THE UNIVERSITY OF SHEFFIELD
DEPARTEMENT OF COMPUTER SCIENCE

“Optimisation Techniques for Gas Turbine Engine Control Systems”

Dissertation Main Report

By:

Salem Adra

This report is submitted in partial fulfilment of the
Requirement for the degree of
MSc (Eng) in Advanced Software Engineering

Project Supervisors: ***Dr Kirill Bogdanov***
Dr Ian Griffin

Submission Date: **27TH August 2003**

Plagiarism Declaration:

All sentences or passages quoted in this dissertation from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations which are not the work of the author of this dissertation have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this, amounts to plagiarism and will be considered grounds for failure in this dissertation and the degree examination as a whole.

Name*:* **Salem Adra**

Signature*:* **Date*:* 27th August 2003**

Abstract

"Optimisation Techniques for Gas Turbine Engine Control Systems" is an external project in collaboration with the Control & Systems University Technology Centre supported by Rolls-Royce at the Automatic Control & Systems Engineering Department at the University of Sheffield. A Multi Objective Genetic Algorithm (MOGA) optimizer was implemented to optimise different jet engines' parameters influenced by several variables such as altitude measure, fuel flow, thrust and the amount of power requested by the pilot. The project's core is to improve the MOGA technique currently used by implementing a Memetic Algorithm (MA), a more elaborate Evolutionary Algorithm that consists basically of hybridizing a Genetic Algorithm (GA) with a local search technique. The resulting strategy will be implemented in an optimization tool that is currently being piloted by Rolls-Royce (RR) for controller parameter tuning.

In this project, three of the most popular types of local search techniques used in MAs are successfully implemented, tested and contrasted. A detailed analysis of their performance is described, highlighting their major differences, advantages and disadvantages. These local search techniques are: Hill Climbing technique, Simulated Annealing technique and Tabu Search technique.

This report provides wide background information about Genetic Algorithms and a critical evaluation of Memetic Algorithms. Finally, suggestions are made for future testing and improvements.

Acknowledgments

Many thanks go to my supervisors Dr Kirill Bogdanov and Dr Ian Griffin for their guidance, support and patience throughout the project, as I would like to thank my tutor Dr Tony Simons for all his assistance and appreciated advice.

Finally I would like to dedicate this dissertation to my parents, whom without their encouragements, supports and financial sacrifice none of this work could have been possible.

Contents

List of Figures

List of Tables

CHAPTER 1-INTRODUCTION

1 Introduction

1.1 Introduction

For testing the efficiency and accuracy of response of jet engines, Control engineers at Rolls-Royce (RR) spend a lot of time measuring and collecting data fed back by control systems and sensors. This is done in order to calculate and reduce errors, consequently optimising engines and control system parameters for better performance. Fig. 1.1 illustrates a typical situation that is often encountered when testing engines:

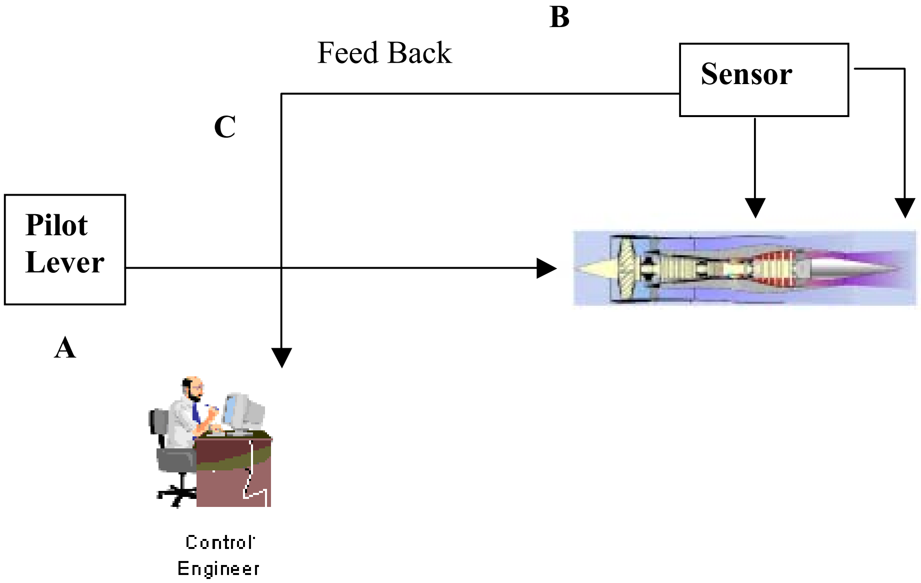


Figure 1.1: Engines Testing Situation

In Fig 1.1, the pilot lever requests a certain amount of power from the engine by moving to a certain angle. Ideally, we want the engine's thrust to exactly match the amount requested by the lever. A sensor is placed within the engine to measure the amount of thrust produced by the engine in response to the pilot's demand. The measurements recorded by the sensor are fed back to the control system, which calculates the error between the requested amount of power and the actual value produced. The control system will then act to reduce the error in achieved thrust by adjusting the fuel flow rate to the engine. The parameters of the control system determine how effective this error reduction process is. The role of the Control Engineer is to identify a set of controller parameters that will provide optimal control performance in terms of error reduction. In practice, the process of identifying an optimal set of controller parameters for an engine is a time consuming and expensive process. An alternative approach to solving this problem is the use of an optimization routine. An optimization routine would employ a simulation environment to automatically tune the controller parameters in a quicker and more cost-effective manner whilst also providing a control system that improves the performance of the engine.

At the Control & Systems University Technology Centre (UTC) in the Department of Automatic Control & Systems Engineering (AC&SE), an optimisation tool, based in Matlab and Simulink is being developed, which enables the user to tune controller parameters in a simulation environment. The Control Engineer defines a set of objective functions, against which the control system's parameters are tuned. The controller parameters are then optimised in an attempt to meet the goals that have been defined for each objective. The optimisation routine employed in the tool is a Multi Objective Genetic Algorithm (MOGA). Genetic Algorithms (GAs) are a generational, evolutionary technique that imitates the biological evolution of living species. The optimisation tool is implemented to optimise three specific parameters (Gain (K), Lead Time

(Tnum) and Lag Time (Tden)). The user specifies target or goal values for each of the different performance objectives. These include time domain objectives such as overshoot, overshoot time and other objectives. The performance of each candidate set of controller parameters will be evaluated in Simulink for every new optimisation values of the three parameters reflecting their performance in the real domain.

The search range for the parameters and the software path for the MOGA will be specified as well by the user using the Graphical User Interface (GUI). A specific search range for each parameter is defined around an initial value or guess, usually proved to be a good solution from previous engines. The genetic algorithm then searches for the optimum control system performance by adjusting each controller parameter within a neighbourhood defined as being a fixed percentage either side of the initial guess.

1.2 Project Aim

The aim of the project is to try to improve on the Genetic Algorithms used in the actual system by implementing a Memetic Algorithm (MA). This is a further extension to multiobjective algorithms and relatively a more recent technique whose application in certain domains is still under further investigations and research. Memetic algorithms are a more elaborated version of multiobjective algorithms, and their implementation is a challenge that may “hopefully” provide better optimisation results.

1.3 Structure of This Report

First this report opens with a detailed literature review that ensures an overview of Genetic Algorithms, defining these algorithms and illustrating a comparison with other traditional techniques. In this literature review, the general background for the project will be discussed, the major elements of the Genetic Algorithm will be explored and a definition of the four essential functions of Genetic Algorithms, which are successively the selection function, the crossover function, the mutation function and the reinsertion function, will be described. The termination of Genetic Algorithms and their different applications will also be allocated proper sections for their discussion. The last sections of the literature review will be dedicated to the illustration of a comprehensive presentation of Memetic Algorithms, pinpointing their additional functionality and differences from typical Genetic Algorithms using a simple Pseudocode example.

After the Literature Review, the main core of this project will be presented in details in the Requirements and Analysis Chapter, where the actual optimization system, used at the Automatic Control & Systems Engineering Department at the University of Sheffield, will be clearly elucidated. The system’s requirement will be then detailed, coupled with a requirements’ analysis pinpointing the major issues that should be considered while designing the system and implementing the requirements.

The system’s design will be then demonstrated in Chapter 4, using some functional diagrams and examples’ illustrations that will clarify the intended system’s design, to be implemented throughout the project. The 3 local search techniques to be hybridized with the multiobjective genetic algorithm will be clearly explained and contrasted.

In Chapter 5, the implementation process of the memetic algorithm will be discussed, illustrating some code’s portions for better understanding. Mainly, three major issues will be discussed in this Chapter; the neighbourhood setting process, the local search functionality, and the acceptance steps of the local search algorithms. At the end of this Chapter, the testing process of the implemented local search techniques is ensured, illustrating some output results from the optimisation process, and using a simple created scenario to validate the correct functionality of the memetic algorithm.

Chapter 6 will be dedicated for the results and the major findings’ listings. A statistical analysis will be made, highlighting the performances’ differences of the genetic algorithm and the memetic algorithm. Future work is also described at the end of this Chapter.

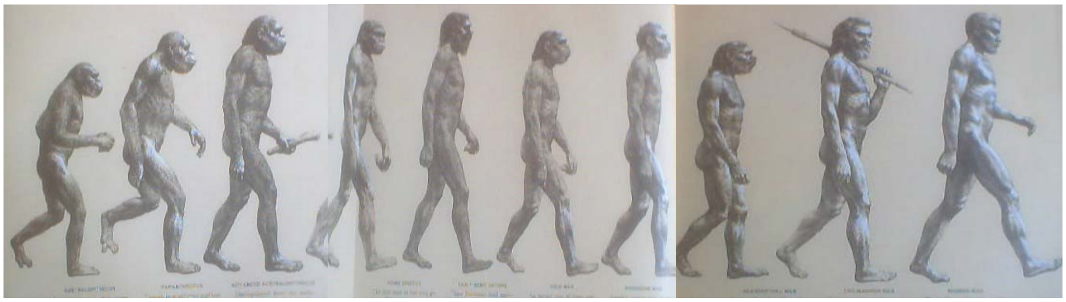
At the end of this report, a summarizing conclusion will be stated, recapitulating the major results and findings of the project.

CHAPTER 2- LITERATURE REVIEW

2 Literature Review

2.1 Introduction: What are Genetic Algorithms?

According to the Darwinism view, the human physical and mental status that we are privileged with is the chronological result of successive biological evolutions. Prehistorically, the principle of "survival of the fittest" was applied to primates' populations, where the best individuals were able to survive several biological crises and climatic breakdowns, adapting to new environments. This has resulted the humans' actual position at the end of this evolutionary chain.



The Genetic Algorithm is a stochastic global search method that imitates this process of natural biological evolution, operating on "populations" of potential solutions by applying the law of the jungle where the survival is for the fittest, hopefully producing better approximations to a given application's solution. Until a stopping criterion is reached (e.g. certain number of generations or a mean deviation in the population), a new set of approximations is created at each generation.

2.2 Motivation for the Use of GAs

The motivation for using GAs is partly dependent upon the nature of the application. In other words, if the optimization problems under investigation are reasonably well behaved, then the obvious and best choice will be the use of the conventional deterministic techniques, such as deterministic gradient-based-and simplex-based search methods. Unfortunately, these conventional deterministic optimization techniques face major difficulties in several scenarios, such as when the objective function is discontinuous or characterized by many local optima and points at which gradients are undefined, or when the estimation problem involves many parameters that interact in highly non-linear ways. In these situations, heuristic methods like GAs are a powerful alternative for exploring search spaces and finding good solutions that cannot be detected by conventional numerical techniques.

2.3 How Genetic Algorithms work?

Starting from a population of potential solutions ensured by the user, the GA works on improving these solutions by filtering out relatively bad ones and discovering better approximations by searching the neighbourhood around some solutions and applying operators borrowed from natural genetics. The populations' individuals are encoded as strings composed over some alphabet and are assimilated to "chromosomes", the basic unit of genetics. The chromosomes' values or "genotypes" are mapped in a "one-to-one" fashion onto the decision variable domain or "phenotype". Binary representations (i.e. {0, 1}), integer representations and real valued representations are different kinds of chromosomes' representations in GAs, although the first one is the most commonly used representation.

Example of Binary representation:

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

Figure 2.1: Binary Representation

Chromosome X1 is encoded with 10 bits whereas X2 is constituted of 15 bits. The number of bits determines the resolution of the search within the search space. Although the search process of GAs operates uniquely on this encoding of the chromosomes, it is only with their decoding into the respective phenotypic values that the meaning within the decision space can be determined. The fitness and performance of the chromosomes in their application domain can then be determined by assessing the objective function values. A function "phen" is allocated the functionality of mapping the encoding of a chromosome into its phenotypic value, and then an objective function, formulated by the user and specific to the domain of the application, is processed using the phenotypic values represented by the chromosomes that were outputted from the "phen" function. Note that this function won't be necessary in cases where real value representations are used. Each objective function's role is to assess the fitness of an individual within the application's environment and will consequently attach to it a value or score reflecting its performance with respect to that objective. The objective function value for each objective is then converted into a fitness value. The fitness values will be the basis of the individual's selection for mating to form the next generations of potential solutions. Generally the fitness function's role is to transform the values produced by the objective function into non-negative values reflecting probabilities. Once the fitness values are assigned to the individuals, they can be chosen to recombine with a probability according to their fitness values. After their selection, the chromosomes will be recombined accordingly similar to the reproduction process in biological evolution, thus exchanging genetic information between each other, based on the assumption that certain individuals genes' parts produce on average fitter individuals.

There are several types of recombination operators but the simplest one is that of the single-point crossover described in Figure 2.3. Consider the following two parent binary chromosomes (Figure 2.2):

P1 = 1 0 1 0 0 1 1 1 1 0

P2 = 1 0 0 1 1 0 1 0 0 1

Figure 2.2: Binary Chromosomes

An integer position along the length of the chromosome is randomly selected between 1 and the chromosome's length (L) minus one. (Note that a chromosome is indexed like an array, i.e. 0 is the index of the first element. This range is logical, because crossing over 2 chromosomes at index 0 will simply give back the same parents). Consider a crossover using P1 and P2 at index 5. This will result in the following offspring:

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
P1 =	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**
P2 =	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**
O1 =	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**
O2 =	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**

Figure 2.3: Single Point Crossover

Note that when the pairs of chromosomes are chosen to breed, the crossover operator is applied with a certain probability and hence not necessarily on all the strings of a population. After recombination, another operator, the mutation operator, is applied to the offspring. Again, the application of this operator is not guaranteed but based on a user-defined probability. This is generally a low value just like in the natural world. The mutation operator will cause a single bit to change its state, i.e. in case of binary representations, 0→1 and 1→0. For example, mutating offspring O2 at its last index will generate the following individual:

O2m = 1 0 0 1 1 1 1 1 1 1

Mutation is generally a backup operator assuring that the probability of searching a specific subspace of the problem space is never null.

After the selection, recombination and mutation operators, the cycle is repeated for another generation. The objective function is processed again on the encoding of the offspring, evaluating their performance within the application environment, and a respective fitness value for each candidate solution is allocated correspondingly. Generation by generation, the process continues, stochastically propagating genetic material through subsequent generations until a predefined stopping criterion is reached.

2.4 What makes GAs so special?

In Table 2.1, the major elements that differentiate the Genetic Algorithms optimisation technique from other traditional methods are illustrated, essentially noting that GAs provides a population of optimised candidate solutions rather than one unique solution. The choice of a final solution will be left for the user. The use of GAs is particularly appropriate in situations where a problem possesses a family of equivalent optimal (Pareto-optimal) solutions, as with multiobjective optimisation and scheduling problems.

GAs versus Traditional Methods	
GAs	*Traditional Methods*
Search a population of points in parallel	Operate on single points
Do not require derivative or auxiliary information	Generally requires derivative information
Use probabilistic transition rules	Generally use deterministic transition rules
Work on an encoding of the parameter set (except in real-valued representations)	Work on the parameter set itself

Table 2.1: GAs versus Traditional Methods

2.5 Pseudo-code of a simple Genetic Algorithm

The pseudo-code illustrated in Figure 2.4 is an outline of a simple Genetic algorithm (Goldberg, 1989):

Figure 2.4: GA Pseudocode

```
Procedure GA
  Begin
        T=0;                    %T denoting the Time variable
    Initialise P(t);        %Generally Consists of a Random initialisation
                                       of the initial population, at time T=0
        Evaluate P (t);         %Assess the performance of the individuals in
                                       the current population.
        While not finished do
        Begin
                  t = t+1;   %Increment The Time variable By 1
                  Select P (t) from P (t-1);
                  Reproduce pairs in P (t); % denotes the reproduction
                  Evaluate P (t)              phase, reproducing new
         End                                      Offspring by cross over
End
```

2.6 Population Representation

As stated earlier, GAs operate on an encoding of the parameter set, which constitutes a population of potential solutions. In typical situations, it is shown that a population size of 30 to 100 individuals is most commonly used, although a variant of GAs, the “micro GA”, operates on smaller populations’ sizes with more restrictive operators.

Many chromosomes representations are widened and used although the single-level binary representation is the most spread. In this representation, chromosomes are constituted by the concatenation of binary strings inspired from the decision table. Despite its popularity, binary representation suffers from several deficiencies such as the representational bias of this kind of representation, the Hamming distance between adjacent values is constant, and as validated by Caruana and Schaffer (1988), large Hamming distances between adjacent values can mislead the search process locating the optimal solutions. Some solutions though were advocated for overcoming these weaknesses, such as the use of Gray coding or logarithmic scaling while converting the chromosomes’ encoding into their domain’s values, as suggested by Schmitendorgf *et-al* (1992). Alternatively, real valued and integer representations are increasingly achieving usage interests, as they can result in several advantageous effects. Specifically, more efficient GAs will be produced, as the adoption of this representation eliminates the need for converting chromosomes into their phenotypic values, in addition less memory will be needed for the whole process and the loss of precision that can result from the discretisation of phenotypic real values to binary encoding will be totally avoided. After selecting the type of representation that best suits the application environment, the next step is to create an initial population of potential solutions for the GA. Alternative methods for this process are widely adopted. One way of initialising the population consists of creating a random generation of the required number of individuals (Nind) using a random number generator that uniformly distributes numbers in the desired range from the set {0, 1}, each of the same specific length, i.e. number of bits (Lind). An alternative way is the “extended random initialisation” procedure (Bramlette, 1991) whereby a given number of random initialisation trials is processed for each individual, which will be initialised to the best performance trial. In situations where the nature of the application is well understood in advance, or where the GA is used in conjunction with knowledge based systems, the initialisation can be held in the vicinity of good solutions previously known, like it is the case in this project.

2.7 The Objective and Fitness Functions

The objective functions reflect the raw performance of each individual in the problem domain. Two approaches can be adopted, minimisation and maximisation. In the case of minimisation the fittest individual will be allocated the smallest numerical value resulting from each objective function, and vice versa in the case of maximisation. This function only reflects the raw performance of each individual. The fitness function then transforms the objective function’s values into a non-negative measure of relative fitness, which will be used by the GA for selection and breeding purposes, i.e.:

$$\mathbf{F(x) = g(f(x))}$$

Where “f” is the objective function, “g” transforms the value of the objective function into a non-negative value, and “F” is the resultant fitness value. Many versions are adopted for the Fitness function; the proportional fitness function (equation 2.1) and the linear transformation (equation 2.2) are the most often used ones.

$$F(X_i) = \frac{f(Xi)}{\sum_{i=1}^{Nind} f(Xi)}$$

Proportional fitness function **(2.1)**

Nind is the population size, Xi is the phenotypic value of individual i

While this function (the proportional fitness function) ensures that each individual is allocated a probability for reproduction proportional to its relative fitness, it does not take into consideration negative objective function values.

$$F(x) = a*f(x) + b \qquad \textbf{\underline{Linear Transformation}} \qquad \textbf{(2.2)}$$

"a" is positive scaling factor in case of maximisation and negative vice versa

"b" is an offset ensuring that the resulting fitness value is positive

This method (Equation 2.2) though has uncovered another kind of undesired situation, the case of rapid convergence towards possible sub-optimal solutions.

A suggested solution (Baker, 1985) consists of adding constraints on the reproduction range by limiting the number of offspring an individual can produce to a certain maximum, so that no individuals will generate an excessive number of offspring, and thus preventing "premature convergence". The fitness function can be used in several environments and problem domains whilst the objective function is domain-specific and should be thus created by the user.

2.8 The Selection Operator

2.8.1 Introduction

Having allocated a reproduction expectation value for every individual, the next step will consist of a probabilistic selection of pairs of individuals for reproduction, based on their relative fitness values.

Roulette wheel selection, Steady-State selection, Stochastic Universal selection and Rank selection are all different kinds of selection methods commonly used in a wide range of application domains. In order to assess the quality of the selection process, Baker (1987) introduced three performance measures for selection algorithms: bias, spread and efficiency. Bias is defined as the absolute difference between the actual selection probability resulting from the selection process, and the expected selection probability reflected in its relative fitness value. The optimal zero bias is therefore achieved when the individual's actual and expected number of trials are identical. While bias is a good measure of accuracy, spread is another more flexible measure of consistency, defining a specific acceptable range for the actual number of trials for each individual that spans both sides of the expected value by a certain offset. Finally, like its name indicates, the "efficiency" measure is a process aiming to efficient selection processes, ideally targeting to zero bias while maintaining a minimum spread range.

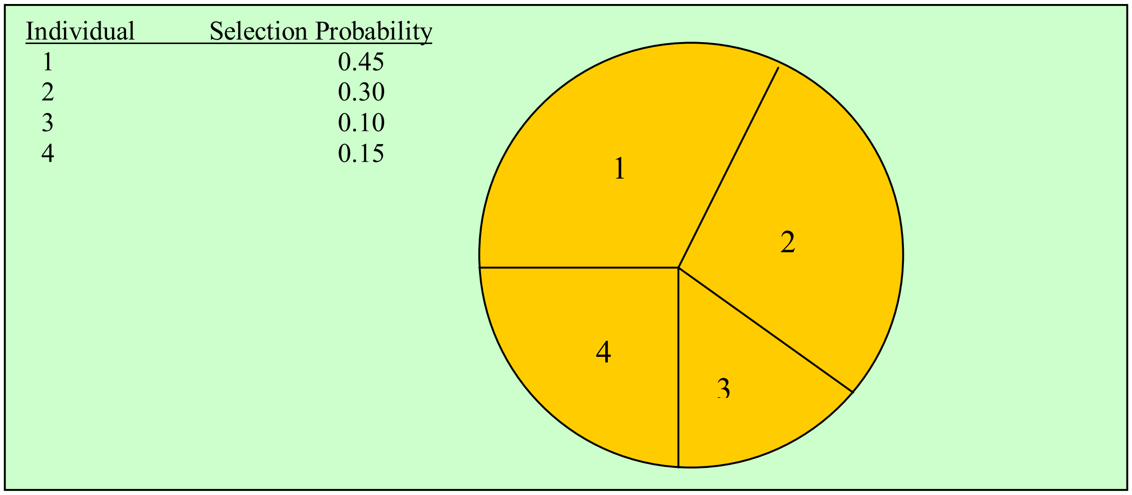
2.8.2 Roulette Wheel Selection

One of the most commonly used selection schemes is the roulette-wheel selection, also called stochastic sampling with replacement (SSR). The individuals are allocated contiguous intervals of length's range [0 sum], where "sum" determines the sum of the individuals' expected selection probability or raw fitness. A random number is generated in the range [0 sum] and the individual whose segment spans the random number is selected. The process is repeated iteratively until the desired number of individuals is obtained.

Table 2.2 shows the selection probability for 4 individuals, Individual 1 being the most fit individual that occupies the largest interval, whereas individual 3 is the weakest one and correspondingly occupies the smallest interval (see figure 2.5).

Number of individual	1	2	3	4
Selection probability	**0.45**	**0.30**	**0.10**	**0.15**

Table 2.2: Selection probability



Individual	Selection Probability
1	0.45
2	0.30
3	0.10
4	0.15

Figure 2.5: Roulette Wheel Selection

Consequently most fit individuals will occupy larger intervals and will have higher probabilities for being selected to breed and propagate to the next generations.

In the Roulette wheel method described above, the segment size and thus the selection probability remain invariant through the whole process; logically any individual with segment size > 0 could entirely fill the next population. In order to avoid this situation and decrease the chances of early convergence, another selection method, Stochastic Sampling with Partial Replacement (SSPR) enhances SSR by reducing (generally by 1.0) the interval's size of an individual once selected, which is set to nil if it becomes negative.

2.8.3 Stochastic Universal Sampling

Stochastic Universal Sampling (SUS), despite the Roulette Wheel selection, is another selection method, which uses N equally spaced pointers for selecting the individuals (Figure 2.6), where N denotes the number of selections needed. This method starts with a random shuffle of the population's placement over the wheel. Then a single number is generated randomly from the range [0 Sum/N]. This number will constitute the position of the first pointer on the wheel, and the N-1 pointers left will form a series of equally spaced pointers, usually based on the following distribution: [ptr, ptr +1,… ptr + N-1], and consequently, the individuals whose intervals are pointed at by one of the N pointers will be chosen for reproduction. This method seems to be more efficient and time saving compared to the Roulette wheel selection, which uses one pointer, selecting thus just one individual at a time. Selection methods are very numerous and each one possesses its different characteristics, advantages and drawbacks.

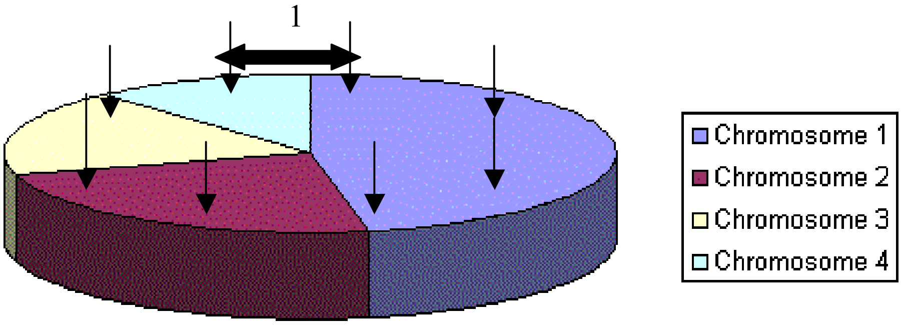


Figure 2.6: SUS with 8 pointers

2.9 The Crossover Operator

2.9.1 Introduction

Crossover is the commonly used technique for implementing the reproduction process, and is the basic operator for the creation of new individuals in a generation. Its notion is essentially borrowed from the natural reproduction process in the real world, where an offspring inherits specific parts from both his parents. Just like the selection process, Crossover has many versions and types, the single-point crossover described in the introduction being the simplest. An extension of the single point crossover is the Multi-point Crossover, which will be described in the section 2.9.2

2.9.2 Multi-point Crossover

The major idea behind this kind of crossover is that instead of one single point crossover, "M" unduplicated crossover positions are selected randomly in the range {1, 2, … L-1} (L denotes the length of the Chromosome), and sorted in an ascending order, once again noting that a chromosome is indexed starting at the index zero. The bits spanning consecutive crossover positions will be than exchanged between the parents producing consequently offspring sharing specific parts of both parents, and who will be assessed and allocated a fitness value based on their performance later on in the GA. Figure (2.7) illustrates a Multi-point crossover at two positions:

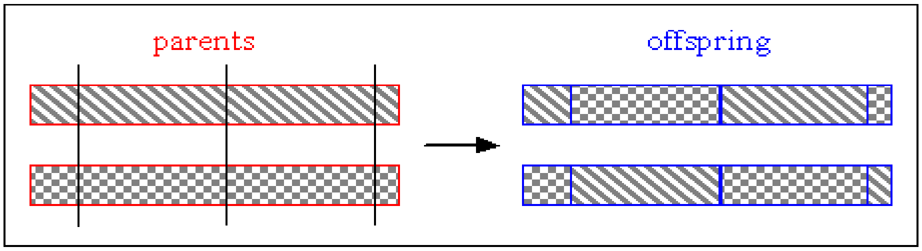


Figure 2.7: Multi-point Crossover

2.9.3 Uniform Crossover

This kind of Crossover (Syswerda, 1989) is more general than both the single point and the Multi-point Crossovers where precise locations between crossover points to be switched by the parents are identified. Uniform Crossover (Figure 2.8) generalises this notion by making any single bit of a chromosome a potential crossover point. This is due to a randomly created Crossover mask of the same size as the chromosomes, which will be applied to the parents chosen for recombination by choosing bits from the parents based on an agreed parity of the Crossover Mask.

Parent 1	0	1	1	1	0	0	1	1	0	1	0
Parent 2	1	0	1	0	1	1	0	0	1	0	1
Mask	0	1	1	0	0	0	1	1	0	1	0
Offspring	1	1	1	0	1	1	1	1	1	1	1

Figure 2.8: Uniform Crossover

For each bit, the parent who contributes its variable to the offspring is chosen randomly with equal probability. In the previous example (Figure 2.8), taking the bit from parent 1 if the corresponding mask bit is 1 or the bit from parent 2 if the corresponding mask bit is 0 has produced the offspring.

Other Crossover operators are also widely known and used in several varieties of problems. The **“*Shuffle* “operator** (Caruana et al, 1989) operates on a pair of chromosomes chosen for reproduction by selecting a single random crossover point just like in the single point crossover, although in this kind of crossover, before the designated bits of the parents are exchanged to form an offspring, they are randomly shuffled, and the final part of this operator will consist on “unshuffling” the bits of the resultant offspring. The “***Reduced Surrogate”*** (Booker, 1987) operator is another kind of crossover, which constrains the process to always generating new offspring by adding restriction to the location of crossover points. Alternatively, real valued chromosome representations possess a whole series of crossovers operators such as the “***Intermediate Recombination”*** (Mühlenbein and Schlierkamp-Voosen, 1993)**,** which consists of producing offspring phenotypes intermediate to the parents’ numerical values and the “***Line Recombination”*** (Mühlenbein and Schlierkamp-Voosen, 1993) which is an extension of the Intermediate Recombination adding some constraints on the process. The choice of individuals’ representation and consequently the version of crossover operator is a challenging issue, which directly depends on the nature of the problem under investigation.

2.10 The Mutation Operator

After a crossover is performed, mutation takes place. Just like in natural evolution, Mutation is a random process, which consists of replacing a certain allele of a gene resulting in a new genetic structure. Mutation is intended to prevent the situation where all solutions in a population fall into a local optimum of the solved problem, and thus it constitutes a background operator which guarantees that the probability of searching a specific subspace is not zero. Mutation operation randomly changes the offspring resulting from crossover. Typically, it is applied with low probabilities in the range 0.001 and 0.01. In the case of binary encoding, mutation consists of switching few randomly chosen bits from 1 to 0 or from 0 to 1.

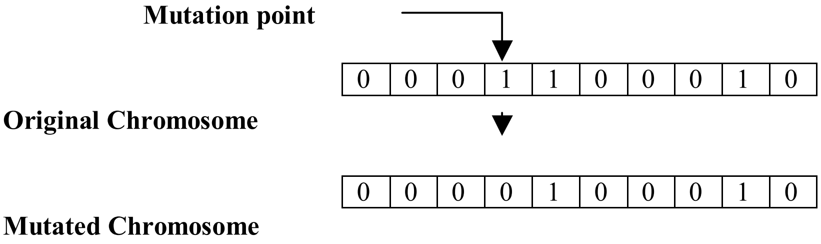


Figure 2.9: Mutation

With Real value and Integer representations two conventional methods are applied, either by perturbing the chromosomes values or by randomly selecting new values within a specific allowed range. For these kinds of representations (Integer and Real value), it is argued (Tate and Smith, 1993) that higher mutation rates can be more desirable and lead to better solutions than the reduced mutation rates. Another variation of the standard Mutation operator consists of directing the mutations towards low fitness individuals in order to explore new subspaces without taking the risk of losing information from fit individuals.

2.11 The Reinsertion Operator

After selecting individuals for reproduction based on their fitness values, crossover and mutation processes have resulted new individuals in the population. Now the fitness value can be determined for the individuals of the new generation. Sometimes though the recombination process produces fewer individuals than the number of individuals of the previous population, resulting a fraction difference. Note that the number of individuals in a population should be conserved through all the generations. This fraction difference between the initial number of individuals and the new one is called a generation gap (De Jong and Sarma, 1993), and a process of insertion will be then required in order to maintain the number of individuals in a population, although other reasons such as the fact of producing higher number of offspring will necessitate the reinsertion scheme. The most commonly used strategy of Reinsertion is to fill the generational gap by inserting the best-fit individuals from the previous population. Although this strategy seems logical enough, it has been proved (Fogarty, 1989) that no significant convergence differences are noted with other reinsertion schemes such as inserting randomly or even the least fit individuals. However, a more successful strategy is to reinsert the oldest members of a population, i.e. the individuals that were able to survive and propagate through several generations, consequently demonstrating good performances and qualities.

Termination of the GA And some useful GA's Applications

2.12 Termination of the GA

Due to the stochastic nature of the GA's search and optimisation techniques, it is usually ambiguous to determine specific convergence criteria, as at some stages the fitness of a population may be stabilized for a number of generations, and thus the timing for a convergence is unpredictable. A commonly used practice consists of terminating the Genetic algorithm after a certain predefined number of generations, which will enable the user to test the quality of the resultant population of solutions against the problem domain, and restart the GA for another specific number of generations if the quality of the previous solutions was not acceptable or of a certain standard. Another termination strategy is to stop the GA when a particular point in the search space is encountered or a mean deviation in the population is reached.

2.13 GA's Applications

Genetic algorithms have been proved to be of great efficiency and usability in different problems and domains such as in Machine Learning for evolving simple programs to solve various problems; they have been also used in totally different domains, like arts, for evolving pictures and music.

Genetic Algorithms have been successfully implored in the following domains:

- Non-linear dynamical systems - predicting, data analysis
- Designing neural networks, both architecture and weights
- Robot trajectory
- Evolving LISP programs (genetic programming)
- Finding shape of protein molecules
- Functions for creating images

To get an idea about some problems solved by GAs, here is a short list of some applications (source: Evonet's website funded by the European Commission, http:// www.evonet.com//):

"Genetic Algorithms in Parametric Design of Aircraft" (Bramlette and Bouchard), is a project devoted to the optimization of aircraft designs by optimizing lists of special parameters. The project was based on an approach that uses real number representation's genetic algorithm, which starts by generating a large number of individuals constituting the initial population and processing only the good members. The work's domain is aeronautical engineering and is relatively similar to the "Optimisation Techniques for Gas Turbine Engine Control Systems" project.

Other interesting problems of various domains solved by genetic algorithms:

- "Dynamic Anticipatory Routing in Circuit-Switched Telecommunications Networks" (Cox et al), is a project which deals with an optimization problem of the routing of telephone networks in order to minimize the costs to West US. The project's approach was based on a performance comparison of an order-based genetic algorithm and several other optimization techniques, into the problem's domain. The project achieved two major results: Firstly, Genetic Algorithms are highly successful especially when the problem under investigation is of complex nature. Secondly, the hybridization of Genetic algorithms with other local search techniques is a potential improvement of performance, which is basically the essence of investigation of this MSc project.

- "A Genetic Algorithm Applied to Robot Trajectory Generation" (Davidor), is a project, whose core is the application of genetic algorithm techniques in the study case of a path planning problem of a robot arm, moving from one point in the space to another. The project uses variable-length chromosomes and devises some novel crossover operators.
- "Strategy Acquisition with Genetic Algorithms" (Grefenstette), is an application concerned with maneuvering a simulated airplane in order to evade a simulated missile. The project applied variable length chromosomes and has employed a special genetic algorithm, named "SAMUEL", capable of learning new techniques.
- "Genetic Synthesis of Neural Network Architecture" (Harp and Samad) is an interesting project that aimed to encode neural network architectures on binary chromosomes; the approach has deployed a genetic algorithm with variable length chromosomes.
- "A Genetic Algorithm Approach to Multiple Fault Diagnosis" (Liepens and Potter), is a project that has implemented a genetic algorithm for finding the most plausible combination of causes for alarms in a microwave communication system. The approach employed binary representations of chromosomes and has deduced that higher performances could be achieved with hybrid systems.
- "A Genetic Algorithm for Conformational Analysis of DNA" (Lucasius et al), is a project that has employed bit strings to encode molecular structures, in an environment aiming to develop a genetic algorithm for determining the structure of a sample DNA, based on spectrometric data.
- "Interdigitation: A Hybrid Technique for Engineering Design Optimization" (Syswerda), is a project that aimed to explore the use of genetic algorithms to solve the problem of scheduling interrelated activities in a laboratory. The approach of this project used an order-based chromosome to represent schedules.

2.14 GA Discussion

One of most advantageous scheme of GAs is their parallelism, i.e. the strategy that GAs use for travelling search spaces using an entire population of individuals rather than optimising just a single value. Their operation on encodings of the variables (genotype) rather then the variable itself (phenotype) is another creative and distinctive usage of GAs, which reduces the likelihood of getting stuck at a local extreme like it is the case with some other methods.

Genetic Algorithms are relatively easy to implement. Once the basic GA is implemented, the implementation of a new chromosome object is quite straightforward and problem-dependent. The same encoding scheme is generally used in GAs independently from the application's domain; an objective function is however required and should be implemented by the user. Conversely, for several reasons, choosing and implementing the chromosomes' representation and objective function can be confusing and difficult. On the other hand, one of GAs' disadvantages is their complex computational time as they can be very slow in some applications especially when compared to other traditional methods' time complexity. But the possibility of terminating the computation at any time eases the problem and makes the longer run time acceptable especially with faster computers.

Overview of Memetic Algorithms

2.15 Introduction: What are Memetic Algorithms?

"Memetic Algorithm" is a concept first introduced in 1989 (Pablo Moscato, 1989), the term "Memetic" having its roots in the word "meme" introduced in Richard Dawkins' book "The Selfish Gene" (1990) and which denoted the "unit of imitation" in cultural transmission. The essential idea behind Memetic Algorithms is the use of local search heuristics within a population-based strategy, such as Genetic Algorithms. Memetic Algorithms share most of the GAs characteristics although they introduce a new improvement procedure based on local search in the neighbourhood of newly generated individuals resulting from the recombination operators. The main difference between Genetic Algorithms and Memetic Algorithms is the approach and view of the information's transmission techniques. Whereas genetic information carried by genes is usually transmitted intact to the offspring (e.g. Genetic Algorithms), "memes" the base unit of Memetic Algorithms are typically adapted by the individual transmitting them. Just like in the real world, while individuals can inherits from their parents the eyes and skin's colour genetically transmitted to them intact, they will independently adopt new cultural dependent characteristics through their life span, such as their intelligence, their ability to drive and swim etc.

2.16 Why do we need Local Search?

Local Search techniques are mostly useful for controlling exponentially growing solution spaces, as it is the case in most practical problems. Another benefit of local search techniques is their ability to deal with the ambiguity of some problems' models and solve it, as well as they have proved to assure ease of use of problem specific knowledge compared to other classical optimisation techniques.

2.17 Memetic Algorithm Pseudocode

```
Initialise P randomly (P = Population)
For i = 1 to m (m = population size)
    Perform local search in the neighbourhood of i
        (i being the current individual)
        Evaluate fitness of i and its neighbourhood explored
        Make i the best individual found
End For
Repeat
    Select parents from P
    Generate offspring applying recombination
        to the parents selected
    If an individual is selected to undergo mutation,
        then apply local search
        Evaluate fitness of current individual and its neighbours
        Adopt best individual
Until stopping condition is reached
```

Figure 2.10: Memetic algorithm pseudocode

The Pseudocode described in figure 2.10, is the Pseudocode of a Memetic Algorithm coupled to a Genetic Algorithm (Digalakis and Margaritis, 2000)

2.18 Memetic Algorithm Applications

Although the common view of Memetic Algorithms categorizes them as a GA hybridised with a local search procedure, the concept of Memetic Algorithm is more general than that, as they can be used with any other meta-heuristic technique, and not just Genetic Algorithms, and they have been found to be of significant efficiency (Moscato, 1999) in certain domains like multiobjective optimisations compared to the Genetic Algorithms' performance. Memetic Algorithms were used in a variety of applications domains and they have showed efficiency in performance leading to good quality results. Table 2.3 is a list of general applications where Memetic Algorithms were applied and proved to be very efficient:

Memetic Algorithm Applications
Scheduling Problems
Transport Problems
Logistic Problems
Network Optimisation
Process Optimisation
Space Craft Trajectory Optimisation
Bio informatics
Planning Problems
Timetabling Problems
Evolvable Hardware And Hardware Design

Robotics
Telecommunications
Mechanical And Structural Engineering

Table 2.3: Memetic Algorithm Applications

Relatively similar work to this MSc project has been conducted (Kosmas Knödler et al) at the Department of Computer Architecture at Wilhelm-Schickard-Institute of Computer Science (contact: knoedler@informatik.uni-tuebingen.de, website: www.ra.informatik.uni-tuebingen.de.). The project, whose industrial partner is the BMW group Munich, consisted on implementing a Memetic Algorithm which aims to get the Optimal Calibration of modern automotive combustion Engines. In a brief description, the memetic algorithm used in the "Optimal Calibration of modern automotive combustion Engines" project consisted of a Genetic Algorithm hybridized with a "Hill Climbing" local search which sets a neighbourhood for some special parameters and iterates until a stopping criterion is met, while exploring the neighbourhood space, only accepting good solutions. Note that this Hill Climbing approach will be explored in this MSc project in addition to other commonly used techniques of local search.

2.19 Memetic Algorithm Advantages and Disadvantages

Memetic Algorithms have proved to be very useful and efficient in single-objective combinatorial optimisation. The main reason behind this success seems to be the process of local search, which has been proved (Borges and Hansen, 1998) to bring better solutions in several applications domains, compared to the performance of traditional Genetic Algorithms. The extension of Memetic Algorithms to multiobjective optimisation problems is very promising due to the fact that the process of local search exploits the global convexity, which has been used in single-objective optimisation to explain the high concentration of good solutions to some combinatorial optimisation problems in only a small fraction of the entire solution space. On the other hand, their use in nonlinear continuous multiobjective combinatorial optimisation problems is not very straightforward, especially because in these situations, the "global convexity" property does not hold anymore (Jaszkiewicz, 1998). Another notable disadvantage of Memetic Algorithms is the considerable amount of time and memory needed for the improvement of their performance (Jaszkiewicz, 2000).

2.20 Comparison between Memetic Algorithm and Genetic Algorithms

As an optimization technique, Genetic Algorithms simultaneously examine and manipulate a set of possible solutions. GAs constitute a robust technique, which can deal successfully with a wide range of problem areas, including those that are difficult to handle by other methods. They are not guaranteed to find the global optimum solution to a problem, but they are generally good at finding "acceptably good" solutions to various problem domains. Generally, where specialized techniques exist for solving particular problems, they are more likely to out-perform GAs in both speed and accuracy of the final result.

Although Genetic Algorithms are good at exploring the solution space due to their search from a set of candidate solutions and not just from a single point, they are not well suited for fine-tuning structures that are close to optimal solutions. Incorporation of local improvement operators into the recombination step of a Genetic Algorithm is essential to deal with such situations. Memetic Algorithms (MAs) are evolutionary algorithms that apply a separate local search process to refine individuals. Combining global and local search is a strategy used by many successful global optimization approaches, and MAs have in fact been recognized as a powerful algorithmic paradigm for evolutionary computing. In particular, the relative advantage of MAs over GAs is quite consistent on their complex search spaces.

CHAPTER 3- REQUIREMENTS AND ANALYSIS

3 Requirement and Analysis

3.1 Current System

The Multi Objective Genetic Algorithm's optimizer used in the Optimisation techniques for Gas Turbine Engine Control Systems project is a global search technique, which explores a relatively vague vector spaces of parameters' values by operating on individuals composed of a certain number, N, of parameters, while optimizing the values of a certain number, M, of objectives.

At the beginning of the Genetic Algorithm, each of the N parameters constituting the population's individuals is bounded by an invariable upper and lower limit value which should not be violated all along the optimization process, i.e. the Genetic Algorithm will operate on the individuals parameters by exploring new values within the predefined range. In this project, a binary representation of chromosomes is applied, and thus the multi objective genetic algorithm starts by creating a binary population composed of a certain fixed number of individuals. The GA assesses then the fitness of the newly created population by running the objective function on each of the individuals inhabiting the population. A fitness score is consequently allocated for each individual, which will be the basis of selection for the recombination process and the propagation to the next generations. Figure 3.1 summarizes the general functionality of the MOGA technique used in the system

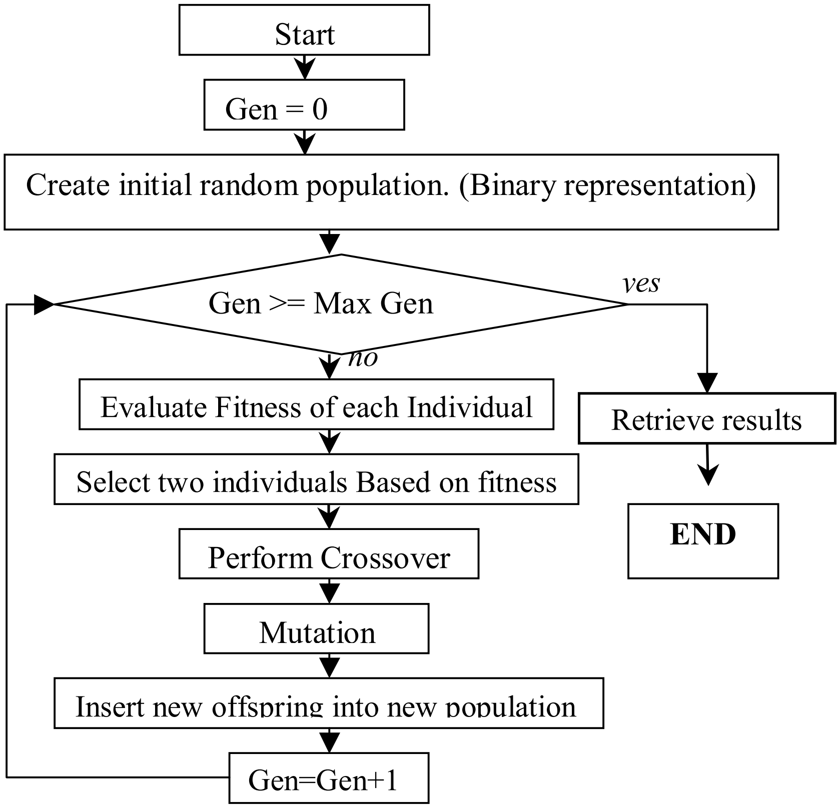

Start
Gen = 0
Create initial random population. (Binary representation)
Gen >= Max Gen
yes
no
Evaluate Fitness of each Individual
Select two individuals Based on fitness
Perform Crossover
Mutation
Insert new offspring into new population
Gen=Gen+1
Retrieve results
END

Figure 3.1: MOGA

1st- First an initial population is randomly generated and encoded into binary strings representing the genotypic values of its inhabitants.

2nd- A predefined function in the MOGA toolbox converts the binary representations of the population into real number values denoting the phenotypic values.

3rd- Each individual in the population is assessed through the objective function. (In Simulink).

4th– Each individual is assigned a fitness value (usually the value of the fitness value is determined in the range [0 1] denoting a probability) reflecting its performance assessed through the objective function.

5th- Couples of individuals are selected stochastically for reproduction. Each individual has a probability of being selected for recombination proportional to its fitness value, accordingly individuals with higher fitness values will have higher chance of being selected for breeding and vice versa.

6th- The genotypic representations of the selected individuals are bred via specific techniques such as crossover. Basically, this part denotes the genetic exchange between individuals.

7th- The mutation operator operates in a stochastic fashion on the newly generated offspring resulting from crossover; usually this operator takes place with relatively low probability just like in the biological environment.

8th- The objective function takes place again assessing the performance of the new individuals. The mechanism restarts again, following the same logical order, until a stopping criterion is met.

3.2 System Requirement

The main system requirement is to hybridize the Multi Objective Genetic Algorithm technique with a local search process, which aims to fine-tune locally the solution space vector of each individual's parameter for the exploration of new good solutions, thus improving the optimization results. Actually, research (Baker, 1998) has proved that Genetic Algorithm alone, or Evolutionary Algorithm in general, are not well suited for fine tuning optimization, partially because they tend to operate on large vector spaces of solutions, and thus in special situations starting with initial good guesses close to the global optimum, it is more likely that genetic algorithms will consume a lot of time evaluating and exploring sub-optimal solutions mainly due to their lack of local search tuning. This mechanism of hybridizing a genetic algorithm with a local search technique is more commonly known as a memetic algorithm.

3.3 Requirement Analysis

When it comes to hybridizing a genetic algorithm with a local search technique, several design issues will have to be carefully considered (Krasnogor, 2002).

As its name denotes, local search explore “locally” a subspace of values, or in other word the neighbourhood of a potential solution. The question that arises first is: How do we define the neighbourhood? Unfortunately this question cannot have a single answer, but it’s essentially related to the application domain under investigation.

Once the neighbourhood is defined, the other design issues that should be analysed when engineering a Memetic Algorithm are the following:

- Which individuals in the population should be improved by local search?

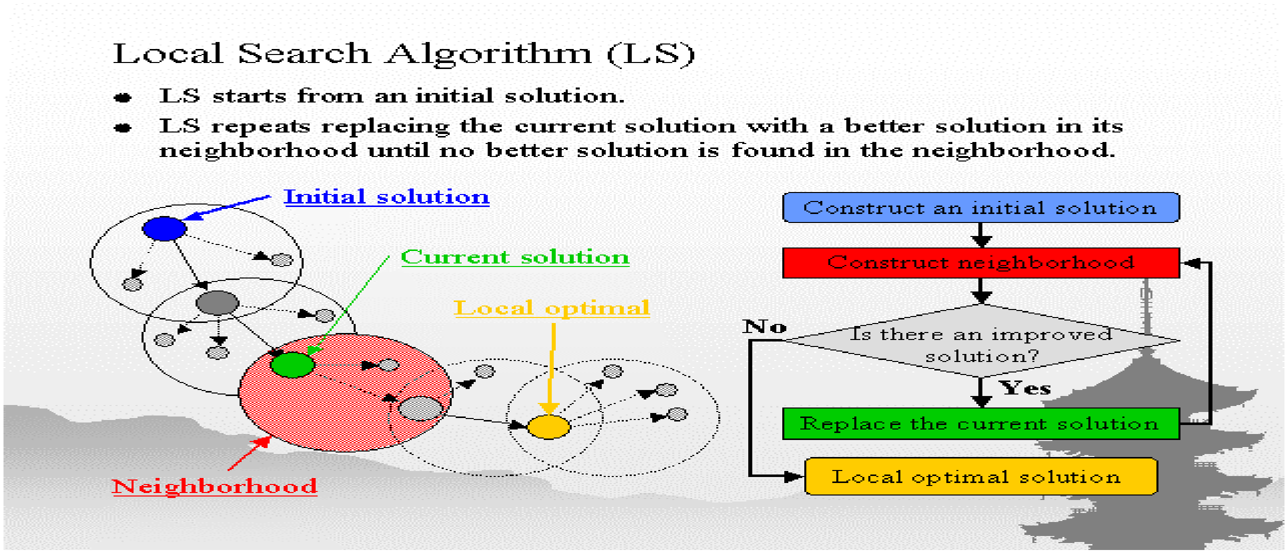
For the purpose of this project and due to its experimental nature it was decided to apply the local search algorithm on the entire population, and to experiment other approaches such as directing the local search towards relatively fit individuals in order to improve their performance and get better offspring later on if the time permits.

- What kind of local search technique should be hybridized to the genetic algorithm?

According to statistics and research (Krasnogor, 2002), it has been showed that the most commonly used local search techniques in Memetic Algorithm are the Hill Climbing techniques, Simulated annealing techniques and the Tabu Search techniques, although other extensions and techniques (GRASP, FAN…) are also used in a wide range of applications For this project it was decided to experiment with the three most commonly used local search techniques; Hill Climbing, Simulated Annealing and Tabu Search.

- How long should the local search run?

It turned out that this question is related to the nature of the local search technique used in the system. Experimentation and trials were dedicated to solve this conflict and further information will be explained in the following Chapters when the three kinds of local search will be discussed in details.



Local Search Algorithm (LS)

- LS starts from an initial solution.
- LS repeats replacing the current solution with a better solution in its neighborhood until no better solution is found in the neighborhood.

Figure 3.2: Outline of a General Local Search Algorithm

Note: The previous diagram is taken from the Internet.
The Author’s email is:umetani@amp.i.kyoto-u.ac.jp,

CHAPTER 4- DESIGN

4 Design

4.1 Introduction

Many Local search techniques were considered for the hybridization process with the genetic algorithm used in the actual system, and it was decided to start by adding the local search improvement phase after the recombination (Crossover) process for each individual of the population. The entire optimization process will be reshaped consequently to the following logical order at each generation:

1st- The objective function is processed for the entire population.
2nd- A fitness value is assigned to each individual based on its performance.
3rd- Couples of individuals are selected for recombination.
4th- Recombination process takes place, generating new offspring.
5th- The population is shifted to the local search process where each individual can be improved locally.
6th- A new population, potentially improved by the local search technique, is returned back to the global search phase of the genetic algorithm.
7th- The mutation operator applies to the newly returned population from the local search phase. And the cycle of Global search intersected with the Local Search- restarts again.

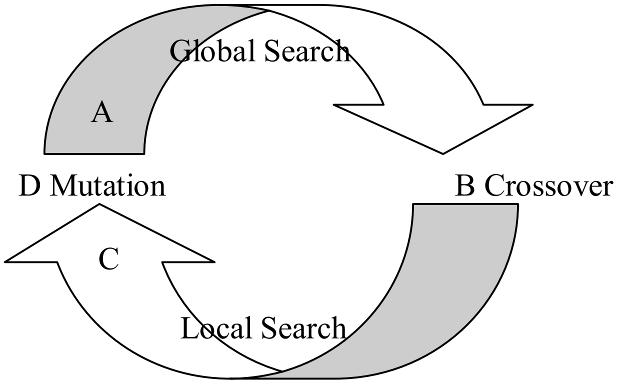


Figure 4.1: Global search/Local search cycle

The local neighbourhood of each parameter constituting an individual is to be set by the local search algorithm, which receives as one of its argument the global range of definition for each parameter from the Genetic Algorithm. After defining the local range of search for each individual's parameter, the local search algorithm starts to explore the neighbourhood in a random fashion by perturbing the parameters of each individual by a certain allowed fraction within the local range, creating an entire new population. The objective function will be processed consequently on the new potential solutions assessing their performance, and ultimately an acceptance phase depending on the nature of the local search technique used will decide whether to switch the old population received from the Genetic Algorithm with the newly generated population in the local search or not.

The functional diagram shown in figure 4.2 summarizes the entire process of a GA coupled with a local search technique:

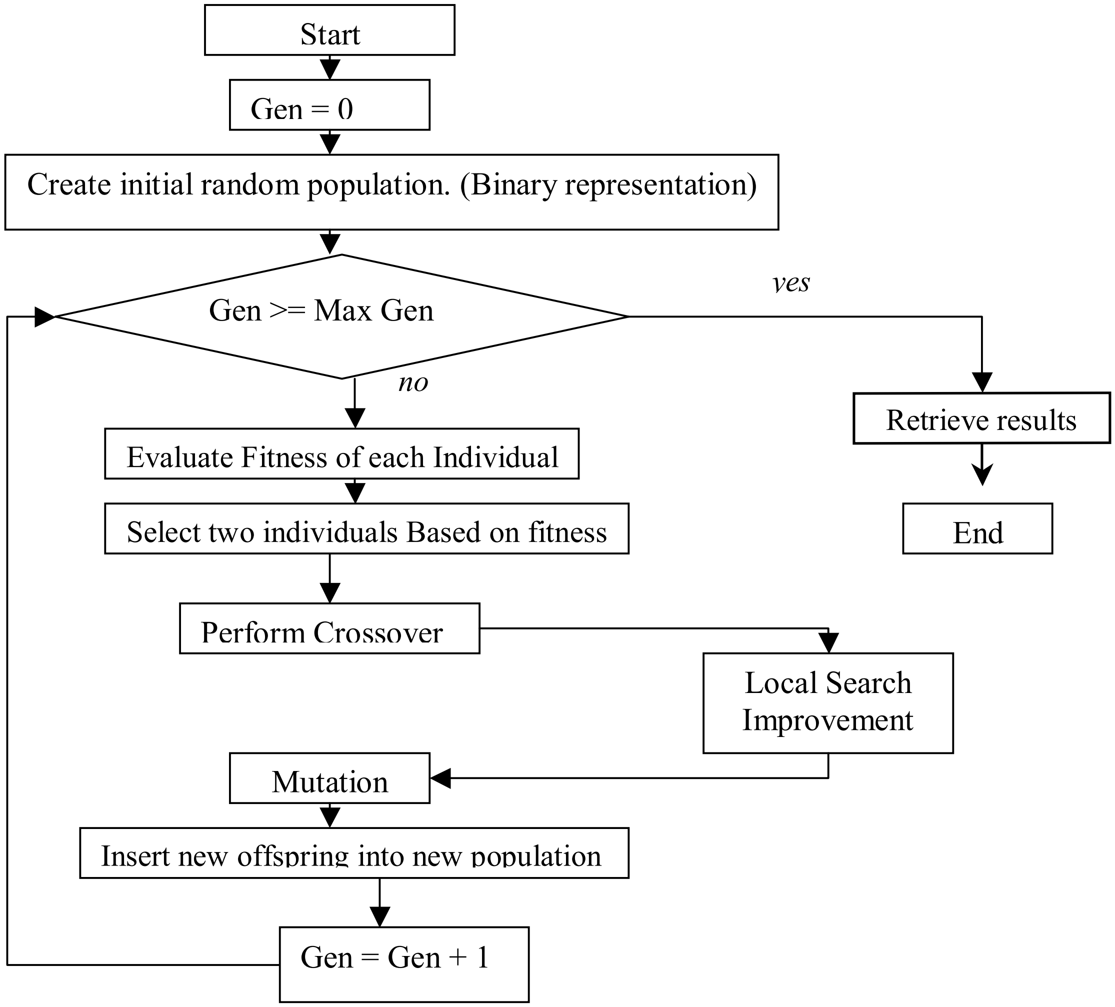

Start
Gen = 0
Create initial random population. (Binary representation)
Gen >= Max Gen
yes
no
Retrieve results
End
Evaluate Fitness of each Individual
Select two individuals Based on fitness
Perform Crossover
Local Search Improvement
Mutation
Insert new offspring into new population
Gen = Gen + 1

Figure 4.2: Genetic Algorithm Combined with local search

In the case of this MSc project, the neighbourhood of an individual parameter's value was decided to constitute 10% of the global search range of that specific parameter. For example, let's suppose an individual phenotype is formed of 10 parameters (Figure 4.3),

Parameters									
P1	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P10**

Figure 4.3: Individual's Phenotype.

And that the first parameter of an individual is set to have a phenotypic range of [0 20] which is fixed during the whole optimization process, i.e. the global search applied by the MOGA for the first parameter of an individual is only defined between 0 and 20:

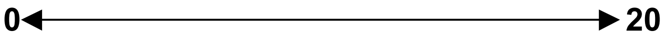


Figure 4.4: Global search Range for parameter P1

Let's suppose at one point of the MOGA process, the individual set to be passed to the local search algorithm for improvement has a numerical value of 5 for its first parameter P1. The local search will have to search a range of width equal to 10% of the MOGA global search around the value of P1 constituting the actual individual.

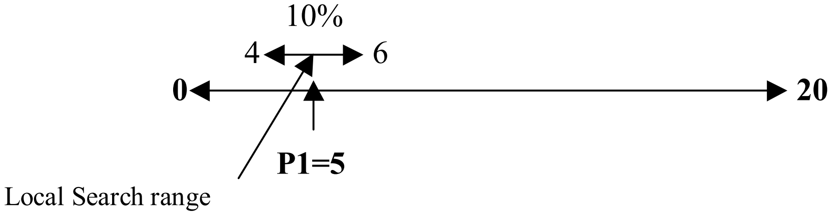


Figure 4.5: Actual P1 value of the individual under processing

In the following three sections, the three local search techniques hybridized with the genetic algorithm optimizer will be discussed in details.

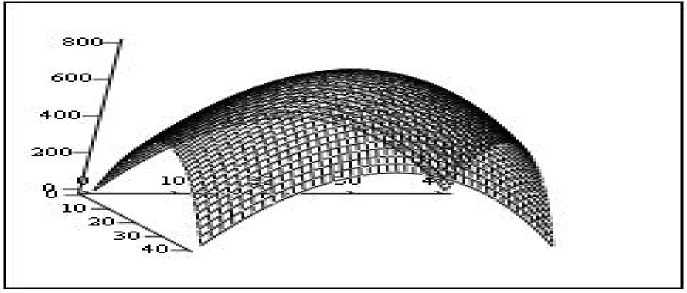
4.2 Hill Climbing Local Search

Hill Climbing is basically considered as the simplest heuristic local search technique.
It is usually applied in problem domains where the ultimate objective is finding a goal state regardless of the path followed to reach it. This local search technique is not an optimal one as it may choose suboptimal paths to find a solution and is usually described as a "Greedy" search mechanism. The time complexity for Hill Climbing algorithm is usually proportional to the cost of the longest path for a solution (Spears, 2001).
Hill Climbing technique is a process based on iterative improvements. Its functionality is pretty much simple and straightforward. Improving a given state **A**, the Hill climbing local search systematically generates a neighbourhood for that state and moves to the first neighbour with lower cost in case of minimization problems and vice versa in case of maximization. The process terminates when the current state has no neighbours with better fitness. Figure 4.6 illustrates a simple Hill Climbing Pseudocode (Knödler et al):

```
Procedure Hill Climbing
Repeat N times
    Generate neighbouring solution S' from S
    If Fitness(S') > Fitness(S)
    Then S = S'
End Repeat;
Return S;
```

Figure 4.6: Hill Climbing Pseudocode in a maximization problem

Hill Climbing seems logical and beneficial especially in situations where the search space is of simple nature with no more than a single maximum or minimum; see Figure 4.7:



.

Figure 4.7: Search space with a single maximum point.

Unfortunately real search spaces can be very complex and represent many local maxima or minima; example Figure 4.8

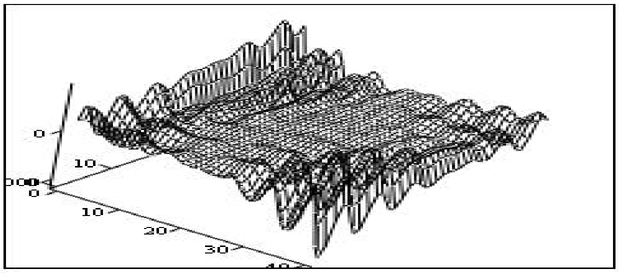


Figure 4.8: Complex Search space.

In these situations, Hill Climbing local search faces the problem of getting trapped at local maxima or minima, totally different from the global ones (Spears, 2001).
Figure 4.9 shows a situation where a hill climbing local search get trapped at a local minimum and not being able to find a neighbour with lower fitness in a minimization problem.

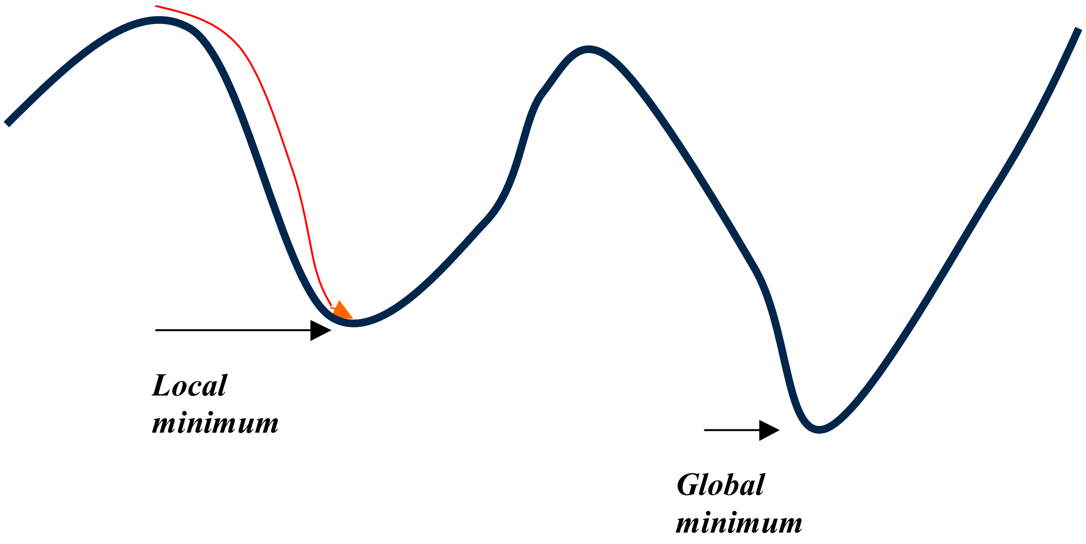


Figure 4.9: Local search process trapped at local minimum

Another Local Search technique, which deals with this problem, is the Simulated Annealing technique described in section 4.3.

4.3 Simulated Annealing Local Search

4.3.1 Annealing Process

“Annealing” is a term essentially borrowed from thermodynamics, and it denotes the process where a solid material is melted under high temperatures and then gradually cooled by slowly reducing the temperature, or the process in which a liquid freezes and crystallizes. The key point of the annealing process is the slow decrease of temperature, which allows the liquid molecules to lose their thermal mobility gained at high temperatures and form a pure ordered crystal, which constitutes the minimum energy for the system. On the other hand, if the cooling process wasn't slow enough, the liquid ends up in suboptimal energy levels.

4.3.2 Simulated Annealing

Simulated Annealing also known as Monte Carlo annealing, probabilistic hill-climbing and stochastic relaxation was first introduced by Metropolis et al in 1953. It is an optimization method originated from the Annealing process of thermodynamics, where a solid achieves thermal equilibrium after being cooled gradually under slowly decreasing temperatures. The primary advantage of simulated annealing is its ability to escape local optima, in complex spaces representing different local optima.

4.3.3 How does Simulated Annealing Works?

Just like the Hill Climbing technique, Simulated Annealing (S.A) starts by perturbing a potential solution or state by moving to another state in its neighbourhood. The objective function is calculated for the new state and compared to the objective function of the previous state. In case of minimization, if the new state has a lower objective function than its predecessor, the new state is accepted replacing consequently the older one, and basically a step downwards is taken towards the global or local minima. On the other hand if it turns out that the new state's performance is worse than its predecessor, oppositely to the hill climbing technique which directly rejects bad steps, simulated annealing may accept the new worse state based on a probabilistic model, i.e. a step upwards may be allowed in a minimization problem or a step downwards may be accepted in maximization situations (Sundermann, 1996).

The simulated annealing process requires an initial temperature variable T to start with and a cooling schedule. At each temperature T a certain predefined number, N, of improvement iterations is performed on the individuals of a population. T will gradually be decreasing by a small constant multiplication factor K that constitutes the cooling schedule. For example, a logical Simulated Annealing situation may start with an initial temperature T =1, a cooling schedule as follows: $T \leftarrow KT$ where a suitable value for K might be 0.95 and a number of iteration N at each temperature equal to 100.

Let's consider the two following cases:

The simulated annealing algorithm is processing a state. First the state's value is perturbed by moving to another local state in its neighbourhood. The objective function is performed on the new state to assess its performance.

1st case: The performance of the new state is better than its predecessor

Result:

The new state is accepted independently of the temperature value T; totally similar to the Hill Climbing process.

2nd case: **The performance of the new state is worse than its predecessor**

Result:

First a random number N denoting a random probability is drawn from a uniform random distribution on the interval [0, 1].

Then the following Metropolis criterion is checked deciding whether to accept the new state or not:

$$\textbf{If } (P=\exp(-\Delta E/T))>N \qquad \textbf{(4.1)}$$

Where T denotes the actual temperature and ΔE denotes the difference between the fitness value or "Energy" of the new state and the previous state:

$$\Delta E = \text{Fitness (New State)} - \text{Fitness (Old State)} \qquad \textbf{(4.2)}$$

Note that in the case of a minimization problem, a state is better than another state if its fitness value is lower than the other state's fitness value and vice versa in maximization situations. In other words a bad state will be only accepted if the Metropolis criterion (equation 4.1) is valid, and consequently a step upwards may be taken to escape local minima; see Figure 4.10.

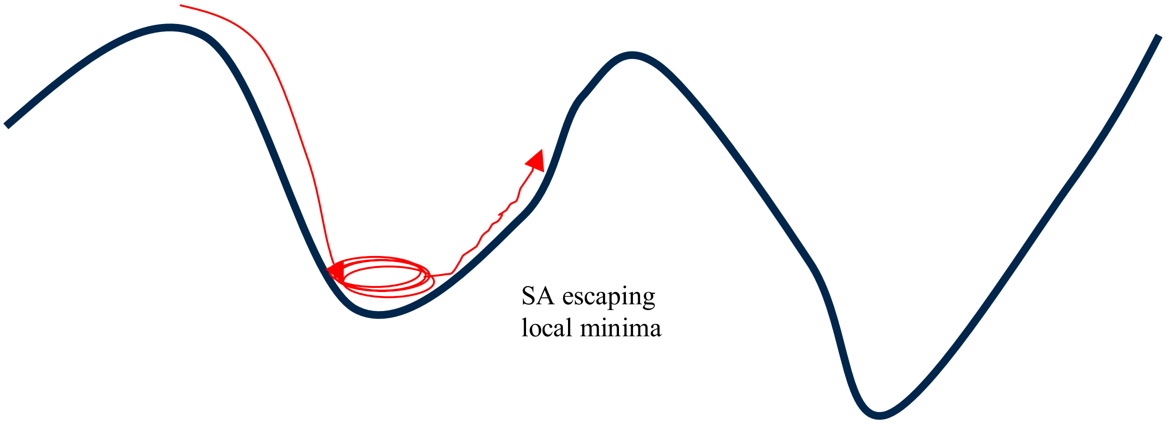


Figure 4.10: Simulated annealing process escaping local minima

The idea behind this acceptance criterion (Spears, 2001), is that due to the decrease of temperature value all along the process, at very low temperatures, ultimately the exponential function, Exp ($-\Delta E/T$), will be tending to zero as the equation $-\Delta E/T$ will be tending to minus infinity, and consequently the validity of the criterion will be nearly impossible, and thus the simulated annealing process will be acting like a Hill Climber local search at low temperatures, basically by only accepting good solutions.

Although Simulated Annealing algorithms are relatively simple to implement, careful attention should be taken when configuring the two important parameters constituting the cooling schedule which represents the core of the simulated annealing process, these two parameters are the step size for the temperature perturbation, i.e. the temperature minimization factor K and the initial temperature T. Most researchers suggest step sizes and T values settings that allow approximately 80% of the poor moves to be accepted, although these two parameters are totally subjective to the nature of the application domain. Extensive results about widely used Simulated Annealing schedules and their corresponding performances can be found at the following website: http:// www.ingber.com. Other elements to consider while implementing a Simulated Annealing algorithm are that although SAs are proved to be efficient and reliable in several applications domains, they require much more response time or objective function evaluations then other techniques which may be tedious and time consuming in some applications' environments, especially where the objective functions are of complex nature and time consuming. On the other hand, unlike other optimization techniques that attempt to make intelligent moves in the solution space, SA, which has been termed a "biased random walk", takes the steps in a completely random fashion.

A variation of the simulated annealing process (Spears, 2001) replaces the Metropolis Criterion:

$$P = e^{(-\Delta E/T)}$$

With the logistic function

$$P = \frac{1}{(1 + e^{-dE/T})} \quad \textbf{(4.3)}$$

also known as the sigmoid function (Figure 4.11), as an acceptance criterion for taking uphill or downhill moves. Simulated Annealing algorithms, which use the sigmoid function, are usually called “Stochastic Hill Climber”.

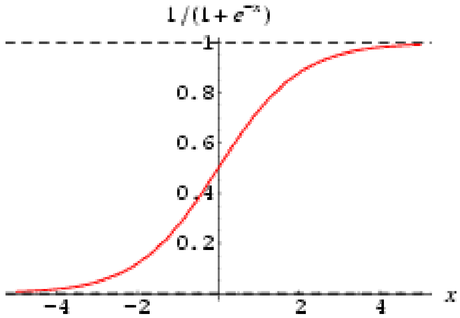


Figure 4.11: The Sigmoid Function

4.3.4 Simulated Annealing Pseudocode

The pseudocode in Figure 4.12 illustrates a basic simulated annealing process of a minimization problem:

```
Procedure simulated annealing

Begin
 Initialize temperature T
 Starts at a current point or solution
 Best_Solution ← Current_Solution
 Repeat for a certain number of iterations N
    Select a new point or Solution randomly in the neighbourhood of the Current_Solution
    If Performance (New_Solution) < Performance (Current_Solution)
      Current_Solution ← New_Solution

    Else if random [0,1) > $e^{(-(performance(New\_Solution)-performance(Current\_Solution))/T)}$
      Current_Solution ←New_Solution

Temperature T ← schedule (temperature)
 Until Temperature < halting-criteria
End
```

Figure 4.12: Simulated Annealing algorithm.

4.4 “Tabu Search” Local Search

4.4.1 Tabu search: Introduction

Despite their simplicity, “Hill Climbing” techniques have uncovered the problem of getting stuck at local optima in search spaces of complex natures representing multiple points of inflection. In order to solve that common problem, Simulated Annealing technique has provided the possibility of accepting bad solutions in order to go uphill to escape a local minimum or downhill in case of local maxima. Unfortunately Simulated Annealing technique has uncovered yet another problem which could not be solved by the added expressive functionality of the technique. It has been demonstrated that in certain specific scenarios, Simulated Annealing can get trapped at local optima regions (Prasetio and Rhone, 2002). Due to the random nature of the step-taking process of the Simulated Annealing technique, in some situations, the SA algorithm might end up oscillating in a local optimum region by constantly making steps forwards followed by steps backwards or upwards and then downwards. The following scenario better illustrates the problem:

In a minimization problem, the ultimate goal of a Simulated Annealing algorithm is to make steps towards the global minima while accepting upwards steps based on a probabilistic model in order to avoid entrapment at local minima. But due to the random nature of the movement, at local minima, the algorithm may accept a bad move and consequently go up the hill escaping the local minima, and then go back again to the local optima region by a random backward step, ending up in a cycling situation which may not halt ever.

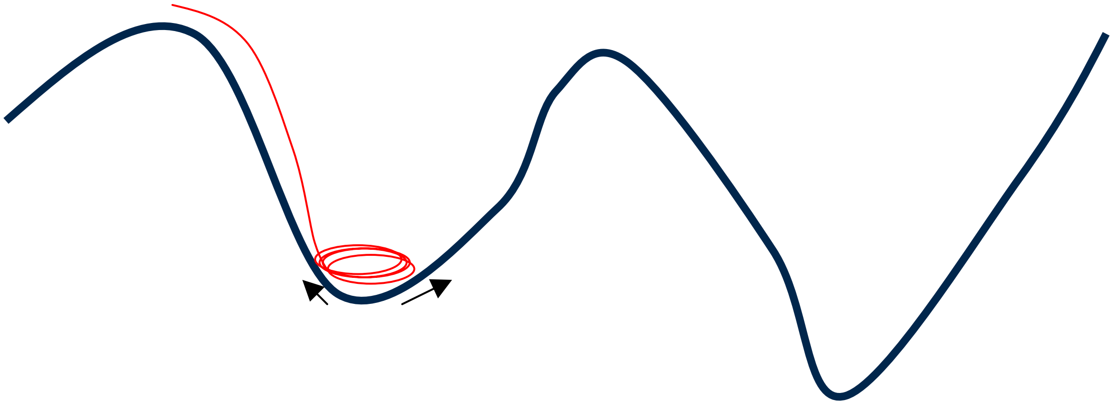


Figure 4.13: SA trapped in a cycling situation, which might run indefinitely

In order to solve this conflict it was necessary to set a penalizing process, which forbids certain paths to be taken, and eliminates these cycling situations, which is the basic idea behind "Tabu Search" algorithms.

Tabu search is a meta-heuristic search function, a relatively new technique originated around 1977 (Glover, 1977), and which is still being researched and under investigations. As mentioned previously, the main additive improvement of Tabu search techniques is their ability to escape indefinite cycling situations. Its main concept is that there is no point of accepting a bad solution or step unless it is taken to escape cycling scenarios by avoiding a path already visited. Thus the need for a certain internal memory, which holds the paths already visited, emerges. Tabu search algorithms keep track of the previously visited paths by recording recent moves into a Tabu list, which constitutes the main element of a Tabu search algorithm. Tabu derived from the word "taboo" is generally used to describe something which is prohibited and inviolable. The Tabu list, usually implemented as an array, stack or queue, can be thought of as traffic law that forbids some paths, usually previously visited ones, to be taken, which ultimately eliminates the risk of oscillation or cycling. Figure 4.14 (Prasetio and Rhone, 2002) summarizes the iterative functionality of Tabu search algorithms:

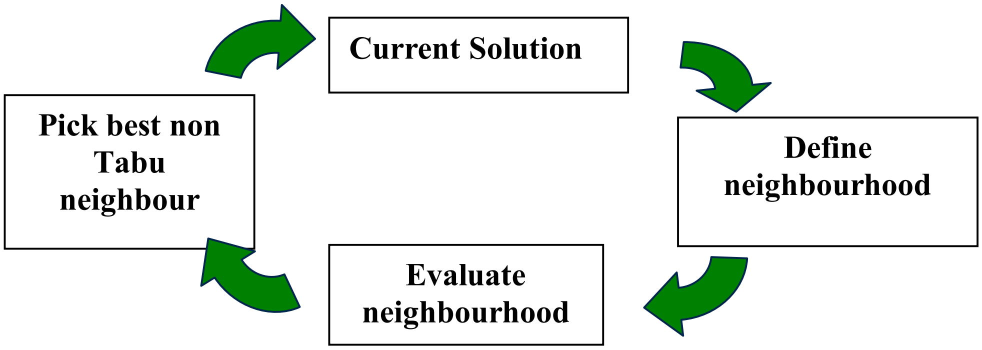


Figure 4.14: Tabu Search Functionality

The major issues that should be considered when designing a Tabu search algorithm are basically related to the Tabu list, its organization, its length and implementation. That is besides the issues that relate to the greedy search algorithm itself, and the type of movement that should be applied for “jumping” from a solution to another. The Tabu list’s length is particularly a very delicate parameter, which should be carefully configured, as a wrong choice of the list’s length may lead to an inefficient algorithm. Research (Thesen, 1998) has shown that a Tabu list’s length in the range 7 to 15 is usually a suitable choice for a wide range of applications.

4.4.2 Pros and Cons of Tabu Search

To put in a nutshell, Tabu search algorithms are usually a very efficient approach for tracking good solutions most likely undetectable by other mechanisms. The Tabu search is not bounded by linearity, as local optima situations and cycling problems are preventable. In general, compared to other optimization techniques, Tabu search algorithms usually yield better quality results. On the other hand, Tabu search algorithms possess their own constraints. When designing a Tabu search, the designer should keep in mind that these optimization techniques do not guarantee optimality, rather the decision whether to stop the search process or not remains to the user to decide. In addition, just like Simulated Annealing techniques, Tabu search mechanisms requires a huge amount of performance evaluation, and consequently the running time of these algorithms in applications with complex objective functions may be unbearable.

4.4.3 Tabu search Pseudocode

Figure 4.15 illustrates a basic Tabu search Pseudocode, (Arne Thesen, 1998)

```
Initialize
    Identify initial Solution
    Create empty TabuList
    Set BestSolution=Solution
    Define TerminationConditions
    Done=false
Repeat
    If Value of Solution > Value of BestSolution Then
        BestSolution=Solution
    If no TerminationConditions have been met yet Then Begin
        Add Solution to TabuList
        If TabuList is full Then
            Delete oldest entry from TabuList
        Find NewSolution by some transformation on Solution
        If no NewSolution was found OR
            If no improved NewSolution was found for long time Then
                Generate NewSolution at Random
        If NewSolution not on TabuList Then
            Solution = NewSolution
    End
    Else Done = true
Until Done = true
```

Figure 4.15: Tabu Search Pseudocode

CHAPTER 5- IMPLEMENTATION AND TESTING

5 Implementation and Testing

5.1 “Hill Climbing” Local Search Implementation

The first approach for hybridizing the multiobjective genetic algorithm technique used in the current system was to couple it with a simple Hill Climber local search which aims to make local improvements on the population’s individuals at each generation of the global search phase constituting the MOGA. As mentioned earlier in Chapter 4, it was decided to integrate the local search improvement just after the recombination process and before the mutation operator takes place. In other words, the out-coming population from the crossover process is passed to the local search mechanism in order to improve its performance and return it back to the genetic algorithm phase, where the mutation operator takes over again, by modifying, based on a probabilistic model, some bits of the newly returned population from the local search process, ensuring the possibility of searching any region in the search space.

The logicality of the Hill Climber local search is very simple; the algorithm defines the neighbourhood for each individual constituting the population passed from the MOGA for local improvement. More accurately, the local search algorithm sets the range or neighbourhood for each of the variables composing an individual. In the current system, the phenotypic representation of an individual is constituted of ten variables:

Variable									
V1	V2	V3	V4	V5	V6	V7	V8	V9	V10

Table 5.1: Individual’s Phenotype.

Note that the local search algorithm as well as the crossover operator, the objective function and the fitness function operates on the phenotypic representation of the individuals, i.e. the real valued encoding. Instead of processing just one individual at a time, the local search algorithm process the entire population, by setting the neighbourhood for each of the ten variables composing each individual in the population. The total number of individual per population was set to 50 individuals, which is a typical population length in genetic algorithms. The population is thus to be thought of as a matrix or two dimensional array composed of fifty rows denoting the total number of individuals, and ten columns denoting the number of variables which constitutes an individual:

Number of variables per individual = 10										
Number of individuals 50	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10

Table 5.2: A population of 50 Individuals

Each of the individuals variables are bounded by an upper global limit and a lower global limit defined at the beginning of the genetic algorithm. These global limits are totally problem-related and should be conserved all along the process; otherwise the algorithm might end up processing irrelevant data which might be totally illogical and irrational. It was decided that for the local search process, the search range should be reduced to 10 percent of the global search range at each variable. For a better understanding let's consider the following scenario:

Table 5.3: GlobalMax:

50	**70**	**20**	**100**	**50**	**88**	**60**	**30**	**90**	**100**

Table 5.4: GlobalMin:

-50	**0**	**-40**	**20**	**0**	**0**	**10**	**-30**	**-20**	**0**

GlobalMax and **GlobalMin** are two arrays containing correspondingly the upper and lower limits of each of the 10 variables constituting an individual.

And let's suppose that the following reduced (for simplicity) population was generated by the crossover operator and passed to the Hill Climber local search to improve it:

Table 5.5: Actual Population

10	**2**	**18**	**50**	**30**	**30**	**40**	**-10**	**19**	**60**	**Individual 1**
30	**50**	**-15**	**40**	**20**	**20**	**20**	**10**	**0**	**35**	**Individual 2**

Since the local range is locally set to be equal to 10 percent of the global range, the Hill Climber local search will first set the upper local and lower local limits for each of the two individuals' variables by generating two matrices with the same dimensions as the population, one registering the local lower limits for the variables and the other storing the local upper limits:

In the case of the previous reduced population of two individuals, the Hill Climber algorithm will produce the following local neighbourhoods.

Table 5.6: Setting the local search neighbourhood

50	**70**	**20**	**100**	**50**	**88**	**60**	**30**	**90**	**100**	**GlobalMax**
10	**2**	**18**	**50**	**30**	**30**	**40**	**-10**	**19**	**60**	**Individual 1**
-50	**0**	**-40**	**20**	**0**	**0**	**10**	**-30**	**-20**	**0**	**GlobalMin**

The width of the local range for each variable is equal to the absolute difference of the GlobalMax and the GlobalMin of that variable multiplied by 0.1:

Local Range width= (Abs (GlobalMax-GlobalMin)*0.1)

10	**7**	**6**	**12**	**5**	**8,8**	**7**	**6**	**11**	**10**	**Local Range Width**

Table 5.7: Local Range Width

Local upper and lower limits for Variable 1 of Individual 1:

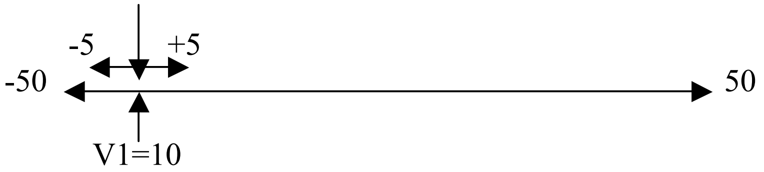


Figure 5.1: Local Neighbourhood Setting case 1

So basically the upper local limit for variable 1 (V1) will be equal to **10 + 5 = 15** and its local lower limit is equal to **10 - 5 = 5**. So the local range width is to be thought of as sliding range all along the global range width, and which aims to be centred at the actual variable's value in order to determine its local upper and lower range limits. In extreme cases where the actual variable's value is at a smaller distance than the half of the local range width to an upper or smaller global range value, the local upper or lower limit will be, depending on the situation, set to the global limit while the other limit will be set to the difference of the local width range and the distance between the current variable's value and its nearest global limit, the second variable (V2) in the previous example summarizes the point:

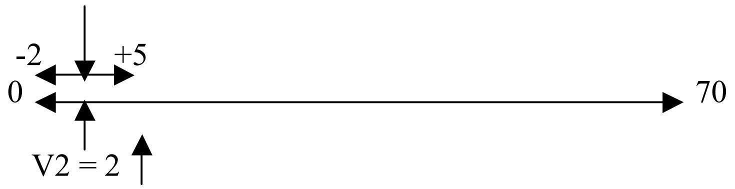


Figure 5.2: Local Neighbourhood Setting case 2

Since the distance between V2 and its global lower limit is less than its local width range divided by 2 (local width range = 7), the local lower limit for V2 is set to be equal to its global lower limit, while its local upper limit is equal to its local width range minus its distance to the global lower limit→ V2's local limit = 7 – 2 =5

Consequently the Hill Climber local search will set up the local bounds for each variable of the actual solution by storing it in the corresponding matrices:

Table 5.8: Actual Population

10	2	18	50	30	30	40	-10	19	60	**Individual 1**
30	50	-15	40	20	20	20	10	0	35	**Individual 2**

Table 5.9: LocalMin

5	0	15	44	25	25,6	36,5	-13	13,5	55	**Individual 1**
25	46,5	-18	34	15	16,4	16,5	7	-5,5	30	**Individual 2**

Table 5.10: LocalMax

15	7	21	56	35	34,4	43,5	-7	24,5	65	**Individual 1**
35	53,5	-12	46	25	24,4	23,5	13	5,5	40	**Individual 2**

In order to function properly and set the local search range for each individual, the implemented Hill Climber local search required the allocation of nine parameters, among which are the two arrays "ubounds" and "lbounds" storing the global upper and lower limits of the variables which are defined at the beginning of the genetic algorithm.

Figure 5.3 illustrates the code portion that creates the local neighbourhood for each individual:

```
for i=1:nind
      for j=1:nvar
        if (phen(i,j) + delta(j)/2 <= ubounds(j)) & (phen(i,j) - delta(j)/2 >= lbounds(j))
          local_ubounds(i,j) = phen(i,j) + delta(j)/2;
          local_lbounds(i,j) = phen(i,j) - delta(j)/2;
        else if (phen(i,j) + delta(j)/2 > ubounds(j))
            temp_prime = ubounds(j) - phen(i,j);
            temp_prime_prime = delta(j) - temp_prime;
            local_ubounds(i,j) = ubounds(j);
            local_lbounds(i,j) = phen(i,j) - temp_prime_prime;
          else if (phen(i,j) - delta(j)/2 < lbounds(j))
              temp_prime = phen(i,j) - lbounds(j);
              temp_prime_prime = delta(j) - temp_prime;
              local_lbounds(i,j) = lbounds(j);
              local_ubounds(i,j) = phen(i,j) + temp_prime_prime;
            end
          end
        end
      end
    end
```

Nind = number of individuals of the population
Nvar = number of variables that constitutes an individual
Phen = A matrix of dimension (Nind, Nvar) containing the actual population.
Delta = An array of length Nvar containing the local range width for each variable.
lbounds = An array of length Nvar containing the global lower bound of the variables.
Ubounds =An array of length Nvar containing the global upper bound of the variables.
Local_lbounds = A matrix of dimension (Nind, Nvar) containing the local lower bounds of the variables constituting the individuals of the entire population.
Local_ubounds = A matrix of dimension (Nind, Nvar) containing the local upper bounds of the variables constituting the individuals of the entire population.

Figure 5.3: Local neighbourhood setting's code

After having set the neighbourhood for each of the population's inhabitants, the next task of the Hill Climber local search is to create a new population by randomly perturbing the variables of each individual in its local range previously determined, for the ultimate goal of improving the performance of the actual population.

In Matlab the following command shown in equation (5.1)

$$\mathbf{X = a + (b - a) * rand\ (n)} \qquad \mathbf{(5.1)}$$

generates a matrix named X of dimensions n by n whose elements are generated by a uniform distribution of random numbers on a specified interval, in this case [a, b]. In general the function **rand** generates arrays and matrices whose elements are uniformly distributed in the interval (0, 1). Multiplying the output of rand (n) by (b-a) consequently generates random numbers in the range (0, width of the interval (a, b)). Finally adding the lowest bound of the desired interval (in this case a), slides the random numbers generated between zero and the width of (a, b) to the desired interval, i.e. (a, b).
For the Hill-Climbing algorithm, the goal was to randomly modify each individual's variable in a predefined local range, i.e. (LocalMin, LocalMax).
Figure 5.4 illustrates the process of perturbation of the individuals' variables in order to move to another state or solution in the hill-climbing algorithm:

```
for i=1:nind
   for j=1:nvar
      temp_ind(i,j) = phen(i,j);
      while(temp_ind(i,j) == phen(i,j))
         a = temp_lbounds(i,j);
         b = temp_ubounds(i,j);
         temp_ind(i,j)=a + (b -a) * rand(1);
      end
   end
end
```

Nind: number of individuals in the population
Nvar: number of variables constituting an individual
Phen: matrix of dimensions (Nind, Nvar) containing the entire Actual population
temp_ind: matrix of dimensions (Nind, Nvar) containing the Modified population

Figure 5.4: Step taking process.

A new potential population is thus created. At this point of the Hill Climbing algorithm, the objective function should be applied to the new population in order to assess its performance in the application domain, to decide whether to accept the new population or not. Note that the objective function is a problem-related process, which should be designed by the user, not the Hill Climbing designer.

In the current system, the objective function, already implemented at the Automatic Control and Systems Engineering Department at The University of Sheffield, assigns a fitness score to each individual, by simulating its performance onto engines' diagrams (implemented in Simulink).

Figure 5.5 illustrates the objective function and the fitness function being applied to the new population created by the Hill Climbing algorithm:

```
for indno=1:nind;
    objv_temp(indno,:) = xobjvfn(local_ind(indno,:));
  end;

  E_new=rank_prf(objv_temp,goalv,priorityv);
```

Figure 5.5: performance evaluation

Xobjvfn is the objective function, which simulates the engines response to the new population. The objective function assesses one individual at a time (the new individuals being stored in *local_ind* matrix), and evaluates its performance by controlling and testing a number of objectives. Note that this function assigns a raw fitness value to each individual, and returns a matrix *obj_temp* of dimension (Nind, Nobj) where Nind denotes the number of individuals in the population, and Nobj denotes the number of objectives functions being evaluated. The function *rank_prf* takes as one of its arguments the objective matrix produced by the objective function, and transforms the raw objective values of each individual into fitness scores. Basically the problem is a minimization situation, i.e. the individual with the least fitness value is the fittest one. The smallest fitness value that can be produced by *rank_prf* is zero. Thus an individual is better than another one if its fitness value, assigned by the *rank_prf* function, is lower, or in another words if the difference of fitness functions is negative. At the end the fitness of the new population is compared to the fitness of the old one, and only the new individuals with better fitness values then their predecessors are accepted (see Figure 5.6), which is the typical acceptance process of a Hill Climbing algorithm.

```
for i=1:nind
   if(delta_E(i)<0)  %it means the new individual performed better...the best rank is zero
        phen(i,:) = temp_ind(i,:);  % the new individual replaces the old one
        E_old(i) = E_new(i);        % the new individual's fitness replace the old one's.
    end
end
```

Figure 5.6: Acceptance process.

5.2 "Simulated Annealing" Local Search Implementation

The second approach for hybridizing the genetic algorithm of the current system with a local search process consisted on implementing a Simulated Annealing (SA) algorithm which is basically a more advanced and sophisticated process compared to the Hill Climber techniques, especially because of its added feature which allows the escapology from local optima. At the heart of the Simulated Annealing mechanism is an analogy with essential principles of thermodynamics, which describes the process by which solid materials are melted and then allowed to cool down through a slowly decreasing temperature's environment. The Simulated Annealing techniques and the Hill Climbing techniques represent a lot of similarities especially due the randomness of the "step taking" process from one solution to another. Similarly to the Hill Climbing local search, the Simulated Annealing improvement algorithm was interleaved between the crossover operator and the mutation operator in the genetic algorithm process. First the SA local search algorithm starts by defining the local range of search for each of the variables constituting the populations' individuals. The same neighbourhood setting process employed in the Hill Climber local search was applied in the Simulated Annealing approach.

Differently to the Hill Climbing approach, which starts by perturbing the population after having defined the local neighbourhood, the Simulated Annealing algorithm, requires the initialization of a temperature variable, *Temp***,** and a cooling schedule which decides the decreasing rate of the temperature all along the process. Note that the initial temperature initialization and the allocation of a cooling schedule are the major two requirements of a Simulated Annealing algorithm, they are totally problem related and thus one should make careful consideration choosing these features. Due to the experimental nature of the project, it was decided to initialize the initial temperature to 1 (Figure 5.7), which is a recommended value, based on previous research and experimentation in the field. The temperature decrease step was set to 0.02, which is a relatively small and suitable decrease step, for an annealing schedule. The minimal temperature was set to zero, although it would be more suitable in other programming environments (other then Matlab) to make it 0, 02 in order to avoid special warnings or error messages especially with operations including divisions, (i.e. division by zero) when the temperature reaches its minimum value.

```
%set initial temperature
 Initial_Temperature=1;
 % set up temperature schedule
 T_schedule=[1:-.02:0];
```

Figure 5.7: Initial temperature and temperature schedule settings

Another essential element of Simulated Annealing techniques is the initialization of a number of search iterations N that should be accomplished at each temperature. In the workspace of this project it was decided to experiment with a number of iterations equal to 100 for each temperature. After the initialization of the initial temperature Temp, the temperature schedule and the number of search improvement's trials per each temperature are set; the process starts by randomly modifying the actual population by moving to adjacent values in the local search range, applying the same approach used in the Hill Climber algorithm (Figure 5.4). The next step of the process consists of the objective function's assessment for the new randomly created population; note that the Simulated Annealing is a very long and time consuming process due to the requirement of huge amounts of objective function's evaluations. In the case of the previous initializations, (*Initial_Temp =1*, *Temp_schedule: Temp→Temp – 0, 02 until 0, and number of iteration N per temperature =100*), the objective function was evaluated 5000 times for one single generation of the global search phase of the MOGA.

The final phase of the SA local search consists of the acceptance step, which oppositely to the Hill Climbing's acceptance step, which only accept good solutions, might accept bad individuals based on the Metropolis criterion, which enables the local search process to escape situations where it get trapped at local optima. (Equation 4.1)

Figure 5.8 illustrates the acceptance step of the implemented simulated annealing local search:

```
                 R =rand(1);%returns a random number between 0 and 1
                 For i=1:nind            %
                     if(delta_E(i)<0)    % it means the new individual has performed
    Same as                              % better then its predecessor. The best rank is 0
  Hill Climbing          phen(i,:) = temp_ind(i,:);
 Acceptance step         E_old(i) = E_new(i);

                    elseif(exp(-delta_E(i)/Temp)>R) % accepts a worst individual if the
                                                    % Condition is valid
                         phen(i,:) = temp_ind(i,:);
                         E_old(i) = E_new(i);
                    end
                end
```

Figure 5.8: Acceptance step of the simulated annealing algorithm

5.3 "Tabu Search" Local Search Implementation

The Tabu local search algorithm was the third approach for hybridizing the multi objective genetic algorithm used in the actual system, with a local search improvement phase. From a logical point of view, the tabu search mechanism seemed to be the best choice of local search process, mainly because of its expressive power to escape undesirable situations, such as the entrapment at local optima, a situation which the Hill Climber techniques occasionally suffer from, and the cycling situations at local optima regions which may jeopardize the termination of local search processes such as the Simulated Annealing algorithms. On the other hand, due to the random chaotic distribution of the solution's values in the search space, the option of fitting a defined function to the solutions' values in the local range space is not the suitable choice. Consequently no assumptions could have been made about the data points distribution in the local spaces, and thus the chances that Simulated Annealing technique and the Tabu search approach could be inappropriate for this application's domain are probable, especially if the solution space does not actually present any local optima regions; in these situations the expressive features of the Tabu search and the Simulated Annealing approaches would not be too invested, but unfortunately, their performances will be costly in time due to their huge requirement of objective function's assessment.

The implemented Tabu search algorithm, similarly to the previous local search approaches (Hill Climbing and Simulated Annealing), was introduced after the crossover operator of the genetic algorithm. A potential population of individuals is passed by the MOGA to the Tabu search algorithm, for improvement. The Tabu search algorithm starts then by initializing a Tabu list, the major element of Tabu search algorithms, and which is to be thought of as an internal memory which keeps records of the previously visited solution points in the local neighbourhood, in order to impose the notion of discovering new data points and avoid cycling situations. On the other hand, memory cannot possess unlimited capacity, or in this case, storage capacity, so it was decided to limit the length of the Tabu list to ten solutions' values pre-visited at a time, as research (Thesen, 1998) has showed that large Tabu list lengths may be too restrictive for the search process and may lead to insufficient exploration mechanisms. The Tabu list (Figure 5.9) was represented by a three dimensional array, of dimensions (Nind, Nvar, 10), where Nind denotes the number of individuals in the population, Nvar denotes the number of variables constituting an individual and 10 which denotes the length of the tabu list:

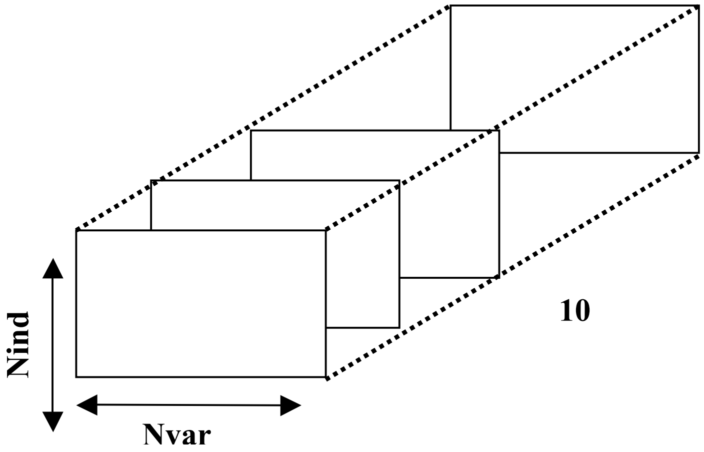


Figure 5.9: Tabu list Empty

The initial population passed from the genetic algorithm and which is contained in a matrix of dimension (Nind, Nvar) fills the first level of the tabu list (Figure 5.10), denoting by that the first visited set of solutions' values which should not be visited again for at least 10 exploration's steps.

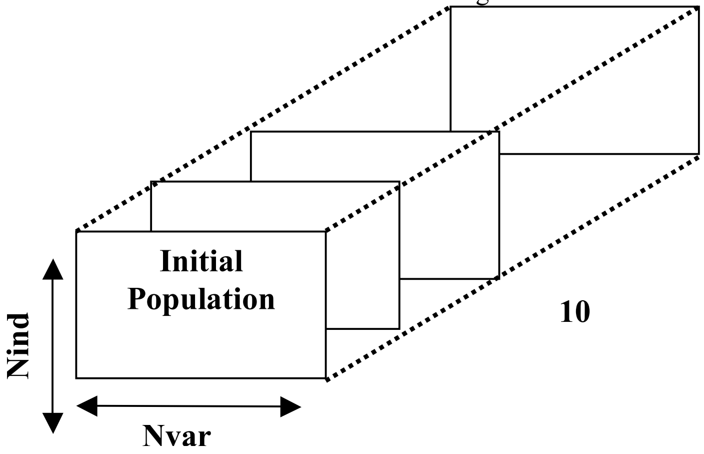


Figure 5.10: Tabu list with a Filled slot

Figure 5.11 illustrates the implemented piece of code designated with the initialization of the tabu list:

```
tabu_lis t= zeros (Nind,Nvar,10);   %First the tabu list is initialized and filled with zeros

index = ones(nind,1);               % Index is an array which holds the index of the next
                                    %empty slot in the tabu list for all the individuals,
                                    %(i.e. maximum 10)
tabu_list (:,:,1) = phen;           % the first slot of the tabu list is occupied by the
                                    %initial population passed from the MOGA
index(i)=index(i)+1;                %Incrementing the index of the next empty slot
```

Figure 5.11: Tabu list initialization

After initializing the Tabu list, the Tabu search algorithm sets the neighbourhood for the entire individuals of the population, randomly perturbs the variables of each individual, moving by thus to a neighbour population, and then assesses the performance of the newly created population. Note that the previously described processes are the same processes used in the Hill Climbing technique and the Simulated Annealing local search.

The final step of the Tabu search algorithm consists of the acceptance step, noting that the core of the Tabu search technique is that there is no point in accepting bad solutions unless to escape a subspace recently explored (cycling problems) or get out of local optima. The acceptance step is decomposed into five cases illustrated below: (Note: The code for the acceptance step process can be found in Appendix 2.)

- **Case 1**
- Conditions
    1. The new individual has performed better than its predecessor
    2. The new individual is different from all the individuals stored in the Tabu list of that individual, which basically means that the new individual is a good solution, which has not been recently explored.
    3. The Tabu list for that individual is not full.
- Result: Accept the new individual; insert it in the Tabu list at the index of the next empty slot and increment the index by one.

- **Case 2**
- Conditions
    1. The new individual has performed better than its predecessor
    2. The new individual is different from all the individuals stored in the Tabu list of that individual, which basically means that the new individual is a good solution, which has not been recently explored.
    3. The Tabu list of that individual is full
- Results:
    1. Accept the new individual
    2. Delete the first slot of the Tabu list, in other words, delete the oldest visited solution
    3. Decrement the index of the next empty slot
    4. Insert the new individual in the tabu list at the index of the next empty slot and increment the index by one.

- **Case 3**
- Conditions:
    1. The new individual has performed worse than its predecessor or is a duplicate of one of its Tabu list elements
- Results:
    1. Pick randomly another individual in the allowed neighbourhood.
    2. Calculate its new performance
    3. Increment a Timer variable by one; (Timer initially is equal to zero)
    4. Check case1 and case 2 again

- **Case 4**
- Conditions:
    1. The new individual has performed worse than its predecessor
    2. The Timer variable is equal to ten; it means no improved individual has been found for ten iterations.
    3. Tabu list for that individual is not full
- Result:
    1. Accept the bad individual; the core of this step is escape potential cycling situations or local entrapment by accepting a bad solution after a number of fruitless iteration to find a better individual in order to explore new subspaces.
    2. Insert the bad individual at the next empty slot in the Tabu list and increment the index.

- **Case 5**
- Conditions:
    1. The new individual has performed worse than its predecessor
    2. The Timer variable is equal to ten; it means no improved individual has been found for ten iterations.
    3. Tabu list for that individual is full

- Result:
  1. Accept the bad individual; the core of this step is escape potential cycling situations or local entrapment by accepting a bad solution after a number of fruitless iteration to find a better individual in order to explore new subspaces.
  2. Delete the first slot of the Tabu list, in other words, delete the oldest visited solution
  3. Decrement the index of the next empty slot
  4. Insert the new individual in the Tabu list at the index of the next empty slot and increment the index by one.

5.4 Local Search Testing

The local search techniques hybridized with the genetic algorithm of the actual system are relatively straightforward numerical techniques, which do not have any interaction with the user, in the way they do not require any input arguments. They are just internal processes, hybridized with a genetic algorithm optimization technique; they get their arguments from the global search process of the MOGA. In other words, traditional testing techniques such as the category partition method which deals with a system as a black box, or even random exhaustive testing would not be suitable testing approaches for the actual system. In order to investigate the correctness of the local search techniques, it was more suitable to pin out the resulting data from the several phases constituting the local search algorithms. For example, testing the "local neighbourhood setting" process was verified by taking snapshots of the process's functionality:
Figure 5.12 represents the numerical output values of the local neighbourhood settings and the resulting new individuals for individual 1, 2 and 3 from the hill-climbing algorithm:

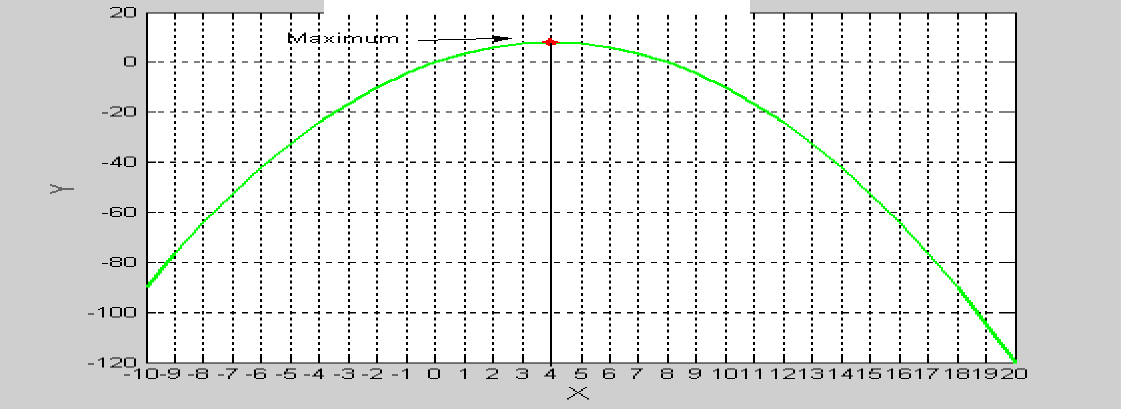
```
Hill Climbing
ubounds =           ubounds is the global upper limit vector for the 12 variables constituting an individual
Columns 1 through 12  denoting the 12 variables
50.0000 1.0000  50.0000 1.0000  50.0000 1.0000 50.0000  1.0000   0.0050   0.1000   0.0050  50.0000

lbounds =           lbounds is the global lower limit vector for the 12 variables constituting an individual
Columns 1 through 12 denoting the 12 variables
-50.0000  0.0010 -50.0000  0.0010 -50.000  0.0010  -50.0000  0.0010  -0.0050  -0.1000 -0.0050  -50.0000

delta =             local search width = 10 % of global search width for each variable.
Columns 1 through 12
10.0000  0.0999 10.0000  0.0999  10.000  0.0999  10.0000  0.0999  0.0010   0.0200   0.0010  10.0000

phen =              Phenotype values of the 3 first individual of the population
-31.2589  0.0025  34.3900  0.3737 -4.7555  0.0421 -28.2269  0.1131   0.0005  -0.0350   0.0031   4.3435
-31.2482  0.0040 -26.1242  0.0022  23.1670  0.0301 -13.5080  0.0048 -0.0045   0.0365   0.0029  -30.1877
27.6287   0.0065  -21.9814  0.2233  -16.1601  0.9194  31.6846  0.0410  0.0046  -0.0750   0.0050  -19.8253

temp_ubounds =      Local upper limit for each of the 3 individuals' variables
-26.2589  0.1009 39.3900  0.4236  0.2445  0.1009  -23.2269   0.1631   0.0010  -0.0250   0.0036   9.3435
-26.2482  0.1009 -21.1242  0.1009  28.1670   0.1009  -8.5080  0.1009   -0.0040  0.0465   0.0034  -25.1877
32.6287  0.1009  -16.9814  0.2733  -11.1601  0.9694  36.6846  0.1009  0.0050  -0.0650   0.0050  -14.8253

temp_lbounds =      Local lower limit for each of the 3 individuals' variables
-36.2589  0.0010  29.3900  0.3237 -9.7555  0.0010  -33.2269  0.0632   0.0000  -0.0450   0.0026  -0.6565
-36.2482  0.0010  -31.1242  0.0010  18.1670  0.0010  -18.5080  0.0010  -0.0050  0.0265   0.0024  -35.1877
22.6287   0.0010  -26.9814   0.1734  -21.1601  0.8695  26.6846  0.001    0.0040  -0.0850  0.0040  -24.8253

temp_ind =          Three new individuals created in the allowed local range
-29.0900  0.0066  35.8059  0.3475  -7.0627  0.0011  -31.2085  0.1560   0.0004  -0.0284   0.0030   5.4730
-31.2337  0.0417  -22.8564  0.0987  24.2902  0.0992  -10.7099  0.0539  -0.0041  0.0451   0.0025  -31.1113
30.0300  0.0064  -19.2173   0.2395  -12.3289  0.9464  31.2367   0.0665   0.0050  -0.0685  0.0041  -16.0519
```

Figure 5.12: Neighbourhood setting and population's modification processes

The previous pinned out data is printed out during an execution of the hybridized MOGA technique with the Hill Climbing local search, and it demonstrates the accuracy and the correctness of the “neighbourhood setting” process and the random movement of the population to an adjacent population in the predefined local range. For the random perturbation step of the actual population, an assumption about the correctness and the reliability of the internal random generator function “Rand” in Matlab was made, although future testing may be carried to investigate the accuracy of randomness of that generator. (More details, discussions and results analysis in Chapter 6).

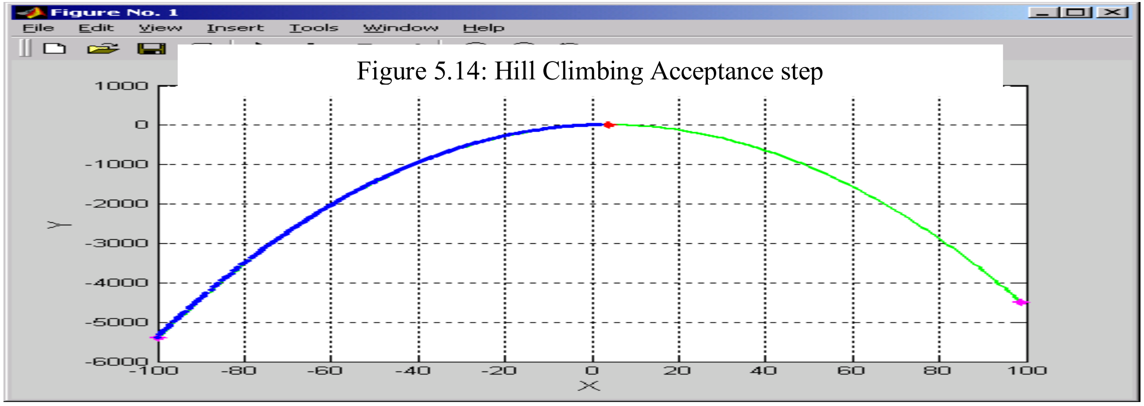
In addition, in order to test the correctness of the local search functionalities, the Hill Climbing local search was tested on a simple created scenario. The scenario consisted of a simple parabolic function $F(x) = -\frac{1}{2} \times x^2 + 4x$ (Figure 5.13), which represents a single maximum point reached for X=4.

Figure 5.13: $F(x) = -\frac{1}{2} \times x^2 + 4x$



The objective of this scenario was to prove the reliance of the Hill Climbing local search, by investigating its behaviour. It was decided to set the domain of definition for the function F to [-100 100] for the purpose of this test. The Hill Climbing local search started with a solution point with abscissa X = -100 and ordinate Y=F (-100). The local search process consisted of a 10000 number of iterations. At each iteration, a step was made by randomly modifying the abscissa of the current point or solution in the range [current abscissa –1 current abscissa +1]. The function F (x) was then evaluated at the new abscissa, which is only accepted if its ordinate value (i.e. F (x) value, which basically denotes the fitness function) is higher than the ordinate of the previous abscissa. The values of the accepted Xs were pinned out on the screen throughout the entire local search process and stored in an array, as well as their corresponding function’s evaluations. The arrays storing the x values and their matching y values were primarily initialized to zeros. After the maximum (x = 4) was reached, the Hill Climbing local search did not accept any x values, while storing the number –101 in the array storing the x values to denote that all the x values picked during the process, after the solution x=4 was reached, were void.

Figure 5.14: Hill Climbing Acceptance step

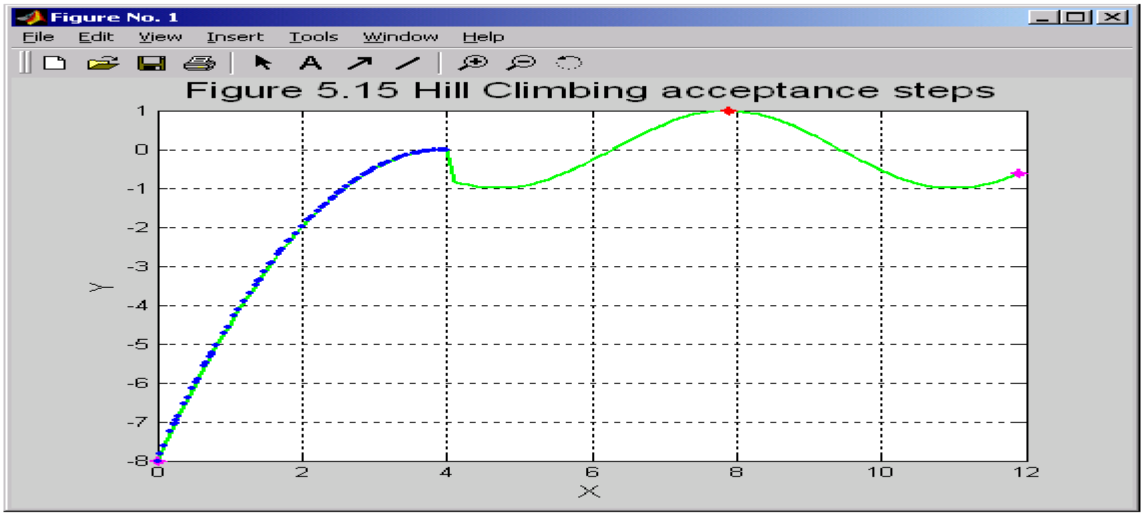


Although Figure 5.14 demonstrated the correctness of the hill climbing local search and its reliability to get to the maximum of the function F, the impracticality of this local search was demonstrated by changing the problem's scenario ($F(x) = -\frac{1}{2} \times x^2 + 4x$), by using a system of equations instead, whose graph on the domain [0 12] represents two different maximums, (a local maximum and a global maximum).

$$\begin{cases} F(x) = -\frac{1}{2} \times x^2 + 4x - 8 & \text{If x} <= 4 \\ F(x) = \sin(x) & \text{If x>4} \end{cases} \tag{5.2}$$

Although this system of equations is defined for all the real numbers, a reduced domain of definition [0 12] was used for the purpose of the testing.

Figure 5.15 is the resulting plot from the execution of the hill climbing local search to get to the maximum of the previous system of equations, and it shows that after getting to the value x=4, the algorithm was stuck, and no further steps were accepted.

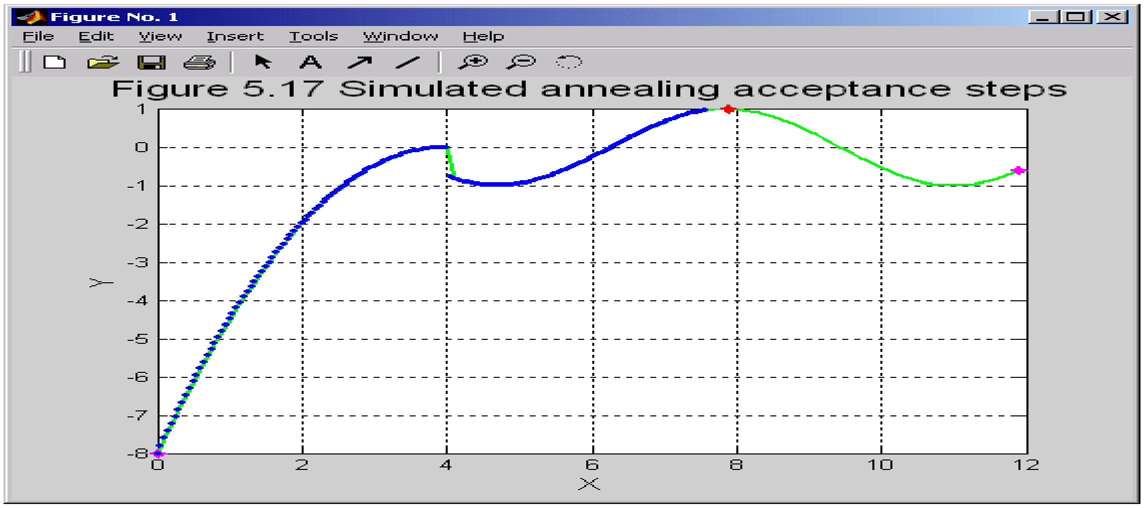


On the other hand, the same problem's scenario was applied using the Simulated Annealing process, which accepts downhill's steps based on a probabilistic model, and the Tabu search process, which keeps record of previously visited steps in an internal fixed length memory (Tabu list). Fortunately Simulated annealing and Tabu search techniques have both demonstrated their ability to escape the local maximum at x=4, and were able to reach the global maximum after a certain sufficient number of iterations. But the remarkable thing was that Simulated Annealing process has taken much more time to get to the global max compared to the performance of the Tabu search process, and that's due to the cycling situations that the SA process was facing all the way through the search space. In other words, SA kept on oscillating around some values, by making forward steps followed by backwards steps, especially at low temperatures, where the chance of accepting downhill's steps is relatively high, the fact which essentially delayed the convergence of the SA algorithm towards the global max. This processing delay was totally inexistent during the Tabu search process, mainly because the Tabu search algorithm was forbidding backwards steps towards data points already visited and stored in the tabu list. Downhill's steps were only accepted in the case where no improved solution was found for a specific amount of time; the idea behind this acceptance step is to explore new search spaces and escape local optima. The following printed messages (Figure 5.16) were pinned out during the execution of SA processing solutions points around the local maxima. X=4:

```
ans =
x accepted=3.8,and its y values is=-0.02
ans =
x accepted=3.9,and its y values is=-0.005
ans =
Bad solution accepted X=4.1
ans =
Bad solution accepted X=4.2
```

Figure 5.16: SA acceptance steps

Figure 5.17 illustrates the acceptance steps of the simulated annealing. The same scenario was applied with the Tabu search process, and similar acceptance steps (Figure 5.17) behaviour was reported although the process took much less time to converge when Tabu Search process was used. (Note: Testing code may be found in appendix 2.).



CHAPTER 6- RESULTS AND DISCUSSIONS

6 Results and Discussions

6.1 Introduction

The objective of this project was to investigate the usability of memetic algorithms in the application domain, and compare their performances with the multiobjective genetic algorithm's performance. On the other hand, due to the stochastic nature of evolutionary algorithms (i.e. genetic algorithms and memetic algorithms) and the pareto-optimality characteristic of the solutions' fitness into the application's domain, there was no guarantee that the memetic algorithm would outperform the genetic algorithm. In order to make an informed comparison between the performances of the MOGA and the memetic algorithm, it was not convenient to make the comparison based on the numerical results produced by a single execution of each of the two algorithms; instead a certain number of iterations of the two algorithms were essential in order to analyse the results. Accordingly, it was decided to run each of the four processes, (1→MOGA, 2→MOGA/Hill Climbing, 3→MOGA/Simulated annealing, 4→MOGA/Tabu Search), 20 times, make a record of their results, i.e. the individuals' fitness's and the best numerical values achieved for each of the objectives, and finally produce a statistical analysis of their results. Regrettably, executing the four previous algorithms for a certain number of iterations (20), using the essential problem's application, was unbearable due to the enormous amount of execution time needed for each of the processes to converge, which is principally linked to the complex objective function (based in Simulink) used in the initial system. As a result, it was decided to test the performances of the four algorithms using a simpler problem's domain; the latter reduced problem used a phenotype representation of the individuals composed of two variables, optimizing the values of four objectives while using a predefined mathematical set of equations as an objective function. Sections 6.2, 6.3, 6.4 and 6.5 illustrate the major findings and results of each of the four processes.

6.2 MOGA Results and Discussion

The multiobjective genetic algorithm was executed 20 consecutive times. Each execution was terminated after 100 generations of the MOGA algorithm, which was processing a population constituted of 40 individuals, each composed of 2 variables, while optimizing the values of 4 objectives. After each of the 20 executions of the MOGA algorithm, the best values of the four objectives, achieved throughout the 100 generations of each execution and produced by the objective function for each of the 40 individuals, were stored in a matrix named "bestobjv". From one execution of the MOGA to its successor, a matrix named "bestobjv_con" concatenated the best objectives values achieved for each execution (i.e. the "bestobjv" matrices of each execution), while keeping record of the number of the best objectives reached for each execution. The fitness scores of the population's individuals were also stored in corresponding matrices. Note that a relative fitness function ("rank_prf.m") was employed in the system, ranking the objective function's results of each individual according to their performance and suitability in the application's domain, compared to the performances of the other individuals of the same generation, while assuming a minimization problem (i.e. rank 0 is best). Another fitness function named "ranking_mo.m" was then applied to the output of the relative fitness function "rank_prf.m" to assign global fitness scores for each individual of the population.

Figure 6.1 illustrates the average values of the best objectives achieved after each of the 20 executions of the MOGA algorithm, terminated after the processing of 100 generations at every execution.

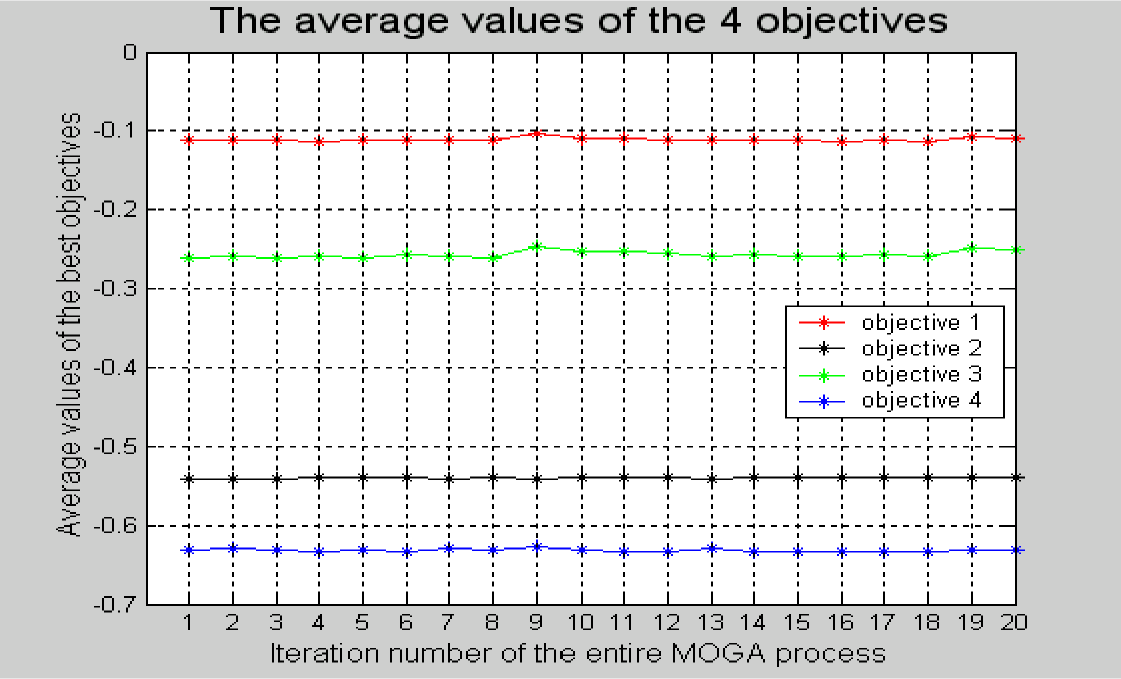


Figure 6.1: The best average values of the 4 objectives (MOGA)

It was remarkable that the average values of the best-achieved objectives throughout the 20 executions of the MOGA were relatively consistent and stable, occasionally presenting minor fluctuations such as around the 4th and 9th execution. The total average values (i.e. the average value of the best average values at each execution of the MOGA) for each objective are presented in Table 6.1.

Total Average Values			
Objective 1	**Objective 2**	**Objective 3**	**Objective 4**
-0.1108	**-0.5398**	**-0.2564**	**-0.6319**

Table 6.1: Total Average Values (MOGA)

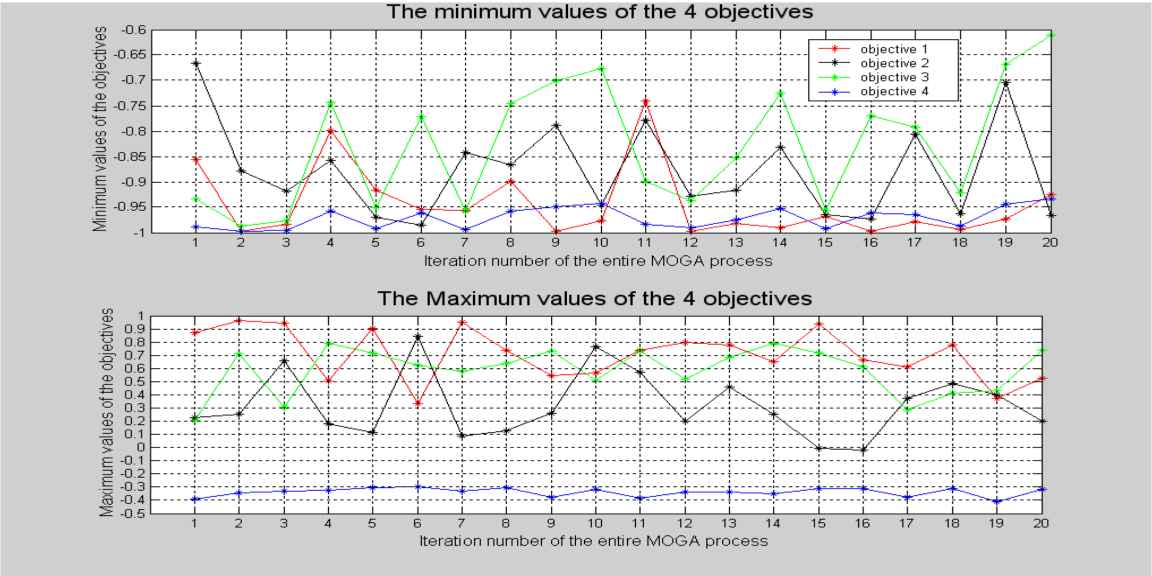


Figure 6.2 Minimum and Maximum values at the 100th generation (MOGA)

Figure 6.2 illustrates the minimum and the maximum values attained at the 100th generation for each of the four objectives at each of the 20 executions of the MOGA. It is remarkable that the minimum and maximum values variations of objective 4 were more or less stable throughout the 20 executions compared to the other objectives' behaviour, only varying in a range width of 0.1. (Maximum values variation range [-0.4 -0.3], Minimum values variation range [-1 -0.9]). On the other hand, the three other objectives represented relatively unpredictable variations of minimum and maximum values attained, occasionally showing conspicuous changes from one execution to another. Note that the graphs illustrated in Figure 6.2 correspond to the maximum and the minimum values of the objectives achieved at the 100th generation of each execution which is not necessarily the generation where the best objectives' values where achieved. Figure 6.3 illustrates the minimum and the maximum values of the best objectives achieved throughout the 100 generations of each of the 20 executions:

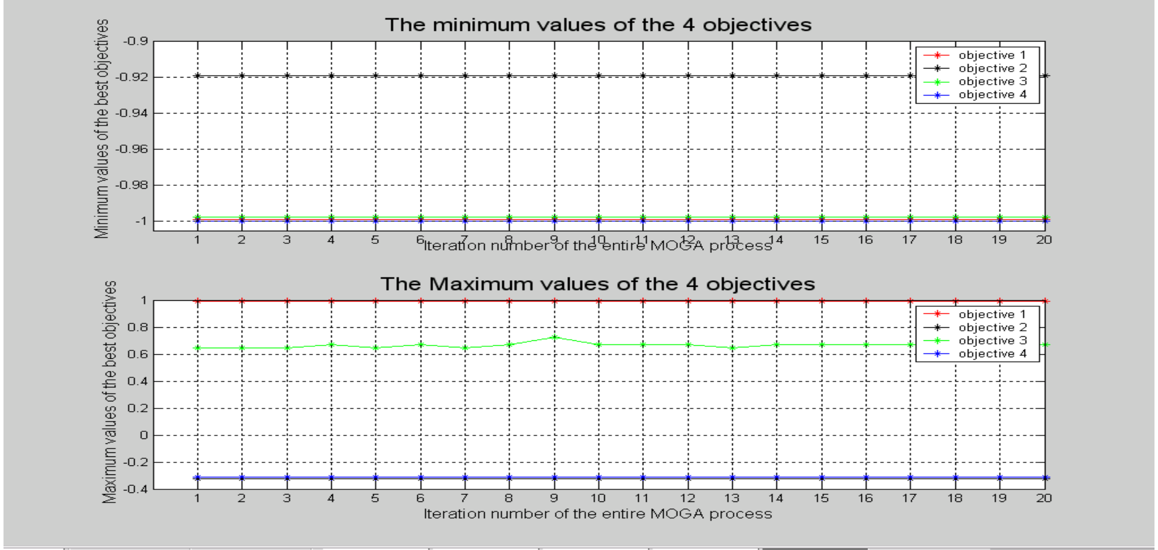


Figure 6.3: Minimum and Maximum values of the best objectives (MOGA)

It was very obvious that the minimum and the maximum values of the best objectives were much more stable compared to the minimum and maximum values of the same objectives at the 100th generation of each execution. The minimum values of objectives 1, 3 and 4 were relatively fluctuating around the value -1; where as the minimum values of objective 2 were isolated around the value –0.92. On the other hand, the maximum values of objective 2 and 4 were more or less mutual, while objective 1 presented the highest maximum values and objective 2 possessed a more fluctuating curve, fitting the maximum values attained for that objective all along the 20 executions.

6.3 Hill Climbing Results and Discussion

The MOGA hybridized with the Hill Climbing local search was tested using similar criteria to the MOGA process. The MOGA/Hill Climbing algorithm was executed 20 times, while storing the best results of the objectives functions (best objectives values) and the individuals' fitness's. Each execution has processed 100 generation of the MOGA, while performing 100 iterative steps of local improvement for each generation. These local search improvements were based on a Hill Climbing algorithm, which explores subspaces of solutions, while only accepting improved individuals.

Figure 6.4 illustrates the average values of the best objectives attained for each of the 4 objectives at each of the 20 executions:

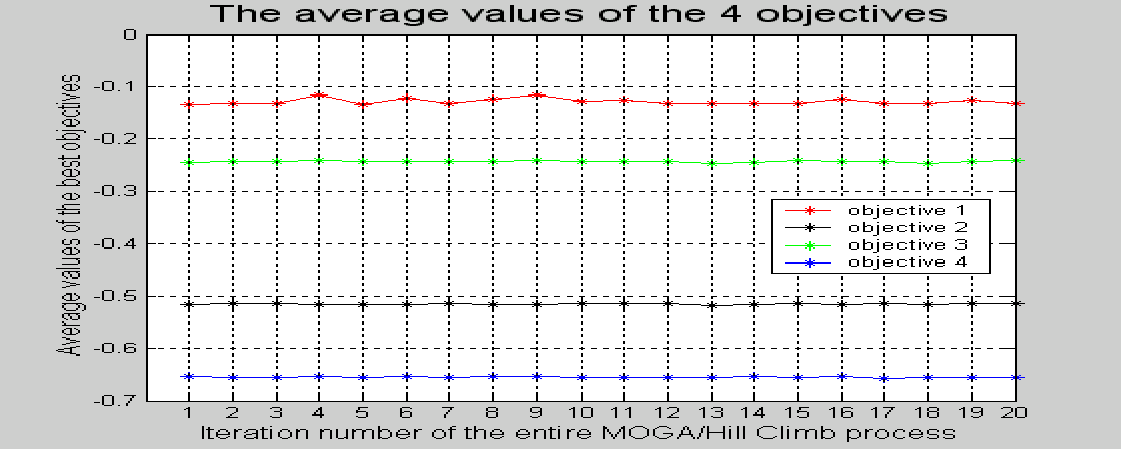


Figure 6.4: The average values of the 4 objectives (MOGA/Hill Climbing)

The average values of the best objectives achieved throughout the MOGA/Hill Climbing technique were pretty much consistent with results of the MOGA, with no major changes to be mentioned, although the variations of the values of objective 1 have presented sharper fluctuations while objective 3 presented more stabilized variation behaviour, compared to the MOGA output.
The total average values (i.e. the average value of the best average values at each execution of the MOGA/Hill Climbing algorithm) for each objective are presented in Table 6.2.

Total Average Values			
Objective 1	**Objective 2**	**Objective 3**	**Objective 4**
-0.1285	**-0.5158**	**-0.2428**	**-0.6556**

Table 6.2: Total Average Values (MOGA/Hill Climbing)

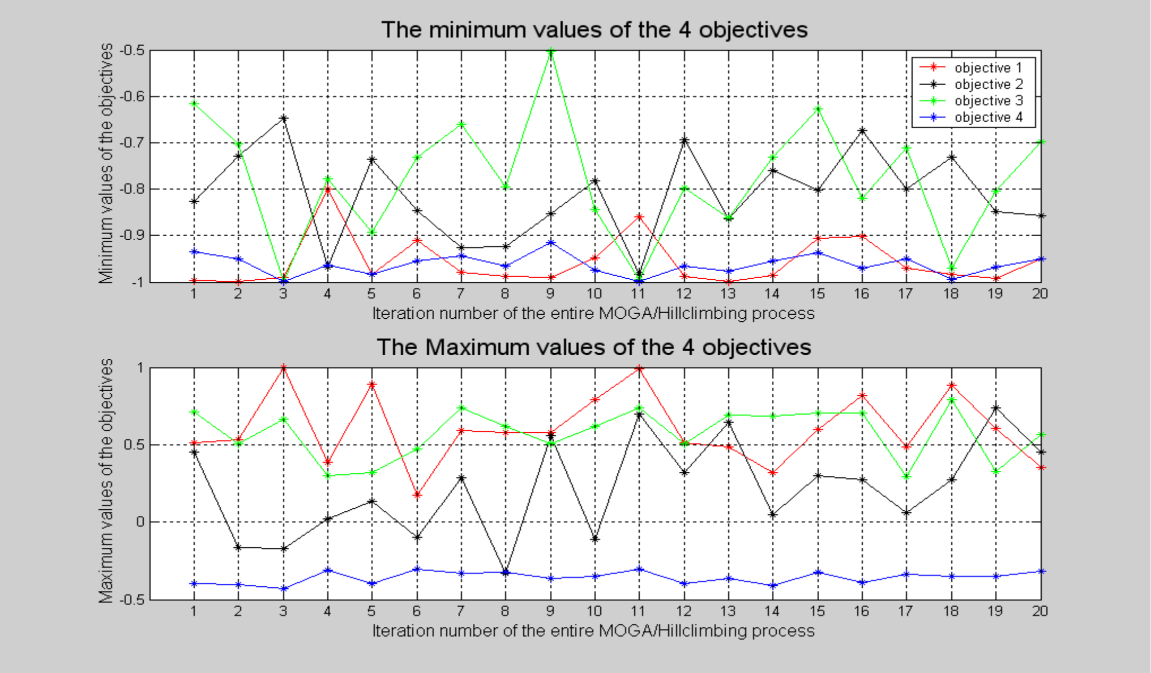


Figure 6.5: Minimum and Maximum values of the 4 objectives (MOGA/Hill Climbing)

Figure 6.5 illustrates the minimum and the maximum values attained for each of the four objectives at the 100th generation of each of the 20 executions of the MOGA/Hill Climbing process. The minimum and maximum values variations of objective 4 were approximately of the same stability as in the 100th generations of the MOGA process throughout the 20 executions, again varying in a range width of 0.1. (Maximum values variation range [-0.4 -0.3], Minimum values variation range [-1 -0.9]). The three other objectives, represented a relatively unpredictable variations of minimum and maximum values attained, occasionally showing conspicuous changes from one execution to another similarly to the MOGA process. Figure 6.6 illustrates the minimum and the maximum values of the best objectives achieved throughout the 100 generations of each of the 20 executions:

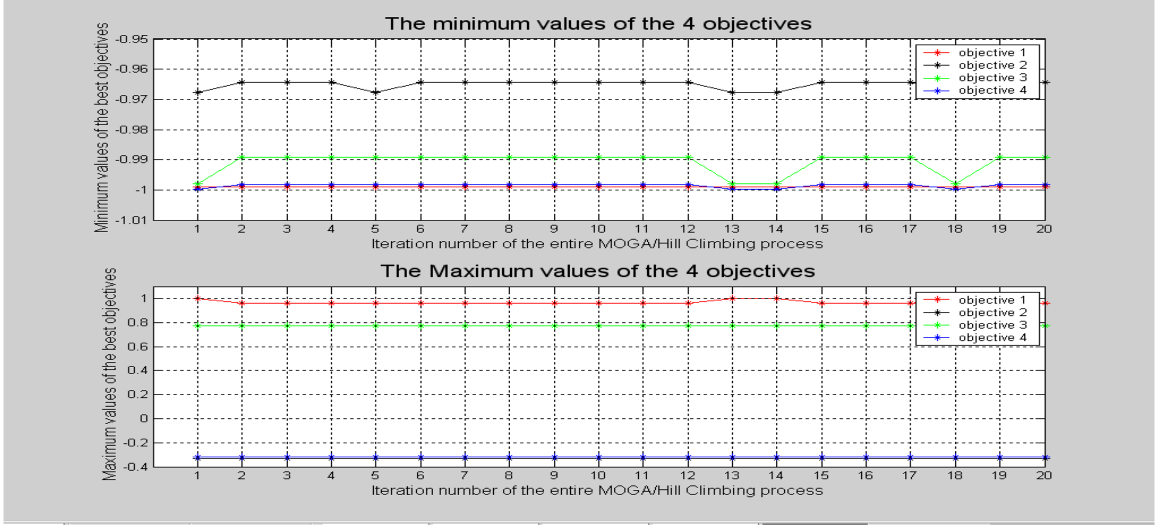


Figure 6.6: Minimum and Maximum values of the best objectives (MOGA/Hill Climbing)

Figure 6.6 pointed up some differences between the behaviour of the best objectives resulting from the MOGA and the MOGA/Hill Climbing techniques. In the MOGA/Hill Climbing algorithm, objective 2 presented altering minimum values for the best objectives values, diverging at more than one occasion from the minimum values of the objectives 1 and 4 towards the value –0.99, which was not the case in the MOGA. On the other hand, in the MOGA/Hill Climbing algorithm, Objective 2 presented some instability concerning its minimum values and Objective 1 presented some fluctuations in its maximum values' behaviour, which were absent in the MOGA.

6.4 Simulated Annealing Results and Discussion

The MOGA hybridized with the Simulated Annealing local search was tested using similar criteria to the MOGA process. The MOGA/Simulated Annealing algorithm was executed 20 times, storing the results of the objectives functions (best objectives values) and the individuals' fitness's. Each execution has processed 100 generation of the MOGA, while performing 100 iterative steps of local improvement for each of the 50 temperatures constituting the cooling schedule of the Simulated Annealing algorithm, at each generation. These local search improvements are based on the Simulated Annealing algorithm, which explores subspaces of solutions, accepting improved individuals, and occasionally accepting bad ones based on a probabilistic model. Figure 6.7 illustrates the average values attained for the best objectives at each of the 20 executions:

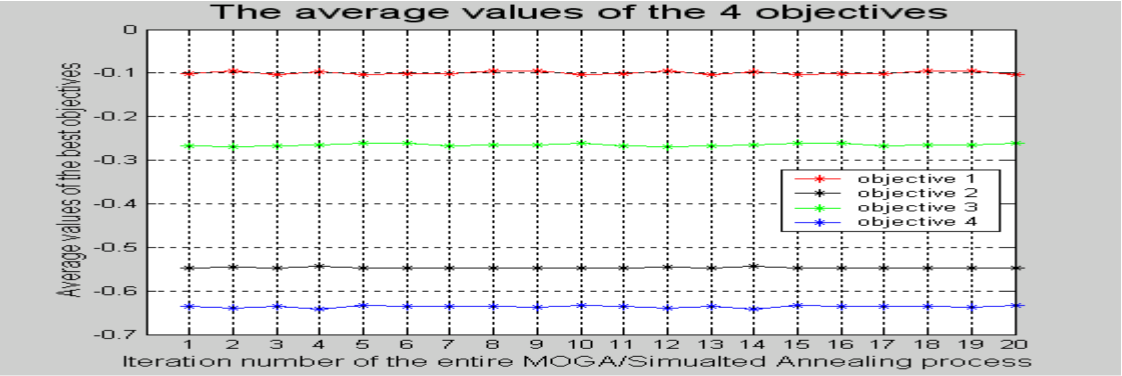


Figure 6.7: The average values of the 4 objectives (MOGA/SA)

The total average values (i.e. the average value of the average values at each execution of the MOGA/Simulated Annealing algorithm) for each objective are presented in Table 6.3.

Total Average Values			
Objective 1	**Objective 2**	**Objective 3**	**Objective 4**
-0.0999	**-0.5466**	**-0.2647**	**-0.6363**

Table 6.3: Total Average Values (MOGA/SA)

Figure 6.8 illustrates the minimum and the maximum values attained for each of the 4 objectives at the 100th generation throughout the 20 executions of the MOGA/Simulated annealing process. The results were relatively more similar to the MOGA results than the MOGA/Hill Climbing results.

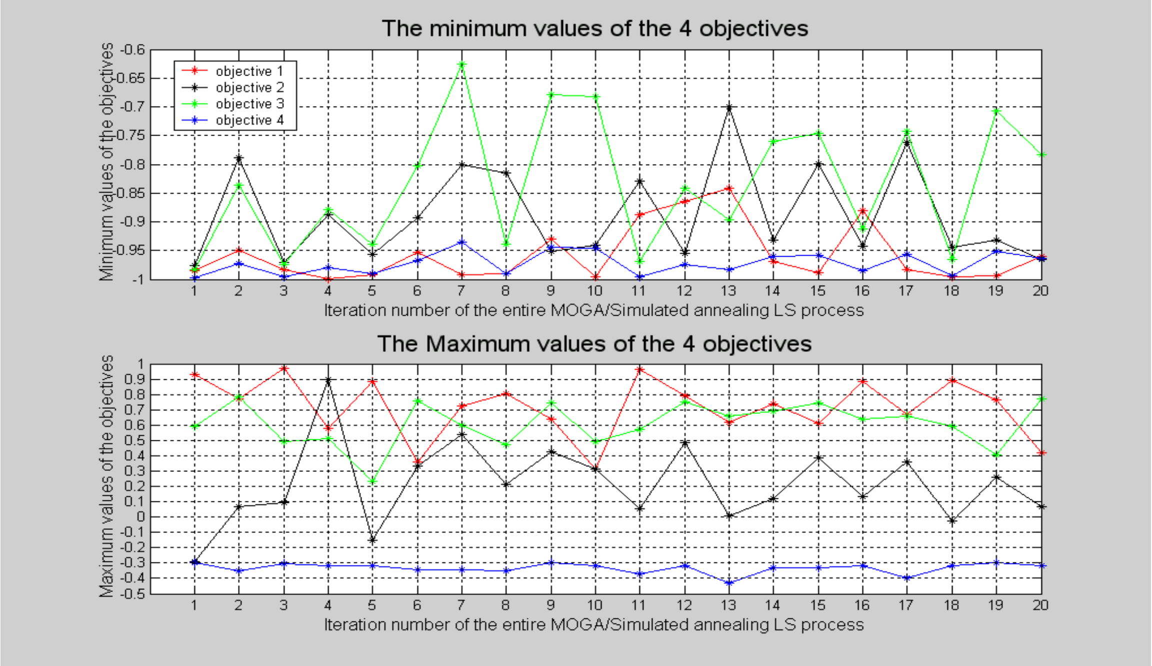


Figure 6.8 Minimum and Maximum of the 4 objectives (MOGA/SA)

Figure 6.9 illustrates new differences between the behaviour of the best objectives resulting from the MOGA, the MOGA/Hill Climbing and the MOGA/Simulated Annealing techniques, especially concerning the minimum values variations of the 4 objectives. In the MOGA/Simulated Climbing algorithm, objective 2 presented alternating minimum values for the best objectives values between the minimum values of objective 1 and 4, more frequently diverging from the minimum values of the objective 1 towards the value –0.995, while the minimum value of objective 4 was totally different from the outcome of the previous techniques (MOGA, MOGA/Hill Climbing), in the way that its value was more or less stable, but this time around the value –0.995, differently from the minimum values of objective 4. The behaviour of the maximum values of the 4 objectives was considerably similar to their behaviour in the MOGA and the MOGA/Hill Climbing algorithms.

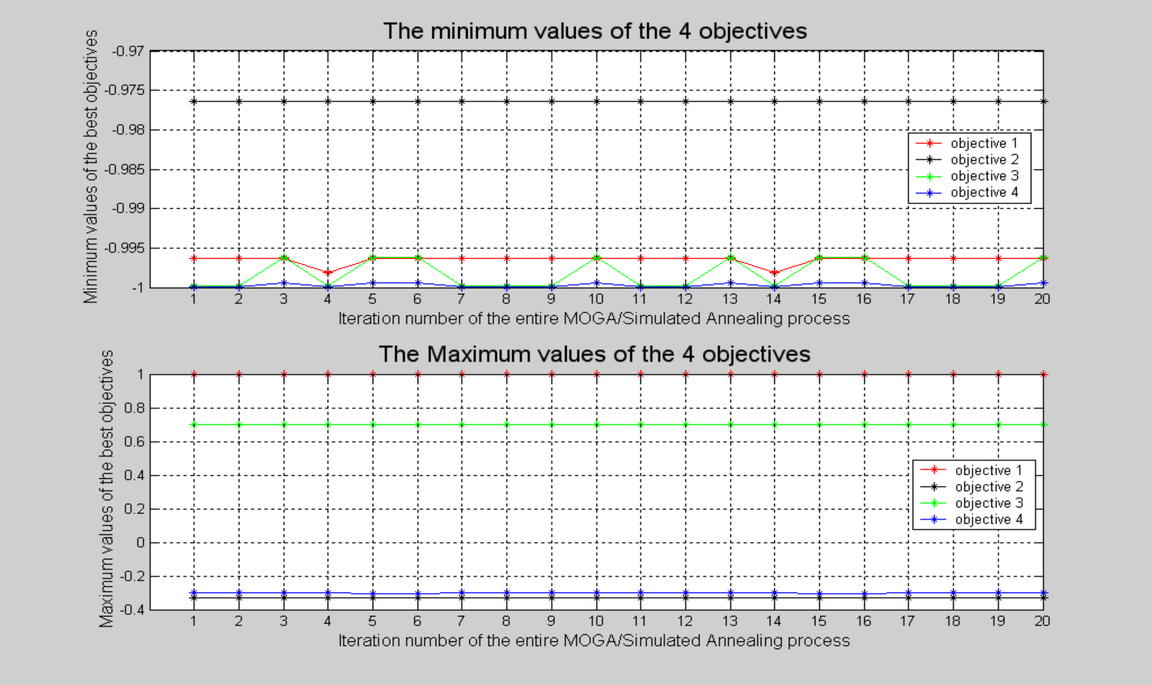


Figure 6.9: Minimum and Maximum values of the best objectives (MOGA/SA)

6.5 Tabu Search Results and Discussion

The MOGA hybridized with the Tabu search local search was tested using similar criteria to the MOGA process. The MOGA/Tabu search algorithm was executed 20 times, storing the best results of the objectives functions (best objectives values) and the individuals' fitness's. Each execution has processed 100 generation of the MOGA, while performing 100 iterative steps of local improvement for each generation. These local search improvements were based on a Tabu search algorithm, which explores subspaces of solutions, while accepting ameliorated individuals or bad ones in order to escape cycling situations and entrapments at local optima.

Figure 6.10 illustrates the average values attained for each of the 4 objectives at each of the 20 executions:

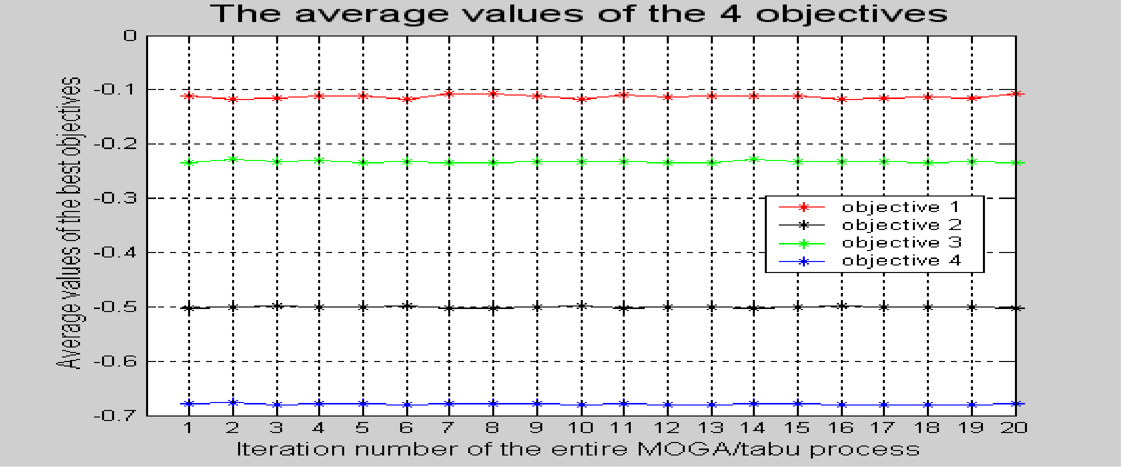


Figure 6.10: The average values of the 4 objectives (MOGA/Tabu search)

The total average values (i.e. the average value of the average values at each execution of the MOGA/Tabu Search algorithm) for each objective are presented in Table 6.4.

Total Average Values			
Objective 1	**Objective 2**	**Objective 3**	**Objective 4**
-0.1129	**-0.5007**	**-0.2325**	**-0.6792**

Table 6.4: Total Average Values (MOGA/Tabu search)

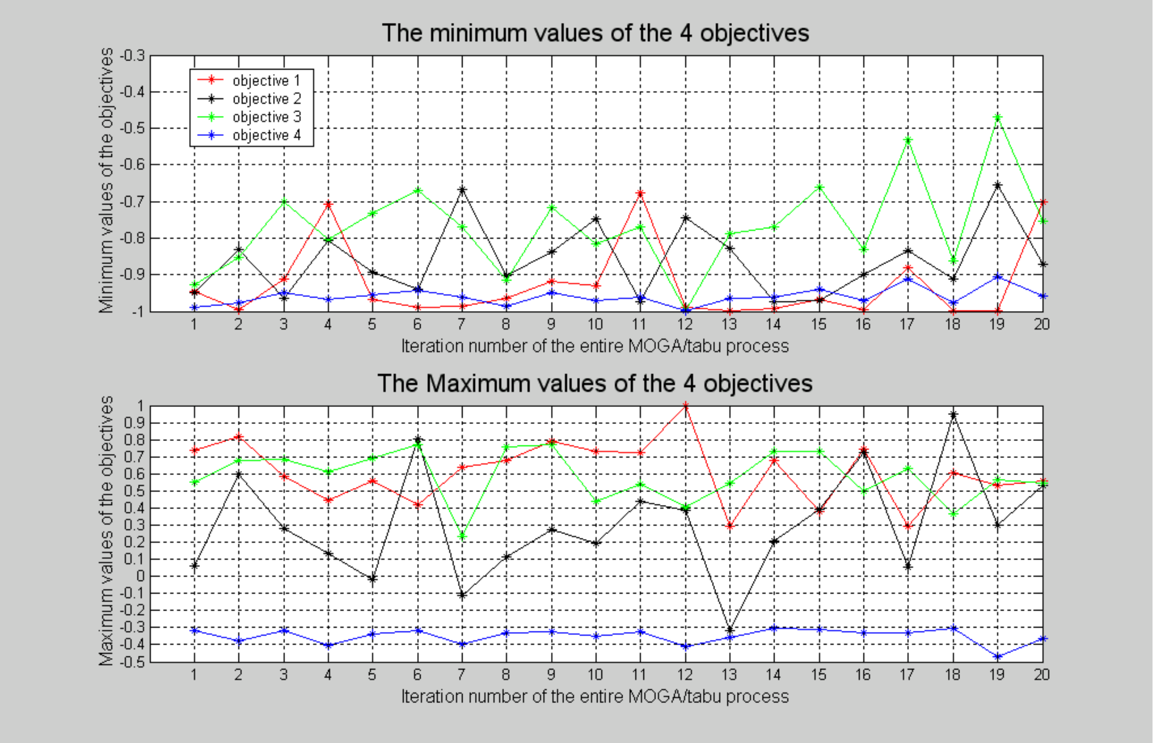


Figure 6.11 Minimum and Maximum of the 4 objectives (MOGA/Tabu search)

From figure 6.11 it was remarkable that the chaotic behaviour of the results were pretty much consistent with the other techniques' results.

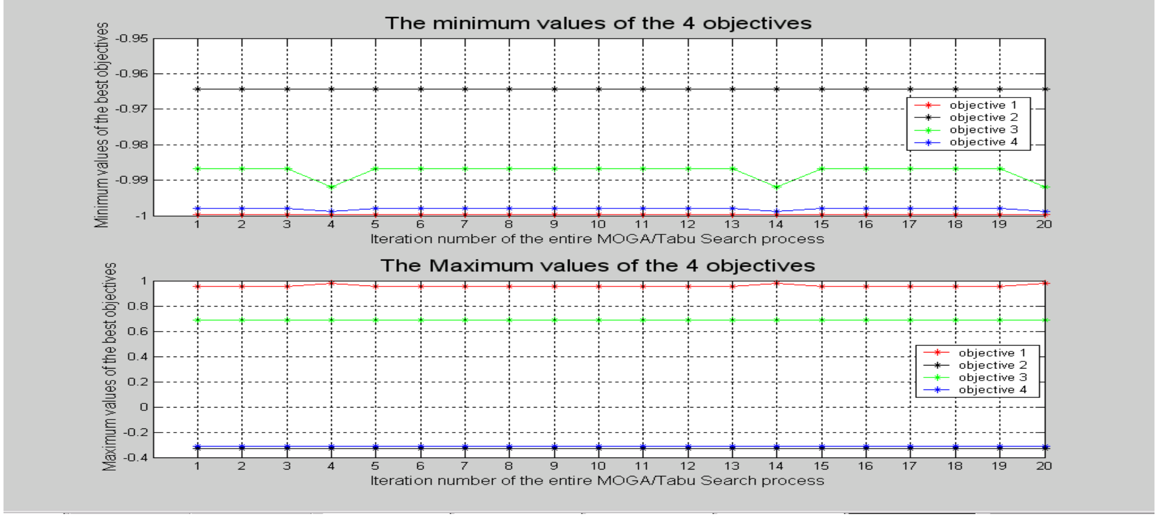


Figure 6.12: Minimum and Maximum values of the best objectives (MOGA/Tabu search)

The maximum values' behaviour of the four objectives resulting from the MOGA/Tabu Search process was relatively similar to the results of the previous described techniques, although from figure 6.12, it was perceptible that the minimum values of objective 2 was slightly detached from the values of objectives 1 and 4, without any intersections points with the minimum values' variation curves of these objectives (1 and 4). This stability and resemblance of maximum values' behaviour throughout the 4 processes, (MOGA, MOGA/Hill Climbing, MOGA/Simulated Annealing, MOGA/Tabu Search), is logically a preferable situation, and it denotes a competitive performances of these processes, with no process trespassing a certain global maximum value of the objectives, while behaving differently at the minimum values regions, by exploring different local spaces and competing to decrease the values of the objectives to mark an improved performance.

6.6 Results' Statistical Analysis

Table 6.5: Total averages Analysis for the best objectives obtained after 20 executions of each process

	Objective1	Objective 2	Objective 3	Objective 4
MOGA	**-0.1108**	**-0.5398**	**-0.2564**	**-0.6319**
MOGA/ Hill Climbing	**-0.1285**	**-0.5158**	**-0.2428**	**-0.6556**
MOGA/Simulated Annealing	**-0.0999**	**-0.5466**	**-0.2647**	**-0.6363**
MOGA/Tabu Search	**-0.1129**	**-0.5007**	**-0.2325**	**-0.6792**
Best Average (Lowest)	**Hill Climbing**	**Simulated Annealing**	**Simulated Annealing**	**Tabu Search**

From the results analysis (Table 6.5), it was very obvious that the memetic algorithm has beaten and outperformed the MOGA optimizing the values of the four objectives, by producing lower averages values for each objective, after 20 executions of each of the four processes. In particular, the MOGA hybridized with the Simulated Annealing improvement mechanism was able to get improved average values for the best values attained for 2 out of 4 objectives (objectives 2 and 3).

Table 6.6: Minimum achieved among the best objectives:

	Objective1	Objective 2	Objective 3	Objective 4
MOGA	**-0.9991**	**-0.9193**	**-0.9976**	**-0.9996**
MOGA/ Hill Climbing	**-0.9990**	**-0.9677**	**-0.9980**	**-0.9997**
MOGA/Simulated Annealing	**-0.9963**	**-0.9764**	**-0.9998**	**-1.0000**
MOGA/Tabu Search	**-0.9997**	**-0.9644**	**-0.9920**	**-0.9987**
Best result (lowest)	**TS**	**SA**	**SA**	**SA**
Worst result (highest)	**SA**	**MOGA**	**TS**	**TS**

HC = Hill Climbing- SA=Simulated annealing-TS=Tabu Search

Again, from Table 6.6, it was reflected that the memetic algorithm has outperformed the MOGA by getting better results minimizing the 4 objectives. More specifically, it was remarkable, that the MOGA/Simulated Annealing process has uniquely reached minimal values for each of the objectives 2, 3 and 4 that were not reached by the other techniques. On the other hand, it was also very notable that the MOGA has scored the worst performance for the optimization of objective 2, by achieving a minimum value of–0.9193 for objective 2, which was relatively by far outperformed by the three memetic algorithm approaches.

Table 6.7: Maximum achieved among the best objectives:

	Objective1	Objective 2	Objective 3	Objective 4
MOGA	**0.9938**	**-0.3217**	**0.7244**	**-0.3109**
MOGA/ Hill Climbing	**0.9963**	**-0.3306**	**0.7704**	**-0.3163**
MOGA/Simulated Annealing	**0.9997**	**-0.3317**	**0.7017**	**-0.2997**
MOGA/Tabu Search	**0.9818**	**-0.3281**	**0.6873**	**-0.3106**
Best result (lowest)	**TS**	**SA**	**TS**	**HC**
Worst result (highest)	**SA**	**MOGA**	**HC**	**SA**

HC = Hill Climbing- SA=Simulated annealing-TS=Tabu Search

From table 6.7, it was deduced that the memetic algorithm has once again outperformed the MOGA for the optimisation of the 4 objectives, in the way that the memetic algorithm has reached lower maximum values for the best objectives achieved throughout 20 executions compared to the maximum values attained by the MOGA for the same 4 objectives. To put in nutshell, after 20 executions of the 4 processes, each allowed to run for 100 generations, the best objectives values achieved have represented lower minimum values (i.e. better objectives were explored) and lower maximum values (i.e. less bad objectives were dealt with) in the case of the memetic algorithm, which globally denotes a better quality results compared to the data produced by the MOGA.

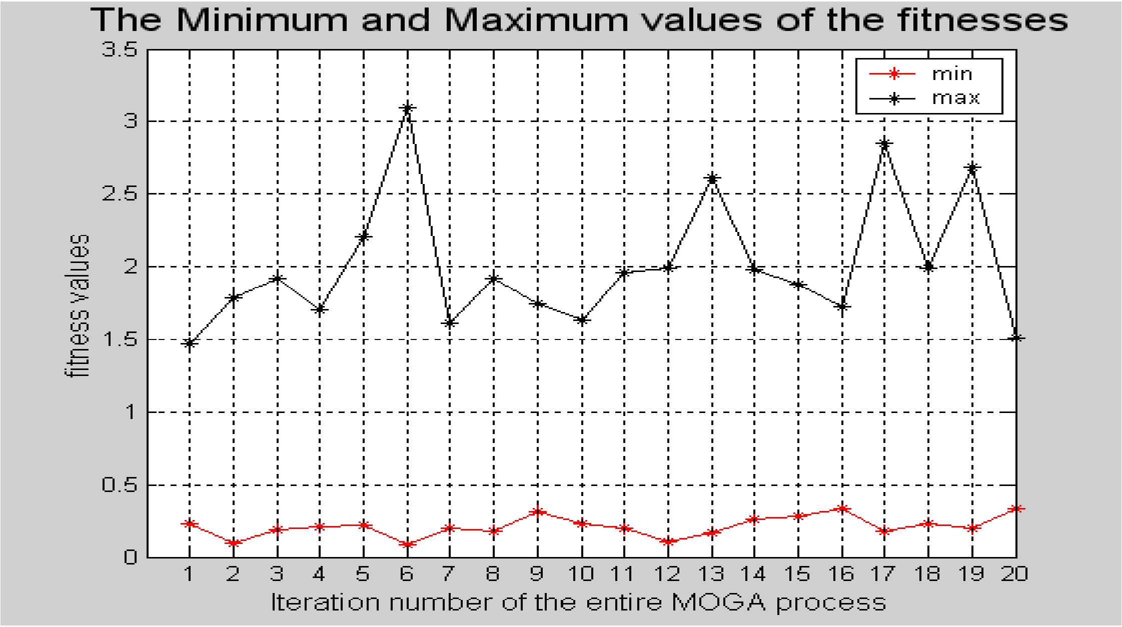


Figure 6.13: Minimum and Maximum fitness variation throughout 20 executions of MOGA

Figure 6.13 highlights the minimum and the maximum values' behaviours of the fitness values assigned to the 4 objectives all along the 20 executions of the MOGA. Noting that the graphs shown in figure 6.13 represent the minimum and the maximum values of the fitness scores assigned to the population's individuals (solutions) by the function "ranking_mo.m" which assigns global fitness values to the individuals based on a maximisation fashion, i.e. the best individual will be allocated the highest fitness value. It was remarkable that the maximum fitness values were relatively unstable compared to the minimum values' behaviour, with the maximum being achieved at the 6th execution of the MOGA. This relative minimum stability reflects a desirable optimisation characteristic, which denotes that the effort and the exploration process is rather exploring spaces representing good fitness in order to improve their performances rather than experimenting with low-fitness individuals.

Table 6.8: MOGA Fitness

Execution Number	Best Fitness	Worst Fitness
1	1.4713	0.2321
2	1.7863	0.0950
3	1.9177	0.1902
4	1.7012	0.2124
5	2.2019	0.2165
6	3.0952	0.0903
7	1.6084	0.1964
8	1.9225	0.1763
9	1.7415	0.3094
10	1.6317	0.2281
11	1.9591	0.2014
12	1.9904	0.1056
13	2.6115	0.1653
14	1.9799	0.2596
15	1.8821	0.2788
16	1.7249	0.3305
17	2.8481	0.1770
18	1.9863	0.2333
19	2.6846	0.1955
20	1.5048	0.3305

Best Fitness Achieved=3.0952 at the 6th execution, Worst Fitness Achieved=0.0903 at the 6th execution.

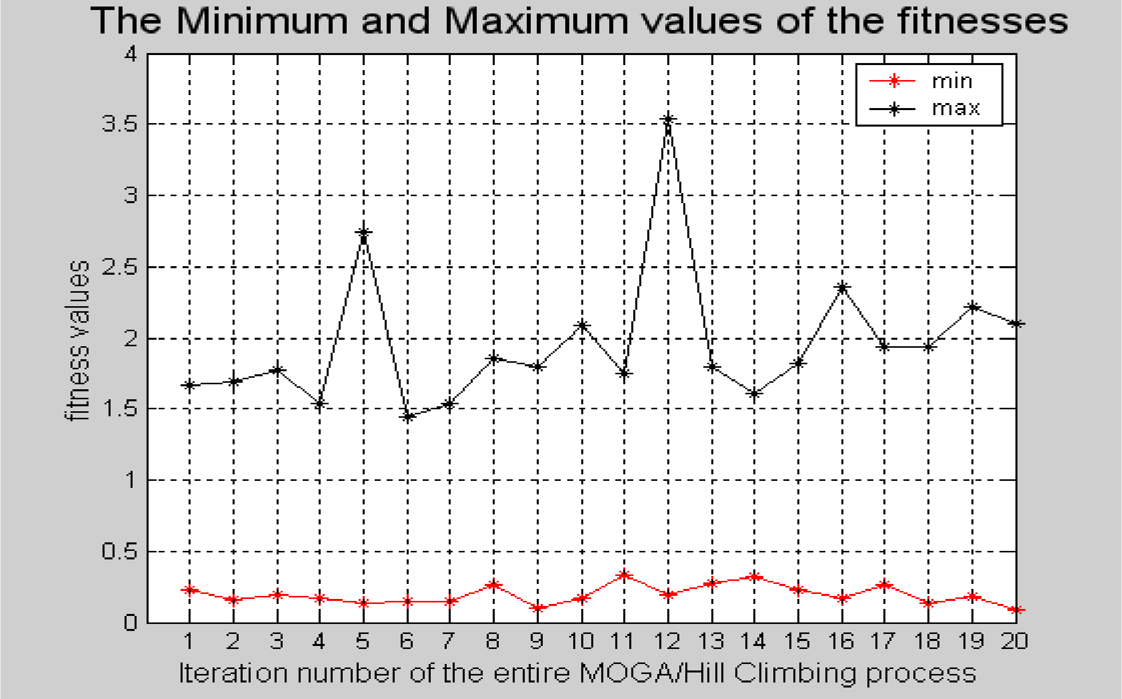


Figure 6.14: Minimum and Maximum fitness variation throughout 20 executions of MOGA/Hill Climbing

The same characteristics concluded from the graph of the minimum and the maximum values' behaviour of the fitness values attained during the MOGA process (Figure 6.13) can be linked to the behaviour variations of the minimum and the maximum values of the fitness scores achieved by the MOGA/Hill Climbing algorithm (Figure 6.14), although in the latter algorithm, the maximum value of the fitness was achieved at the 12th execution.

Table 6.9: MOGA/Hill Climbing Fitness

Execution Number	Best Fitness	Worst Fitness
1	1.6617	0.2233
2	1.6868	0.1615
3	1.7700	0.1984
4	1.5365	0.1686
5	2.7376	0.1386
6	1.4491	0.1405
7	1.5388	0.1515
8	1.8535	0.2642
9	1.7951	0.0995
10	2.0851	0.1690
11	1.7461	0.3305
12	3.5394	0.1875
13	1.7954	0.2692
14	1.6094	0.3222
15	1.8224	0.2338
16	2.3531	0.1697
17	1.9339	0.2616
18	1.9392	0.1289
19	2.2138	0.1787
20	2.1002	0.0914

Best Fitness Achieved=3.5394 at the 12th execution, Worst Fitness Achieved=0.0914 at the 20th execution.

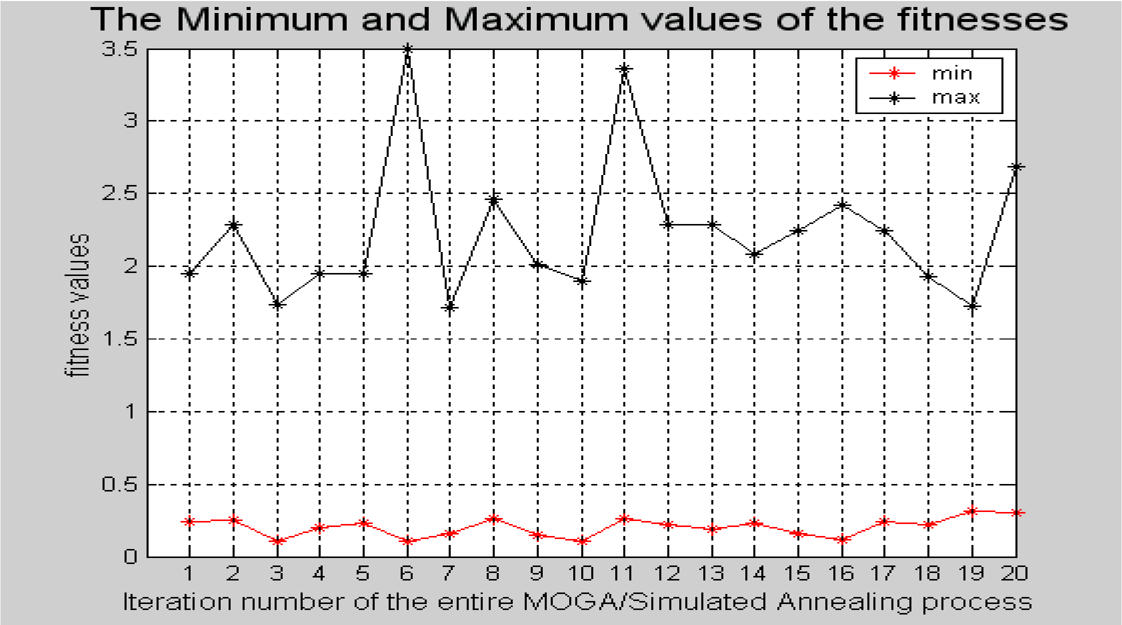


Figure 6.15: Minimum and Maximum fitness variation throughout 20 executions of MOGA/Simulated Annealing

The maximum fitness value was attained at the 6th execution of the MOGA/Simulated Annealing process, similarly to the MOGA process, although this is just a total coincidence based on the stochastic nature of the evolutionary algorithm (Genetic and Memetic algorithms).

Table 6.10: MOGA/Simulated Annealing Fitness

Execution Number	Best Fitness	Worst Fitness
1	1.9478	0.2390
2	2.2876	0.2546
3	1.7312	0.1051
4	1.9455	0.2007
5	1.9514	0.2264
6	3.4900	0.1040
7	1.7093	0.1617
8	2.4603	0.2633
9	2.0143	0.1451
10	1.8951	0.1035
11	1.7461	0.1051
12	1.9455	0.2546
13	1.7954	0.1040
14	1.6094	0.3222
15	1.8224	0.2338
16	2.3531	0.1697
17	1.9514	0.2616
18	1.9392	0.1451
19	2.2138	0.1787
20	1.9478	0.1451

Best Fitness Achieved=3.4900 at the 6th execution, Worst Fitness Achieved=0.1035 at the 10th execution.

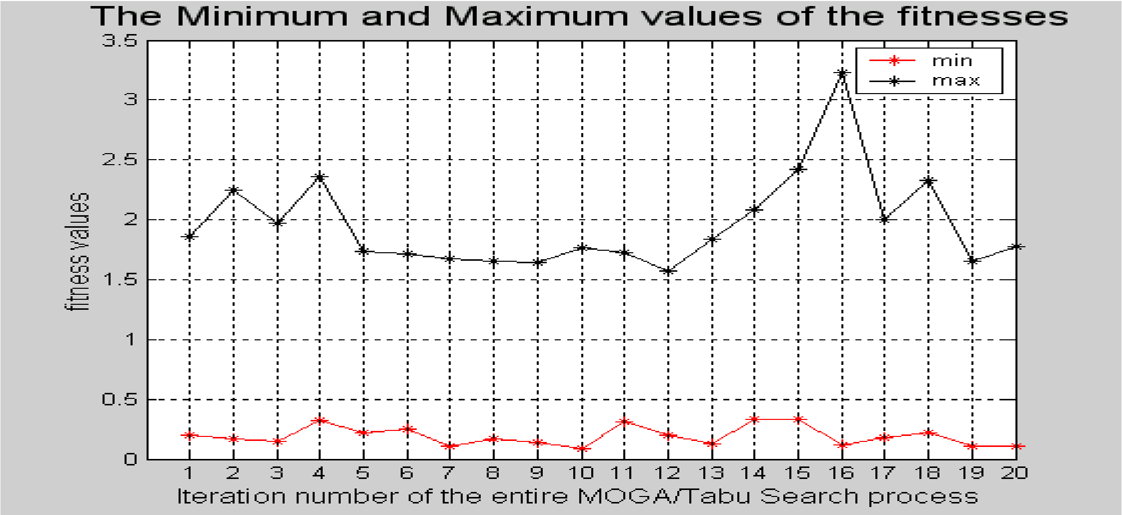


Figure 6.16: Minimum and Maximum fitness variation throughout 20 executions of MOGA/Tabu

From Figure 6.16, it is notable the way the maximum fitness was continuously increasing from the 12th execution of the MOGA/Tabu Search until the 16th execution where the fitness has achieved its global maximum value. The minimum values' behaviour being always consistent to the other techniques' results, bounded between 0 and 0.7.

Table 6.11: MOGA/Tabu Search Fitness

Execution Number	Best Fitness	Worst Fitness
1	1.8539	0.2020
2	2.2467	0.1731
3	1.9690	0.1534
4	2.3577	0.3217
5	1.7297	0.2209
6	1.7109	0.2516
7	1.6772	0.1031
8	1.6545	0.1686
9	1.6391	0.1354
10	1.7680	0.0825
11	1.7266	0.3083
12	1.5750	0.2012
13	1.8399	0.1233
14	2.0778	0.3305
15	2.4243	0.3305
16	3.2304	0.1175
17	2.0051	0.1755
18	2.3274	0.2191
19	1.6545	0.1115
20	1.7805	0.1054

Best Fitness Achieved=3.2304 at the 16th execution, Worst Fitness Achieved=0.0825 at the 10th execution.

Table 6.12: Fitness Statistics

	MOGA	MOGA/TS	MOGA/SA	MOGA/HC
Worst Fitness Achieved	0.0903	0.0825	0.1035	0.0914
Technique with the worst Fitness achieved (Lowest)		√		
Technique with the highest worst fitness achieved (highest)			√	
Best Fitness	3.0952	3.2304	3.4900	3.5394
Technique with the best Fitness achieved (highest)				√
Technique with the lowest best fitness achieved (lowest)	√			

From Table 6.12 it was demonstrated based on the statistical results obtained after 20 executions of the MOGA and the three different versions of the memetic algorithm implemented (MOGA/Hill Climbing, MOGA/Simulated Annealing, MOGA/Tabu Search) that the MOGA techniques had allocated a maximum fitness value of 3.0952 to its best solutions produced, which is the minimum fitness value compared to the best fitness scores allocated to the best solutions resulted from the hybridized systems (memetic algorithm). In addition, the MOGA/Hill Climbing algorithm has achieved the highest best global fitness value; its value was 3.5395, "0.4442" higher than the best fitness achieved by the MOGA, which is relatively a big difference, and a good fitness improvement. On the other hand, the MOGA/Simulated Annealing algorithm was the best approach in terms of getting the highest worst fitness among the 4 processes, which denotes a "lift up" or an improvement of the global optimisation quality.

6.7 Future Work

The hybridization of the Multi Objective Genetic Algorithm employed in the “Optimisation techniques for Gas Turbine Engine Control Systems” project with a local search improvement step can be deemed a success in respect of the fact that three different kinds of local search techniques were successfully implemented and integrated with the genetic algorithm .On the other hand, due to the time constraints and the experimental nature of the project, further experimental processes would surely benefit the system. The memetic algorithm implemented through out the project consisted of the integration of the local search techniques between the recombination process, Crossover, and the mutation operator. Due to the huge amount of objective functions evaluations required by the local search techniques, the investigation of the local search process’s performance at distinct points of the genetic algorithm were unfeasible. A susceptible future experimentation may consist of the hybridisation of the local search process at different points of the genetic algorithm, such as before the recombination step or after the mutation operator. A result comparison can then be established to evaluate the performance of the memetic algorithm at the distinct phases of the MOGA, which may highlight certain performance improvements. On the other hand, instead of locally improving each individual of the population, it might be praiseworthy investigating the direction of the local search improvement step towards relatively fit individuals, who have higher chances for survival by recombining and propagating to the next generations.

Another area of investigation which surely would improve the performance of the memetic algorithm is the employment of Meta-modelling techniques, such as neural networks approaches, to improve the performance of the objective functions used for the evaluation of the solutions’ performance into the problem’s domain. Currently, the memetic algorithm is consuming unbearable amounts of time in order to converge towards acceptable good solutions, which is totally normal, due to its requirement of huge amounts of objective function’s evaluation while exploring the local spaces, in order to decide whether to accept a new solution or not. Meta-modelling techniques are efficient, in the way they speed up the evaluation process of the individual’s fitness, by improving the objectives functions’ performances.

A the end, it is also recommendable to verify the randomness of the random generator function “Rand”, integrated in Matlab, which was used in the local search processes of the implemented system in order to randomly move from a solution to another in the local search space. Throughout the project, a total reliance was assumed about the correctness of that random generator, even the MOGA toolbox previously implemented at the Automatic Control & Systems Engineering at the University of Sheffield, uses the same random generator for actions such as the creation of random binary population of individuals for the MOGA to start with, assuming the efficiency of the random generator. Nevertheless, randomness’ characteristics such as determinism, cycle length, uniformity and correlation (Knuth, 1981) might be verified and tested on the Rand function of Matlab. Noting that there exist three types of random numbers; the truly random numbers, such as found by counts of Geiger measuring radioactive decay, the pseudo-random numbers, which although they posses the appearance of randomness, they display repeatable patterns or periods of re-occurrence, and finally the quasi random numbers, which are numbers chosen to fill solutions spaces with maximum distances between points. The best way for testing the randomness of the random generator employed in the system, is to investigate the use of several distinct random generators and make sure that the results are adequate and independent of the generator, another testing technique may consists of the application of some hash or disturbance functions on the output of the random generator in order to check out any repetitive or similar patterns.

CHAPTER 7- CONCLUSION

7 Conclusion

The implemented system can be deemed a success in respect of the fact that the main core of the project has been met. The most widely used local search techniques, Hill Climbing, Simulated Annealing and Tabu search, have been successfully implemented, tested and integrated with the multi objective genetic algorithm optimization technique that was employed in the actual system sponsored by the Control & Systems University Technology Centre supported by Rolls-Royce at the Automatic Control & Systems Engineering Department at the University of Sheffield.

The experimentation results produced by the memetic algorithm have exposed an improved optimization performance compared to the traditional results of the MOGA. Enhanced objectives values were achieved reflecting better fitness values in their application's domain. More technically, it was remarkable that the multiobjective genetic algorithm hybridized with the Simulated Annealing local search improvement, was distinctively producing the lowest objective values for 3 out of 4 of the objectives being optimized, outperforming by a relatively good lead the optimisation results of the same objectives produced by the MOGA. A statistical comparison based on data results produced by 20 executions of the MOGA and the memetic algorithms (i.e. MOGA/Hill Climbing, MOGA/Simulated Annealing, MOGA/Tabu Search) has exhibited the fact that the best average values of the best global objectives values achieved by the multiple algorithms belong to the different versions of the memetic algorithm (MOGA/Hill Climbing, MOGA/Simulated Annealing, MOGA/Tabu Search), depending on the objective, in other words, the best average value of the best global values of objective 1 were linked with the "MOGA/Hill Climbing" memetic algorithm version, while the best averages for objectives 2 and 3 go to the MOGA/Simulated Annealing version, and finally the best average value for objective 4 goes back to the MOGA/Tabu Search technique. The MOGA could not compete with the other techniques or represent a best average value for any of the 4 objectives throughout 20 executions of these algorithms. On the other side, it was also notable, that the MOGA was producing the worst values for objective 2 in specific, by resulting a range of data solutions representing both higher minimum and maximum bounds compared to the range of the values produced for the same objective by the other techniques of the memetic algorithm, noting that the optimization process consisted of a minimization problem.

To put in a nutshell, the memetic algorithm implemented has highlighted better results and outperformed the traditional MOGA technique, which is basically the basic core of this experimental project. Unfortunately, mainly due to the time constraints imposed on the development cycle of this project, further experimentations may be made in the future, which might uncover better results, especially if the local search improvement steps were allocated more running time for processing new local spaces.

T. Bäck, F. Hoffmeister and H.-P. Schwefel, "*A Survey of Evolution Strategies*", Proc. ICGA 4, pp. 2-10, 1991.

J. E. Baker, "*Adaptive Selection Methods for Genetic Algorithms"*, Proc. ICGA 1, pp. 101-111, 1985.

J. E. Baker, *"Reducing bias and inefficiency in the selection algorithm*", Proc. ICGA 2, pp. 14-21, 1987.

Richard Baker, "Genetic Algorithms in Search and Optimization", 1998.

L. Booker, *"Improving search in genetic algorithms*", In *Genetic Algorithms and Simulated Annealing*, L. Davis (Ed), pp. 61-73, Morgan Kauffmann Publishers, 1987.

M.F. Bramlette, *"Initialization, Mutation and Selection Methods in Genetic Algorithms for Function Optimization"*, Proc ICGA 4, pp. 100-107, 1991.

Mark F. Bramlette and Eugene E. Bouchard, "Genetic Algorithms in Parametric Design of Aircraft".

R.A. Caruana and J.D. Schaffer, *"Representation and Hidden Bias: Gray vs. Binary Coding"*, Proc. 6th Int. Conf. Machine Learning, pp153-161, 1988

R. A. Caruana, L. A. Eshelman, J.D. Shaffer, "*Representation and hidden bias П: Eliminating defining length bias in genetic search via shuffle crossover",* In Eleventh International Joint Conference on Artificial Intelligence, N. S. Sridharan (Ed.), Vol.1, pp. 750-755, Morgan Kauffman Publishers, 1989.

Andrew Chipperfield, Peter Fleming, Hartmut Pohlheim, Carlos Fonesca, *"Genetic Algorithm Toolbox*", Department of Automatic Control Engineering and Systems Engineering, University of Sheffield

Carlos A. Coello, David A. Van Veldhuizen and Gary B. Lamont, "*Evolutionary Algorithms for Sloving Multiobjective Problems*", pp. 372-376, Kluwer Academic Publishers, 2002.

Louis Anthony Cox, Jr., Lawrence Davis, and Yuping Qiu, Dynamic, "Anticipatory Routing in Circuit-Switched Telecommunications Networks".

Yuval Davidor, "A Genetic Algorithm Applied to Robot Trajectory Generation".

Digalakis and Margaritis, 2000 Memetic Algorithm Pseudocode

T.C. Fogarty, *"An Incremental Genetic Algorithm for Real-Time Learning"*, Proc. 6th Int. Workshop on Machine Learning, pp. 416-419, 1989.

D. E. Goldberg and J. Richardson, *"Genetic Algorithms with Sharing for Multimodal function Optimization"*, Proc. IGCA 2, pp. 41-49, 1987.

D.E. Goldberg, *Genetic Algorithms in search, Optimization and Machine Learning*, Addison Wesley Publishing Company, January 1989.

J.J Grefenstette, "*Incorporating Problem Specific Knowledge into Genetic Algorithms*", In Genetic Algorithms and Simulated Annealing, pp. 42-60, L. Davis (Ed.), Morgan Kauffmann, 1987.

John J. Grefenstette, "Strategy Acquisition with Genetic Algorithms".

Steven A. Harp and Tariq Samad, "Genetic Synthesis of Neural Network Architecture".

C. Z. Janikov and Z. Michalewicz, “An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms”, Proc. IGCA 4, pp. 31-36, 1991

K. A. De Jong, *Analysis of the Behaviour of a class of Genetic Adaptive Systems*, PhD Thesis, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1975.

K.A. De Jong and J. Sarma, “Generation Gaps Revisited”, In Foundations of Genetic Algorithms 2, L. D. Whitley (Ed.), Morgan Kaufmann Publishers, 1993.

Kosmas Knödler, Jan Poland, Andreas Zell, Alexander Mitterer and the BMW Group Munich, “Using Memetic Algorithms for Optimal Calibration of modern automotive combustion Engines”

D. Knuth, *Art of Computer Programming*, Vol2. Seminumerical Algorithms, Second edition. Addison-Wesley, Reading, Massachusetts, 1981.

Natalio Krasnogor, Tutorial on memetic algorithms, 2002.

Gunar E. Liepens and W. D. Potter, “A Genetic Algorithm Approach to Multiple Fault Diagnosis”.

C.B. Lucasius and G. Kateman, “Towards Solving Subset Selection Problems with Aid of Genetic Algorithm”, *In Parallel Problem Solving from Nature 2*, R Männer and B. Manderick, (Eds.) pp. 239-247, Amsterdam: North Holland, 1992

H. Mühlenbein and D. Schlierkamp-Voosen, “Predictive Models for the Breeder Genetic Algorithm”, Evolutionary Computation, Vol. 1, No. 1, pp. 25-49, 1993.

Yanto Prasetio and Eddie Rhone, “Tabu Search: Overview and Example”, 2002.

W.E. Schmintendorgf, O. Shaw, R. Benson and S. Forrest, *“Using Genetic Algorithm for Controller Design: Simultaneous Stabilization and Eigenvalue Placement Region”*, Technical Report No. CS92-9, Dept. Computer Science, College of Engineering, University of New Mexico, 1992.

W. M. Spears and K. A. De Jong, “*On the Virtues of Parameterized Uniform Crossover”*, Proc. ICGA 4, pp.230-236, 1991.

W. M Spears and K. A. De Jong, “*An Analysis of Multi-Point Crossover*”, In *Foundations of Genetic Algorithms*, J. E. Rawlins (Ed.), pp. 301-315, 1991.

Diana Spears, “More Search Algorithms”, 2001.

Erik Sundermann, “Simulated Annealing”, email: Erik.Sundermann@advalvas.be

G. Syswerda, “*Uniform crossover in genetic algorithms*”, Proc. ICGA 3, pp. 2-9, 1989.

D.M. Tate and A.E Smith, *“Expected Allele Convergence and the Role of Mutation in Genetic Algorithms”*, Proc. ICGA 5, pp.31-37, 1993

Arne Thesen, “Design and Evaluation of Tabu Search Algorithms for Multiprocessor Scheduling”, Kluwer Academic Publishers, Madison USA, 1998.

A. H. Wright, *“Genetic Algorithms for Real Parameter Optimization”*, In *Foundations of Genetic Algorithms*, J.E. Rawlins (Ed.), Morgan Kaufmann, pp. 205-218, 1991.

http://www.cs.sandia.gov/opt/survey/sa.html
http://www.cs.sandia.gov/opt/survey/ts.html
http://www.csep1.phy.ornl.gov/csep/mo/node28a.html
http:// www.ingber.com.
http://www.irirdia.ulb.ac.be/~meta.html
http://www.isic.ecs.soton.ac.uk/isystems/evolutionary/notes/evol/simulated_annealing.html
http://www.mit.edu/people/refreshh/1.713paper/node10.html
http://www.petaxp.rug.ac.be/~erik/research/research_part2.html
http://www.ra.informatik.uni-tuebingen.de
http://www.sce.carleton.ca/netmanage/tony/ts.html
http://www.winforms.phil.tu_bs.de/winforms/research/tabu/.html

Memetic Algorithms Home Page:

http://www.densis.fee.unicomp.br/~moscato/memetic_home.html

Report preparation

The report preparation phase involved designing the memetic algorithm, implementing the local search techniques, integrating them with the multiobjective genetic algorithm optimisation technique and iteratively testing the developed system. Time has been carefully allocated for writing, checking and updating the final dissertation report.

Date	*Milestone*
1st June	Project Start Date
20th June	Research and Design Phase Completed
8th July	MOGA/Simulated Annealing Implemented and Tested
16th July	MOGA/Hill Climbing Implemented and Tested
25th July	MOGA/Tabu Search Implemented and Tested
5th August	Results Analysis and Comparison Accomplished
20th August	Report First Draft Completed
27th August	Project Hand-in Date

***FORMAT CONVENTIONS*:** *This is Code* *% This is a comment*

Simulated Annealing Testing Code

```
% Author: Salem Adra
% Testing Version
% Simulated Annealing local search

disp(‘Simulated Annealing’)

Num_Steps=10000;
T_schedule=[1:-.02:0];
no_schedule=length(T_schedule);
x=zeros(1,120);
var=0;

for i=1:120
   x(i)=var;
   var=var+0.1;
end

y=zeros(1,120);
for i=1:120
   if i>41
      y(i)=sin(x(i));
   else
      y(i)=(-1/2)*(x(i)*x(i))+4*x(i)-8;
   end
end

clf
plot(x,y,'g', ...
   x(1),y(1),'m*', ...
   x(120),y(120),'m*', ...
   x(80),y(80),'r*', 'LineWidth',2);
title('Figure 5.17 Simulated annealing
       acceptance steps’,’FontSize’,16);
xlabel('X','FontSize',12);
ylabel('Y','FontSize',12);
grid on;
hold on

xsolutions=zeros(nm,1);
ysolutions=zeros(nm,1);
xold=0;
yold=(-1/2)*(xold*xold)+4*xold-8;
xsolutions(1)=xold;
ysolutions(1)=yold;

h = plot(xold,yold,'r.','LineWidth',1);
set(h,'EraseMode','None');
```

Hill Climbing Testing Code

```
 %Author: Salem Adra
 %Testing Version
 %Hill Climbing Local Search

 disp(‘Hill Climbing’)

 Num_Steps=10000;


 x=zeros(1,120);
 var=0;

 for i=1:120
    x(i)=var;
    var=var+0.1;
 end

 y=zeros(1,120);
 for i=1:120
    if i>41
       y(i)=sin(x(i));
    else
     y(i)=(-1/2)*(x(i)*x(i))+4*x(i)-8;
    end
 end

 clf
 plot(x,y,'g', ...
    x(1),y(1),'m*', ...
    x(120),y(120),'m*', ...
    x(80),y(80),'r*', 'LineWidth',2);
    title('Figure 5.16 HillClimbing'
 'acceptance steps', “FontSize’,16);
 xlabel('X','FontSize',12);
 ylabel('Y','FontSize',12);
 grid on;
 hold on

 xsolutions=zeros(nm,1);
 ysolutions=zeros(nm,1);
 xold=0;
 yold=(-1/2)*(xold*xold)+4*xold-8;
 xsolutions(1)=xold;
 ysolutions(1)=yold;

h = plot(xold,yold,'r.','LineWidth',1);
    set(h,'EraseMode','None');
```

Simulated Annealing Testing Code

Cont

```
pause(1)
count=2;
for tj=1:no_schedule
 Temp=T_schedule(tj);
 for nm=1:Num_Steps,
  xnew = xold-0.1 + 0.2*rand(1,1);
  if xnew>4
    ynew=sin(xnew)
  else
    ynew=(-1/2)*(xnew*xnew)+4*xnew-8;
  end
  r=rand(1);
  if(ynew>yold)   %it means the new
                  % individual performed better
     xold=xnew;
     yold=ynew;
     set(h,'Color','b')
     set(h,'XData',xold,'YData',yold);
     pause(1)
     xsolutions(nm)=xold;
     ysolutions(nm)=yold;
     sprintf('x accepted=%g,and its y values is=%g',…
     ,[xold yold])
  else if (exp((ynew-yold)/Temp)>r)
     xold=xnew;
     yold=ynew;
     set(h,'Color','b')
     set(h,'XData',xold,'YData',yold);
     pause(1)
    xsolutions(nm)=xold;
    ysolutions(nm)=yold;
    sprintf('Bad solution accepted X=%g\n',[xold])
  end
```

Hill Climbing Testing Code

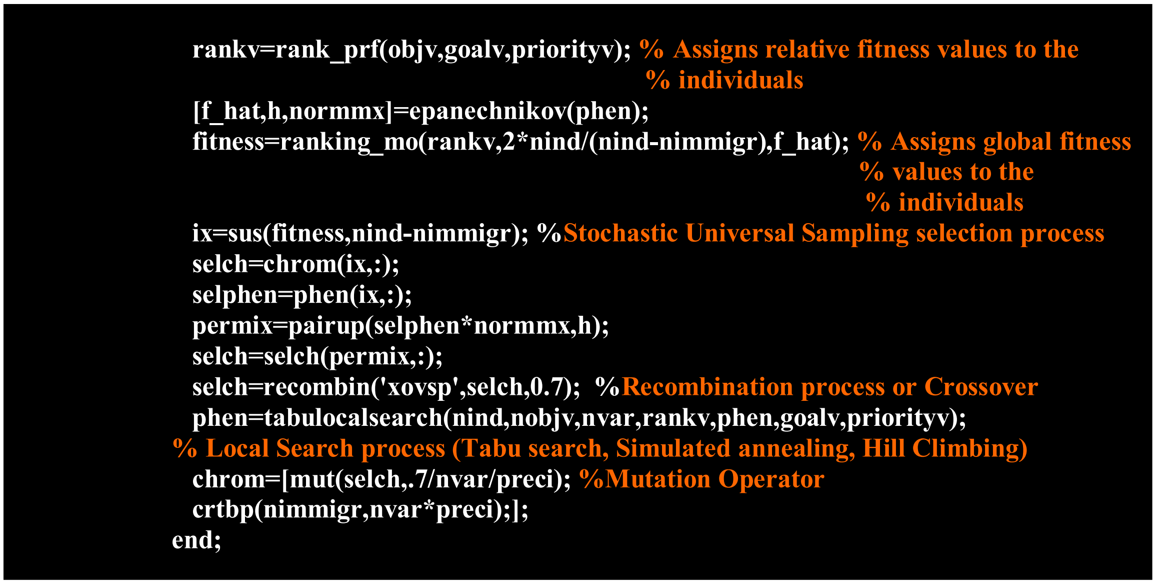
Cont

```
pause(1)
count=2;
for nm=1:Num_Steps,
   xnew = xold-0.1 + 0.2*rand(1,1);
   if xnew>4
     ynew=sin(xnew);
   else
      ynew=(-1/2)*(xnew*xnew)+4*xnew-8;
   end



     if(ynew>yold)    %it means the new
                 % individual performed better
           xold=xnew;
           yold=ynew;
           set(h,'Color','b')
           set(h,'XData',xold,'YData',yold);
           pause(1)
           xsolutions(nm)=xold;
           ysolutions(nm)=yold;
           sprintf('x accepted=%g,and its
          y values is=%g',[xold yold])
    end
   if xold == 4 % i.e .local max acheived
      xsolutions(nm)=-12;%→invalid entry
      sprintf('Local Max achieved,No more
 acceptance steps, current X=%g\n',[xnew])
   end
end
```

Application Domain: MOGA Hybridized with a local search process

```
while gen<maxgen,
  drawnow;
  gen=gen+1;
  phen=bs2rv(chrom,fieldd); % This function decode the phenotypic representation
                            % of the individuals into real numbers.
  for indno=1:nind;
    objv(indno,:)=xobjvfn(phen(indno,:)); % Assessment of the objective function
  end;
  [ix,bestix] = find_nd(objv,bestobjv);
  bestobjv = [bestobjv(logical(bestix),:) ; objv(logical(ix),:)];% Stores best objectives
                                                                 % values
  bestphen = [bestphen(logical(bestix),:) ; phen(logical(ix),:)]; % Stores best
                                                                  % individuals
```

Application Domain: MOGA Hybridized with a local search process Cont



```
  rankv=rank_prf(objv,goalv,priorityv); % Assigns relative fitness values to the
                                         % individuals
  [f_hat,h,normmx]=epanechnikov(phen);
  fitness=ranking_mo(rankv,2*nind/(nind-nimmigr),f_hat); % Assigns global fitness
                                                         % values to the
                                                         % individuals
  ix=sus(fitness,nind-nimmigr); %Stochastic Universal Sampling selection process
  selch=chrom(ix,:);
  selphen=phen(ix,:);
  permix=pairup(selphen*normmx,h);
  selch=selch(permix,:);
  selch=recombin('xovsp',selch,0.7);  %Recombination process or Crossover
  phen=tabulocalsearch(nind,nobjv,nvar,rankv,phen,goalv,priorityv);
% Local Search process (Tabu search, Simulated annealing, Hill Climbing)
  chrom=[mut(selch,.7/nvar/preci); %Mutation Operator
  crtbp(nimmigr,nvar*preci);];
end;
```

EA	Evolutionary Algorithm
GA	Genetic Algorithm
GUI	Graphical User Interface
HC	Hill Climbing
LS	Local Search
MA	Memetic Algorithm
MOGA	Multi Objective Genetic Algorithm
NIND	Number of Individuals
NVAR	Number of Variables
PHEN	Phenotype
RR	Rolls-Royce
SA	Simulated Annealing
SSPR	Stochastic Sampling with Partial Replacement
SSR	Stochastic Sampling with Replacement
SUS	Stochastic Universal Sampling
TS	Tabu Search