis, Stellenbosch University.

[27] Venter L & Visagie SE, 2013, *Petrochemical blending problem instances*, [Online], [Cited June 27th, 2013], Available from http://hdl.handle.net/10019.1/80952.

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