Sensitivity to Parameters and General Applicability of Genetic Algorithms and Simulated Annealing Algorithms for Mapping Data to Multicomputers

By
Jalal Y. Kawash
B.Sc., American University of Beirut

THESIS

Submitted in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science
at the Lebanese American University
December 1994

Signatures Redacted

Dr. Nashat Mansour (Advisor)
Assistant Professor of Computer Science
Lebanese American University

Signatures Redacted

Dr. Wadih Jureidini
Associate Professor of Mathematics and Computer Science
American University of Beirut

Signatures Redacted

Dr. George Nasr
Assistant Professor of Electrical and Computer Engineering
Lebanese American University

Abstract

We analyze the sensitivity to parameters and the general applicability of genetic algorithms and simulated annealing algorithms for mapping data to distributed-memory multicomputers, using the loosely synchronous computation model. The analysis includes sensitivity to user parameters, fault tolerance capability, and applicability to different multicomputer topologies. The user parameters are either objective function dependent or algorithm dependent. The fault tolerance capability is demonstrated by using the mapping algorithms for mapping data to a multicomputer that has some failed processors. We assume a hypercube multicomputer architecture in most experiments. However, comparative results for mesh, array, ring, tree, star graph, and fully connected topologies are presented. The mapping algorithms used are sequential hybrid genetic algorithm, versions of a distributed genetic algorithm, sequential simulated annealing algorithm, and a simulated parallel simulated annealing algorithm. The experimental results verify that these algorithms are insensitive to user parameters in wide ranges, completely fault tolerant, and unbiased towards particular multicomputer topologies. These results support the conjecture that physical optimization algorithms are flexible and have general applicability, where these properties are necessary for the automation of the mapping process.

Acknowledgments

First of all, special thanks go to Professor Nashat Mansour, my advisor, for all the help, encouragement, and support he provided me. In the absence of a research environment, I could not have completed this work without him.

I am grateful to the second readers Professor Wadih Juriedini and Professor George Nasr. I greatly thank Miss. Rima Slim for her help especially her comments and continuous reviews of these chapters. I also thank my colleagues Gabriel Aghazarian and Khaled Fakih for the valuable discussions, the A.C.C. supervisor Mr. Tarek Dana, the assistant A.C.C. supervisor Mr. Ali Aleywan, and all those who assisted me to accomplish this thesis, in a way or another.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Let *ALGORITHM* be an algorithm intended to solve a problem with an underlying data set, *DATA*, on a multicomputer, *MCOMP*. Then, the data mapping problem is the problem of partitioning *DATA* into mutually exclusive subsets, each of which is mapped to a processor node in *MCOMP*. The aim is minimizing the execution time of the parallel *ALGORITHM* on *MCOMP*.

The assumptions made in this work are the following:
(a) *MCOMP* is a distributed-memory message-passing multicomputer whose processors are connected by a static point-to-point interconnection network [Hwang 1993].
(b) The Routing used in *MCOMP* is wormhole routing [Hwang 1993].
(c) The computation model used is loosely synchronous computation model in which processors perform compute-communicate cycles in a SPMD (Single Program, Multiple Data) scheme [Fox et al. 1988].

Under such assumptions, the execution time on *MCOMP* is determined by the execution time of the slowest processor. Thus, the data mapping problem is obviously an optimization problem, and achieving data-parallelism involves nearly-equal distribution of computation workloads over the processor nodes of *MCOMP*, as well as the minimization of the amount of their inter-processor communication. The data mapping problem is NP-complete, and no optimal solutions can be found in reasonable time. Instead, several methods have been proposed to find acceptable and near-optimal solutions.

The methods that have been proposed to solve the data mapping problem fall into two major categories: heuristics and physical optimization algorithms. Heuristic procedures are mainly geometry based methods. The physical optimization algorithms are derived from natural sciences. Heuristics include Recursive coordinate bisection, recursive spectral bisection, recursive graph bisection, mincut-based, scattered decomposition, nearest-neighbor techniques, greedy algorithms, and clustering methods [Berger and Bokhri 1987; Chrischoides et al 1991; Ercal 1988; Fox 1988; Karmer and Muhlenbein 1989]. On the

other hand, physical optimization algorithms include neural networks [Hopfield and Tank 1986; Fox and Furmanski 1988], simulated annealing [Kirkpatrick et al. 1983; Rutenbar 1989], genetic algorithms [Holland 1975; Goldberg 1989], and mean-field annealing [Bultan and Aykanat 1992]. Genetic algorithms are derived from the theory of natural evolution in Biology. Simulated annealing is based on the process of solid annealing in statistical mechanics.

Heuristics for data mapping are fast. However, most of them are biased towards certain problem structures and multicomputer topologies. Most heuristics are not guided by an objective function the fact that leads to an unbalanced treatment of the computation and communication terms. Physical optimization algorithms, particularly genetic algorithms and simulated annealing, are slow in their execution. However, they do not make apriori assumptions concerning the underlying problem, and they are guided by an objective function [Mansour and Fox 1991, 1992, 1994a, 1994b]. It has been claimed that they are flexible and of general applicability [Mansour and Fox 1992]. Moreover, it is conjectured that these algorithms do find suboptimal mappings for different situations, such as mapping to faulty architectures and to a variety of multicomputer or topologies. However, these claims and conjectures have not been supported by sufficient experimental work.

In this work, we explore and analyze, for the first time, the behavior of genetic and simulated annealing algorithms for different situations. Our analysis has three dimensions. The first dimension deals with the degree of sensitivity of a mapping algorithm to its parameters which require user intervention. The second constitutes the fault tolerance capability of a mapping algorithm. An algorithm is said to be fault tolerant if it is capable of mapping to an incomplete (faulty) *MCOMP* architecture. The third dimension studies whether a mapping algorithm is biased towards a particular *MCOMP* topology. Our experimental results verify that genetic algorithms and simulated annealing algorithms are insensitive in wide ranges to most user parameters. They are completely fault tolerant, and they apply to different multicomputer topologies.

This thesis is organized as follows. Chapter 2 defines the data mapping problem in detail along with our assumptions based on which an objective function is devised. In chapter 3, we outline the genetic algorithms used in the analysis. The genetic algorithms are either sequential or distributed. Chapter 4 outlines the simulated annealing algorithms we use. Chapters 5 and 6 explores the first dimension of the analysis. In chapter 5, we study the sensitivity of genetic algorithms to user parameters. In chapter 6, we repeat the same work for simulated annealing. Fault tolerant mapping, the second dimension, is explored in

chapter 7 for genetic algorithms as well as simulated annealing. In chapter 8, we address the third and last dimension, bias towards multicomputer architecture. Finally, we present our conclusions and remarks in chapter 9.

Chapter 2

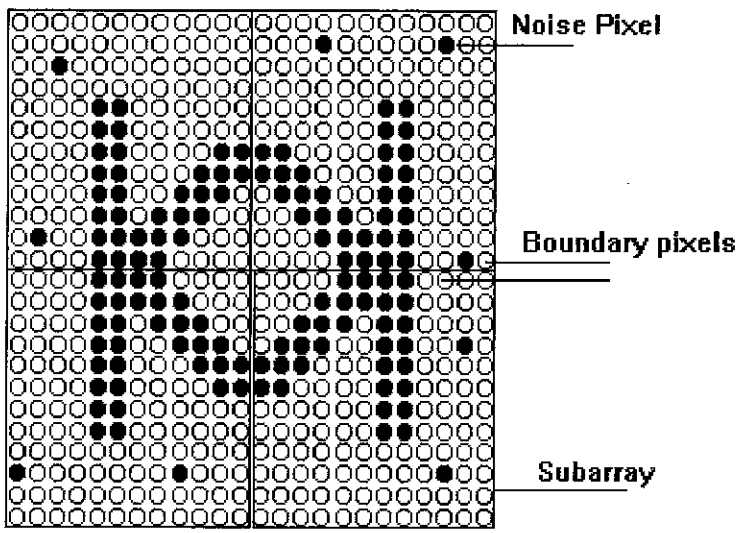
Data Mapping Problem

Let *ALGORITHM* be an algorithm intended to solve a given problem with an associated data set, *DATA*, on multicomputer *MCOMP*. The data mapping problem is the problem of partitioning *DATA* into mutually exclusive subsets, and mapping each of these subsets to a processor in *MCOMP*. The aim is minimizing the execution time of *ALGORITHM* on *MCOMP*. The execution time of the parallel program depends on the algorithm itself, the underlying data set, the computation model, and the multicomputer machine.

In this chapter, we define the data mapping problem in details. We also present our assumptions and the objective function used.

1. Data and Multicomputer Representation

The data set can be represented mathematically as a graph $G_C(V_C,E_C)$. $G_C(V_C,E_C)$ will be referred to as the computation graph. The set of vertices V_C corresponds to data objects on which computations are to be performed. Elements of E_C, the set of edges, represent the computation dependencies among the data objects. These dependencies are specified by the algorithm which is intended to run on the given data set.
As an example, consider the problem of image processing described in figure 1. Figure 1(a) shows a shape digitized on a 24x24 array of pixels. It shows also some noise pixels. The problem is to eliminate these noise pixels using the Jacobi algorithm shown in figure 1(b). Each pixel (data object) in the given data set will be represented as a vertex in the computation graph corresponding to this problem as shown in figure 2. The presented Jacobi algorithm identifies a noise pixel by determining the values of the eight surrounding pixels. Thus, the computation value of a data object depends on the values of its direct neighbors. Moreover, this dependence should be translated in the form of edges in the computation graph.



(a)

$$\text{Pixel\_Value}(x,y) = [\ \text{Pixel\_Value}(x-1,y) + \text{Pixel\_Value}(x-1,y-1) + \text{Pixel\_Value}(x-1,y+1) + \text{Pixel\_Value}(x,y+1) + \text{Pixel\_Value}(x,y-1) + \text{Pixel\_Value}(x+1,y) + \text{Pixel\_Value}(x+1,y-1) + \text{Pixel\_Value}(x+1,y-1)\ ] / 8$$

(b)

Figure 1. (a) A shape digitized on a 24x24 array of pixels. The array, which is decomposed into 4 sub-arrays, has few noise pixels. (b) A Jacobi algorithm to eliminate noise pixels.

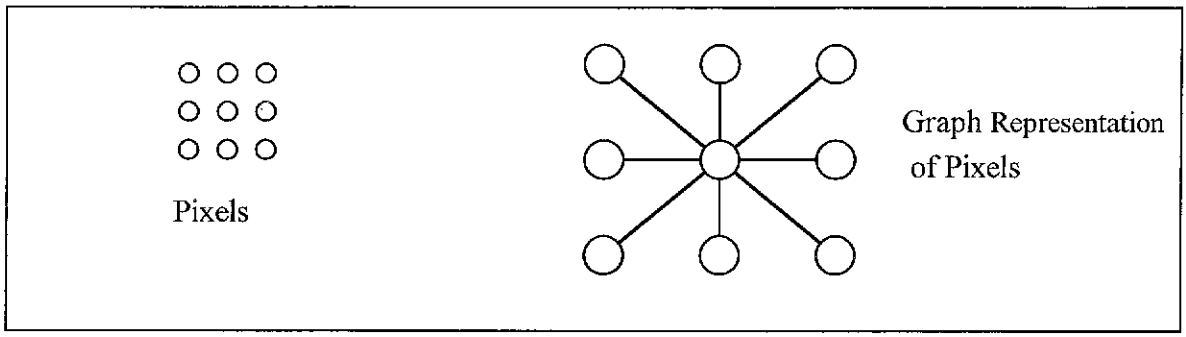


Figure 2. Representation of a pixel as a computation graph vertex and data dependencies as edges.

The multicomputer topology can be represented also by the graph $G_m(V_m,E_m)$. In this graph, V_m constitutes the set of multicomputer nodes, while E_m constitutes the actual physical interconnections between the nodes. The graphs in figure 3 represent two cubes of 4 nodes and 8 nodes respectively.

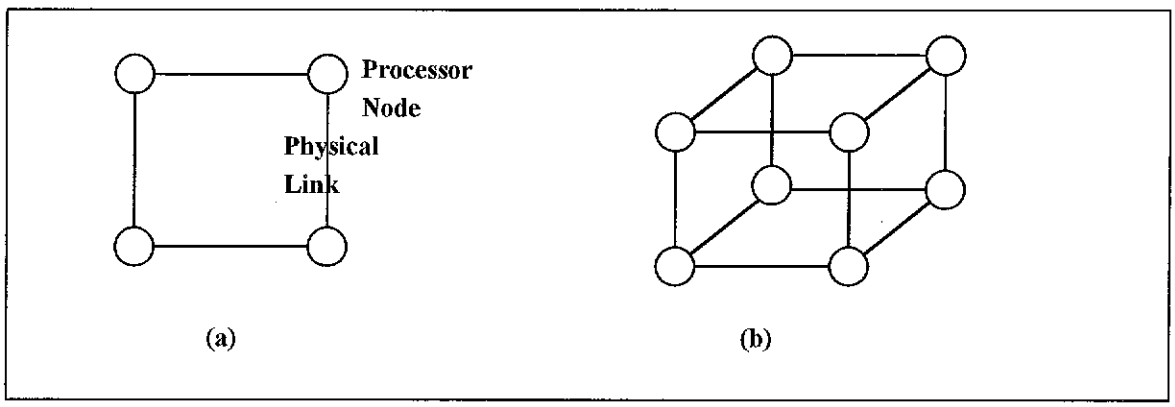


Figure 3. (a) 4-node cube graph. (b) 8-node cube graph.

2. The Computation Model

To devise an appropriate objective function, we assume a loosely synchronous computation model. This model is assumed in this work because the majority of large scale computations fall into this category [Fox et al. 1988]. In such a model, processors perform compute-communicate iterations, i.e., each processor runs the same algorithm on a different subgraph, then they exchange information pertaining to boundary vertices before proceeding with computation. Therefore, the total parallel execution time is determined by the slowest processor.

3. Problem Definition

The data mapping problem, also called the data allocation problem, refers to partitioning G_C into $|V_m|$ subgraphs each of which is mapped, or allocated, to a node in G_m. Thus it is an optimization problem which constitutes the determination of the onto function,

$$f: V_C \rightarrow V_m$$

such that the execution time of *ALGORITHM* represented by an objective function is minimized [Mansour and Fox 1992]. The mapping configuration is represented by the array MAP[ ], where

MAP[v] = p when the vertex v is among the vertices mapped to processor p.

Figure 1(a) proposes a mapping solution to the image processing problem. It is partitioned into 4 sub-arrays so that each node in the multicomputer in figure 2(a) can run the Jacobi algorithm on a 12x12 sub-array. The solution aims at balancing the load of the processors by mapping equal number of pixels to each processor node. Each processor, now, runs the Jacobi algorithm on the sub-array allocated (mapped) to it in much the same way a sequential processor runs it on the whole array. The main difference is that the data set of a node in a multicomputer is smaller, and each processor node will be obliged to do some communication with other processors to determine the values of the boundary vertices.
For example, a pixel, (x,y) on the boundary, mapped to processor *p1* (refer to figure 1(a)) has at least one neighbor, say (x,y+1), mapped to processor *p2* with $p1 \neq p2$. To determine Pixel_Value(x,y), processor *p1* has to request Pixel_Value(x,y+1) which is only known by processor *p2*. Thus, *p2* has to send it to *p1*, so that the later may proceed with computation.

4. The Objective Function

OF_{typ} is a typical objective function corresponding to the time taken by the slowest processor in computing and communicating in a loosely synchronous computation model [Mansour and Fox 1992, 1994a, 1994b].

$$OF_{typ} = Max_{p \in Vm}\{W(p) + C(p)\} \quad \textbf{(1)}$$

$W(p)$ corresponds to the computation workload of processor p. It is given by

$$W(p) = \sum_{v \in V_c} w(v).\delta(v,p) \quad \textbf{(2)}$$

where

$$\delta(v,p) = \begin{cases} 1 \text{ if } v \text{ is mapped to } p \\ 0 \text{ otherwise} \end{cases}$$

$w(v)$ is the computation time per vertex v. It is given by

$$w(v) = t_{float} \,.\, \lambda \,.\, \theta(v) \quad \textbf{(3)}$$

where t_{float} is the time taken by a processor to perform a floating point operation. Hence, it is the smallest reasonable time of a machine operation. λ corresponds to the number of computation operations per an edge in E_C in one iteration. $\theta(v)$ is the degree of vertex v in G_C. Substituting equation 3 in equation 2 yields

$$W(p) = t_{float}.\lambda.\sum_{v \in V_c} \theta(v).\delta(v,p) \quad \textbf{(4)}$$

Let $S_v(p) = \sum_{v \in V_c} \theta(v).\delta(v,p)$. That is, $S_v(p)$ denotes the number of edges in E_C which are mapped to processor p. Thus,

$$W(p) = t_{float}.\lambda.S_v(p) \quad \textbf{(5)}$$

$C(p)$ in equation 1 corresponds to the amount of communication processor p is performing. Clearly, $C(p)$ depends on the multicomputer specifications ,and hence it will differ from one machine to another. An expression for $C(p)$ which is suitable for modern

multicomputers which implement wormhole routing is given in equation 6 [Hey 1990; Hwang 1993]. It depends on the message latency and the number of processors a given processor communicates with. Recall that the processor nodes in a hypercube are numbered in such a way that the Hamming distance of two nodes gives the number of hops between them. Except for chapter 9, we assume a cube topology in this work.

$$C(p) = t_{float} \cdot \sum_{q \in V_m} \left[\rho . B(p,q) + \sigma + \tau . H(p,q) \right] . \mathrm{sgn}\left(B(p,q) \right) \qquad \textbf{(6)}$$

ρ is the machine time needed to communicate one word. It is given by the number of "t_{float}"s (i.e., it is divided by t_{float}) required to communicate a word. σ is the message start-up time divided by t_{float}. τ is the number of "t_{float}"s representing the communication time per unit distance. Clearly, ρ, σ, and τ are machine dependent parameters. $B(p,q)$ is the number of vertices mapped to p and are boundary with q. It is given by

$$B(p,q) = b. \sum_{\substack{v,u \in V_c \\ <v,u> \in E_c}} \delta(v,p) . \delta(u,q) \qquad \textbf{(7)}$$

b represents the number of values to be communicated per vertex. $H(p,q)$ in equation 6 corresponds to the Hamming distance between processors p and q. Finally

$$\mathrm{sgn}(x) = \begin{cases} 1 , x > 0 \\ 0 , x = 0 \end{cases}$$

$C(p,q)$ is considered to be a reasonable estimation. It is not precise, but it is popular in the mapping literature. Precise measures are sometimes impossible to express accurately. For example, link contention, communication-computation overlaps, and synchronization delays are hard to quantify.

The speed up (S) of the multicomputer is defined as being the workload of a sequential processor divided by the workload of the slowest processor in the multicomputer.

$$S = \frac{\text{Workload of a single processor}}{\text{Workload of the slowest processor in a multiprocessor}}$$

It is thus

$$S = \frac{\sum_{p \in Vm} W(p)}{OF_{typ}} \qquad \textbf{(8)}$$

The solution quality (η) is determined by the efficiency resulting from the mapping solutions. It is defined as the speed up (S) divided by the number of processors involved $(|Vm|)$. Thus

$$\eta = \frac{\sum_{p \in Vm} (W(p))}{|Vm| . OF_{typ}}$$

$$\eta = \frac{\lambda . \sum_{p \in Vm} S_v(p)}{|Vm| . Max_{p \in Vm} \{\lambda . S_v(p) + C(p)\}} \qquad \textbf{(9)}$$

Minimizing OF_{typ} is computationally expensive because $W(p)$ and $C(p)$ correspond to conflicting requirements. In other words, it gives rise to a min-max criterion. Maximum work load per processor corresponds to minimum communication and vice versa. Nevertheless, OF_{typ} is not smooth, which means that the computation of an incremental change resulted from the remapping of a vertex from one processor to another is expensive because it may require the calculation of the loads of all processors. We will circumvent this problem by using a quadratic objective function, OF_{appr}, which is considered to be a good approximation of OF_{typ} [Mansour and Fox 1992, 1994a, 1994b; Fox et al. 1988].

$$OF_{appr} = \lambda^2 . \sum_{p \in Vm} S_v^2(p) + \mu . \sum_{p \in Vm} C(p) \qquad \textbf{(10)}$$

μ represents the relative importance of the communication term to the computation term. It is given by

$$\mu = \mu_{user} \cdot \frac{\lambda^2 . \sum \theta(v)}{4 \rho |V_m| \sqrt{V_c \left(\sqrt{|V_m|} - 1 \right)} . E_b\_V_b} \qquad \textbf{(11)}$$

The derivation of μ is given in [Mansour 1992]. μ_{user} is a scalar value, $0 < \mu_{user} < 1$, which should be selected by the user based on experience. $E_b\_V_b$ is also a user-defined value which reflects the average ratio of boundary edges to boundary vertices in a mapping solution.

Smooth objective functions are preferred in optimization methods. Moreover, OF_{appr} possesses a locality property which makes the calculation of an incremental change due to remapping of vertices from one processor to another inexpensive. This locality implies that only the processors involved in the remapping and the remapped objects determine the cost. Equation 12 presents ΔOF_{appr} which corresponds to the incremental change resulted from the remapping of v from processor *p1* to processor *p2*.

$$\Delta OF_{appr} = 2\lambda^2 \theta(v) \left[\theta(v) + S_v(p2) - S_v(p1) \right] + \mu[\Delta C] \qquad \textbf{(12)}$$

The use of OF_{appr} is very important to Genetic Algorithms and Simulated Annealing because these algorithms employ vertex remapping extensively.

Chapter 3

Genetic Algorithms for Data Mapping

Genetic algorithms (GAs) are based on the theory of natural evolution where species adapt beneficially to the surrounding environment [Holland 1975; Goldberg 1989]. GAs work on a population of possible solutions. These solutions are called individuals or chromosomes, and they evolve over successive generations seeking an optimal solution. The initial population is selected at random, and individuals in a generation reproduce and mate to produce a new generation propagating favorable characteristics. This allows GA to finally converge to a good optimum.

In this chapter, we briefly describe GAs dedicated to solve the data mapping problem . First, we describe a sequential hybrid genetic algorithm, henceforth referred to as SGA. Then, various versions of a distributed genetic algorithm (DGA) are presented.

1. Sequential Hybrid Genetic Algorithm

An outline of SGA proposed by [Mansour and Fox 1991] is given in figure 1. A data mapping configuration is represented as a chromosome consisting of $|V_C|$ genes. The allele value, i.e. the value assigned to a gene, is an integer between 0 and $|V_m|$ -1, representing a processor. Thus, CHROMOSOME[X] = Y implies that the data object X, an integer between 0 and $|V_C|$ -1, is mapped to processor Y. The fitness of the individual is given by $Fitness = 1/OF_{appr}$. SGA employs panmictic reproduction, where the whole population is considered as a single mating unit. First of all, individuals are sorted according to their fitness values. The fittest is assigned the maximum rank value which is 1.2, while the least fit individual is assigned the lowest rank value which is 0.8. Other individuals are ranked equidistantly between these values. Individuals which were assigned ranking values greater than 1 are assigned single copies in the list of reproduction trials. Then, the fractional parts are treated as probabilities [Baker 1985]. Moreover, the fittest-so-far individual is always

saved, and if it is better than the new generation's best individual, it is reinserted to the new population to replace the worst individual. The genetic operators used in SGA are: two-point crossover at 80% frequency, mutation at 1% frequency, and inversion at 0.5% frequency [Mansour and Fox 1991]. SGA is hybridized in the sense that it incorporates a greedy hill climbing heuristic where each offspring individual performs self hill climbing via the remapping of boundary objects from one processor to another. This perturbation, the remapping, is only accepted if $\Delta OF_{appr} \leq 0$. [Mansour and Fox 1992, 1994b] argues that the remapping of a data object takes place from overloaded processors to underloaded ones only. Finally, SGA converges when there is no improvement in efficiency for a number of generations.

```
Random generation of initial population, size POP;
Evaluate fitness of individuals;
repeat (for GEN generations)
   Set μ and genetic operator rates;
   Rank individuals and allocate reproduction trials;
   for i=1 to POP step 2 do
         Randomly select 2 parents from the list of reproduction trials;
         Apply crossover, mutation, and inversion;
         Hill-climbing by offsprings;
   end for;
   Evaluate fitness of offsprings;
   Preserve the fittest-so-far;
until convergence;
solution = fittest;
```

Figure 1. Outline for SGA.

2. Distributed Genetic Algorithms

In SGA, evolution is panmictic and sequential, which is not the case with natural evolution. In natural evolution, species evolve concurrently over discontinuous separated sub-populations called demes [Crow 1986].
In the DGAs described in this section, the population is structured as a cube graph where a node corresponds to a deme. Demes are numbered in exactly the same way processors are numbered in a hypercube multicomputer [Hwang 1993]. Thus, two demes are said to be neighbor demes if they are at a Hamming distance =1.

2.1. The Algorithm

DGA is outlined in figure 2. Initially demes are generated at random, and each deme is generated using different random number generator seeds. This allows DGA to start from different and diverse points. The population is divided into $|V_m|$ equal size demes which mate independently and discontinuously. This process is repeated for *DRFGEN* generations forming a drift phase. In the migration phase, some individuals migrate from one deme to another. Choosing a fixed size deme, the migrants have to replace some victims in the target deme [Mansour and Fox 1994a, 1994b].

2.2. Drift Phase

The DGA drift phase simply employs SGA in each deme [Mansour and Fox 1994a, 1994b].

2.3. Migration Phase

The migration phase depends on the migration policy used. In general, M% (of the deme) migrants are copied in some neighbor deme replacing M% victims. We present five versions of DGA which differ in the migration scheme, number of migrants, and nature of migrants. In some of the versions illustrated below, a migration direction has to be chosen

along which migration takes place. The migration direction is simply one of the cube dimensions [Tanese 1987, 1989].

```
for DM=0 to |V_m|-1 do
    Random generation of initial deme, DM, of size DEME;
    Evaluate fitness of individuals;
end for;
while (each deme is not converged) do
 /* Drift Phase*/
  for DM=0 to |V_m|-1 do
      repeat (for DRFGEN)
         Apply SGA to DM;
      end repeat;
  end for;
 /*Migration Phase*/
  Apply a migration policy (section 2.3);
end while;
Solution = Fittest;
```

Figure 2. Outline for DGA.

2.3.1. ZDGA

The simplest migration policy is 'Do No Migration' [Tanese 1989] . This algorithm will be referred to as ZDGA where M=0. Here we are relying on the drift phase to do the job. Moreover, we will be maintaining diversity among the demes at its highest peek from the beginning of evolution until its end.

2.3.2. 1wDGA1

A one-way migration policy is described in figure 3. The migration direction is determined in a circular fashion over the dimensions of the cube-like population. Along a selected direction, we have a set of deme couples.

```
Determine migration direction, MD;
for every deme couple (X,Y) on MD do
  Determine S and D demes;
  Copy M% best individuals in S to replace worst M% victims in D;
end for;
```

Figure 3. One-way migration policy.

Each couple consists of a source deme, S, and a destination deme, D, which are at a Hamming distance = 1 apart. Let (X,Y) be a couple of demes on the selected direction, we adopt two policies to identify S and D demes. In the first policy, X is an S deme if and only if the average fitness of X is greater than that of Y. The DGA corresponding to this policy will be referred to as 1wDGA1a. In the second policy, X is an S deme if and only if X contains the fittest individual among all individuals in X and Y. 1wDGA1b denotes the corresponding DGA.

2.3.3. 2wDGA1

This policy is depicted in figure 4. The pair of demes (X,Y) exchanges M% of their best individuals replacing their worst M% individuals . That is why this policy is called a two-way migration policy, and the corresponding DGA will be referred to as 2wDGA1 [Tanese 1987, 1989] .

```
Determine migration direction, MD;
for every deme couple (X,Y) on MD do
  Exchange M% best individual copies between X and Y;
  (copies replace the worst individuals)
end for;
```

Figure 4. Two-way migration policy.

2.3.4. SBDGA1

Migration in SBDGA1 is based on the shifting balance theory [Wright 1977]. In each neighborhood, the deme with the best individual is identified as the S deme, while every other deme in that neighborhood is a D deme. This policy is summarized in figure 5.

```
for every neighborhood,NH do
  Identify the S deme;
  Copy M% best individuals from S to every other deme in NH
    replacing the worst M% individuals;
end for;
```

Figure 5. Shifting balance based migration policy.

2.3.5. More Variations

1wDGA2a, 1wDGA2b, 2wDGA2, and SBDGA2 are four variations which deviate from their ancestors 1wDGA1a, 1wDGA1b, 2wDGA1, and SBDGA1 respectively. In these variations, M is selected at random in each migration cycle, and the M% migrants are chosen at random with replacement. The migration direction is also selected at random in 1wDGA2a, 1wDGA2b, and 2wDGA2.

3. Summary

Genetic algorithms are derived from the theory of natural evolution. Species mate and reproduce propagating their favorable aspects to their offsprings.

In this chapter, we described a sequential hybrid genetic algorithm, SGA, and versions of a distributed genetic algorithm, DGA, all dedicated to solve the data mapping problem. SGA is based on classical GA. The DGA versions presented in this chapter employ a distributed discontinuous population. SGA is reemployed in DGA in its drift phase. The DGA versions differ in the migration policies they adopt. These policies were also described.

Chapter 4

Simulated Annealing Algorithms for Data Mapping

To force a material to come to a low-energy state in statistical mechanics, it is heated to a temperature which allows many atomic rearrangements. It is then cooled slowly and carefully, allowing it to achieve thermal equilibrium at each temperature, until the material freezes and crystallizes. This process is called annealing, and simulated annealing (SA) algorithms are based on the idea of this annealing process [Kirkpatric et al 1983; Rutenbar 1989]. The simulation of the behavior of the physical system at each temperature in the cooling process is simulated by the Metropolis algorithm. SA starts with a randomly generated initial solution. In each iteration, the solution is randomly perturbed. If such a perturbation results in a negative or zero change in the system energy, then it is accepted. To prevent premature convergence when trapped in a bad optimum, the simulated annealing algorithm accepts positive changes in energy with a certain temperature-dependent probability.

In this chapter, we present two simulated annealing algorithms that are adapted to solve the data mapping problem [Hwang and Xu 1990; Mansour and Fox 1992, 1994a]. First, we briefly describe a sequential simulated annealing algorithm, referred to as SSA. Then, we present a simulation of a parallel simulated annealing algorithm, referred to as SPSA.

1. Sequential Simulated Annealing Algorithm

SSA is outlined in figure 1 [Mansour and Fox 1992]. The initial mapping configuration is generated at random. SA determines the initial temperature in such a way that makes the probability of accepting a positive change, $e^{-\Delta E/T}$, very high, 0.8. On the other hand, freezing is accomplished when the probability of accepting a minimum positive increase in

the energy function is very small. The energy function, E, is given by the objective function, OF_{appr}. A perturbation corresponds to randomly selecting a data object and remapping it at random. Accepted perturbations turn to be new starting points to the next perturbation. The perturb-accept or reject process is repeated until equilibrium is achieved.

```
Initial configuration, MAP[ ] = Random mapping;
Determine the initial temperature, T(0);
/* Anneal */
while ( T > THRESH1 and Nacc > THRESH2 ) do
      T= T(i);
      repeat
          Perturb (MAP[ ]);
          E = OFappr;
          if (dE <= 0) then
            Accept move;
            Update Configuration, MAP[ ];
          else
            Generate RND, a random number between 0 and 1;
          if (RND < e^(-ΔE/T)) then
            Accept and update;
         else Reject;
       until Equilibrium;
       Determine K;
       T(i+1) = K*T(i);
end while;
```

Figure 1. Outline for SSA.

The new temperature is determined as a fraction of the previous one. We choose a fixed cooling schedule, K. The best-so-far is persevered to avoid losing good solutions. Finally, SSA converges if N_{acc}=0 or there is no improvement for a threshold of successive moves.

2. Simulated Parallel Simulated Annealing Algorithm

An outline of SPSA is given in figure 2 [Greening 1990; Eglese 1990; Williams 1991; Mansour and Fox 1994a]. SPSA is a simulation of the parallel simulated annealing algorithm presented by [Mansour and Fox 1994a]. The algorithm starts by mapping MAP[] into the multicomputer, *MCOMP*, nodes. This is done by breaking MAP[] into contiguous segments, MS_0, MS_1, ... , such that these segments are of equal sizes (if possible). obviously, this is a naive mapping scheme which is far from optimal, but it is of negligible time. Next, each node generates its own random sub-configuration. Initial and freezing temperatures are determined globally using the global summation of $S_v(p)$. Then, SSA is repeated for every node until all nodes converge.

SPSA deviates from SSA which is highly sequential in the sense that perturbations occur successively. In SPSA, perturbations occur concurrently. A node is locally evaluating (accepting or rejecting) a perturbation with wrong values for $S_v(p)$ and ΔC used to evaluate ΔOF_{appr}. This leads to three types of inconsistencies referring to concurrent perturbations of data objects mapped to the same processor, wrong $S_v(p)$ value in different nodes, and outdated ΔC value in different nodes [Mansour and Fox 1994a]. A correction mechanism is needed to manage for these inconsistencies because if they accumulate they lead to degeneration. In other words, if the accumulation of these inconsistencies is allowed, SPSA will either converge prematurely, or we will be obliged to increase the number of passes to get reasonable solution qualities. Both choices are unacceptable. The later makes SPSA computationally expensive, while the former is unacceptable any way. The SPSA correction scheme is based on the unification of the views of the multicomputer nodes at certain frequencies. These frequencies are f_{mvs}, f_{bdary}, and f_{sv}. f_{mvs} is the frequency at which we update the local views of N_{acc} and N_{att} in each node to reflect their global values. Recall from SSA that N_{acc} and N_{att} are needed in order to determine equilibrium. f_{bdary} is the frequency at which the multicomputer nodes communicate the boundary information. In our simulation, this is done via copying the local parts of nodes, MS_i's , into MAP[], the global configuration. The last frequency, f_{sv} constitutes the frequency at which the global summation of $S_v(p)$ is determined. Too high correction frequencies result in high communication overhead. While too low correction frequencies may lead SPSA to premature convergence. An empirical determination of their values is necessary.

```
/* Mapping MAP[] into MCOMP */
for every node (n) in MCOMP do
    Determine MS_n boundaries SB_n and EB_n, The Start Boundary and End Boundary;
end for;

/* Initial random configuration *.
for every node (n) in MCOMP do
    Generate random configuration in MAP[SB_n..EB_n];
end for;

Determine initial temperature, T(0);
Determine freezing temperature, Tfreeze;
Determine boundary information;
Globally find summation for S_v(p);

while not converge ( every node in MCOMP ) do
for every node (n) in MCOMP do
while ( T(i) > Tfreeze and not converged(node n) ) do
   repeat
        Local SSA step;
        Update N_att and N_acc moves at f_mvs
        Write MS_n into MAP[SB_n.. EB_n] at f_bdary;
        Update S_v(p) at f_sv;
   until (equilibrium);
   if (N_acc < THRESH) then
      Save the best-so-far according to OF_typ;
   end if
   T(i) = K * T(i-1);
end while
```

Figure 2. Outline for SPSA.

3. Summary

Simulated annealing algorithms are based on the annealing process of a solid in statistical mechanics. In this process, the solid is heated and then cooled slowly until it freezes and crystallizes. At each temperature the solid achieves thermal equilibrium.

Two simulated annealing algorithms were outlined in this chapter. The first, SSA, is sequential in which perturbations occur successively. The second, SPSA, is a simulation of a parallel simulated annealing algorithm. SPSA deviates from SSA; thus, the sequential nature of the later leads to inconsistencies in the former. A correction scheme which employs correction frequencies was presented. These frequencies shall be determined experimentally.

Chapter 5

Sensitivity of GAs to User Parameters

In chapter 3, we presented SGA and versions of DGA which find good mapping solutions such that the execution time of *ALGORITHM* on *MCOMP* is minimized. These GAs make use of many parameters that require user intervention. Table 1 summarizes these parameters restating a short definition of each.

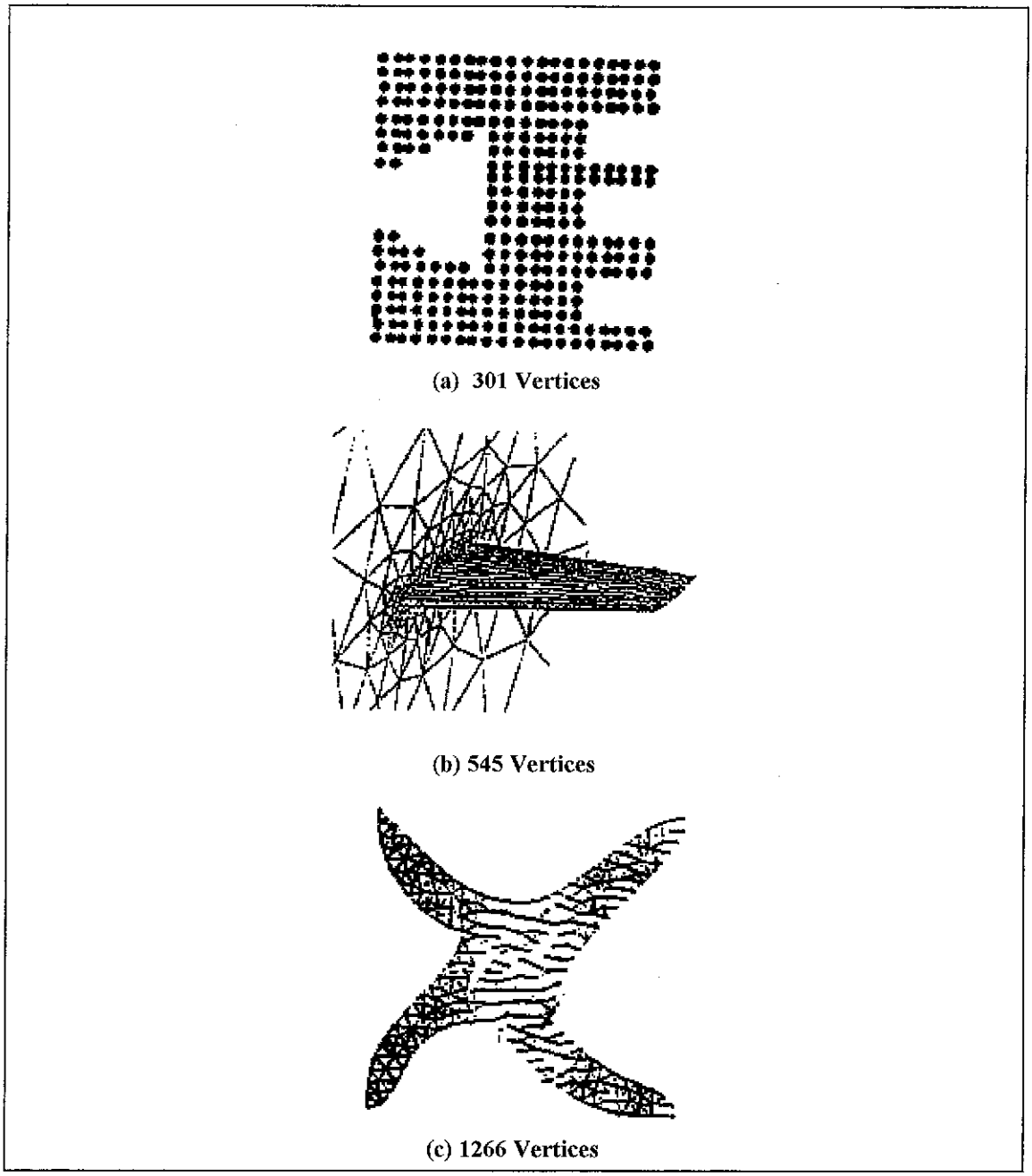
In this chapter, we study experimentally the sensitivity of GA to each of these parameters emphasizing the effect of misestimating their values to GA performance. Three test cases are used in our experiments: TCASE1, TCASE2, and TCASE3. Table 2 summarizes the characteristics of each test case, and the shapes of *DATA* are shown in the mentioned figures. We study the sensitivity of GA in terms of its solution quality and execution time. The solution quality, η, is evaluated using equation 9 in chapter 2. The execution time, t_{exec}, is measured in seconds. We note that TCASE2 and TCASE3 are contracted once and twice respectively using a graph contraction heuristic [Mansour 1992]. Finally, The experiments are conducted on an AViiON 5000 UNIX machine.

Parameter	Description
μ_{user}	Relative importance of communication to computation.
$E_b\_V_b$	Average ratio of boundary edges to boundary vertices.
λ	Computation operations per edge per iteration in *ALGORITHM*.
τ	*MCOMP* communication time per unit distance / t_{float}.
σ	*MCOMP* message startup time / t_{float}.
ρ	*MCOMP* time to communicate one word / t_{float}.
Population Size	Size, in individuals, of the GA population.
Convergence Threshold	Number of generations with no improvement in η.
Migration Policy	Migration phase policies used in DGA versions in chapter 3.
Drift Phase Length	Length of the drift phase in DGA versions.

Table 1. GA parameters whose values involve user's choice.

	$\|V_c\|$	$\|V_m\|$	θ_{avg}	*Contraction*	*Figure*
TCASE1	301	4	3	0	Figure 1(a)
TCASE2	545	8	11.5	1	Figure 1(b)
TCASE3	1266	16	5.5	2	Figure 1(c)

Table 2. Test cases.



(a) 301 Vertices

(b) 545 Vertices

(c) 1266 Vertices

Figure 1. Shapes of *DATA*. (a) TCASE1. (b) TCASE2. (c) TCASE3.

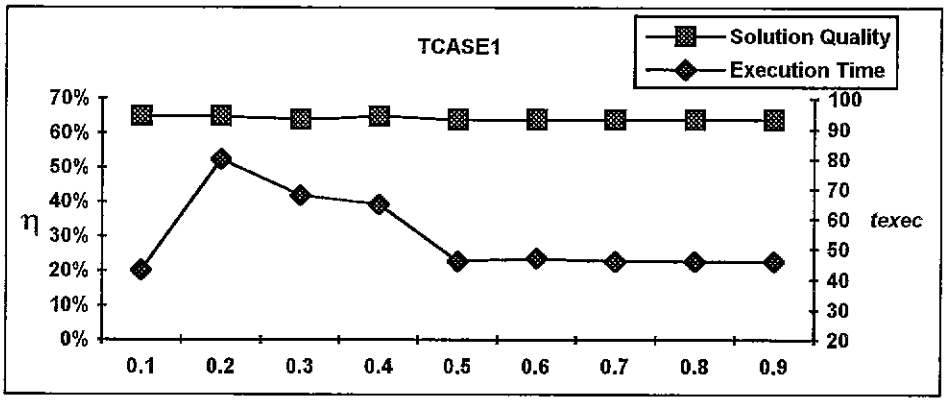


Figure 2. Performance of SGA for different values of μ_{user} (x-axis) using TCASE1.

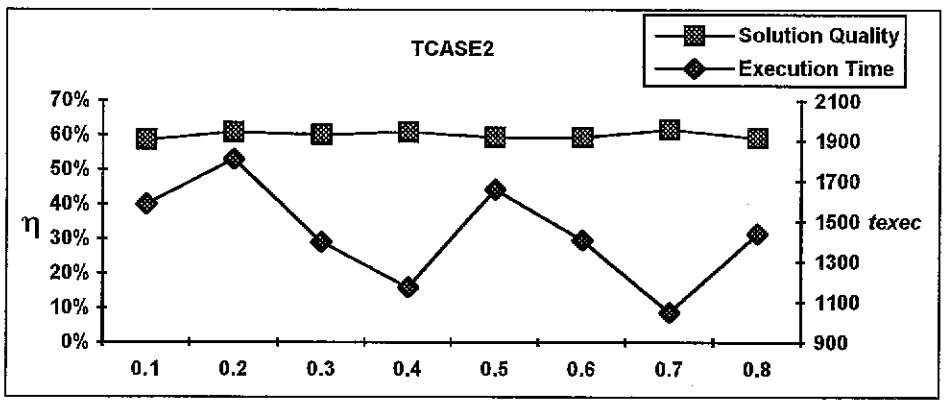


Figure 3. Performance of SGA for different values of μ_{user} (x-axis) using TCASE2.

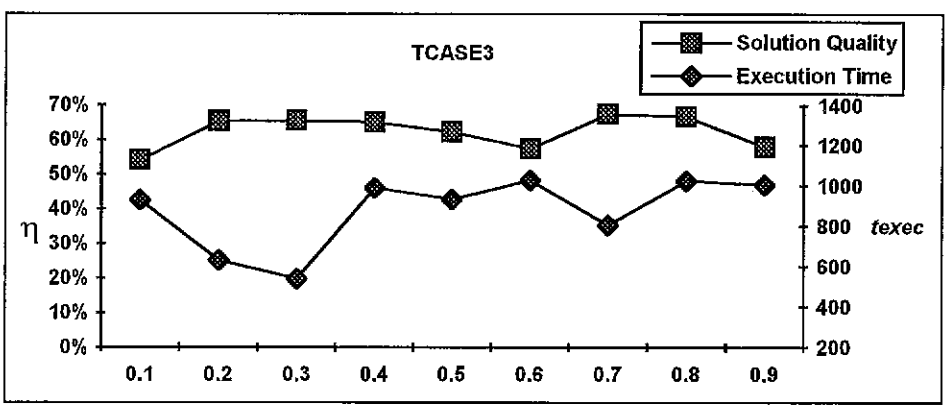


Figure 4. Performance of SGA for different values of μ_{user} (x-axis) using TCASE3.

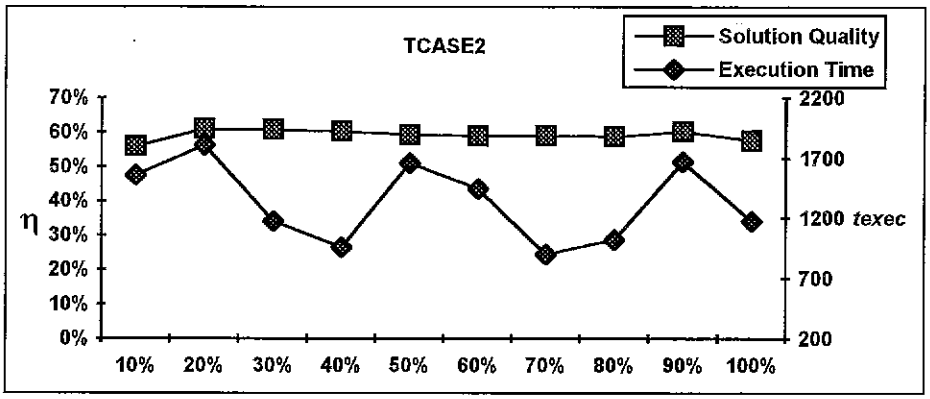


Figure 5. Performance of SGA for different values of E_b _V_b (expressed in terms of θ_{avg} on the x-axis) using TCASE2.

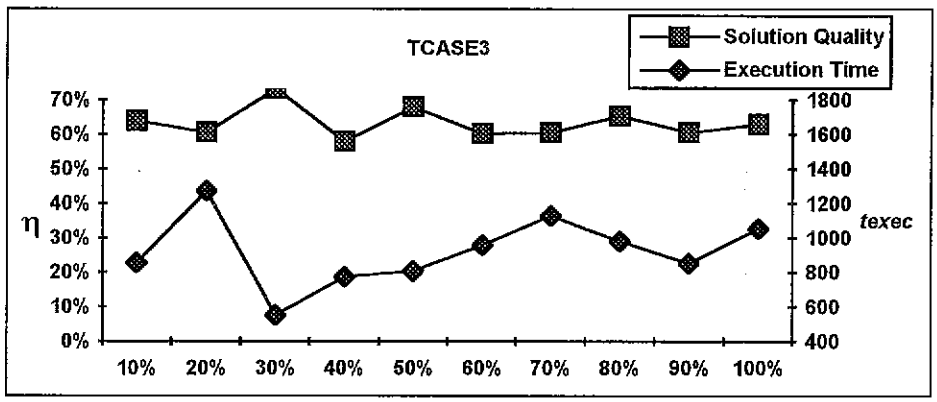


Figure 6. Performance of SGA for different values of E_b _V_b (expressed in terms of θ_{avg} on the x-axis) using TCASE3.

3. Sensitivity to λ

The effects of misestimating λ on SGA solution quality are shown in figures 7 and 8 for TCASE2 and TCASE3 respectively. Assuming the correct λ = 7, SGA is run using wrong estimates for λ. But, its solutions are evaluated using the correct λ.

For both data sets, TCASE2 and TCASE3, SGA shows insensitivity to the choice of λ. Values for λ were underestimated in 85% error or less, and overestimated in 328% error or less. Nevertheless, a 328% overestimation error, λ = 30, in TCASE3 only causes a drop of about 5% in solution quality. This makes SGA highly insensitive to the choice of λ.

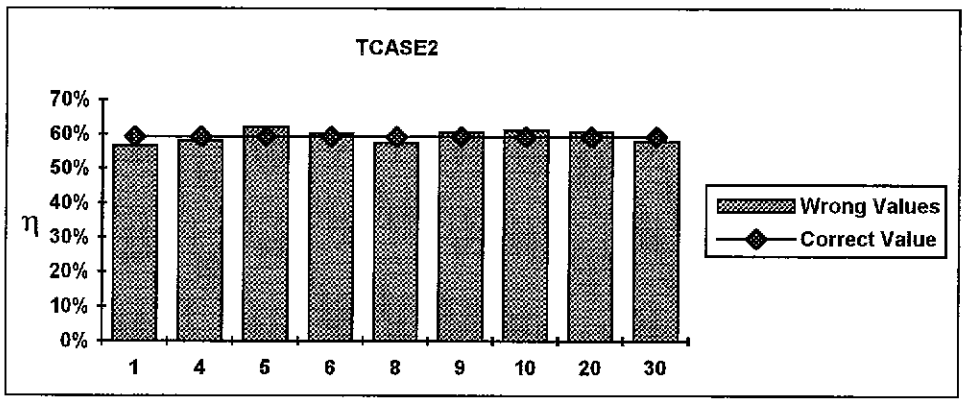


Figure 7. Quality of the solutions found by SGA using wrong values for λ (x-axis) compared to the solution found using correct λ (test case TCASE2).

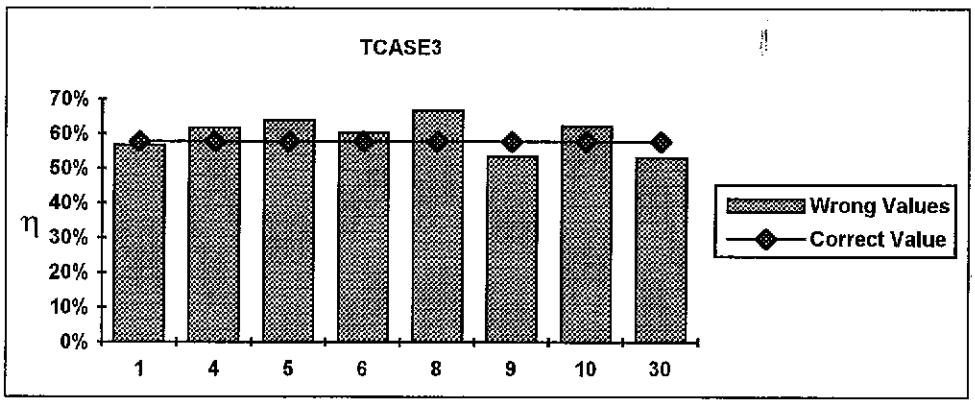


Figure 8. Quality of the solutions found by SGA using wrong values for λ (x-axis) compared to the solution found using correct λ (test case TCASE3).

4. Sensitivity to τ

We assume a correct τ value to be 100. The SGA is run using wrong values. But, its solutions are still evaluated using the correct value.

SGA experimental results for TCASE2 and TCASE3 are shown in figures 9 and 10 respectively. In both cases, we are using overestimation errors less than 200% and underestimation errors less than 90%. SGA is found to be totally insensitive in TCASE2, while TCASE3 favors overestimation. At the extreme, with an underestimation error = 90%, SGA shows a less than 10% decrease in solution quality. Thus, SGA solutions are not highly affected by the choice of τ.

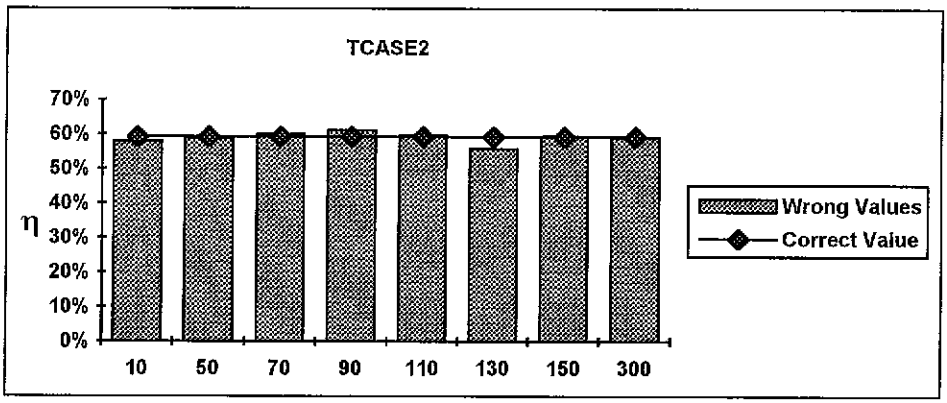


Figure 9. Quality of the solutions found by SGA using wrong values for τ (x-axis) compared to the solution found using correct τ (test case TCASE2).

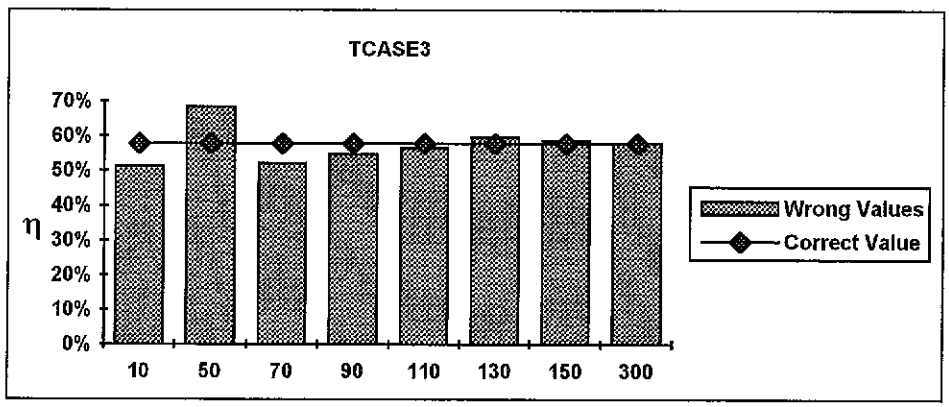


Figure 10. Quality of the solutions found by SGA using wrong values for τ (x-axis) compared to the solution found using correct τ (test case TCASE3).

5. Sensitivity to σ

SGA uses wrong estimates for σ in its execution. However its solutions are evaluated using the correct value for σ, which we assume to be 325.

Figures 11 and 12 show the sensitivity of SGA to σ for TCASE2 and TCASE3 respectively. In TCASE2, σ underestimation errors which are greater than 85% and overestimation errors which are less than 320% show that the SGA is totally insensitive to the choice of σ. Moreover, a 200% overestimation error yields only to a drop of less than 5% in solution quality. TCASE3 favors values for σ which are near the assumed correct value, i.e. between 150 and 450. This makes an underestimation errors' upper limit of more than 50%, and an overestimation errors' upper limit of less than 40% totally acceptable. In short, SGA is not sensitive to the choice of σ with an error range of ±50%

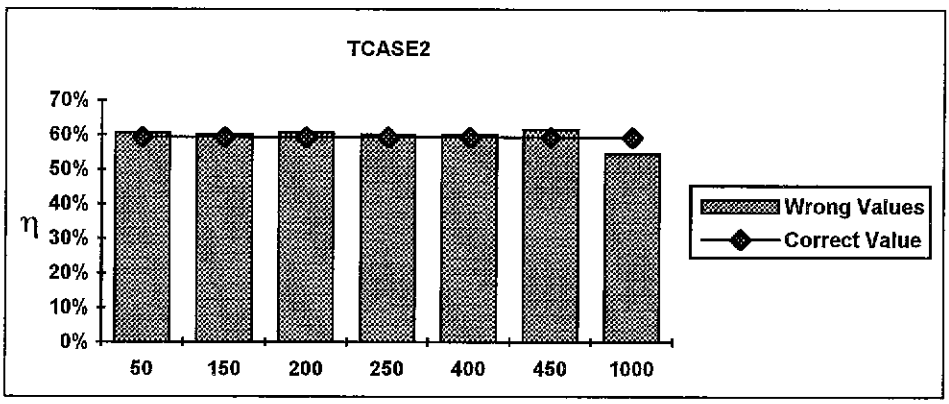


Figure 11. Quality of the solutions found by SGA using wrong values for σ (x-axis) compared to the solution found using correct σ (test case TCASE2).

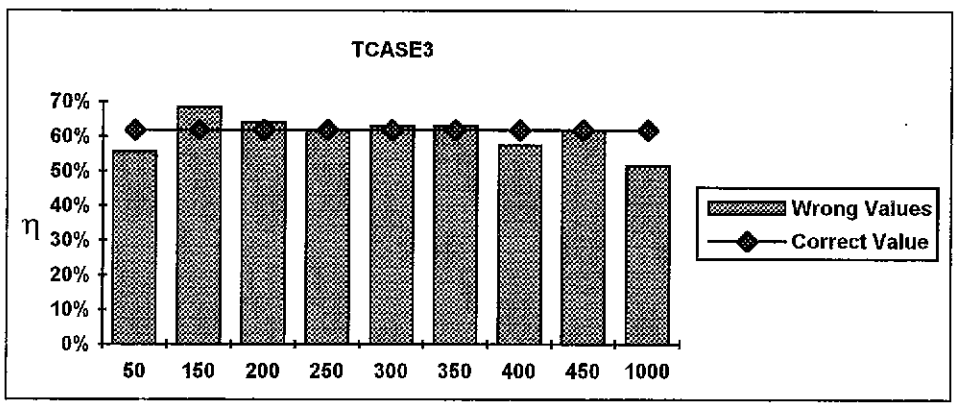


Figure 12. Quality of the solutions found by SGA using wrong values for σ (x-axis) compared to the solution found using correct σ (test case TCASE3).

6. Sensitivity to ρ

The correct value for ρ is assumed to be 15. SGA solutions are found using wrong estimates for ρ, but they are evaluated using the assumed correct value of ρ.

The results are shown in figures 13 and 14 for TCASE2 and TCASE3 respectively. The solution quality in both cases is not sensitive to the choice of ρ although both cases show that a misestimation of ρ may lead SGA to find a better quality solution (more than 10% sometimes). Moreover, worse quality solutions are rare, and they are only worse by less than 5%. In both cases, errors were chosen in a range less than 85% for underestimations, and less than 230% for overestimations. In this wide range SGA is insensitive to the choice of ρ.

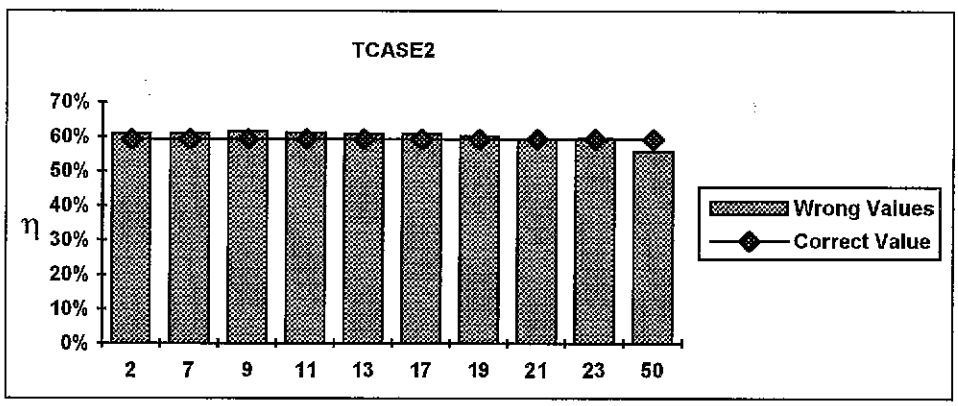


Figure 13. Quality of the solutions found by SGA using wrong values for ρ (x-axis) compared to the solution found using correct ρ (test case TCASE2).

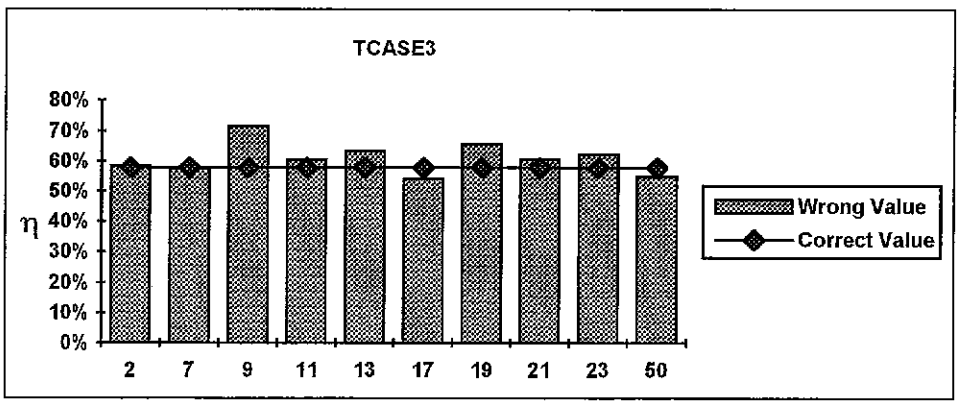


Figure 14. Quality of the solutions found by SGA using wrong values for ρ (x-axis) compared to the solution found using correct ρ (test case TCASE3).

7. Sensitivity to *Population Size*

The population size is one of the critical parameters in any GA. There is no general rule that determines exactly the population size; this parameter is problem dependent, and the user has to set it before executing the GA. A small population leads to less diversity among individuals and possibly to fast convergence to a bad optimum. On the other hand, a large population makes GAs inefficient in terms of time although they are more likely to converge to good optima.

Figure 15 shows the performance of SGA executed on TCASE1 for different population sizes. Too small populations lead SGA to converge to unacceptable solutions in negligible time. On the other hand, the SGA solution quality curve exhibits a straight line-like behavior for reasonable population sizes. The SGA execution time increases steadily with the population size. Therefore, moderate population sizes are preferable. For example, a 100 individual population gives almost the same solution quality given by a 250 individual population in about 4 times less execution time.

The results of SGA for different population sizes in TCASE2 are shown in figure 16. This case shows a straight line-like solution quality curve which implies that SGA is not sensitive in this case to the choice of the population size as far as the solution quality is concerned. However the SGA execution time curve shows an increasing trend which makes large population sizes computationally expensive.

The results of SGA on TCASE3 depicted in figure 17 show a fluctuating solution quality curve. This fact makes the choice of the population size really critical. For instance, a population of 210 individuals yields $\eta = 50\%$ in more than 1500 seconds. Whereas a 200 individual population yields $\eta = 70\%$ in only 600 seconds. The SGA execution time curve shows sharper fluctuations than the SGA solution quality curve at high population sizes although it shows an increasing trend as the population size increases. This implies that SGA execution time is highly sensitive to the population size.

In short, a moderate population size that depends on $|V_m|$ and θ_{avg} is preferable for it offers good solutions in an acceptable time. Therefore, we will use the population sizes 50, 100, 150 for TCASE1, TCASE2, and TCASE3 respectively.

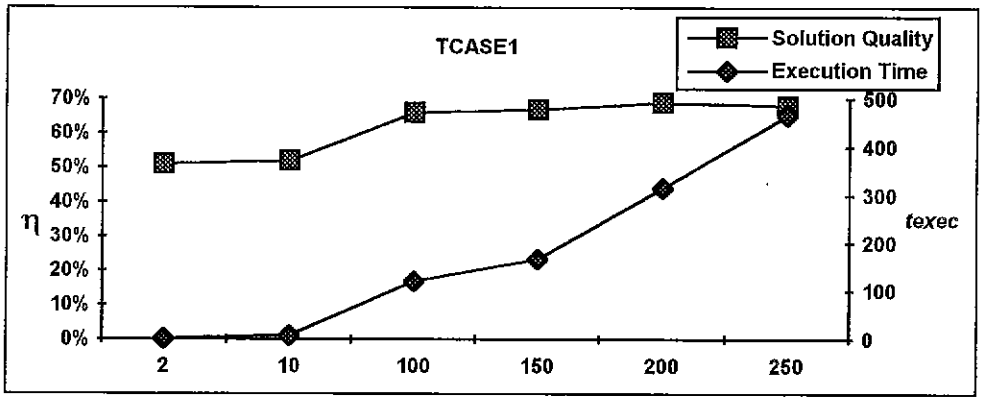


Figure 15. Performance of SGA for different population sizes (x-axis) using TCASE1.

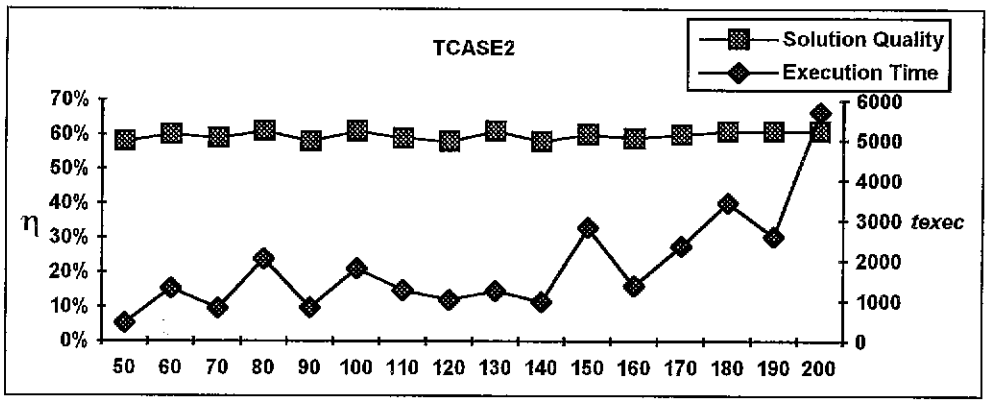


Figure 16. Performance of SGA for different population sizes (x-axis) using TCASE2.

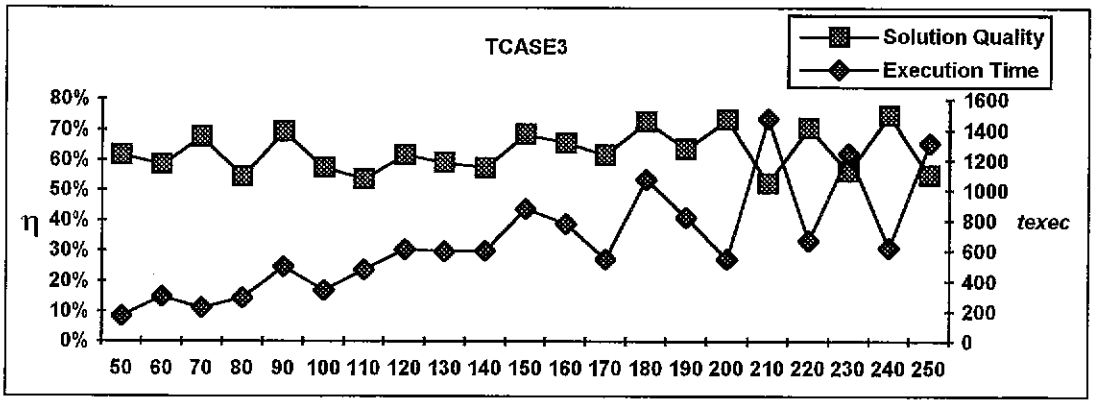


Figure 17. Performance of SGA for different population sizes (x-axis) using TCASE3.

8. Sensitivity to *Convergence Threshold*

SGA converges when there are no efficiency improvements for a number of generations, which we call here the convergence threshold, and refer to as *CT* for short.

Figures 18,19 and 20 show that the SGA solution quality is not affected by the choice of *CT*. However, they also show that the SGA execution time increases steadily and sometimes rapidly as *CT* increases. In general, choosing *CT* to be too small, i.e. less than 10 generations, may drive the SGA to premature convergence. On the contrary, a too large *CT* guarantees a good solution quality, but it makes SGA computationally expensive. In short, *CT* must be large enough to avoid convergence to local optima and small enough to save time. In our implementation, we use *CT*=15 generations.

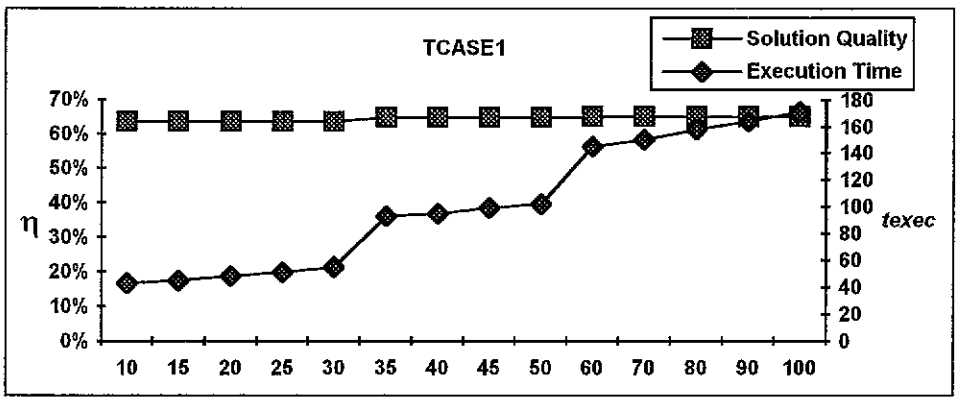


Figure 18. Performance of SGA for different values of *CT* (given in generations on the x-axis) using TCASE1.

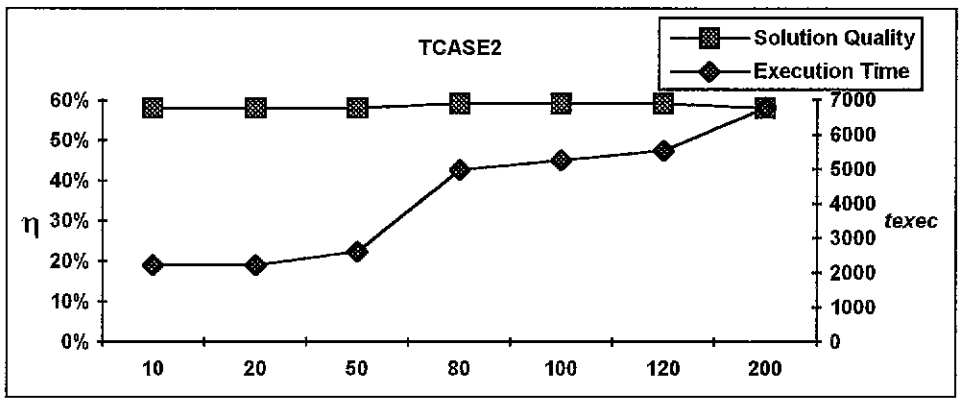


Figure 19. Performance of SGA for different values of *CT* (given in generations on the x-axis) using TCASE2.

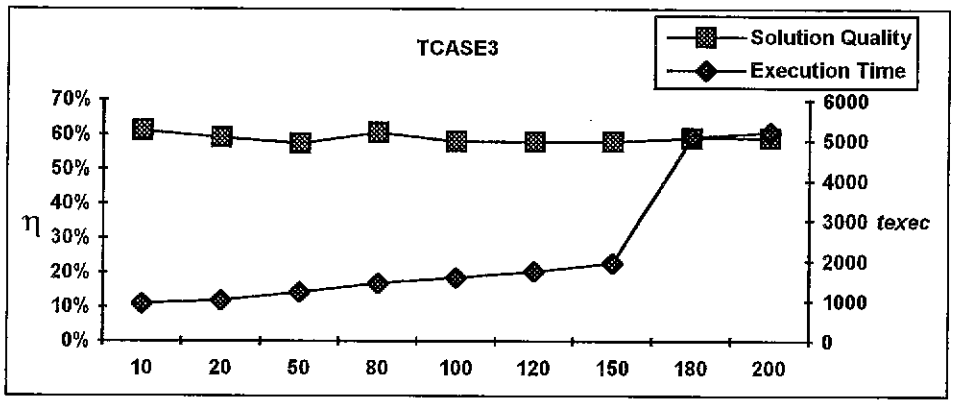


Figure 20. Performance of SGA for different values of *CT* (given in generations on the x-axis) using TCASE3.

9. Sensitivity to *Migration Policy*

The migration scheme, number of migrants, and nature of migrants are the three factors that identify the migration policy of a DGA. In each of the subsequent sections, we study the sensitivity of DGA to each of these three factors. The number of migrants is expressed as a percentage of the deme size, and it is denoted by M. In the implementation of each DGA version, we use $|V_m|$ demes and populations of 96 and 128 individuals for test cases TCASE2 and TCASE3 respectively. This means that the DGA's have 8 demes each of 12 individuals in TCASE2, and 16 demes each of 8 individuals in TCASE3. In most of the subsequent discussion, we incorporate the solution of SGA for comparison purposes using the same parameters as in the DGAs.

9.1. Migration Scheme

Figures 21 and 22 depict the solution quality of various DGA versions for TCASE2 and TCASE3 respectively. For all versions we use M = 30% except for ZDGA where M is always equal to 0. These two figures exhibit high DGA solution quality insensitivity to the migration scheme.

Execution time results, which are depicted in figures 23 and 24 for TCASE2 and TCASE3 respectively, show also that the DGA execution time is insensitive to the migration scheme.

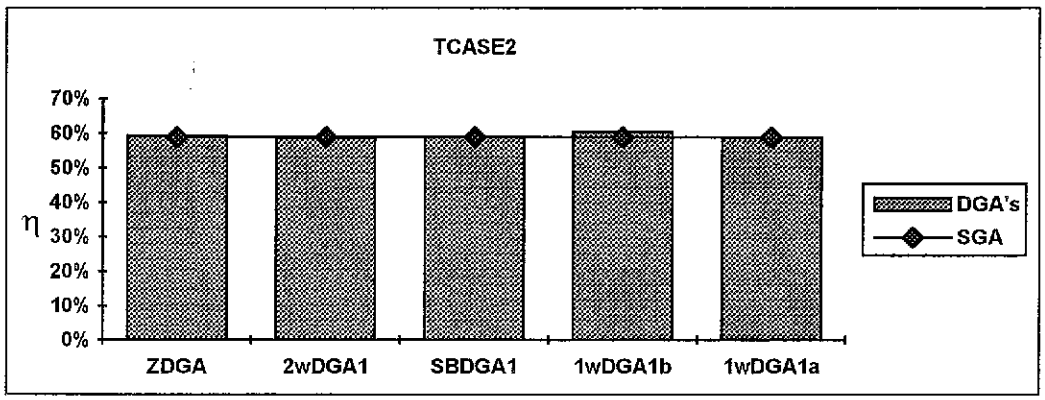


Figure 21. Comparing the solution quality of versions of DGA and SGA using TCASE2.

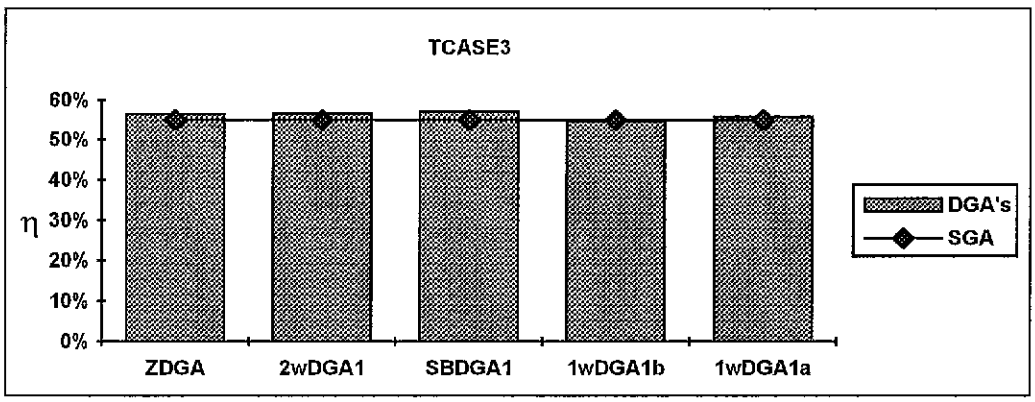


Figure 22. Comparing the solution quality of versions of DGA and SGA using TCASE3.

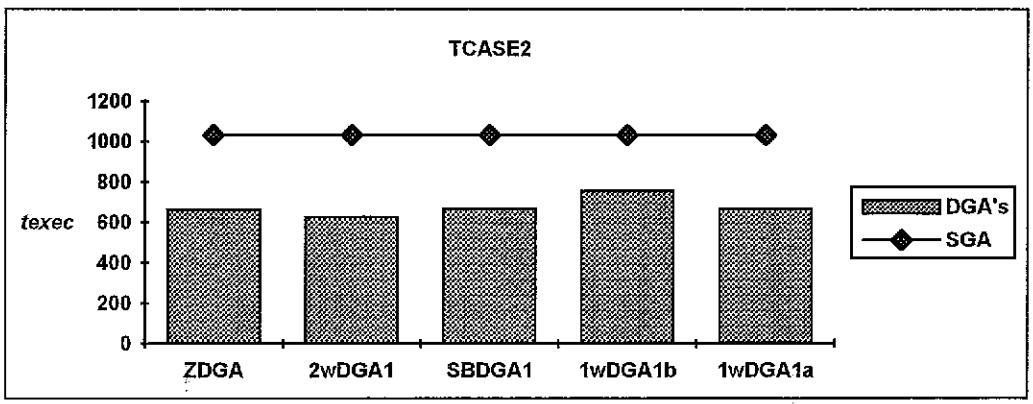


Figure 23. Comparing the execution time of versions of DGA and SGA using TCASE2.

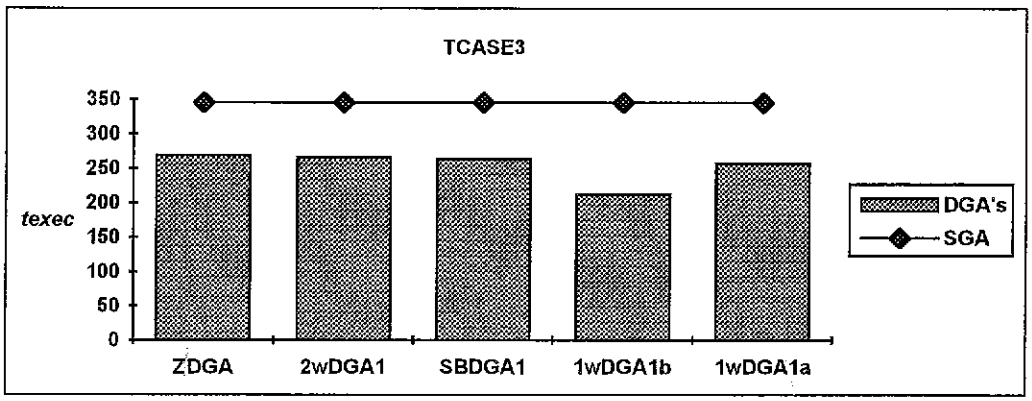


Figure 24. Comparing the execution time of versions of DGA and SGA using TCASE3.

9.2. Number of Migrants

Values of M are chosen in the range 0% to 100%, except for ZDGA, and 2wDGA. For 2wDGA, M > 50% is meaningless, and any DGA with M = 0% resolves to ZDGA.

9.2.1. 1wDGA1b and 1wDGA1a.

The solution qualities of these two versions are shown in figures 25 and 26 for TCASE2 and TCASE3 respectively. Their insensitivity to M can be directly inferred from these figures.

No general rule can be inferred from the execution time curves shown in figures 27 and 28. However, the execution time of these DGA's for any value of M is always less than that of SGA.

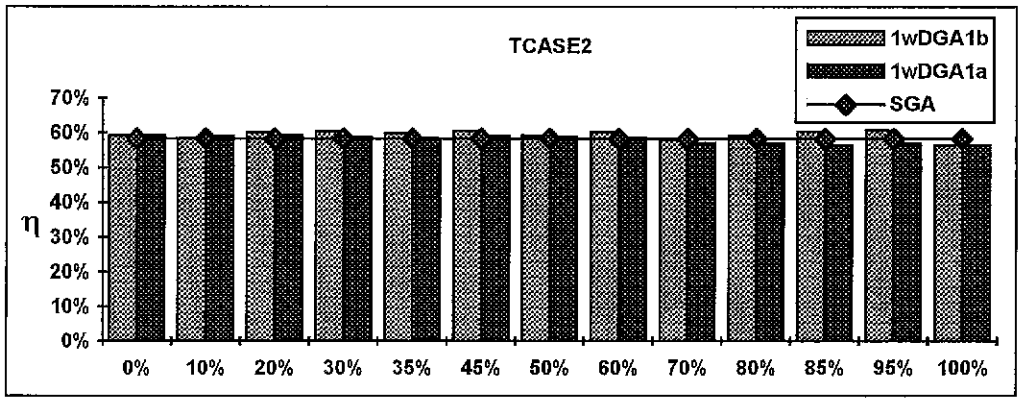


Figure 25. Solution quality of 1wDGA1b and 1wDGA1a for different values of M (x-axis) compared with SGA (test case TCASE2).

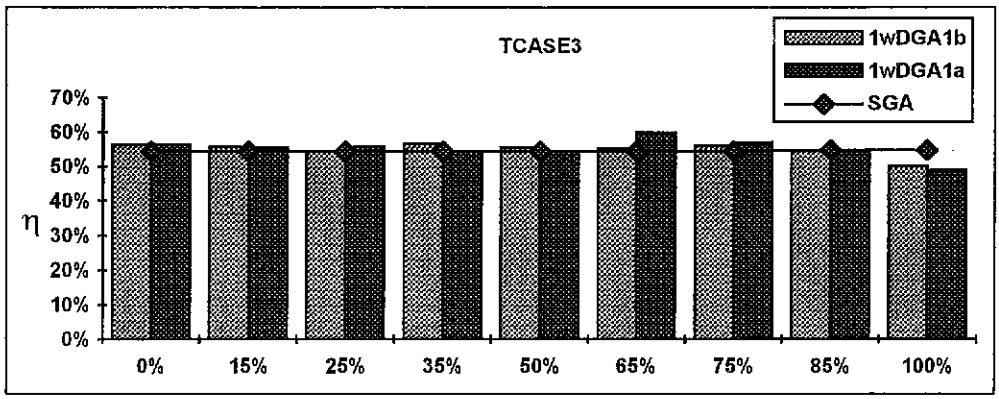


Figure 26. Solution quality of 1wDGA1b and 1wDGA1a for different values of M (x-axis) compared with SGA (test case TCASE3).

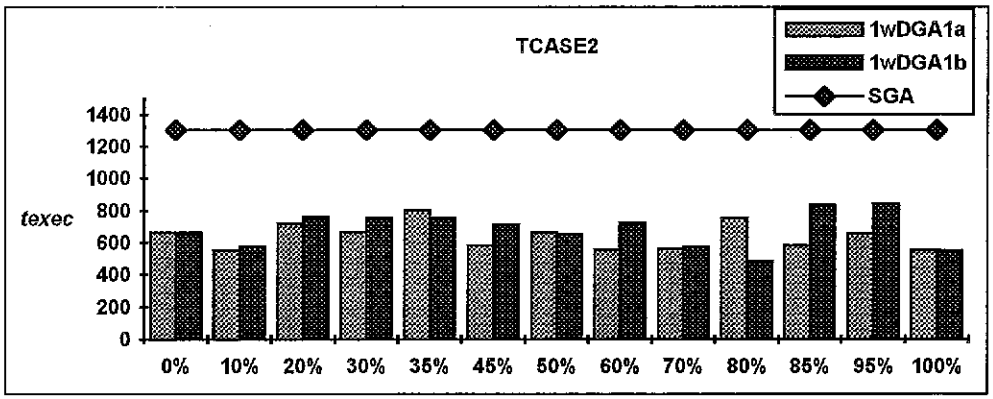


Figure 27. Execution time of 1wDGA1b and 1wDGA1a for different values of M (x-axis) compared with SGA (test case TCASE2).

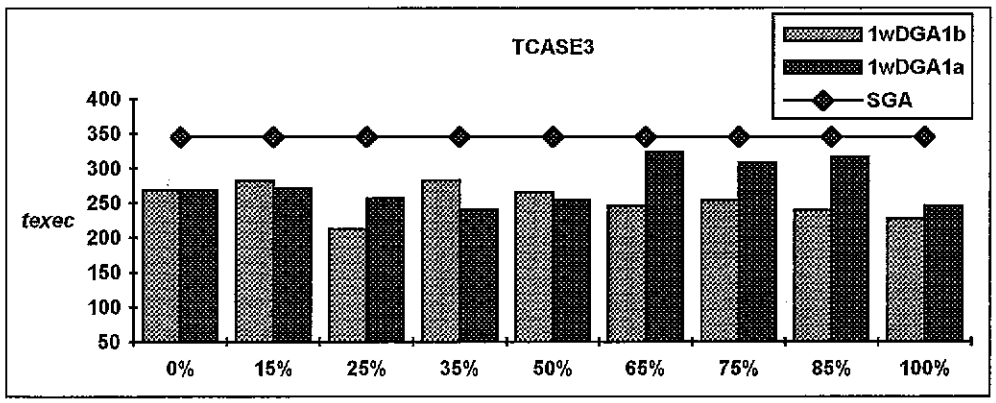


Figure 28. Execution time of 1wDGA1b and 1wDGA1a for different values of M (x-axis) compared with SGA (test case TCASE3).

9.2.2. 2wDGA1.

The solution quality of 2wDGA1 is insensitive to M. Results are shown in figures 29 and 30 for TCASE2 and TCASE3 respectively.

2wDGA1 is also not highly sensitive to M in terms of its execution time. Refer to figures 31 and 32.

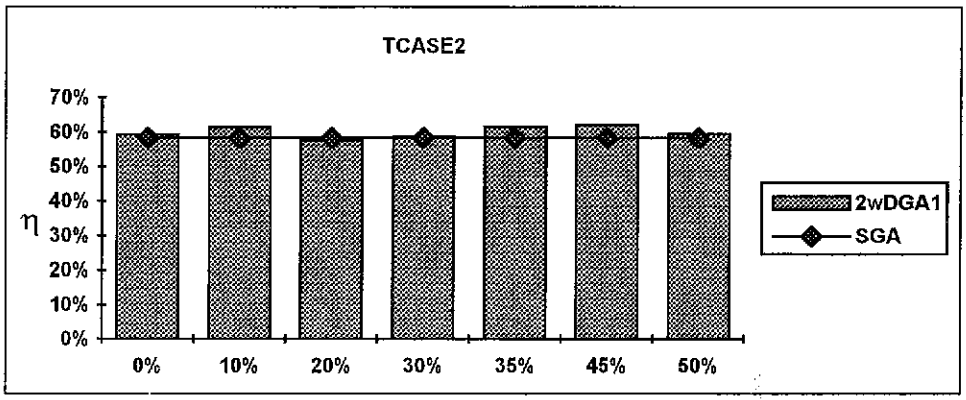


Figure 29. Solution quality of 2wDGA1 for different values of M (x-axis) compared with SGA (test case TCASE2).

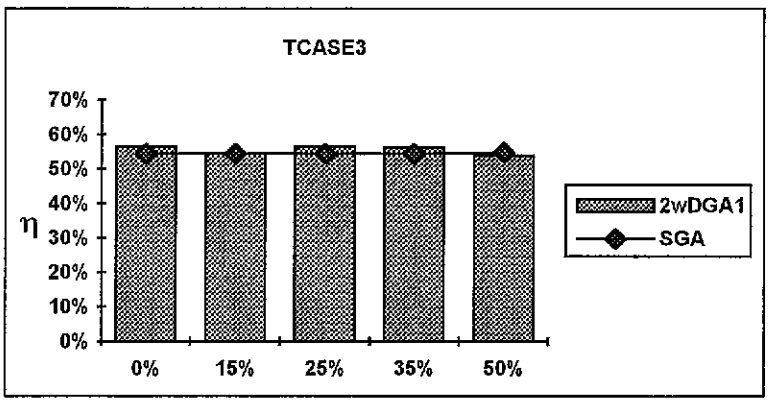


Figure 30. Solution quality of 2wDGA1 for different values of M (x-axis) compared with SGA (test case TCASE3).

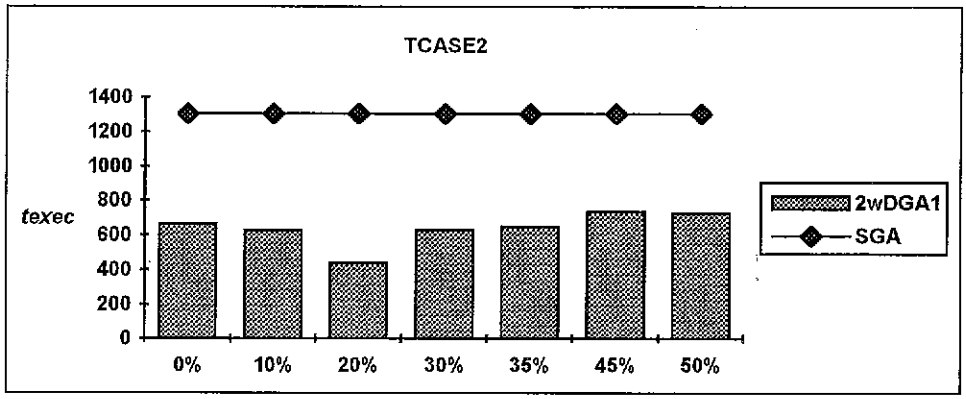


Figure 31. Execution time of 2wDGA1 for different values of M (x-axis) compared with SGA (test case TCASE2).

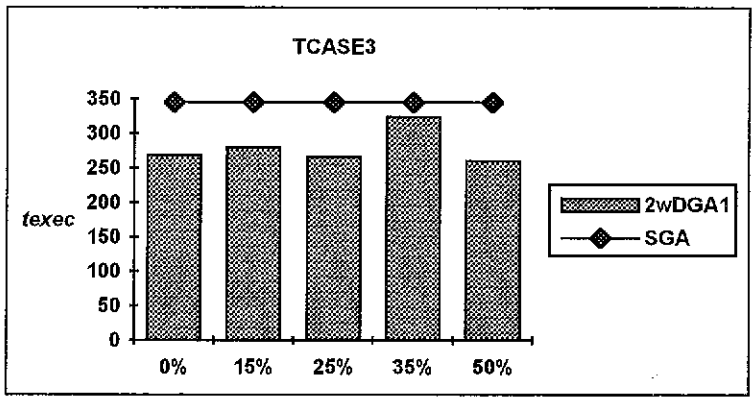


Figure 32. Execution time of 2wDGA1 for different values of M (x-axis) compared with SGA (test case TCASE3).

9.2.3. SBDGA1

SBDGA1 is insensitive to M in terms of solution quality as well as execution time. The experimental results are depicted in figures 33 to 36.

In short, DGA is insensitive to the number of migrants, M.

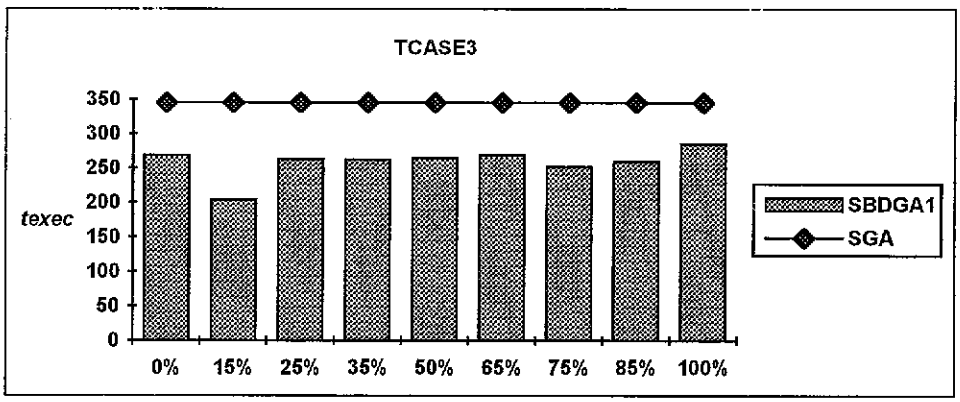


Figure 36. Execution time of SBDGA1 for different values of M (x-axis) compared with SGA (test case TCASE3).

9.3. Nature of Migrants

We introduced 1wDGA2a, 1wDGA2b, 2wDGA2, and SBDGA2 to be able to test the sensitivity of DGA's to the nature of the migrants. In these algorithms, we do not choose the migrants as the best M% individuals. Instead, we just choose random M% individuals with replacement, which is worse than doing it without replacement. Consequently, victims are also chosen in the same random manner. Having shown the insensitivity of DGA's to M, we also choose the number of migrants randomly in each migration phase. Extending this random strategy, we also select the migration direction randomly in 1wDGA2a, 1wDGA2b and 2wDGA2. In this manner, we will be studying the sensitivity of DGA's not only to the nature of the migrants, but also to other contributors such as the migration direction and random generator seeds.

Figures 37 and 38 show the solution quality results of these randomly based migration DGA's compared to ZDGA. The graphs imply that selecting any migrants in any manner along any direction does not lead to any changes in the DGA solution quality.

On the other hand, the execution time curves (figures 39 and 40) show that random-based migrations are most of the time slower than their ancestors.

9.4. Summary

In the previous three sections, we experimentally showed that the DGAs are insensitive to the choice of a migration policy. Nevertheless, we showed that DGAs do not require the existence of such a policy for the used test cases. We note that these results can not be generalized. The hybridized nature of these algorithms is one of the reasons behind this insensitivity. Moreover, we chose a moderate $|V_m|$ in each test case. Had $|V_m|$ been greater, the deme size becomes smaller and the deme individuals turn to be of less diversity the fact that necessitates the existence of a migration policy. Therefore, we will use 2wDGA1 as our DGA representative. This algorithm is expected to have minimal communication overhead had it been implemented on a multicomputer (except if it is compared with ZDGA).

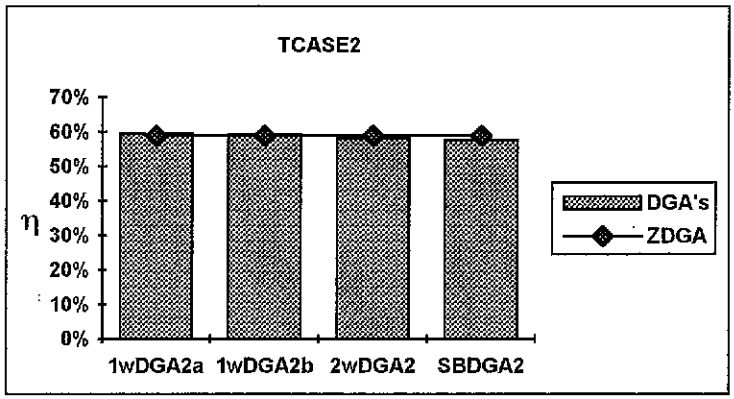


Figure 37. η for versions of DGA using TCASE2.

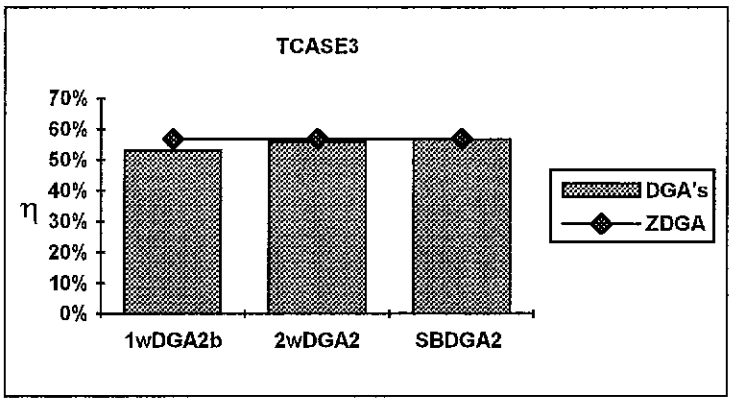


Figure 38. η for versions of DGA using TCASE3.

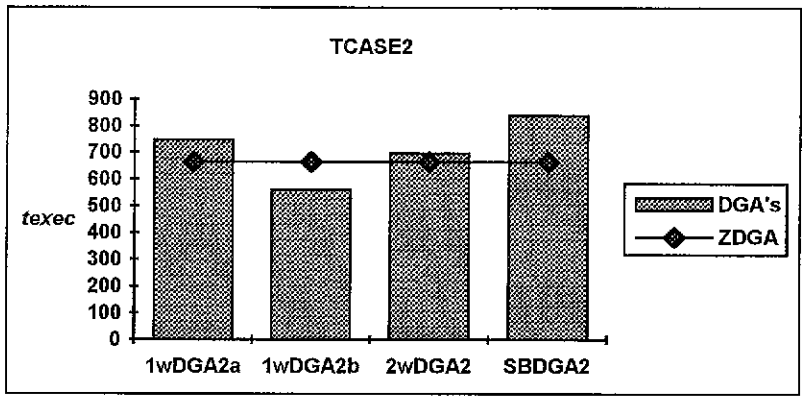


Figure 39. t_{exec} for versions of DGA using TCASE2.

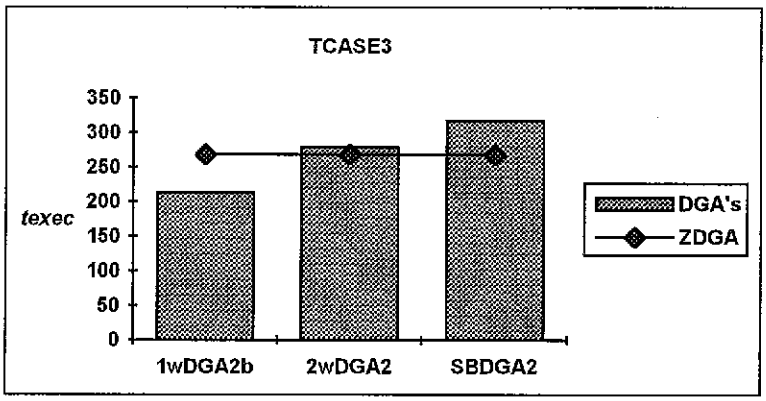


Figure 40. t_{exec} for versions of DGA using TCASE3.

10. Sensitivity to *Drift Phase Length*

The performance of 1wDGA1b, 2wDGA1, and SBDGA is depicted in figures 41 to 46 for different drift lengths measured in generations. Drift length values are chosen in the range of 1 individual to deme size individuals in test cases TCASE2 and TCASE3.

η curves for each of these versions exhibit high insensitivity to the drift length. while t_{exec} curves are affected irregularly the fact that does not lead us to any general conclusion. Therefore, we will use henceforth the drift length value to be (deme size) / 3.

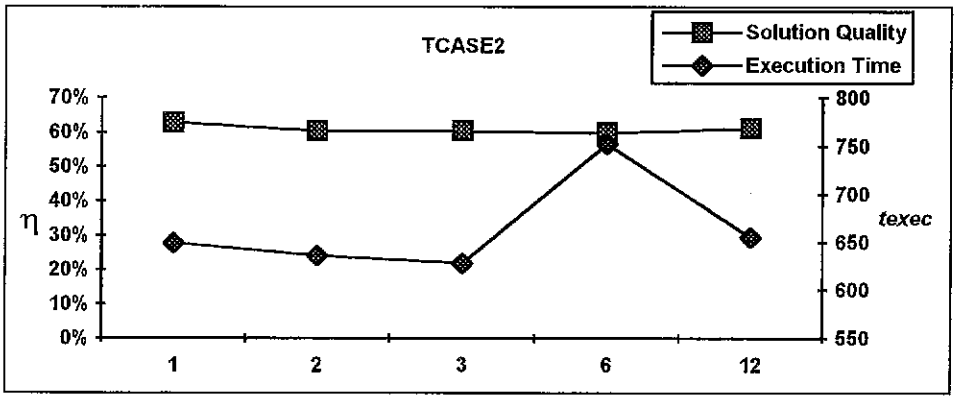


Figure 41. Performance of 1wDGA1b for different drift lengths (given in generations on the x-axis) using TCASE2.

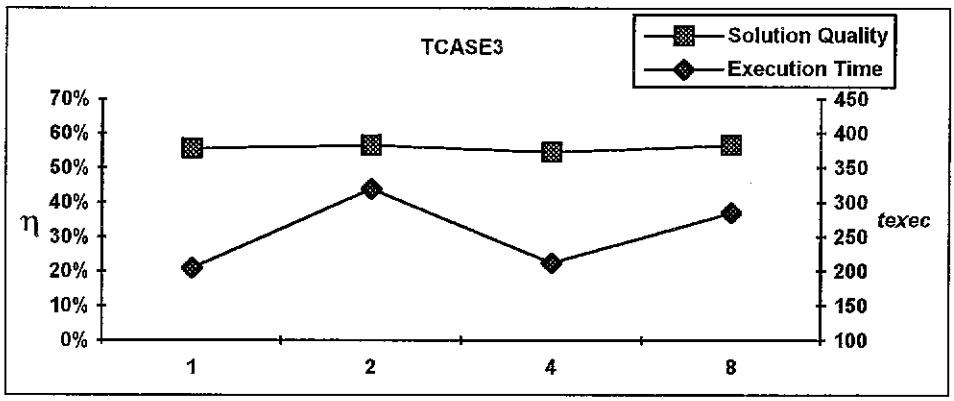


Figure 42. Performance of 1wDGA1b for different drift lengths (given in generations on the x-axis) using TCASE3.

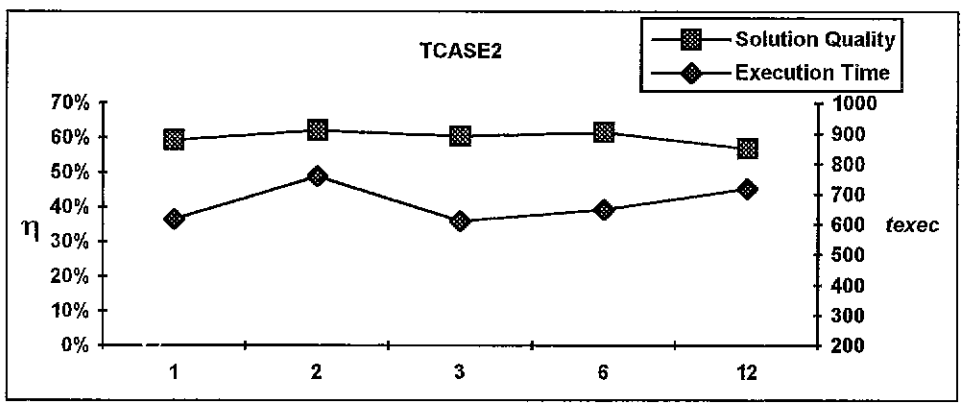


Figure 43. Performance of 2wDGA1 for different drift lengths (given in generations on the x-axis) using TCASE2.

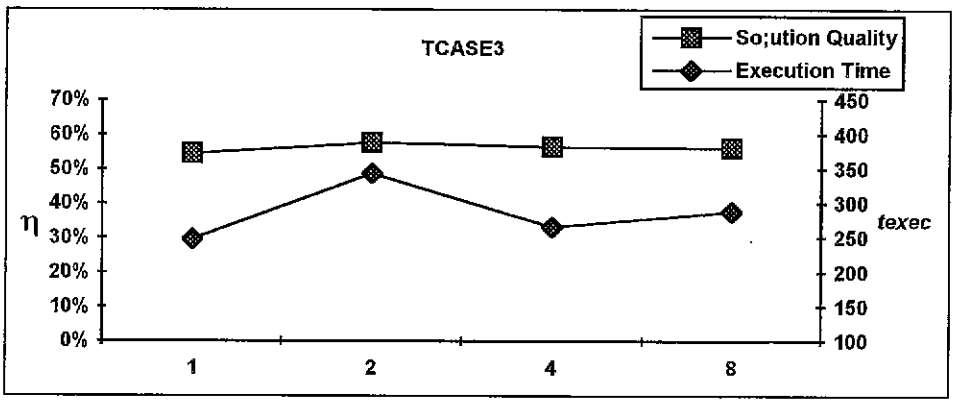


Figure 44. Performance of 2wDGA1 for different drift lengths (given in generations on the x-axis) using TCASE3.

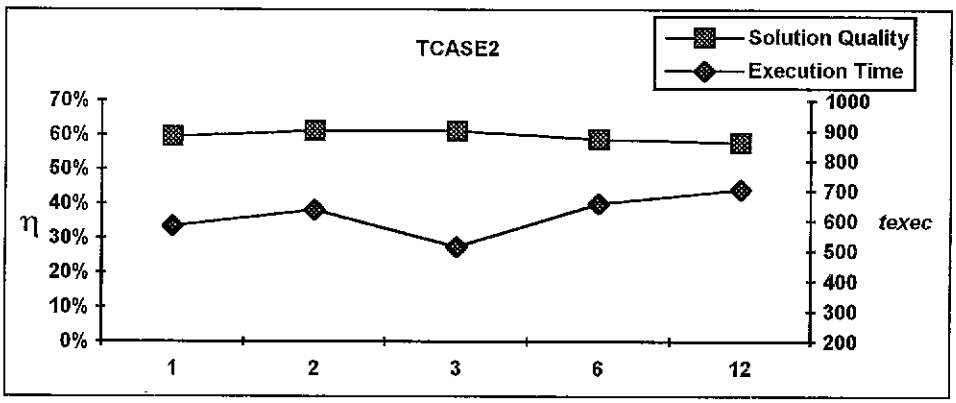


Figure 45. Performance of SBDGA1 for different drift lengths (given in generations on the x-axis) using TCASE2.

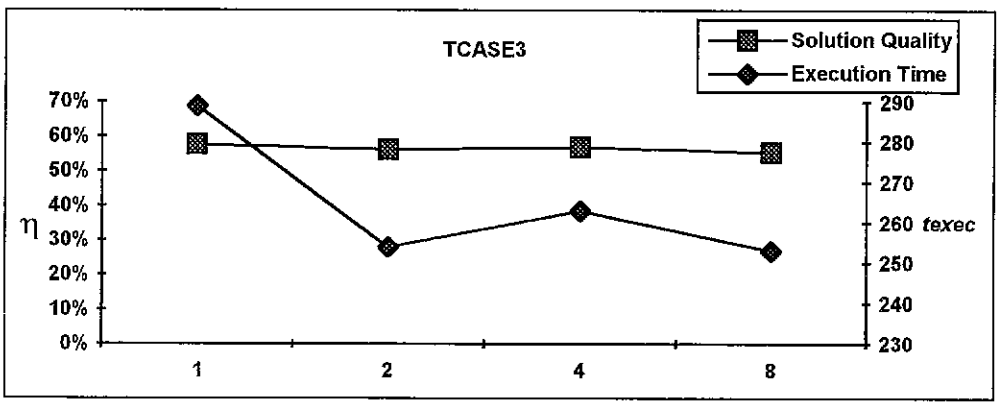


Figure 46. Performance of SBDGA1 for different drift lengths (given in generations on the x-axis) using TCASE3.

11. Summary and Conclusions

In this chapter, we studied the sensitivity of SGA and DGA versions to the parameters whose values involve user's choice. These parameters are summarized in table 1 accompanied with a short definition for each. The test cases employed in this study are presented in table 2. Sensitivity is studied in terms of η, the solution quality, and t_{exec}, the execution time. The results are summarized in table 3.

Parameter	GA	Range	η	t_{exec}	Adopted Value
μ_{user}	SGA	[0.2, 0.8]	I	NR	0.5
$E_b\_V_b$	SGA	[20%, 100%]	I	NR	50%
λ	SGA	[-85%, 328%]	I	-	7
τ	SGA	[-90%, 200%]	I	-	100
σ	SGA	± 50%	I	-	325
ρ	SGA	[-85%, 230%]	I	-	15
Population size	SGA	-	I	S	50,100,150
CT	SGA	≥ 15	I	S	15
Migration Policy (Migration Scheme)	DGA	-	I	I	Two way migration (2wDGA1)
Migration Policy (Number of Migrants)	DGA	-	I	NR	30%
Migration Policy (Nature of Migrants)	DGA	-	I	I	Best Individuals
Drift phase length	DGA	[1, demesize]	I	NR	demesize/3

Table 3. Summary of GA sensitivity results. I : Insensitive. S : Sensitive. NR : No general Rule. (For λ, ρ, σ, and τ the range shows under estimation errors (preceded by a -) and over estimation errors.

GA solution quality is rarely affected by user misestimation of a parameter. SGA is highly insensitive to the objective function related parameters (μ_{user}, $E_b\_V_b$, λ). Moreover, the user misestimation of the machine dependent parameters (τ, σ, ρ) does not affect SGA solution quality. However, parameters such as population size and convergence should be experimentally chosen to yield good solutions in reasonable time. DGA is insensitive to the choice of the migration policy (migration scheme, number of migrants, and nature of

migrants). although we showed also that DGA does not even require the existence of such a policy at all, we believe that this conclusion can not be generalized. Moreover, the drift length choice does not affect DGA solution quality.

The GA execution time results did not lead us, most of the time, to a general conclusion or rule. With most of the parameters, GA execution time is irregularly affected. While with others, such as the population size and convergence threshold, GA execution time increases as the values of these parameters increase.

The overall assessment is that GA's solution quality is not substantially sensitive to the user intervention. However, its execution time may be affected. This fact necessitates the choice of some parameters experimentally.

Chapter 6

Sensitivity of SAs to User Parameters

SSA and SPSA presented in chapter 4 aim at minimizing the execution time of *ALGORITHM* on a multicomputer, *MCOMP*. These SAs depend on several parameters that need to be set by the user. These parameters are restated in table 1 with short definitions.

The sensitivity of SA to user intervention is studied experimentally in this chapter emphasizing the effect of misestimating these parameters on SA performance. The same test cases used in chapter 5 are used in this chapter. The characteristics and shapes of these test cases can be referred to in table 2 and figure 1 both in chapter 5. The SA performance is studied in terms of two measures. The first is solution quality, η, and presented in equation 9 of chapter 2. The second is SA's execution time, t_{exec}, measured in seconds. As it was the case in chapter 5, TCASE2 is contracted once, and TCASE3 is contracted twice [Mansour 1992]. The experiments are conducted on an AViiON 5000 UNIX machine.

Parameter	Description
μ_{user}	Relative importance of communication to computation.
$E_b\_V_b$	Average ratio of boundary edges to boundary vertices.
λ	Computation operations per edge per iteration in *ALGORITHM*.
τ	*MCOMP* communication time per unit distance / t_{float}.
σ	*MCOMP* message startup time / t_{float}.
ρ	*MCOMP* time to communicate one word / t_{float}.
Convergence Threshold	Number of passes with no improvement in η.
f_{bdry}	Frequency at which boundary information is exchanged.
f_{Sv}	Frequency at which Sv(p) is updated.

Table 1. SA parameters whose values involve user's choice.

1. Sensitivity to μ_{user}

The value of μ_{user} is set by the user such that . We show experimentally that SSA is not highly sensitive to the user misestimtation of μ_{user}.

The sensitivity of SSA to μ_{user} is depicted in figures 1 to 3 for the three test cases. In TCASE1, η is totally insensitive to the choice of μ_{user}. η in TCASE2 shows a difference between the highest and lowest points which is less than 10%. However, TCASE3 shows sharp fluctuations in η curve, but the difference between the highest and lowest points is about 12%. Thus, SSA solution quality is not highly sensitive to the choice of μ_{user}.

t_{exec} curves for each of the three test cases are totally irregular leading to no general conclusions. A choice μ_{user}= 0.5 is acceptable for all test cases.

2. Sensitivity to $E_b\_V_b$

$E_b\_V_b$ is expressed in terms of θ_{avg}, and its values are chosen between 10% and 100% of θ_{avg}. η and t_{exec} for these values of $E_b\_V_b$ are shown in figures 4 and 5 for TCASE2 and TCASE3 respectively.

The solution quality is always acceptable for $E_b\_V_b$ between 20% and 90% of θ_{avg} in TCASE2. However, it exhibits large fluctuations in TCASE3 (20% for $E_b\_V_b$ = 10% and 50% of θ_{avg}). Excluding extreme points, $E_b\_V_b$ in the remaining range (20% to 80% of θ_{avg}) show a maximum drop (increase) of 10% in η.

t_{exec} curves lead to no conclusions. In one word, SSA is not highly sensitive to the choice of $E_b\_V_b$ which we choose to be 50% of θ_{avg}.

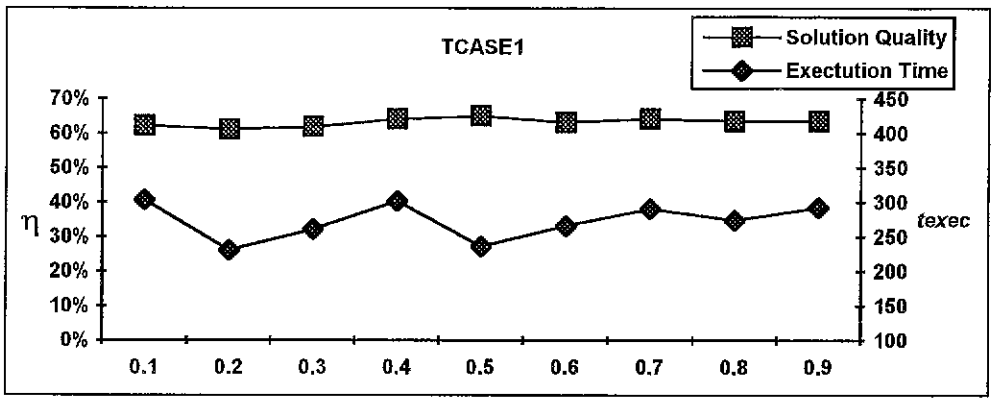


Figure 1. Performance of SSA for different values of μ_{user} (x-axis) using TCASE1.

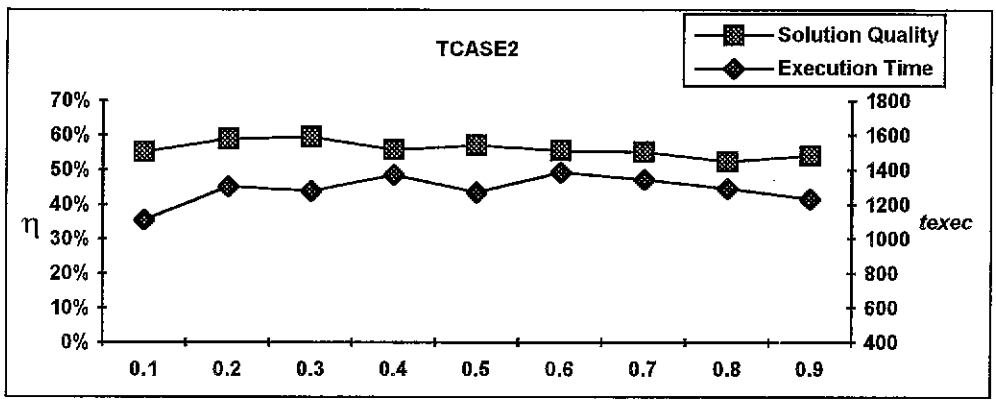


Figure 2. Performance of SSA for different values of μ_{user} (x-axis) using TCASE2.

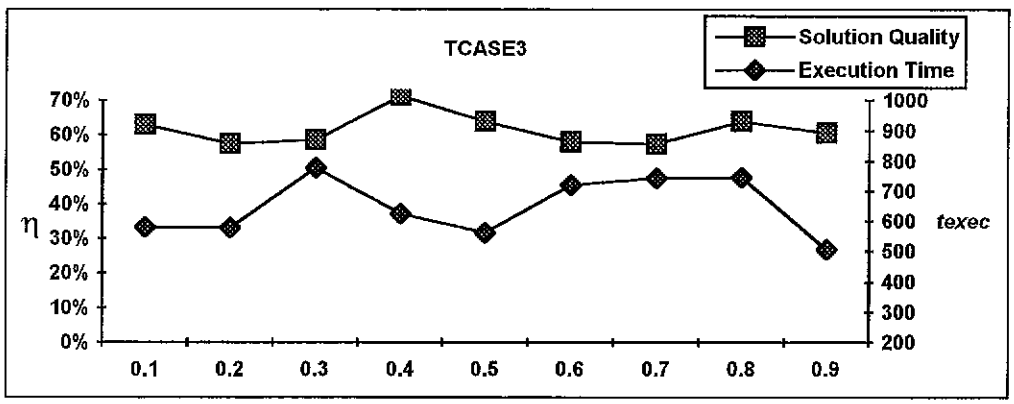


Figure 3. Performance of SSA for different values of μ_{user} (x-axis) using TCASE3.

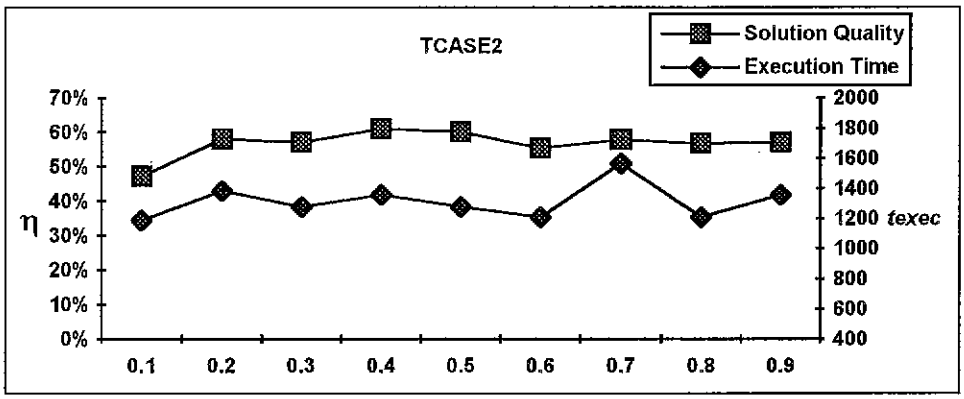


Figure 4. Performance of SSA for different values of $E_b\_V_b$ (expressed in terms of θ_{avg} on the x-axis) using TCASE2.

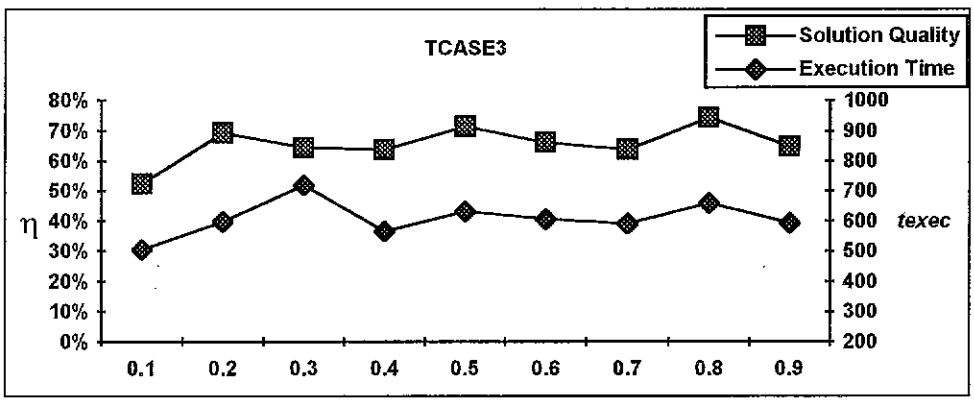


Figure 5. Performance of SSA for different values of $E_b\_V_b$ (expressed in terms of θ_{avg} on the x-axis) using TCASE3.

3. Sensitivity to λ

We assume a correct λ value to be 7. SSA is run using wrong values. But, its solutions are still evaluated using the correct value, 7. These wrong values are overestimated in at most 328%, and underestimated in at most 85% errors.

The experimental results for TCASE2 and TCASE3 are shown in figures 6 and 7 respectively. In both figures, the misestimation has no effect on λ. For example, misestimations in TCASE2 lead to a drop of at most 2%, namely when the underestimation error is 85%. While in TCASE3, an overestimation of 328% drive η to decrease at most 5%. Thus, SSA is highly insensitive to the user misestimation of λ.

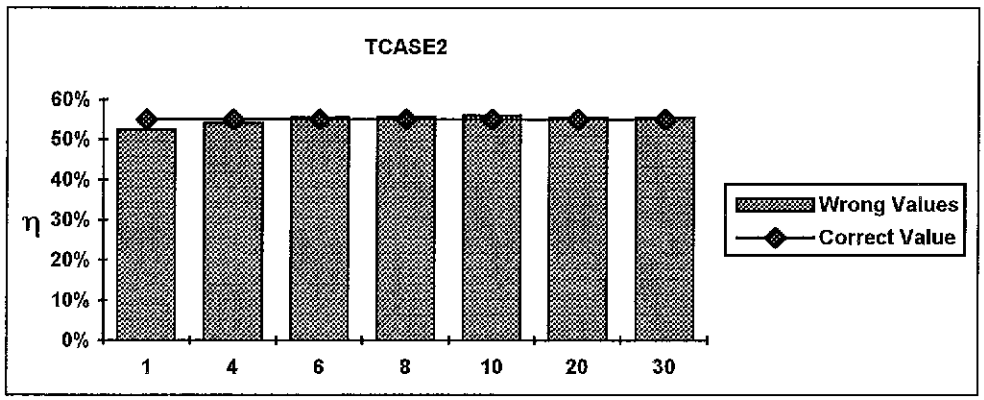


Figure 6. Quality of the solutions found by SSA using wrong values for λ (x-axis) compared to the solution found using correct λ (test case TCASE2).

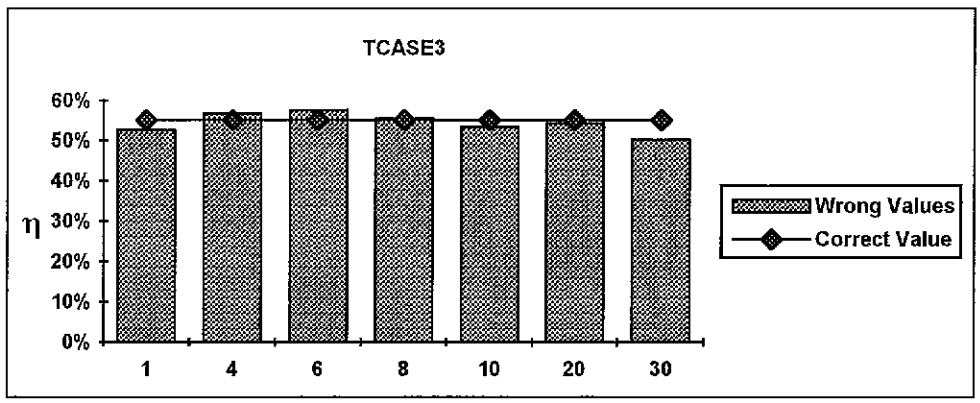


Figure 7. Quality of the solutions found by SSA using wrong values for λ (x-axis) compared to the solution found using correct λ (test case TCASE3).

4. Sensitivity to τ

The effects of misestimating τ are shown in figures 8 and 9 for TCASE2 and TCASE3 respectively. The correct value for τ is assumed to be 100. SSA solutions are found using wrong estimates for τ; however, they are evaluated using the assumed correct value. In both test cases, we use at most 200% and 90% for overestimation and underestimation errors respectively.

In both cases, SSA is insensitive to the mischoice of τ. TCASE2 shows that τ values less than 100, the assumed correct value, lead η to drop at most 2%. Whereas values greater than the correct value, give exactly the same solution generated by the correct value. TCASE3 shows that the choice of τ has no effect at all on η. The only exception is when τ=10 (underestimation error = 90%) where the solution quality is found to be better by less then 2%. In one word, SSA is completely insensitive to the choice of τ.

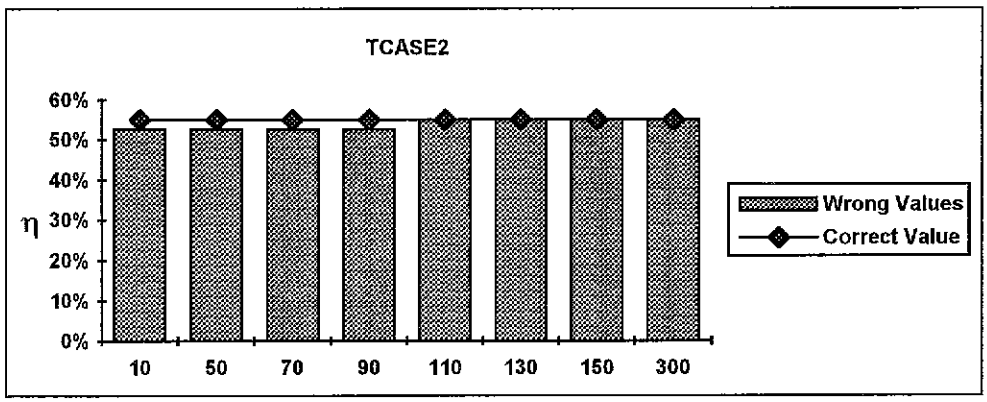


Figure 8. Quality of the solutions found by SSA using wrong values for τ (x-axis) compared to the solution found using correct τ (test case TCASE2).

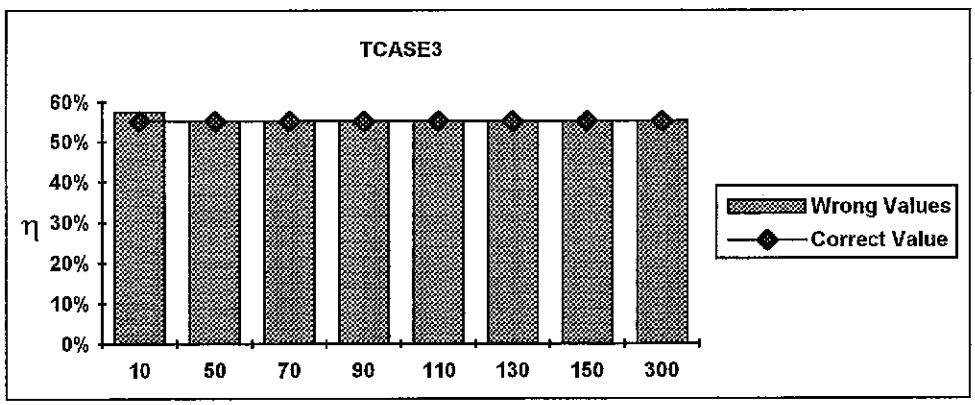


Figure 9. Quality of the solutions found by SSA using wrong values for τ (x-axis) compared to the solution found using correct τ (test case TCASE3).

5. Sensitivity to σ

The correct value of σ is assumed to be 325. SSA solutions are found using wrong estimates for σ, but they are evaluated using the assumed correct value, 325.

Figures 10 and 11 show the sensitivity of SSA to σ for test cases TCASE2 and TCASE3 respectively. SSA in TCASE2 is totally insensitive to the misestimation of σ. For all under and overestimated values, η is always found to be exactly equal to the solution found using the assumed correct value. TCASE3 shows that SSA solution quality is totally insensitive to the choice of σ in the range [150,1000]. Moreover, an underestimation error of 85% only yields η to decrease by about 5%. Therefore, SSA is highly insensitive to the choice of σ.

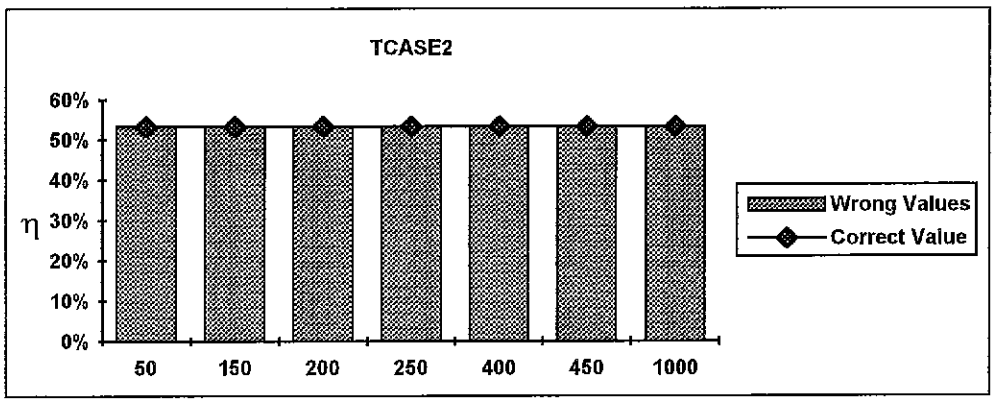


Figure 10. Quality of the solutions found by SSA using wrong values for σ (x-axis) compared to the solution found using correct σ (test case TCASE2).

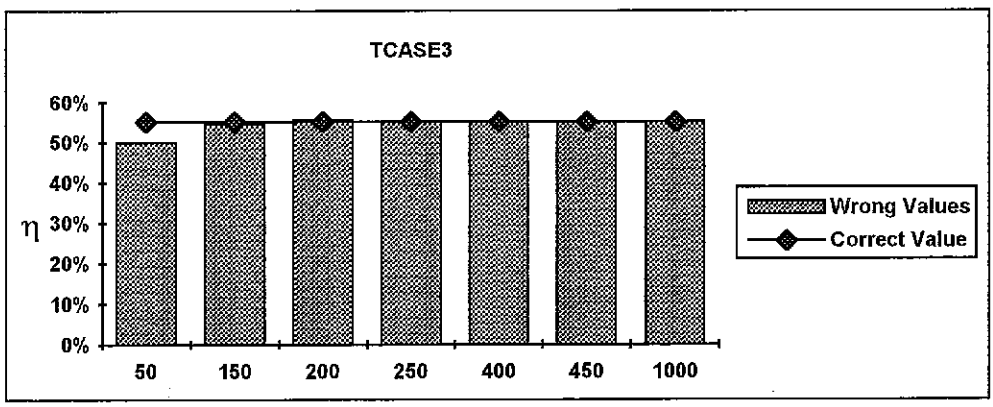


Figure 11. Quality of the solutions found by SSA using wrong values for τ (x-axis) compared to the solution found using correct τ (test case TCASE3).

6. Sensitivity to ρ

SSA uses wrong estimates for ρ n its execution. However, its solutions are evaluated using the correct value for ρ, assumed to be 15. Errors are chosen in a range less than 85% for underestimations, and less than 230% for overestimations.

Experimental results are shown in figures 12 and 13 for test cases TCASE2 and TCASE3 respectively. TCASE2 favors underestimations upon which η increases by less than 5%. Sometimes overestimations lead η to drop less than 2%. TCASE3 also shows the insensitivity of SSA to the choice of ρ. Decreases in η are rare and minimal. For instance, at ρ =19, η shows the maximum drop which is about 5%. Therefore, SSA is not affected by the choice of ρ.

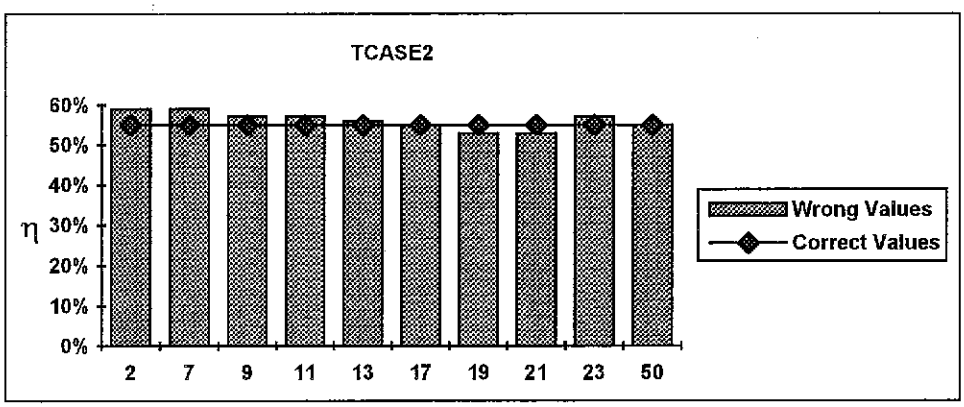


Figure 12. Quality of the solutions found by SSA using wrong values for ρ (x-axis) compared to the solution found using correct ρ (test case TCASE2).

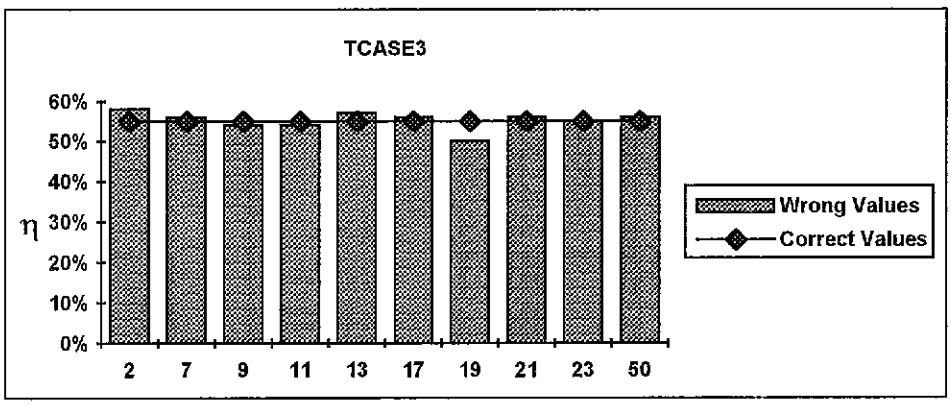


Figure 13. Quality of the solutions found by SSA using wrong values for ρ (x-axis) compared to the solution found using correct ρ (test case TCASE3).

7. Sensitivity to *Convergence Threshold*

SSA converges when there is no improvement in solution quality for a threshold of annealing passes. This convergence threshold is referred to in this section as *CT*.

As *CT* increases, SSA becomes slower while offering no improvement in solution quality. This can be directly inferred from figures 14 to 16. TCASE1 shows that a small *CT* (=5) drives us to sacrifice a 15% drop or more in η. For $CT \geq 10$, η curves are straight lines, while t_{exec} curves are positively sloped. Thus, SSA solution quality is insensitive to the choice of *CT* as far as $CT \geq 20$, and *texec* increases as *CT* increases. We will use $CT = 20$ henceforth, a reasonable choice which yields high solution qualities in a relatively small time.

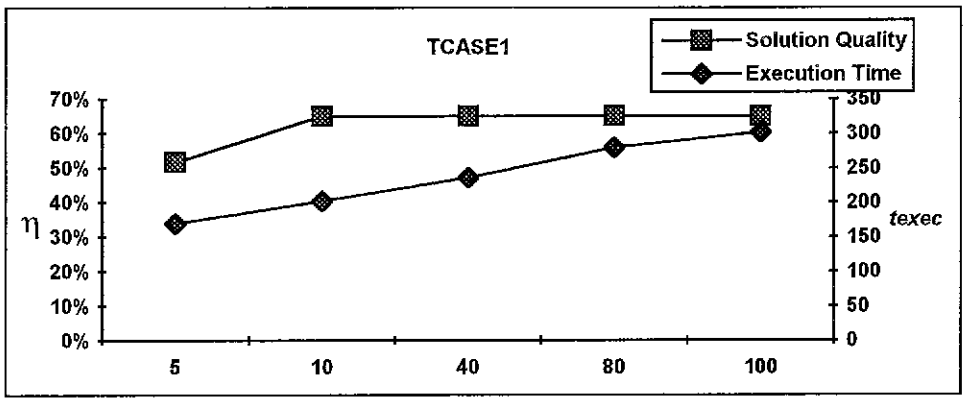


Figure 14. Performance of SSA for different values of *CT* (given in passes on the x-axis) using TCASE1.

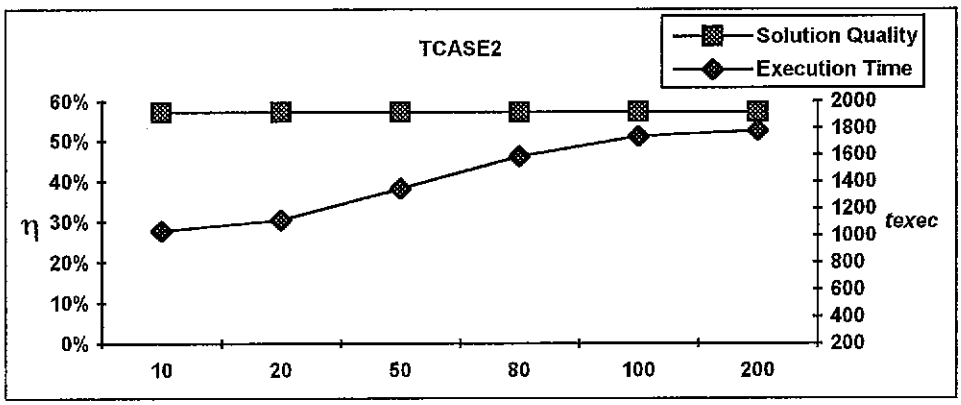


Figure 15. Performance of SSA for different values of *CT* (given in passes on the x-axis) using TCASE2.

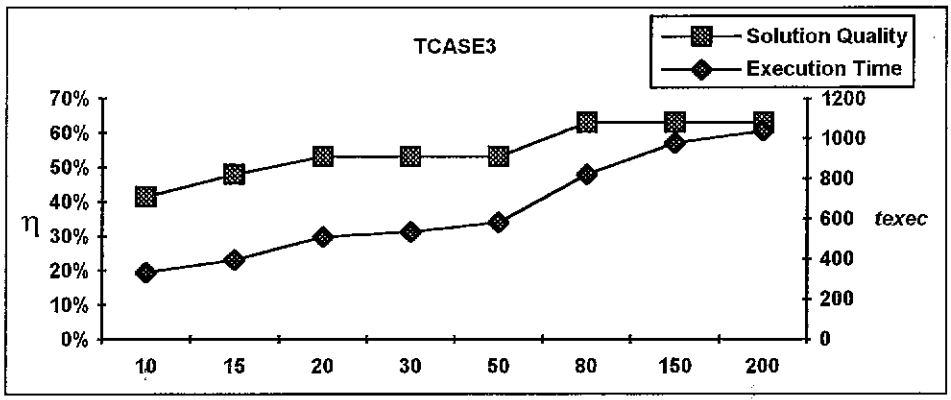


Figure 16. Performance of SSA for different values of *CT* (given in passes on the x-axis) using TCASE3.

8. Sensitivity to *fbdry*

We noted in chapter 4 that *fbdry* should be determined experimentally. In this section, we let $f_{mvs} = f_{sv} = 1/2\ N_{acc}$, this denotes that the number of moves and summation for $S_V(p)$ are communicated once every two accepted perturbations. Figure 17 shows a comparison of the solution qualities of SSA and SPSA for different values of $1/f_{bdry}$ using TCASE2. This shows that SPSA solution quality is most likely to near to that of SSA at high f_{bdry} (low $1/f_{bdry}$). The solutions at low f_{bdry} are still acceptable for $1/f_{bdry} < 500$. The reason for this can be inferred from figure 18 which depicts the total computation time of SPSA denoted as tt_{comp}. tt_{comp} increases as f_{bdry} decreases the fact that gives SPSA the chance to converge to acceptable solutions even though f_{bdry} is too low. for $1/f_{bdry} < 90$, tt_{comp} is acceptable and almost stable. At very high frequencies, $1/f_{bdry} < 10$, SPSA demands too many communication. Figure 19 shows the total number of times SPSA demands communication among all processor nodes. In this figure, tcd_{bdry} and tcd_{sv} denoted the total communication demand for boundary information and $S_V(p)$ global summation respectively. Clearly, at low f_{bdry} (> 70) tcd_{bdry} becomes negligible. However, tcd_{sv} increases steadily. This increase is influenced by the increasing tt_{comp} curve in figure 18. Recall that f_{sv} in this section is set constant. Thus the more SPSA executes, the more $S_V(p)$ is globally summed. Therefore, too low f_{bdry} makes SPSA expensive because of its high computation time. f_{bdry} should be chosen in the middle where computation and communication times are low while solutions are of good quality. This suggests : $1/70 < f_{bdry} < 1/10$.

9. Sensitivity to f_{SV}

We let, in this section, $f_{mvs} = 2$ and $f_{bdry} = f_{sv}$ so that f_{bdry} makes no influence on SPSA. The results for TCASE2 are shown in figures 20, 21, and 22. Figure 20 shows that SPSA solutions are near or better than SSA solutions at high f_{SV}, particularly $1/f_{SV} < 18$. Otherwise, the solution quality shows a decreasing trend for low f_{SV}. tt_{comp} is irregular; however it shows an increasing trend as f_{SV} decreases. Refer to figure 21 for an illustration. The communication demand is shown in figure 22. Note that $tcd_{bdry} = tcd_{sv}$ so we will refer only to tcd_{sv}. tcd_{sv} is very high at high f_{SV} ($1/f_{SV} < 5$) the fact that drives SPSA to be expensive. Therefore, a choice of $5 < 1/f_{SV} < 18$ is recommended.

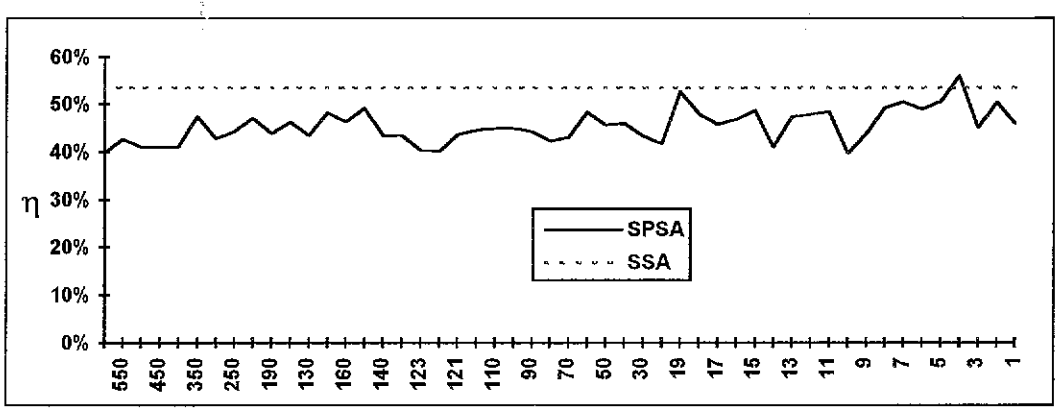


Figure 17. A comparison of the solution qualities of SSA and SPSA for different values of $1/f_{bdry}$ (x-axis) using TCASE2.

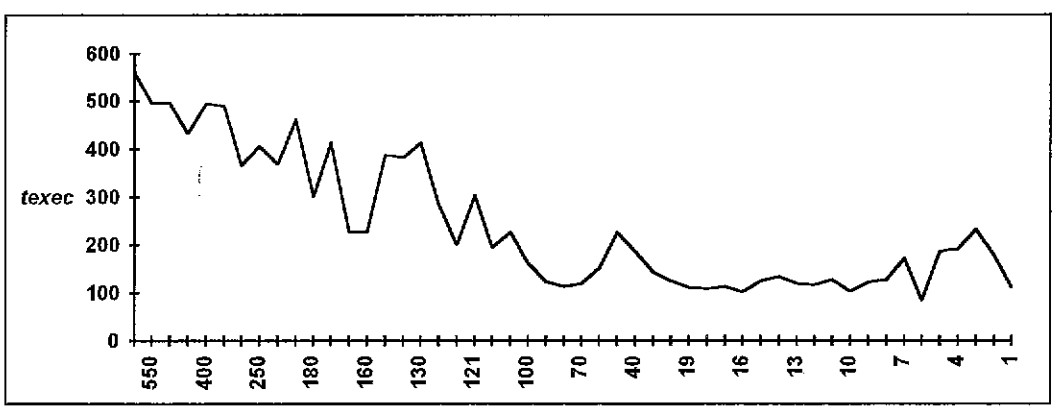


Figure 18. SPSA Total computation time, tt_{comp},, versus $1/f_{bdry}$ (x-axis) using TCASE2.

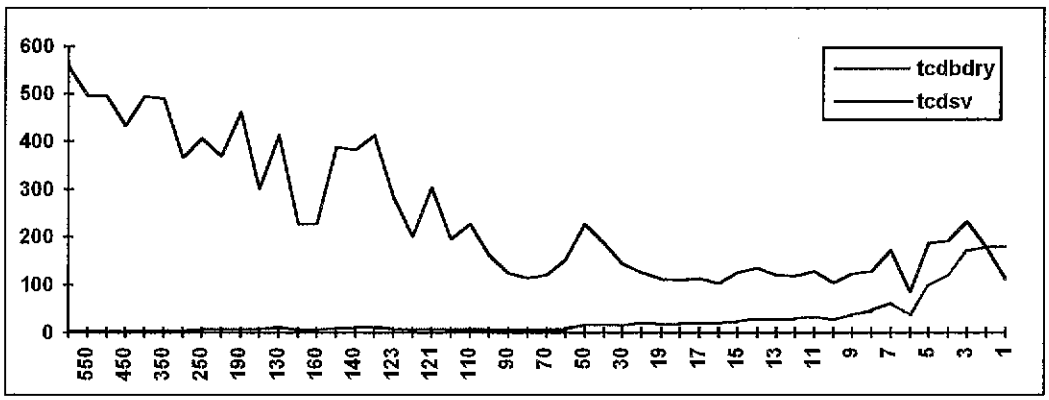


Figure 19. SPSA total communication demands for communicating boundary information and globally summing Sv(p) versus $1/f_{bdry}$ (x-axis) using TCASE2.

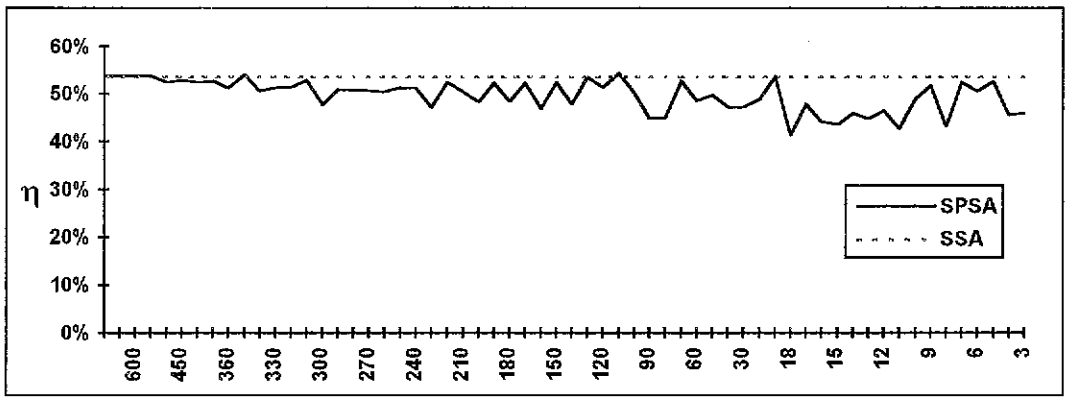


Figure 20. A comparison of the solution qualities of SSA and SPSA for different values of $1/f_{sv}$ (x-axis) using TCASE2.

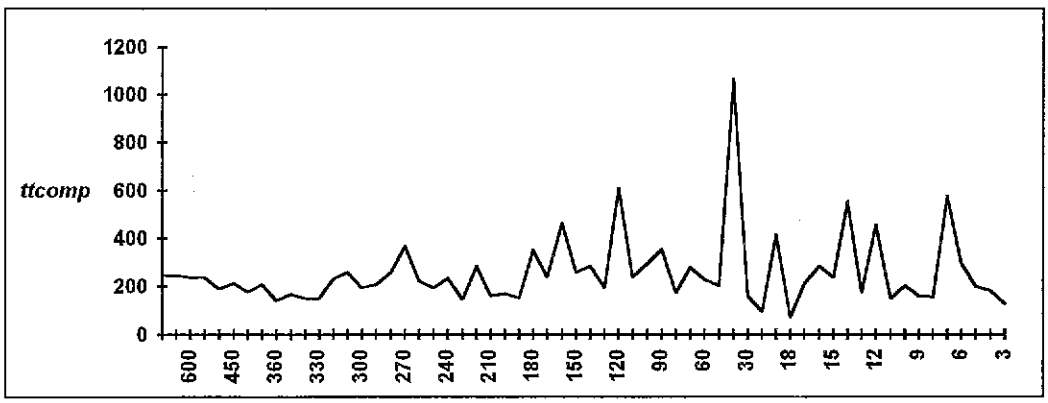


Figure 21. SPSA Total computation time, tt_{comp},, versus $1/f_{sv}$ (x-axis) using TCASE2.

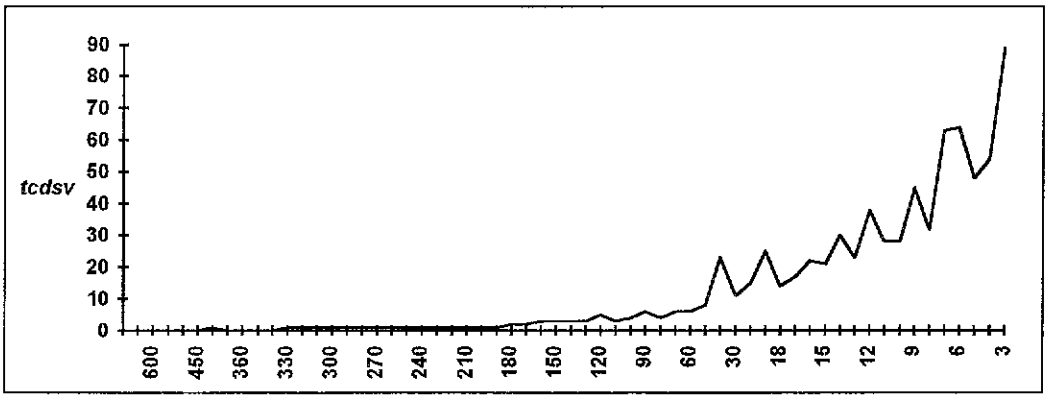


Figure 22. SPSA total communication demand for global summation Sv(p) versus $1/f_{sv}$ (x-axis) using TCASE2.

10. Summary and Conclusions

We studied in this chapter the sensitivity of SAs to the parameters that require user intervention. A summary of these parameters is given in table 1 restating a short definition of each. We employed in the experimental work 3 test cases of different characteristics. these test cases are presented in table 2 of chapter 5. A summary of the experimental results is given in table 2.

Parameter	SA	Range	η	t_{exec}	Adopted Value
μ_{user}	SSA	[0.1, 0.9]	I	NR	0.5
$E_b\_V_b$	SSA	[20%, 100%]	I	NR	50%
λ	SSA	[-85%, 328%]	I	-	7
τ	SSA	[-90%, 200%]	I	-	100
σ	SSA	[-85%,,230%]	I	-	325
ρ	SSA	[-85%, 230%]	I	-	15
CT	SSA	≥ 20	I	S	20
f_{bdry}	SPSA	[1/70,1/10]	S	S	8
f_{sv}	SPSA	[1/18,1/5]	S	S	8

Table 2. Summary of SA sensitivity results. I : Insensitive. S : Sensitive. NR : No general Rule. (For λ, ρ, σ, and τ the range shows under estimation errors (preceded by a -) and over estimation errors.

SSA is highly insensitive to the objective function parameters, i.e., μ_{user}, $E_b\_V_b$, and λ. It is also highly insensitive in terms of its solution quality to the machine dependent parameters , τ, σ, and ρ. The convergence threshold does affect SA solution quality as well as its execution time. It must be determined experimentally. The correction frequencies of SPSA are preferred to be high enough to guarantee good solution quality and low enough to prevent the algorithm from being computationally expensive due to high communication demand.

As it was the case with GAs, execution time results do not lead, most of the time to general conclusions. This is because the execution time is irregularly affected except for the convergence threshold and the correction frequencies where it is affected regularly.

Therefore, SA is insensitive to user intervention to set certain parameters. The solution quality is not substantially affected, but this is not the case with execution time.

Chapter 7

Fault Tolerant Mapping

A mapping algorithm is said to be fault tolerant if it is capable of mapping *DATA* into an incomplete multicomputer, *MCOMP*. In this case, *MCOMP* has at least one faulty processor. Faulty processors are not able to perform computations; however, routing can still be performed through them. This assumption is suitable for modern multicomputers which implement wormhole routing. Recall that wormhole routing is accomplished via special hardware attached to processors and not by the processors themselves [Hwang 1993].

This chapter demonstrates the fault tolerance capability of the physical optimization algorithms : GAs and SAs. 2wDGA1 will be used, and will be referred to simply as DGA. We also use the SAs: SSA and SPSA.

1. Motivation

Most of the proposed heuristics in the data mapping literature are bisection-based methods. Clearly, these algorithms are not fault tolerant [Berger and Bokhri 1987; Chrischoides et al 1991; Ercal 1988; Fox 1988; Karmer and Muhlenbein 1989]. For example, RCB assumes that the underlying topology has N processors where N is always a power of 2 [Chrisochoides et al. 1994].. Had one, say, of these processors fail, the resulting topology is incomplete and N is not a power of two any more. Consequently, These heuristics are not able to find a mapping to an incomplete architecture.

On the other hand, physical optimization algorithms, in particular GAs and SAs, are fault tolerant, and we will experimentally support this belief in the subsequent sections.

2. Implementation Issues

Given a set of faulty processors in *MCOMP*, denoted as V_f, we make two modifications to DGA in order to become fault tolerant. DGA must not generate in its initial population an allele value which is in V_f, i.e. CHROMOSOME[v] $\notin V_f \quad \forall v \in V_c$. Also, the mutation genetic operator must not mutate (change) an allele value such that the new value is in V_f.

Similarly, SSA and SPSA are made capable of fault tolerance after two modifications. The initial mapping configuration must not contain a vertex mapped to a processor in V_f (in other words, MAP[v] $\notin V_f \quad \forall v \in V_c$). Moreover, perturbations can not remap a vertex to a processor in V_f.

These modifications are minor, and with them, DGA, SSA, and SPSA become fault tolerant as we shall see below.

3. Experimental Results

In the following experimental work, we use the same test cases summarized in table 2 of chapter 5. The set of faulty processors, V_f, is chosen every time arbitrarily. $|V_f| = 1, 2,$ and 3 for TCASE1 ($|V_m| = 4$). For TCASE2 ($|V_m| = 8$), $|V_f| = 1, 3, 5,$ and 7. Finally, $|V_f| =$ 1, 5, 7, and 10 for TCASE3 ($|V_m| = 16$). Note that when $|V_f| = 0$, the topology is complete; complete topology cases are incorporated for comparison purposes.

3.1. GA

The results of DGA are depicted in figures 1 to 3. DGA is completely fault tolerant even when *MCOMP* has only one functioning processor, i.e. $|V_f| = 3$ and 7 in TCASE1 and TCASE2 respectively. DGA was able to map the whole computation graph, without being partitioned, into one processor in *MCOMP* with negligible time. Under these conditions, 'the' optimal solution has been found with 100% solution quality. More detailed results are given in Table 1 which shows the number of local vertices and edges, communication cost, and cross edges for each functioning processor. The results in Table 1 show how DGA still balances the combined workloads of computation (represented by the number of edges) and communication (represented by communication cost).

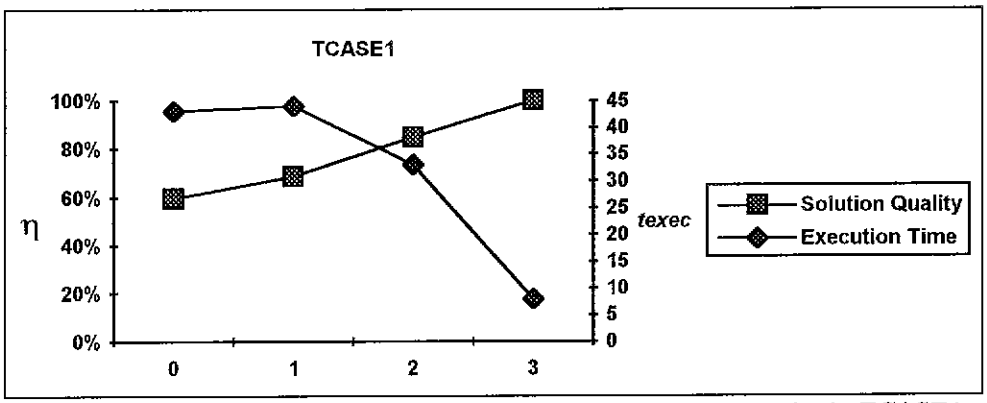


Figure 1. Performance of DGA when mapping to incomplete topologies in TCASE1 (x-axis shows $|V_f|$).

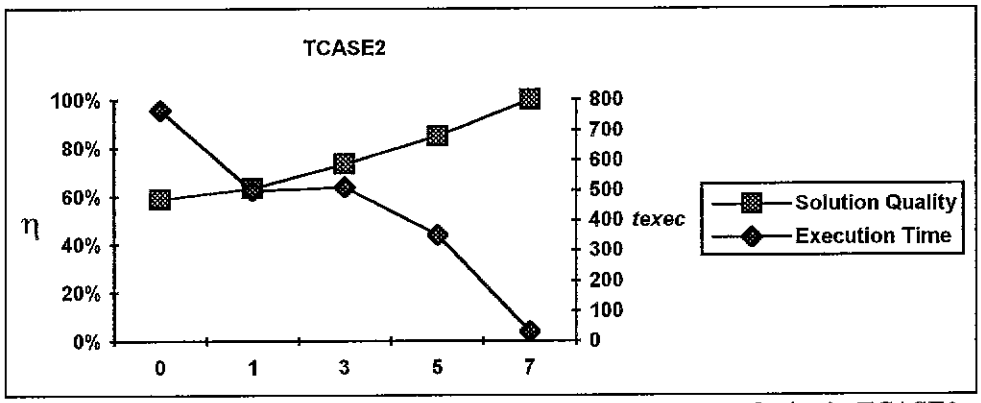


Figure 2. Performance of DGA when mapping to incomplete topologies in TCASE2 (x-axis shows $|V_f|$).

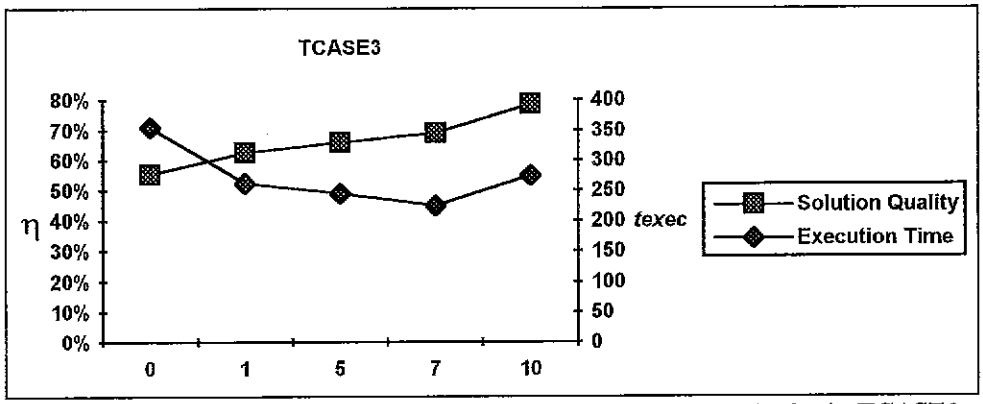


Figure 3. Performance of DGA when mapping to incomplete topologies in TCASE3 (x-axis shows $|V_f|$).

Test Case : TCASE1.

Faulty procs.	η	t_{exec}	Processor	Vertices	Edges	Comm. Cost	Cross Edges
{0}	69%	43	1	100	348	1190	18
			2	98	354	1130	14
			3	103	364	1030	16
{2}	74%	45	0	107	374	695	18
			1	79	297	1285	31
			3	115	395	605	13
{1,2,3}	100%	7	0	300	1066	0	0

Test Case : TCASE2.

Faulty procs.	η	t_{exec}	Processor	Vertices	Edges	Comm. Cost	Cross Edges
{1,5,7}	72%	493	0	108	1246	2720	246
			2	125	1413	1910	183
			3	92	1082	3255	248
			4	97	1145	3380	273
			6	123	1386	2495	196
{0,3,4,6,7}	86%	264	1	191	2168	1865	194
			2	191	2157	1930	157
			5	163	1947	2340	291

Test Case : TCASE3.

Faulty procs.	η	t_{exec}	Processor	Vertices	Edges	Comm. Cost	Cross Edges
{1,2,3,13,14}	65%	266	0	127	684	2055	62
			4	111	640	2300	64
			5	123	681	1250	39
			6	103	574	1495	44
			7	133	703	1280	41
			8	101	564	2760	82
			9	107	619	2555	67
			10	115	635	1585	57
			11	128	712	895	34
			12	116	601	2010	53
			15	102	563	2885	71
{0,3,4,7, 11,12,15}	70%	283	1	135	748	1600	60
			2	137	736	2335	82
			5	140	773	2255	59
			6	137	754	2245	70
			8	150	844	1280	44
			9	150	793	1220	33
			10	137	758	2125	58
			13	137	785	2230	65
			14	143	785	2155	61

Table 1. Detailed results of DGA when mapping to incomplete topologies.

3.2 SA

Fault tolerance capability of SSA is demonstrated in figures 4 to 6. It can be inferred that SSA is also completely fault tolerant even when $|V_f| = 3$ and 7 in TCASE1 and TCASE2 respectively. These are the cases where *MCOMP* has only one functioning processor. The mapping to a single processor was obviously very fast. Detailed results are shown in table 2 involving the number of local edges and vertices mapped to each processor, as well as the cross edges and communication costs of each processor.

SPSA is also fault tolerant. This can be inferred from figures 7 and 8 and the detailed results in table 3.

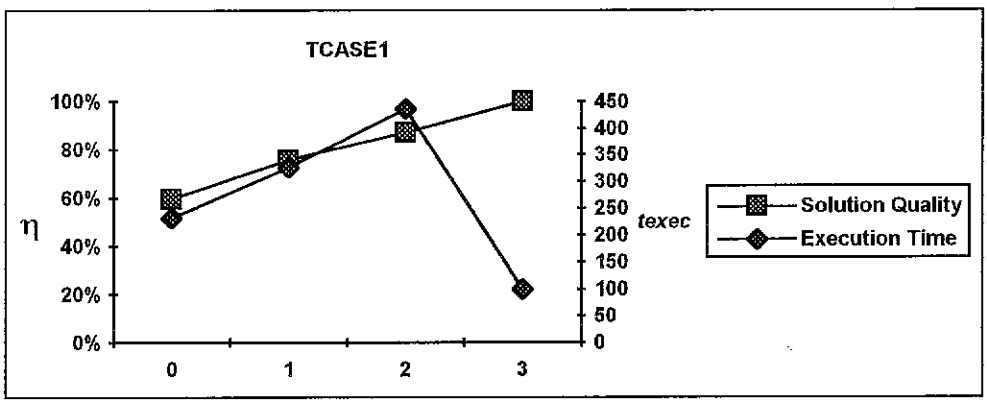


Figure 4. Performance of SSA when mapping to incomplete topologies in TCASE1 (x-axis shows $|V_f|$).

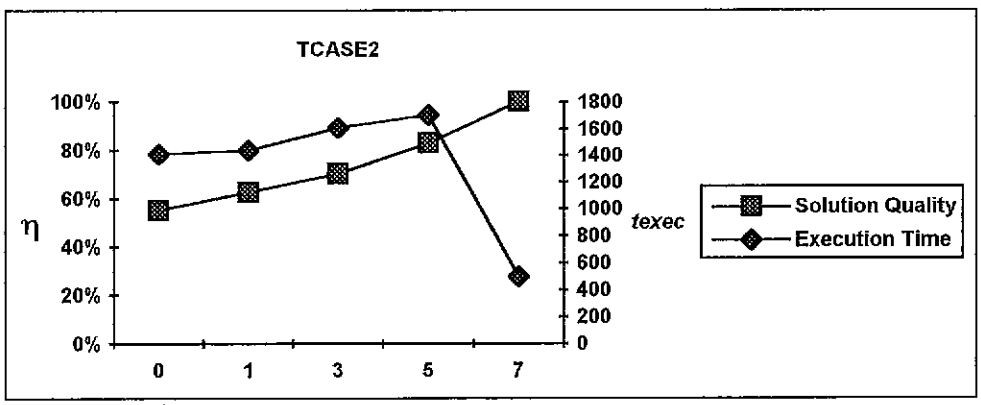


Figure 5. Performance of SSA when mapping to incomplete topologies in TCASE2 (x-axis shows $|V_f|$).

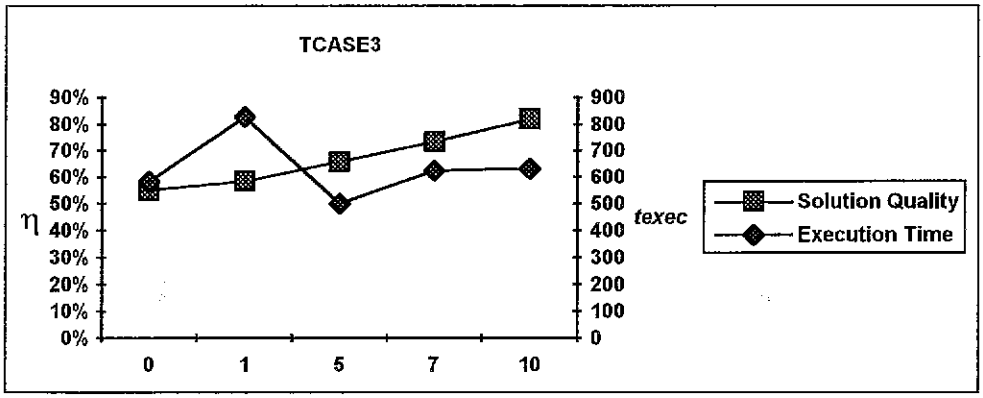


Figure 6. Performance of SSA when mapping to incomplete topologies in TCASE3 (x-axis shows $|V_f|$).

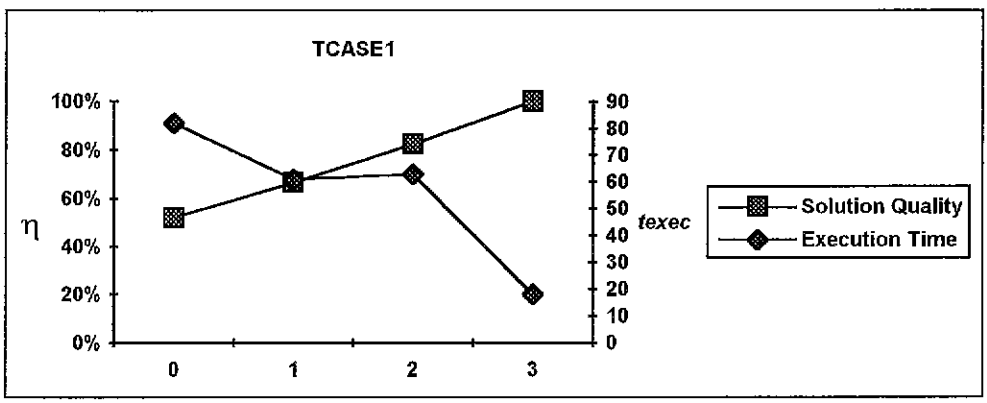


Figure 7. Performance of SPSA when mapping to incomplete topologies in TCASE1 (x-axis shows $|V_f|$).

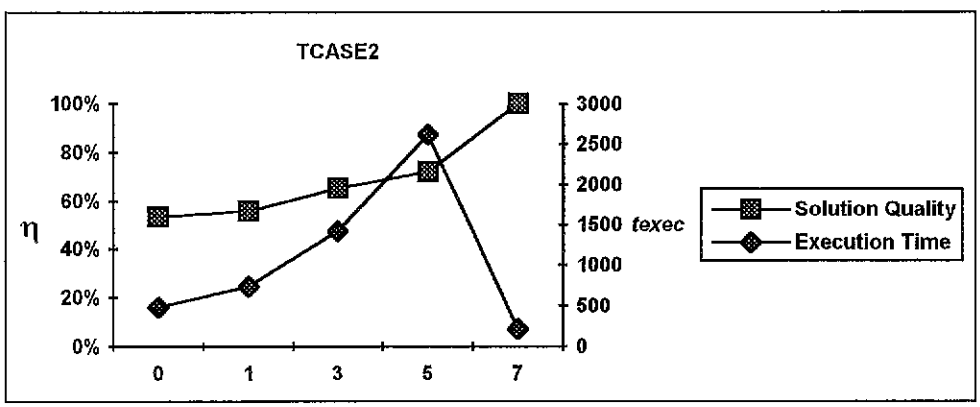


Figure 8. Performance of SPSA when mapping to incomplete topologies in TCASE2 (x-axis shows $|V_f|$).

Test Case : TCASE1.

Faulty procs.	η	t_{exec}	Processor	Vertices	Edges	Comm. Cost	Cross Edges
{0}	76%	327	1	106	372	545	10
			2	108	383	530	7
			3	87	311	1105	17
{2}	74%	289	0	112	393	575	15
			1	85	303	1225	33
			3	104	370	635	18
{1,2,3}	100%	100	0	300	1066	0	0

Test Case : TCASE2.

Faulty procs.	η	t_{exec}	Processor	Vertices	Edges	Comm. Cost	Cross Edges
{1,5,7}	70%	1608	0	100	1071	2245	173
			2	108	1246	3540	336
			3	121	1363	1160	163
			4	99	1193	2695	287
			6	117	1399	2725	283
{0,3,4,6,7}	83%	1701	1	167	2011	2405	319
			2	177	1994	1855	160
			5	201	2267	1815	165

Test Case : TCASE3.

Faulty procs.	η	t_{exec}	Processor	Vertices	Edges	Comm. Cost	Cross Edges
{1,2,3,13,14}	66%	501	0	107	603	2500	77
			4	119	635	1935	89
			5	132	739	1345	59
			6	119	640	1720	44
			7	120	670	1935	84
			8	92	521	2660	117
			9	114	614	1855	56
			10	128	704	1825	60
			11	126	669	2055	95
			12	99	564	1880	54
			15	110	617	1960	73
{0,3,4,7, 11,12,15}	73%	652	1	135	753	1780	51
			2	128	709	1150	39
			5	154	837	605	23
			6	144	756	1060	26
			8	134	750	2020	82
			9	139	793	1845	73
			10	131	734	2025	94
			13	157	849	665	27
			14	144	795	1165	39

Table 2. Detailed results of SSA when mapping to incomplete topologies.

Test Case : TCASE1.

Faulty procs.	η	t_{exec}	Processor	Vertices	Edges	Comm. Cost	Cross Edges
{0}	65%	61	1	120	427	755	27
			2	95	332	695	22
			3	86	307	1420	49
{2}	59%	61	0	96	341	1535	49
			1	100	357	1737	77
			3	105	368	1475	42
{1,2,3}	100%	18	0	300		0	0

Test Case : TCASE2.

Faulty procs.	η	$t_{exec}c$	Processor	Vertices	Edges	Comm. Cost	Cross Edges
{1,2,3}	65%	1419	0	125	1399	2705	233
			4	103	1220	3765	436
			5	79	946	3520	332
			6	94	1114	3385	272
			7	144	1593	2305	169
{0,3,4,6,7}	72%	2626	1	216	2403	3500	541
			2	177	1987	1215	159
			5	152	1882	1865	382

Table 3. Detailed results of SPSA when mapping to incomplete topologies.

4. Summary and Conclusions

In this chapter, we demonstrated experimentally the ability of 2wDGA1, SSA, and SPSA to map to faulty or incomplete architectures. We have shown that only slight modifications are needed to make these algorithms fault tolerant.

Chapter 8

Mapping to Different Topologies

Different computation models and algorithmic characteristic and the variety of multicomputer topologies and architectures make the task of automating the mapping process very difficult. The existence of mapping algorithms that are unbiased towards certain architectures and topologies is very important because it accomplishes a big step towards mapping automation. Through out this work, we were assuming that the underlying multicomputer (*MCOMP*) topology is a cube. However, existing multicomupters have various topologies such as arrays, rings, trees, and meshes [Hwang 1993; Bertsekas and Tistsiklis 1989; Saghi et al. 1993]. In this chapter, we will drop this assumption to demonstrate the ability of GAs and SAs to map to different multicomputer topologies. We will show experimentally that these algorithms are not biased towards a particular topology. The work is motivated by the fact that most proposed heuristics in the mapping context are biased towards particular multicomputer topologies especially cubes. For example, recursive spectral bisection and recursive coordinate bisection can only map to cubes, where the number of processors is a power of two, and possibly to 2^k meshes.

In the following sections, We will provide a summary of famous topologies revising the function $H(p,q)$ presented in equation 6 of chapter 2 for each topology. Then, experimental results are shown. The algorithms used are 2wDGA1 (DGA for short), SSA, and SPSA

1. Major Topologies

The major topologies summarized in this section are depicted in figure 1. We characterize them by the number of nodes, number of edges, diameter, and bisection width. We let $|V_m| = N$. The diameter gives the longest possible distance (in edges) between two nodes in the topology. The bisection width constitutes the number of cross edges resulting after partitioning the topology into two subgraphs.

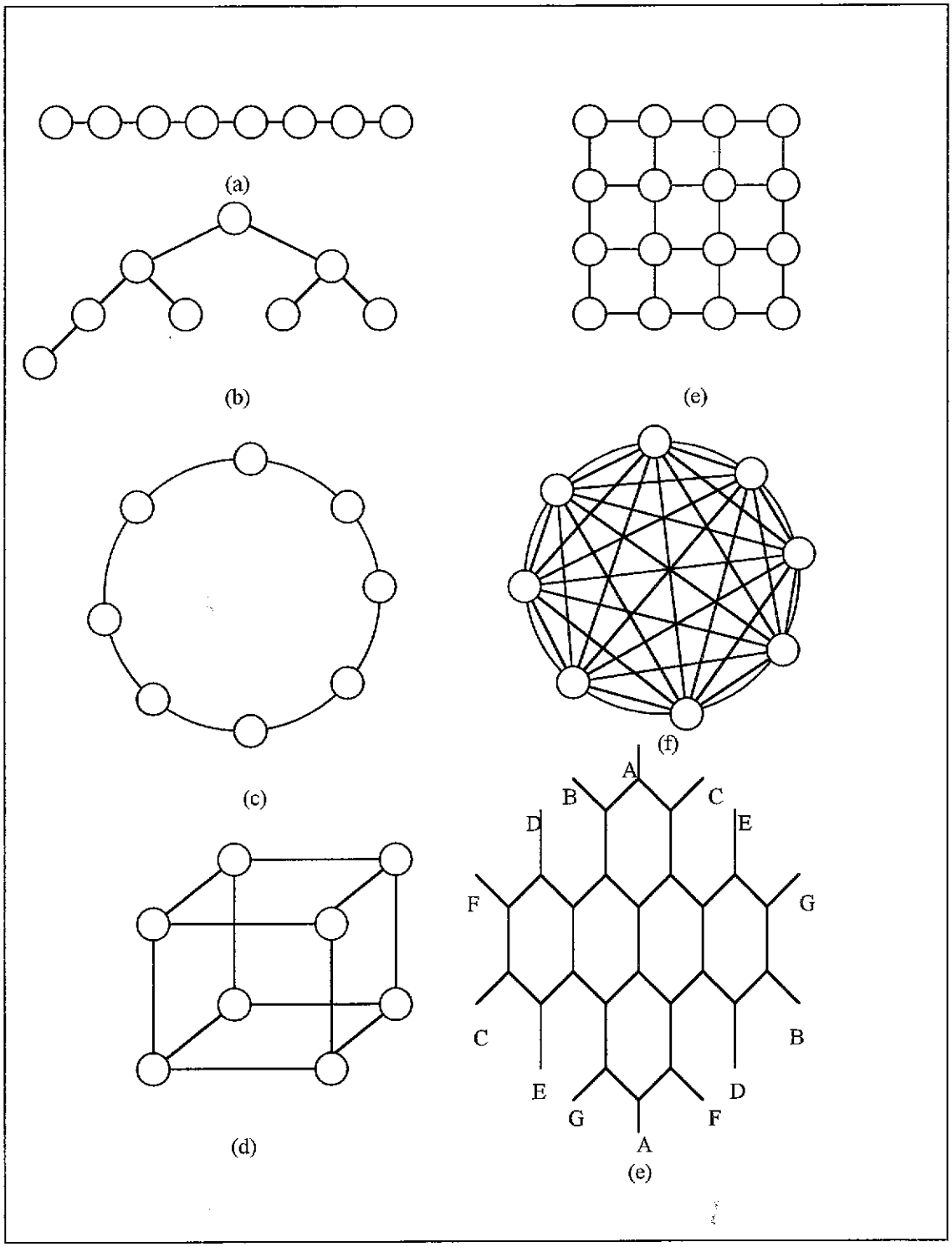


Figure 1. Multicomputer Topologies (a) linear array (b) binary tree (c) ring (d) hypercube (e) 2-D mesh (f) complete graph (g) star graph.

The ***linear array*** is the simplest topology. It consists of N processor nodes, nodes for short, connected by $N-1$ links. The node degree is 2 except for terminal nodes where the degree is 1. The diameter of the linear array is $N-1$ and its bisection width is 1. The distance between any two nodes in a linear array topology is given by :

$H_{array}(p,q) = | p - q |$.

A bidirectional ***ring*** consists of N nodes connected by N links. The node degree and the bisection width are both equal to 2. The diameters is $\lfloor N / 2 \rfloor$, and the distance between two arbitrary nodes is given by

$H_{Bring}(p,q) = Min \{ |p - q|, N - |p - q| \}$.

A ***fully connected*** (or ***complete***) topology is formed out by connecting each node to every other node via a direct link. Thus, the node degree is $N - 1$ and its diameter is 1, the shortest possible diameter. For N nodes, the fully connected topology requires $N(N - 1) / 2$ links. Finally, $H_{full}(p,q)$ is always = 1.

A balanced ***binary tree*** topology requires $N - 1$ links to connect the N nodes. The node degree is at most 3 and at least 1, and the bisection width is 1. The diameter of the topology equals $2 (\lceil log_2 N \rceil - 1)$. The distance between any two nodes in the binary tree is computed using *Floyd's* algorithm [Brassard and Brately 1988]. Therefore,

$H_{tree}(p,q) = Floyd(p,q)$.

Fat trees are variations on conventional trees in which the tree becomes thicker as we go up towards the root. In other words, the channel width becomes larger at the root to alleviate the traffic to the root problem in ordinary trees [Hwang 1992; Hillis and Lewis 1993].

For a dxd mesh (***2D-mesh***) consisting of N nodes, $d = \sqrt{N}$, the diameter = $2 (\sqrt{N} - 1)$, the number of links = $2N - 2\sqrt{N}$, and the bisection width is $\sqrt{N}$. Finally, we assume in 2D-meshes that nodes p and q have the coordinates (x_p, y_p) and (x_q, y_q) respectively then,

$H_{2D\text{-}mesh}(p,q) = | x_p - x_q | + | y_p - y_q |$.

3D-meshes (*dxdxd* mesh) have a diameter of $3(\sqrt[3]{N} - 1)$, a number of links of $3N - 3\sqrt[3]{N}$, with a bisection width of $\sqrt[3]{N}$. Also assuming the coordinates of p are (x_p,y_p,z_p) and of q are (x_q,y_q,z_q) then,

$$H_{3D\text{-}mesh}(p,q) = |x_p - x_q| + |y_p - y_q| + |z_p - z_q| .$$

The mesh properties discussed above assume equal number of processors on all mesh dimensions. these properties can be easily generalized to cope with *d1xd2*(*xd3*) meshes where $d1 \neq d2$ ($\neq d3$). Experimentally, we will make use of such meshes. For example, we will use 2x4, 2x8, and 2x2x4 meshes.

A ***k-cube*** consists of $N = 2^k$ nodes connected by $2N$ links. the diameter is k, the node degree is k, and the bisection width is $2k$ [Desrochers 1988;Hwang 1993; Fox et al. 1988; Bertsekas and Tsitsiklis 1989]. The distance between nodes p and q in cubes has been presented earlier in chapter 2. Recall that

$H_{cube}(p,q) = Hamming\ distance\ (p,q)$.

A ***k-star*** topology [Misic and Jovanovic 1994; Day and Tripathi 1994] consists of $N = k!$ nodes labeled with N permutations of k symbols. There is a link between two nodes in a star if their numbers (which are permutations) differ only in the first position and in one other position. The node degree is $(k - 1)!$, and the diameter is $\lfloor 3/2(k - 1) \rfloor$. Finally,

$H_{star}(p,q) = star\_routing(p,q)$.
star_routing(p,q) is illustrated in appendix A.
The properties of these commonly used topologies is summarized in Table 1.

Topology	$\|V_m\|$	$\|E_m\|$	Node Degree	Bisection Width	Diameter
Linear Array	N	$N-1$	at most 2	1	$N-1$
Ring	N	N	2	2	$\lfloor N/2 \rfloor$
Binary Tree	N	$N-1$	at most 3	1	$2(\lceil \log_2 N \rceil - 1)$
2D-Mesh	N	$2N - 2N^{1/2}$	at most 4	$N^{1/2}$	$2(N^{1/2} - 1)$
3D-Mesh	N	$3N - N^{1/3}$	at most 8	$N^{1/3}$	$3(N^{1/3} - 1)$
k-Cube	$N=2^k$	$2N$	k	$2k$	k
k-Star	$N= k!$		$(k-1)!$		$\lfloor 3/2(k-1) \rfloor$
Fully Connected	N	$N(N-1)/2$	$N-1$		1

Table 1. Summary of the properties of famous multicomputer topologies.

2. Experimental Results

We introduce two additional test cases which constitute variations of the test cases summarized in table 2 of chapter 5. They are TCASE2a and TCASE3a. TCASE2a is the data set of TCASE2 mapped to a multicomputer of 6 (3!) nodes, and TCASE3a is TCASE3 mapped to a multicomputer of 24 (6!) nodes. These two test cases are introduced to demonstrate the ability of DGA, SSA, and SPSA to map to the star graph topology.

We note that a 2-cube and a 2x2 mesh are of equivalent topological properties. However, a 4-node ring is topologically different from these two because of node numbering. (Starting from node 0 and counting clockwise, a 2-cube and 2x2 mesh are numbered 0, 2, 3, and 1, where a ring is numbered 0, 1, 2, and 3).

It should be noted that the objective function we used, as well as all the functions exposed in the mapping literature, do not account for link contention, synchronization overhead, and queuing delay. That is why we were not able to test GAs and SAs against topologies such as the fat tree. Devising an objective function which takes into consideration the mentioned measures is still a research issue. And, any improvement shall take place in this area will have its direct influence on GAs and SAs so that the solutions of such algorithms are more accurate.

2.1. GA

The ability of DGA to map to different multicomputer topologies is illustrated in figures 2 to 6. Detailed results are depicted in tables 2 to 9.

DGA shows comparable solution qualities for all chosen test cases (TCASE1, TCASE2, TCASE3, TCASE2a, and TCASE3a).

In TCASE1, DGA solution details for a 2-cube and a 2x2 mesh are exactly the sane, and they are different for a ring because of the reason stated above. Solutions for array, ring, and tree topologies are better than that of a complete topology in this case because $|V_m|$ is small.

In TCASE2, TCASE3, TCASE2a, and TCASE3a, the complete topology shows the best solutions in spite of the fact that this topology drives the algorithm to an unbalanced treatment of the computation and communication aspects. This fact gives DGA the property of general applicability. This can be inferred form the detailed results in tables 4 to 17 where DGA load distribution is excellent while maintaining minimum communication.

The solution qualities for the array and ring topologies decrease as $|V_m|$ increases due to the poor properties of these architectures at high dimensions.

Solutions for the star graph are excellent and compares to those of a complete graph topology. Finally, we note that solutions for binary tree are also excellent the fact that leads us to conclude that DGA applies also for fat trees. Recall that the objective function we are employing does not take into consideration link contention.

In one word, DGA shows no bias towards a particular multicomputer topology or architecture.

2.2 SA

Results for SSA are shown in figures 6 to 9. Detailed results are exhibited in tables 10 to 17.

Except for the complete topology, SSA solution are comparable for all other topologies. The complete topology leads SSA to treat only the computation part of the objective function after an incremental change. This can be inferred from the detailed results where load distribution is excellent, but communication costs are high (refer to tables 10 to 17).

TCASE1 shows that SSA solutions for 2-cube and 2x2 mesh are the same. Results in TCASE2 are very near to each other. However, array, linear array, binary tree, and 2x8 mesh are of inferior solutions if compared to cube, 4x4 mesh, and 2x2x4 mesh in TCASE3.
Finally, SSA shows excellent mappings o the star graph topology. Therefore, SSA shows no bias towards certain topologies.
SPSA results are depicted in figures 10 to 12 and tables 22 to 24.

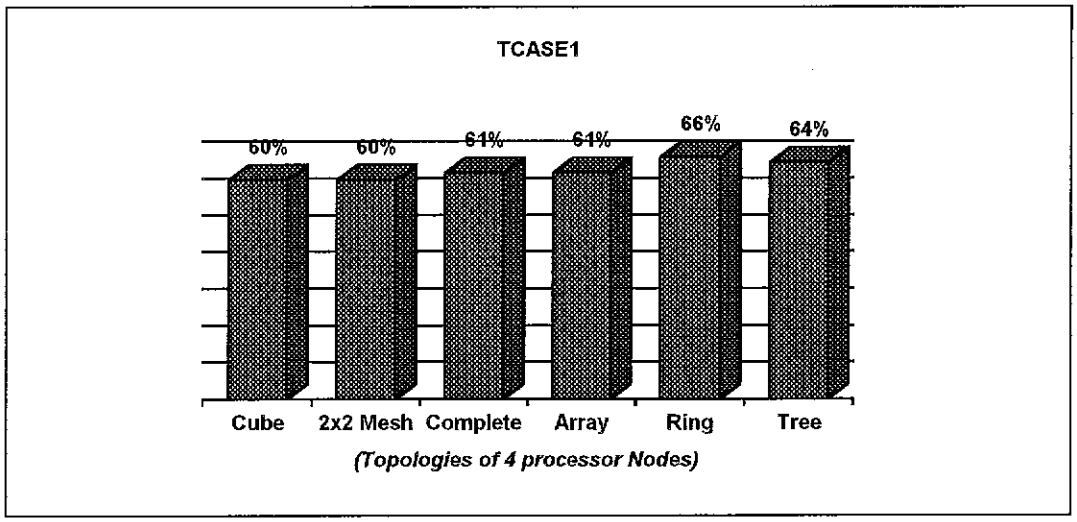


Figure 2. DGA mapping TCASE1 to different multicomputer topologies ($|V_m| = 4$).

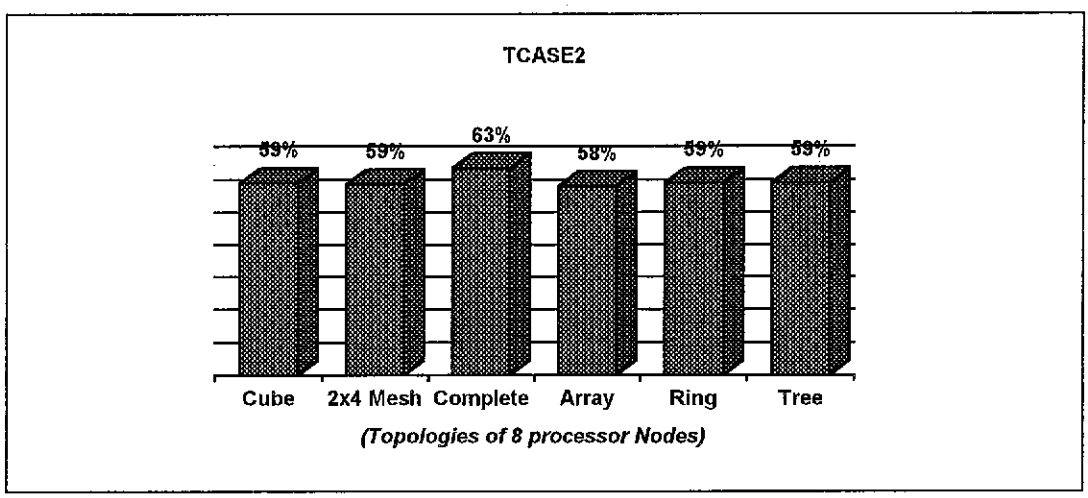


Figure 3. DGA mapping TCASE2 to different multicomputer topologies ($|V_m| = 8$).

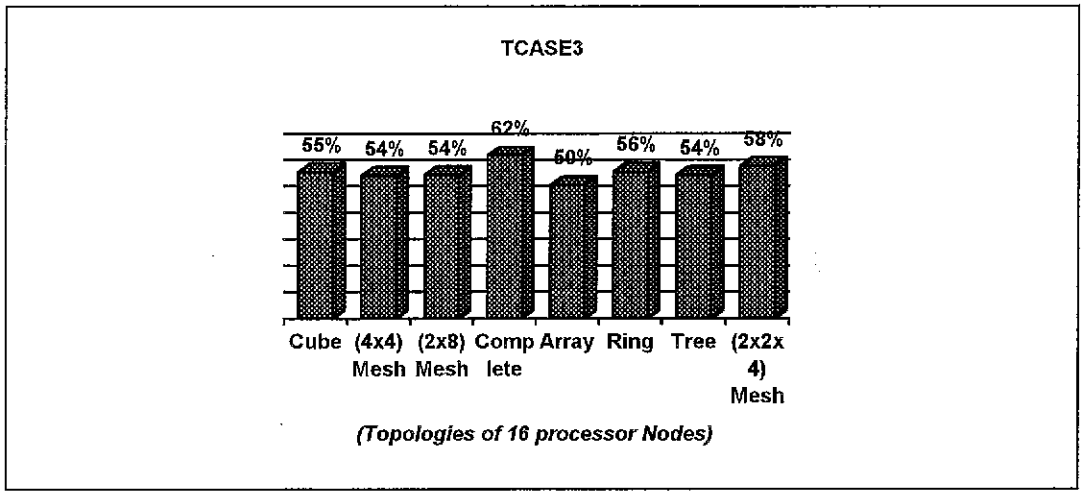


Figure 4. DGA mapping TCASE3 to different multicomputer topologies ($|V_m|$ = 16).

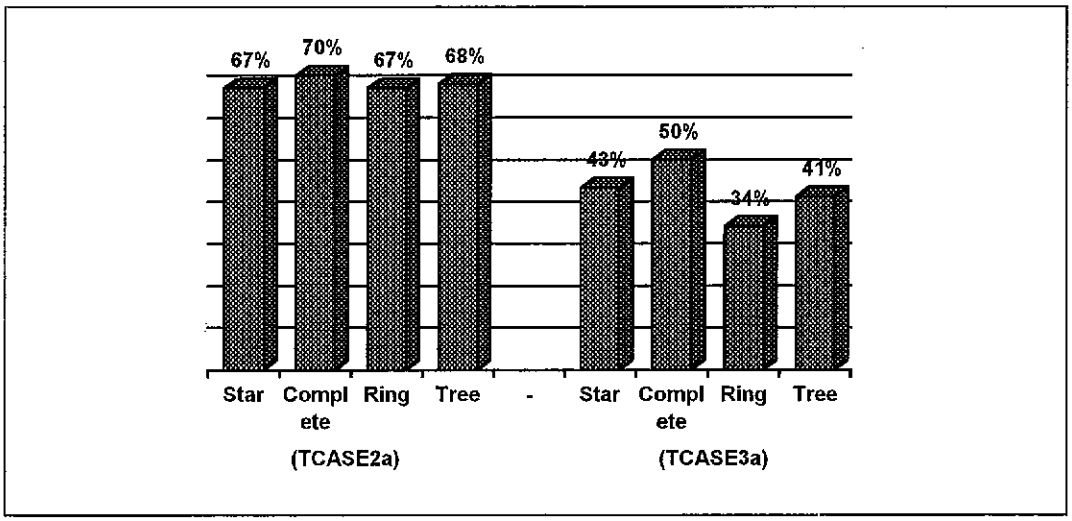


Figure 5. DGA mapping TCASE2a and TCASE3a to different multicomputer topologies ($|V_m|$ = 6 and 24 respectively).

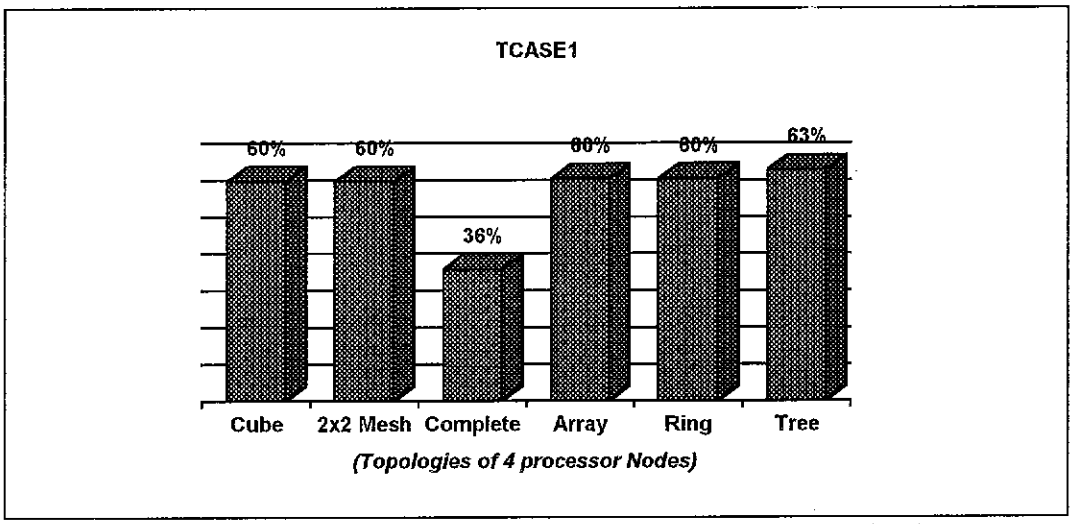


Figure 6. SSA mapping TCASE1 to different multicomputer topologies ($|V_m| = 4$).

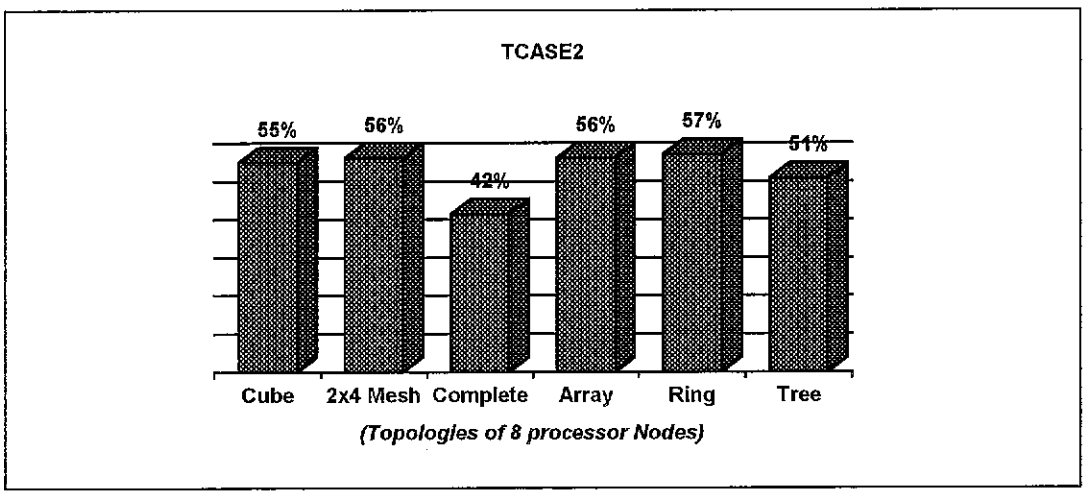


Figure 7. SSA mapping TCASE2 to different multicomputer topologies ($|V_m| = 8$).

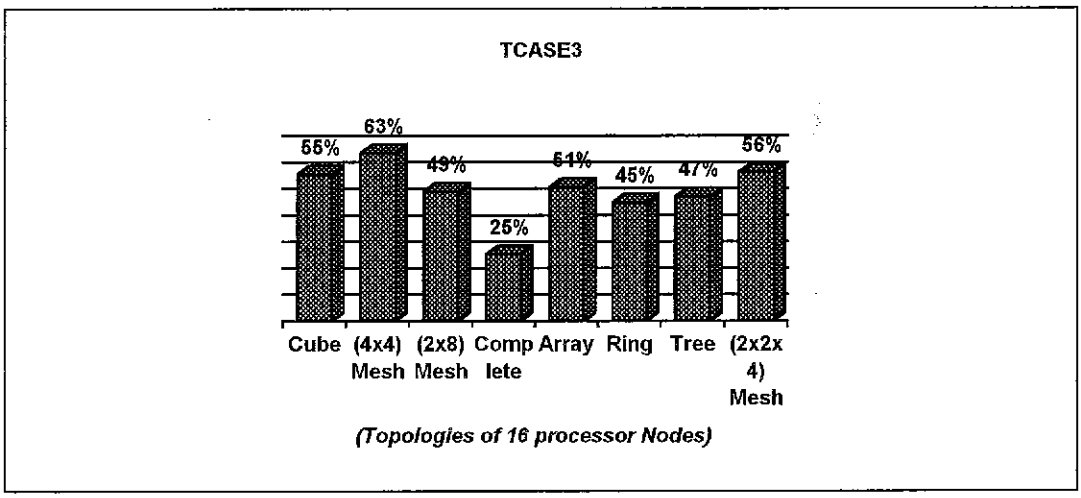


Figure 8. SSA mapping TCASE3 to different multicomputer topologies ($|V_m| = 16$).

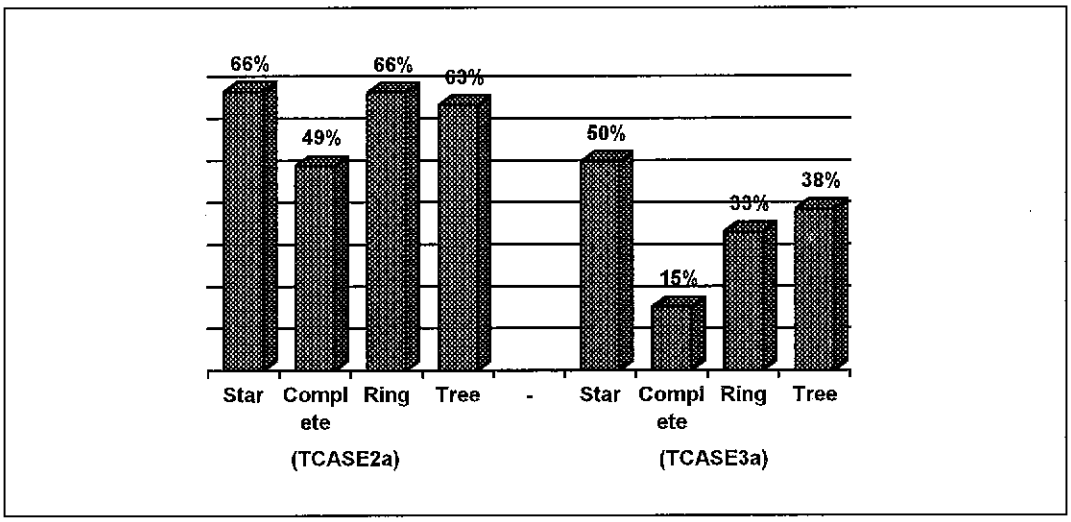


Figure 9. SSA mapping TCASE2a and TCASE3a to different multicomputer topologies ($|V_m| = 6$ and 24 respectively).

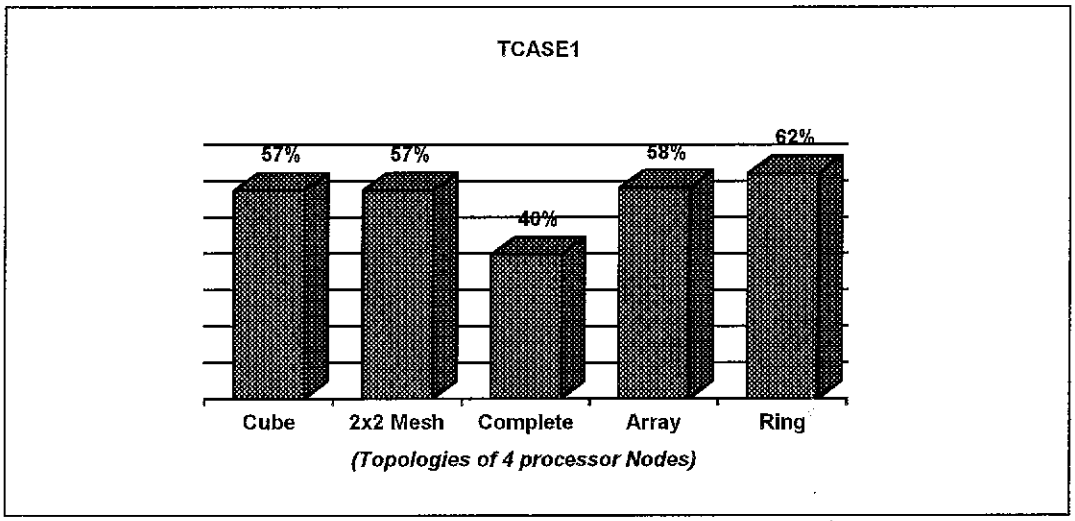


Figure 10. SPSA mapping TCASE1 to different multicomputer topologies ($|V_m| = 4$).

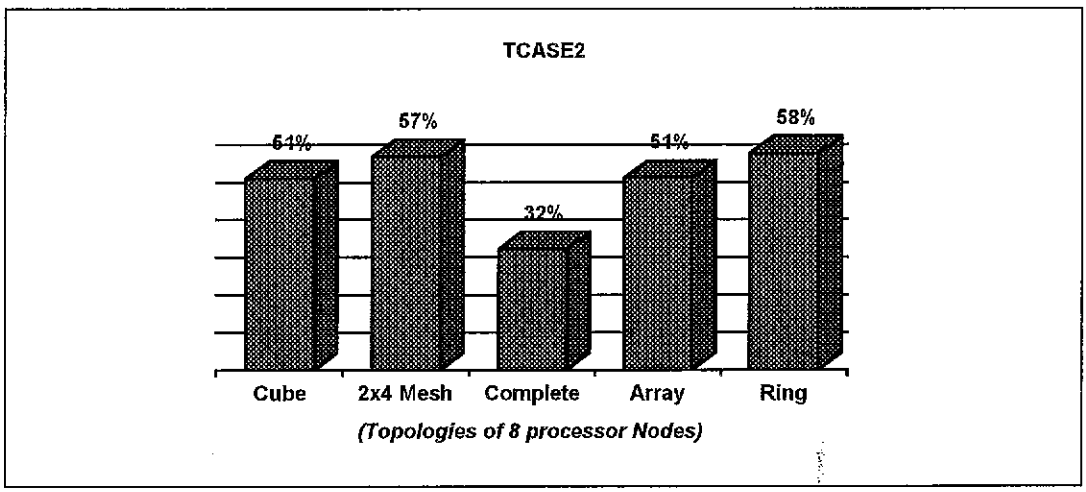


Figure 11. SPSA mapping TCASE2 to different multicomputer topologies ($|V_m| = 8$).

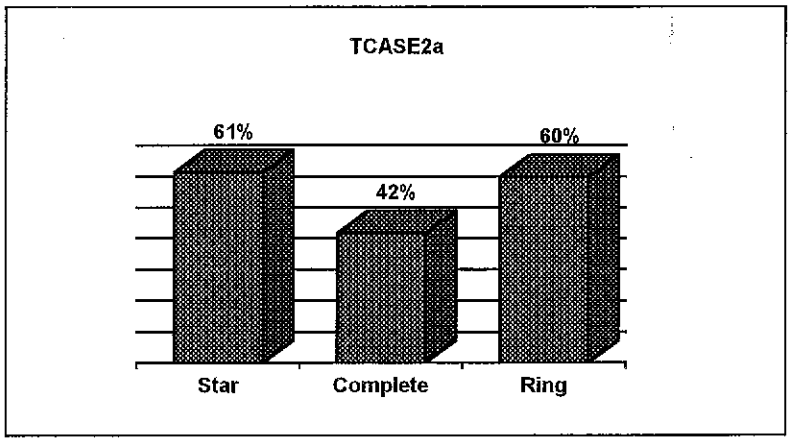


Figure 12. SPSA mapping TCASE2a to different multicomputer topologies ($|V_m| = 6$).

processor	*Array*	*Cube*	*Ring*	*Tree*	*Complete*
p0	(75,272)	(75,272)	(55,205)	(94,333)	(75,272)
p1	(58,189)	(58,189)	(66,228)	(72,258)	(58,189)
p2	(92,327)	(92,327)	(91,320)	(68,242)	(92,327)
p3	(76,278)	(76,278)	(89,313)	(67,233)	(76,278)

Table 2. DGA solutions for TCASE1: local workloads (vertices, edges) for selected topologies.

processor	*Array*	*Cube*	*Ring*	*Tree*	*Complete*
p0	(13,1100)	(20,1220)	(35,1285)	(9,560)	(20,1120)
p1	(15,1115)	(29,1780)	(22,1150)	(18,1090)	(29,1680)
p2	(18,1190)	(9,660)	(10,560)	(8,730)	(9,560)
p3	(20,1205)	(18,1190)	(23,650)	(17,1275)	(18,1090)

Table 3. DGA solutions for TCASE1: communication (cross edges, communication costs) for selected topologies.

Processor	*Array*	*Cube*	*2x4 Mesh*	*Tree*	*Complete*
p0	(74,854)	(60,739)	(72,845)	(76,879)	(54,655)
p1	(74,803)	(59,673)	(56,665)	(60,713)	(67,799)
p2	(75,862)	(68,789)	(82,865)	(69,801)	(64,760)
p3	(62,757)	(66,754)	(57,693)	(56,688)	(81,891)
p4	(75,796)	(86,930)	(76,823)	(78,846)	(54,655)
p5	(56,673)	(65,782)	(67,792)	(84,911)	(88,933)
p6	(64,766)	(65,778)	(70,812)	(53,652)	(65,773)
p7	(65,761)	(76,827)	(65,787)	(69,782)	(72,806)

Table 4. DGA solutions for TCASE2: local workloads (vertices, edges) for selected topologies.

Processor	*Array*	*Cube*	*2x4 Mesh*	*Tree*	*Complete*
p0	(182,3475)	(239,3585)	(207,3425)	(197,2980)	(219,3735)
p1	(145,1925)	(199,4600)	(211,4720)	(219,3770)	(205,2825)
p2	(214,3340)	(193,3045)	(147,2170)	(183,3630)	(192,3145)
p3	(223,3925)	(198,3565)	(227,3650)	(220,4080)	(171,1720)
p4	(170,3670)	(156,2315)	(141,2225)	(154,2655)	(211,3235)
p5	(221,4505)	(204,3765)	(196,2820)	(161,2930)	(149,2130)
p6	(200,4130)	(212,3195)	(196,3535)	(218,4735)	(191,3190)
p7	(215,4070)	(151,2155)	(201,3880)	(184,2705)	(204,2720)

Table 5. DGA solutions for TCASE2: communication (cross edges, communication costs) for selected topologies.

Processor	*Cube*	*4x4 Mesh*	*2x2x4 Mesh*	*Ring*	*Complete*
p0	(84,456)	(67,376)	(78,411)	(78,438)	(68,387)
p1	(90,465)	(83,450)	(86,473)	(67,383)	(99,523)
p2	(87,488)	(88,511)	(68,374)	(69,381)	(73,403)
p3	(70,401)	(84,452)	(78,408)	(80,445)	(79,431)
p4	(76,426)	(86,462)	(79,442)	(93,524)	(81,459)
p5	(86,480)	(68,389)	(72,399)	(85,447)	(83,468)
p6	(67,362)	(77,426)	(78,452)	(72,408)	(82,458)
p7	(89,500)	(80,442)	(82,431)	(78,430)	(74,413)
p8	(76,425)	(90,484)	(75,414)	(68,369)	(82,439)
p9	(81,428)	(72,401)	(77,429)	(81,450)	(85,447)
p10	(68,361)	(81,461)	(81,463)	(90,500)	(91,499)
p11	(80,429)	(77,420)	(79,451)	(66,377)	(76,422)
p12	(91,509)	(63,325)	(90,476)	(93,493)	(64,373)
p13	(77,430)	(87,465)	(79,442)	(91,485)	(87,465)
p14	(70,397)	(85,478)	(84,466)	(80,490)	(68,388)
p15	(74,419)	(78,434)	(80,445)	(75,427)	(74,401)

Table 6. DGA solutions for TCASE3: local workloads (vertices, edges) for selected topologies.

Processor	*Cube*	*4x4 Mesh*	*2x2x4 Mesh*	*Ring*	*Complete*
p0	(28,1190)	(74,2785)	(17,860)	(44,1835)	(57,2150)
p1	(29,1175)	(26,735)	(39,1350)	(61,1855)	(17,560)
p2	(44,1295)	(63,2085)	(60,2580)	(49,2765)	(41,1180)
p3	(51,2295)	(46,2450)	(40,2175)	(33,1920)	(51,1680)
p4	(40,1975)	(42,1295)	(46,2050)	(38,1835)	(49,1665)
p5	(34,1405)	(49,1325)	(43,1380)	(15,645)	(52,1680)
p6	(52,2705)	(70,2225)	(70,2030)	(62,2370)	(38,1150)
p7	(44,1950)	(38,1165)	(45,2035)	(46,2435)	(43,1605)
p8	(35,1320)	(50,2265)	(34,1420)	(57,2850)	(23,620)
p9	(50,2490)	(35,1220)	(59,1770)	(36,1535)	(15,545)
p10	(15,845)	(51,1995)	(47,1935)	(48,1750)	(41,1180)
p11	(61,2070)	(32,1520)	(57,2125)	(51,2680)	(36,1575)
p12	(47,1950)	(39,2175)	(36,1975)	(15,1145)	(53,1680)
p13	(38,1990)	(41,2405)	(36,1120)	(17,1175)	(17,560)
p14	(59,2595)	(64,2300)	(46,1950)	(19,975)	(50,2135)
p15	(49,2065)	(36,1320)	(33,1320)	(47,2450)	(51,2120)

Table 7. DGA solutions for TCASE3: communication (cross edges, communication costs) for selected topologies.

Processor	*Star*	*Complete*	*Tree*	*Ring*
p0	(96,1029)	(101,1153)	(87,1056)	(87,1042)
p1	(80,975)	(100,1120)	(93,1080)	(84,997)
p2	(79,943)	(107,1172)	(95,1075)	(105,1143)
p3	(83,951)	(76,904)	(78,902)	(82,963)
p4	(91,1087)	(80,944)	(92,1013)	(96,1065)
p5	(116,1287)	(81,979)	(100,1146)	(91,1062)

Table 8. DGA solutions for TCASE2a: local workloads (vertices, edges) for selected topologies.

Processor	*Star*	*Complete*	*Tree*	*Ring*
p0	(169,2360)	(203,2325)	(240,3190)	(230,3260)
p1	(249,3740)	(196,2235)	(232,3060)	(227,2485)
p2	(261,3950)	(170,1735)	(233,3140)	(181,2945)
p3	(245,3245)	(270,3020)	(238,4010)	(227,3695)
p4	(229,3085)	(266,2945)	(155,1715)	(155,1615)
p5	(175,2290)	(261,3080)	(212,2270)	(224,3170)

Table 9. DGA solutions for TCASE2a: communication (cross edges, communication costs) for selected topologies.

processor	*Array*	*Cube*	*Ring*	*Tree*	*Complete*
p0	(65,229)	(68,240)	(77,273)	(74,269)	(76,274)
p1	(65,226)	(82,284)	(69,252)	(71,257)	(77,276)
p2	(73,262)	(75,266)	(78,275)	(72,248)	(73,255)
p3	(98,349)	(76,276)	(77,266)	(84,292)	(75,261)

Table 10. SSA solutions for TCASE1: local workloads (vertices, edges) for selected topologies.

processor	*Array*	*Cube*	*Ring*	*Tree*	*Complete*
p0	(19,635)	(22,1090)	(25,1180)	(17,1090)	(198,3240)
p1	(32,1225)	(16,985)	(20,1090)	(23,1120)	(190,3300)
p2	(34,1210)	(36,1270)	(19,1090)	(10,545)	(189,3270)
p3	(21,650)	(30,1195)	(24,1105)	(16,575)	(195,3330)

Table 11. SSA solutions for TCASE1: communication (cross edges, communication costs) for selected topologies.

Processor	*Array*	*Cube*	*2x4 Mesh*	*Tree*	*Complete*
p0	(61,621)	(83,922)	(58,627)	(73,858)	(66,790)
p1	(73,847)	(72,807)	(63,736)	(78,969)	(70,777)
p2	(71,877)	(60,715)	(63,720)	(76,904)	(68,781)
p3	(76,926)	(60,706)	(70,814)	(84,1014)	(70,783)
p4	(69,804)	(72,810)	(72,872)	(45,511)	(67,783)
p5	(71,846)	(67,752)	(64,779)	(66,691)	(68,782)
p6	(72,818)	(76,910)	(66,738)	(43,492)	(68,793)
p7	(52,533)	(55,650)	(89,986)	(80,833)	(67,783)

Table 12. SSA solutions for TCASE2: local workloads (vertices, edges) for selected topologies.

Processor	*Array*	*Cube*	*2x4 Mesh*	*Tree*	*Complete*
p0	(119,1460)	(174,2930)	(163,2230)	(284,4465)	(620,7565)
p1	(267,3425)	(165,2740)	(200,2275)	(345,4005)	(619,7640)
p2	(287,3205)	(243,4020)	(218,4720)	(310,3345)	(631,7715)
p3	(270,3280)	(218,3495)	(234,3585)	(296,3385)	(619,7580)
p4	(304,4040)	(200,2920)	(246,3060)	(141,2180)	(608,7400)
p5	(312,3415)	(200,2365)	(253,3555)	(121,2105)	(628,7520)
p6	(254,2635)	240,3615)	(190,2890)	(150,2240)	(621,7670)
p7	(113,1520)	(238,4810)	(170,2330)	(143,1610)	(597,7550)

Table 13. SSA solutions for TCASE2: communication (cross edges, communication costs) for selected topologies.

Processor	*Cube*	*4x4 Mesh*	*2x2x4 Mesh*	*Ring*	*Complete*
p0	(76,434)	(77,424)	(75,399)	(87,485)	(81,431)
p1	(79,453)	(78,437)	(71,394)	(74,421)	(77,432)
p2	(86,455)	(73,414)	(79,452)	(85,476)	(81,429)
p3	(93,516)	(72,415)	(84,482)	(79,433)	(81,437)
p4	(79,423)	(78,411)	(74,388)	(94,488)	(76,424)
p5	(82,459)	(78,422)	(67,373)	(83,433)	(77,437)
p6	(91,479)	(82,453)	(84,467)	(87,479)	(77,429)
p7	(81,447)	(75,416)	(84,472)	(72,403)	(82,450)
p8	(72,405)	(83,457)	(81,448)	(86,498)	(80,437)
p9	(80,441)	(81,436)	(83,471)	(80,458)	(80,446)
p10	(80,434)	(74,415)	(70,391)	(64,349)	(82,444)
p11	(68,385)	(80,453)	(89,488)	(86,480)	(81,444)
p12	(63,339)	(89,469)	(87,480)	(78,423)	(76,434)
p13	(80,452)	(85,455)	(77,429)	(69,356)	(79,436)
p14	(70,365)	(84,468)	(68,343)	(73,406)	(78,431)
p15	(86,489)	(77,431)	(93,499)	(69,388)	(78,435)

Table 14. SSA solutions for TCASE3: local workloads (vertices, edges) for selected topologies.

Processor	*Cube*	*4x4 Mesh*	*2x2x4 Mesh*	*Ring*	*Complete*
p0	(50,1810)	(36,1150)	(17,560)	(67,2570)	(251,8850)
p1	(69,2340)	(39,1610)	(40,1165)	(83,3555)	(254,8910)
p2	(17,560)	(56,1725)	(54,2265)	(68,1900)	(243,8730)
p3	(34,1220)	(59,1840)	(74,1875)	(77,3780)	(247,8805)
p4	(17,560)	(17,560)	(16,545)	(54,2975)	(240,8805)
p5	(47,1895)	(28,1075)	(83,2360)	(75,3200)	(254,8910)
p6	(19,575)	(39,1750)	(47,2060)	(99,3335)	(255,8955)
p7	(35,1135)	(48,1120)	(44,1180)	(103,3680)	(258,8985)
p8	(45,1620)	(33,1650)	(40,1690)	(114,3005)	(251,8760)
p9	(37,1545)	(48,1695)	(59,1740)	(94,2080)	(264,9000)
p10	(34,1120)	(53,1695)	(53,2220)	(53,1255)	(256,8955)
p11	(49,2305)	(49,1650)	(40,1590)	(44,1180)	(252,8970)
p12	(61,2145)	(17,560)	(32,1600)	(27,1075)	(256,8985)
p13	(62,1870)	(17,575)	(57,2165)	(40,1150)	(268,8790)
p14	(31,1530)	(38,1135)	(43,2670)	(62,1825)	(239,8775)
p15	(49,1650)	(47,17655)	(25,1060)	(42,1210)	(247,8880)

Table 15. SSA solutions for TCASE3: communication (cross edges, communication costs) for selected topologies.

Processor	*Star*	*Complete*	*Tree*	*Ring*
p0	(86,1022)	(96,1040)	(93,1103)	(95,1098)
p1	(92,968)	(87,1026)	(96,1188)	(80,853)
p2	(91,1046)	(93,1057)	(107,1262)	(85,888)
p3	(93,1126)	(90,1048)	(106,1143)	(92,1106)
p4	(105,1277)	(90,1043)	(59,683)	(104,1277)
p5	(78,833)	(89,1058)	(84,893)	(89,1050)

Table 16. SSA solutions for TCASE2a: local workloads (vertices, edges) for selected topologies.

Processor	*Star*	*Complete*	*Tree*	*Ring*
p0	(334,3535)	(770,7435)	(343,4225)	(288,2185)
p1	(142,1025)	(808,7180)	(368,3000)	(133,1535)
p2	(286,2170)	(789,7630)	(286,2170)	(134,980)
p3	(306,2800)	(796,7075)	(153,1730)	(294,2755)
p4	(281,2080)	(789,7180)	(157,1625)	(287,2125)
p5	(133,1550)	(790,7135)	(137,1565)	(330,3475)

Table 17. SSA solutions for TCASE2a: communication (cross edges, communication costs) for selected topologies.

processor	*Array*	*Cube*	*Ring*	*Complete*
p0	(65,236)	(65,235)	(96,341)	(74,256)
p1	(72,252)	(51,177)	(68,236)	(83,296)
p2	(56,196)	(75,176)	(70,251)	(62,222)
p3	(108,382)	(110,378)	(67,238)	(82,292)

Table 18. SPSA solutions for TCASE1: local workloads (vertices, edges) for selected topologies.

processor	*Array*	*Cube*	*Ring*	*Complete*
p0	(28,1295)	(21,1090)	(13,620)	(106,2520)
p1	(38,1240)	(7,500)	(24,1150)	(110,2640)
p2	(26,1720)	(30,1210)	(21,1105)	(90,3385)
p3	(10,545)	(10,605)	(10,545)	(118,2670)

Table 19. SPSA solutions for TCASE1: communication (cross edges, communication costs) for selected topologies.

Processor	*Array*	*Cube*	*2x4 Mesh*	*Complete*
p0	(55,570)	(38,449)	(32,390)	(34,374)
p1	(71,813)	(46,522)	(84,902)	(92,1080)
p2	(71,867)	(85,1020)	(78,936)	(98,1144)
p3	(55,638)	(98,1058)	(69,817)	(40,456)
p4	(86,1021)	(55,604)	(65,762)	(103,1150)
p5	(66,794)	(79,927)	(66,764)	(32,352)
p6	(87,1006)	(78,944)	(67,772)	(48,578)
p7	(54,563)	(66,748)	(84,929)	(98,1138)

Table 20. SPSA solutions for TCASE2: local workloads (vertices, edges) for selected topologies.

Processor	*Array*	*Cube*	*2x4 Mesh*	*Complete*
p0	(112,1460)	(185,4480)	(166,2740)	(288,5000)
p1	(251,2575)	(162,3150)	(160,1675)	(842,8535)
p2	(309,3400)	(256,3105)	(246,3120)	(864,8915)
p3	(282,3845)	(154,3210)	(257,3720)	(374,5630)
p4	(343,3595)	(230,3450)	(262,4215)	(836,9020)
p5	(310,3310)	(231,3555)	(210,3465)	(296,5135)
p6	(282,3395)	(246,4155)	(200,2380)	(458,6125)
p7	(119,1475)	(190,4540)	(169,2200)	(830,8720)

Table 21. SPSA solutions for TCASE2: communication (cross edges, communication costs) for selected topologies.

Processor	*Star*	*Complete*	*Ring*
p0	(84,964)	(106,1197)	(110,1333)
p1	(74,876)	(58,699)	(100,1189)
p2	(99,1147)	(111,1298)	(85,899)
p3	(132,1459)	(101,1169)	(60,627)
p4	(81,992)	(95,1060)	(105,1224)
p5	(75,834)	(74,849)	(85,1000)

Table 22. SPSA solutions for TCASE2a: local workloads (vertices, edges) for selected topologies.

Processor	*Star*	*Complete*	*Ring*
p0	(274,3425)	(893,8155)	(319,2920)
p1	(216,3050)	(577,5695)	(299,3410)
p2	(221,2035)	(964,8455)	(135,980)
p3	(177,1750)	(893,7885)	(115,920)
p4	(300,3500)	(822,7525)	(280,3515)
p5	(240,3230)	(711,6625)	(290,2695)

Table 23. SPSA solutions for TCASE2a: communication (cross edges, communication costs) for selected topologies.

3. Summary and Conclusions

We presented comparative results of DGA, SSA, and SPSA to map to various multicomputer topologies. This include linear arrays, rings, binary trees, cubes, complete graphs, meshes, an star graph topologies. DGA is of general applicability and it shows no bias towards particular topologies. SSA's and SPSA's solutions are also comparable except for the complete topology which leads to unbalanced treatment of the computation and communication terms of the objective function. In one word, GA and SA are of general applicability and they do not show a bias towards particular topologies.

Chapter 9

Conclusions

The sensitivity to parameters and general applicability of genetic algorithms and simulated annealing for mapping data to multicomputers were analyzed. Three dimensions were considered in the analysis: sensitivity to all user parameters, fault tolerance capability, and applicability to different multicomputer topologies. Experiments were conducted for three test cases each has its own properties.

GAs were found to be insensitive in wide ranges to the objective function parameters as well as to the machine dependent parameters. GA parameters, such as the population size and the convergence threshold, must be determined experimentally. This is because they may lead GAs to be computationally expensive. DGAs are insensitive to the migration policy which consists of the number of migrants, nature of migrants, and migration direction. Moreover, GAs are completely fault tolerant even for high failure rates. Finally, GAs are applicable to different multicomputer architectures such as trees, rings, and even the star graph topology. In this sense, GAs are of general applicability.

SAs, on the other hand, were also insensitive to user parameters. Much like GAs, SAs are insensitive to objective function parameters and to machine dependent parameters. Parameters such as convergence threshold must be determined experimentally to yield good solutions in acceptable time. The correction frequencies of SPSA must also be determined experimentally. At low frequencies, solutions are excellent, but communication time is very high. However, at high frequencies, Solutions are still acceptable but computation time is very high. Moreover, SAs are also completely fault tolerant, and finally they show general applicability to different topologies.

This work established an experimental support of the conjecture that GAs and SAs are flexible and of general applicability. The existence of such algorithms is necessary to the automation of the mapping process.

Appendix A

Routing in Star Graph Topologies

Routing in star graphs is accomplished via applying procedure Star_Routing. The procedure incorporates two rules which form a generalization of the rules defined by [Day and Tripathi 1994] which route to the identity permutation *1234 .. n*.

Procedure *Star_Routing*(Source,Destination)

while Source ≠ Destination **do**
    if Source[1] = Destination[1] **then**
        /* Rule 1 */
        Exchange Source[1] with Source[i] such that Source[i] ≠ Destination[i];
        Increase the number of hops (distance);`
    else
        /* Rule 2 */
        Exchange Source[1] with Source[i] where Destination[i] = Source[1];
        Increase the number of hops (distance);`
endwhile.

References

Baker J. E. 1985. Adaptive Selection methods for genetic algorithms. *Int. Conf. on Genetic Algorithms*, 101-111.

Berger M. and Bokhari S. 1987. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE trans. on Parallel and Distributed Systems*. vol. 1, no.1, Jan., 91-106.

Bertsekas D., and Tistiklis J. 1989. *Parallel and Distributed Computation Numerical Methods*. Prentice-Hall.

Brassard G., and Brately P. 1988. *Algorithmics Theory and Practice*. Printice-Hall International, Inc.

Bultan T., and Aykanat C. 1992. A new mapping heuristic based on mean field annealing. *Journal of Parallel and Distributed Computing*,16:292-305.

Chrischoides N., Houstis E. N., Houstis C. E. 1991. Geometry based mapping strategies for PDE computations. *Int. Conf. on Supercomputing*, 115-127.

Chrisochoides N., Mansour N., and Fox G.C. 1994. Performance evaluation of load balancing algorithms for parallel single-phase iterative PDE solvers. Scalable High-Performance Computing Conference, May.

Crow J.F. 1986. *Basic Concepts in Population, Quantitative, and Evolutionary Genetics*. Freeman.

Day K., and Tripathi A. 1994. A comparative study of topological properties of hypercubes and star graphs. *IEEE trans. on Parallel and Distributed Systems*, vol. 5, no. 1, 31-38.

Desrochers G.R. 1988. *Principles of Parallel and Multiprocessing*. Interest Publications, Inc., McGraw-Hill Book Company.

Eglese R.W. 1990. Simulated annealing: a tool for operational research. *Euro. J. operational Research*, 46:271-281.

Ercal F. 1988. *Heuristic Approaches to Task Allocation for Paralell Computing*. Ph.D. Thesis, Ohio State University.

Fox G.C., 1988. A review of automatic load balancing and decomposition methods for the hypercube. *Numerical Algorithms for Modern Parallel Computers*, ed. M. Schultz, 63-76, Sprintg-Verlag.

Fox G.C., Johnson M., Lyzenga G., Otto S., Salmon J., and Walker D. 1988. *Solving Problems on Concurrent Processors*. Englewood Cliffs, N.J., Printice Hall.

Goldberg D.E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, Mass., Addison-Wesely.

Greening D.R. 1990. Parallel simulated annealing techniques. *Physica D* 42:293-306.

Hey A.J.G. 1990. Concurrent supercomputing in Europe. *5th Distributed Memory Computing Conf.*, 639-646.

Holland J.H. 1975. *Adaptation in Natural and Artificial systems*. Univ. of Michigan Press, Ann Arbor.

Hopfield J.J., and Tank D.W. 1986. Computing with neural circuits: A modes. *Science*, 233:625-639.

Hwang K., and Xu J. 1990. Mapping partitioned program Modules onto multicomputer nodes using simulated annealing. *Int. Conf. Parallel Processing*, vol. II, 292-293.

Hwang K. 1993. *Advanced Computer Architecture: Parallelism, Scalability, Programability*. McGraw-Hill, Inc.

Karmer O, and Muhelenbein H. 1989. Mapping Strategies in message-based multiprocessor systems. *Parallel Computing*, 213-225.

Kirkpatrick S., Gelatt C.D., and Vecchi M.P. 1983. Optimization by simulated annealing. *Science*, 220:671-680.

Mansour N. 1992. *Physical optimization algorithms for mapping data to distributed-memory multiprocessors*. Ph.D. diss., Comp. Sci., Syracuse, N.Y.

Mansour N., and Fox G.C. 1991. A hybrid genetic algorithm for task allocation. In *Proc., Internat. Conf. on Genetic Algorithms* (July), pp. 466-473.

Mansour N., and Fox G.C. 1992. Allocating data to multicomputer nodes by physical optimization algorithms for loosely synchronous computations. *Concurrency: Practice and Experience*, 4(7)557-574.

Mansour N., and Fox G.C. 1994a. Parallel Physical Optimization Algorithms for Allocating Data to Multicomputer Nodes. *The Journal of Supercomputing*, 8:53-80.

Mansour N., and Fox G.C. 1994b. Allocating data to distributed-memory multiprocessor by genetic algorithms. *Concurrency: Practice and Experience*, 6(6)485-504.

Misic J., and Jovanovic Z. 1994. Communication Aspects of the star graph interconnection network. *IEEE trans. on Parallel and Distributed Systems*, vol. 5, no. 7, 678-687.

Rutenbar R.A. 1989. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine.*

Saghi G., Siegel H.J., Gray J.L. 1993. Mapping into three classes of parallel machines: a case study using the cyclic reduction algorithm. *Int. Parallel Processing Symposium.*

Tanese R. 1987. Parallel Genetic Algorithm for a Hypercube. *Int. Conference on Genetic Algorithms.*