Applying simulated annealing as an intelligent genetic mutation operator for finding most probable explanations on Bayesian belief networks

Sahr Attia Afara

ADAPTIVE SIMULATED ANNEALING AS AN INTELLIGENT GENETIC MUTATION OPERATOR FOR FINDING MOST PROBABLE EXPLANATIONS OF BAYESIAN BELIEF NETWORKS

SAHAR ATTIA AFARA

2000

APPLYING SIMULATED
ANNEALING AS AN INTELLIGENT
GENETIC MUTATION
OPERATOR FOR FINDING
MOST PROBABLE
EXPLANATIONS ON BAYESIAN
BELIEF NETWORKS

SAHAR ATTIA AFARA

2000

The American University in Cairo

School of Sciences and Engineering

Applying Simulated Annealing as an Intelligent Genetic Mutation Operator for Finding Most Probable Explanations on Bayesian Belief Networks

A Thesis Submitted to

The Department of Computer Science
in partial fulfillment of the requirements for
the degree of Master of Science

by

Sahar Attia Afara
B.Sc. in Computer Science

under the supervision of Dr. Ashraf Abdelbar

November 1999

The American University in Cairo

Applying Simulated Annealing as an Intelligent Genetic Mutation Operator for Finding Most Probable Explanations on Bayesian Belief Networks

A Thesis submitted by Sahar Attia Afara
to Department of Computer Science

November / 1999

in partial fulfillment of the requirements for
The degree of Master of Science
has been approved by

Dr. Ashraf Abdelbar
Thesis Committee Chair / Adviser
Affiliation

[redacted]

Dr. Mohamed Fahmy Tolba
Thesis Committee Reader / examiner
Affiliation

[redacted]

Dr. Mohamed Mohy Mahmoud
Thesis Committee Reader / examiner
Affiliation

[redacted]

[redacted] 6/1/2000
Department Chair / Date
Program Director

[redacted] 16/1/2000
Dean Date
Acting Dean

ACKNOWLEDGMENT

My most sincere gratitude and thanks go to my supervisor Dr. Ashraf Abdelbar. He has provided me with endless support and guidance throughout this research. The friendly atmosphere which he creates with his students allows deep technical discussions to become very enjoyable.

I wish to thank my employer Mentor Graphics Co. for the support which I received from the library and its technical material. I was very fortunate to have the continuous help of our dedicated librarian Rachel Berrington who made all her efforts to find any missing references.

DEDICATION

I dedicate this work to my parents who have continuously provided me with their support and encouragement.

ABSTRACT

Genetic algorithms have emerged as important methodologies for solving difficult problem such as NP-Complete problems. Inspired by a biological phenomena, genetic algorithms in search and optimization have reached a profound acknowledgment of being capable of providing better results than other search methods.

Combining genetic algorithms with local search heuristics in order to incorporate problem specific intertwines have started to emerge as an attraction point in research. Building hybrid intelligent systems which combine multiple search or optimization techniques provide multiple advantages over using one single technique. Hybrid systems tend to incorporate multiple techniques in order to outweigh the disadvantages of separate technique by the combined advantages of multiple ones.

This work attempts to evaluate the combination of simulated annealing and genetic algorithms. This combination is performed through applying simulated annealing as a part of the genetic mutation operator. The research presents a new genetic algorithm which incorporates the basic genetic operators along with the simulated annealing as part of the genetic engine.

The technique is applied to solving Most Probable Explanation problem on Bayesian belief networks. This problem was selected due to its importance in the field of abductive reasoning which is also known as reasoning under uncertainty. Uncertainty plays an important role in many artificial intelligence applications since it maps to the real world where deterministic actions are of minor importance relative to probabilistic actions.

The investigation of combining simulated annealing with genetic algorithms is mainly experimental. The results of this investigation aim at demonstrating how the combination of these two specific techniques is a homogeneous integration of a technique more suitable for certain problems. It also presents a new avenue for researchers in Bayesian belief networks such that hybrid systems could develop into an important approach for solving the problem of Most Probable Explanation.

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

Often computer scientists are faced with the problem of comparing two or multiple algorithms to see which runs faster or takes less memory. There are two approaches to this task. The first is benchmarking: running the two algorithms on a computer and measuring which is faster (or which uses less memory). Ultimately, this is what matters, but a benchmark can be unsatisfactory because it is so specific. The reason for that is it measures the performance of a particular program written in a particular language running on a particular computer with a particular compiler and particular input data. From the single result that the benchmark provides, it can be difficult to predict how well the algorithm would do on a different compiler, computer or data set.

The second approach relies on mathematical analysis of algorithms, independent of the particular implementation and input. The first step in such an analysis is to abstract over the input, to find some parameters that characterize the size of the input. The second step would be to abstract over the implementation, to find some measure that reflects the running time of the algorithm, but not tied to a particular compiler or computer. If algorithms were simple then analysis would be trivial. But two problems make it more complicated. First, it is rare to find a parameter like n that completely characterizes the number of steps taken by the algorithm, as a function of the size of the input. The best we can do is to compute the worst case or the average case. The second problem is that algorithms tend to resist exact analysis. In such cases, we tend to resort to approximations. This analysis is called asymptotic analysis and is denoted by *O()* notation.

Theory of NP-completeness

Analysis of algorithms and the *O()* notation allow us to talk about the efficiency of a particular algorithm. However, they have nothing to say about whether or not there could be a better algorithm for the problem at hand. The field of complexity analysis analyzes problems rather than algorithms. The first gross division is between problems that can be solved in polynomial time and those that cannot be solved in polynomial time, no matter what algorithm is being used. The class of polynomial problems is called *P*. These are sometimes referred to as the easy problems, because this class contains all problems that can be solved in running times of orders such as $O(\log n)$ and $O(n)$. But it also contains those that are solved in $O(n^{100})$, so the name really should not be taken literally.

Another important class of problems is *NP*, the class of non-deterministic polynomial problems. A problem would belong to this class if an algorithm exists that can guess a solution for it and then verify whether or not the guess is correct in polynomial time. The idea is that you either have an experimentally large number of processors so that you can try all the guesses at once, or you are very lucky and always guess right the first time, then the *NP* problems become *P* problems.

One of the big open questions in computer science is whether the class *NP* is equivalent to the class *P* when one does not have the luxury of an infinite number of processors or omniscient guessing [31]. Most computer scientists are convinced that $P \neq NP$, *NP* problems are inherently hard and only have exponential time algorithms. But this has never been proven. Fig. 1 represents the problem existing in the domain of complexity analysis.

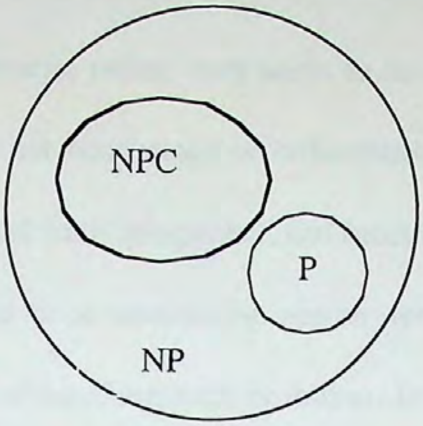

Figure 1: Venn Diagram of P, NP and NPC problems.

Those who are interested in deciding if $P = NP$ look at the subclass of *NP* called *NP*-complete problems. The word complete is used here in the sense of "most extreme" and thus refers to the hardest problems in the class *NP*. It has been proven that either all the *NP*-complete problems are in *P* or none of them is. This makes the class theoretically interesting, but the class is also a practical interest because many important problems are known to be *NP*-complete.

Evolutionary Computation

Evolution is the primary unifying principle of modern biological thought. Classic Darwinian evolutionary theory has become a universally accepted concept. It asserts that the history of the vast majority of life is fully accounted for by only a very few statistical processes acting on and within populations and species. These processes are reproduction, mutation, competition and selection. Evolutionary thought extends beyond the study of life that can be simulated using a computer or other device and put to good engineering purpose. As a consequence, an enormous amount of effort has gone into developing both analytical and numerical optimization techniques. Although there are now many such techniques, large classes of functions still exist which are beyond analytical methods and present significant difficulties for numerical techniques. Unfortunately, these functions are not bizarre, theoretical constructs; rather, they seem to be quite common place and show up as functions which are not continuous or differentiable everywhere, functions which are non-convex, multi-modal (multiple peaks), and functions which contain noise.

As a consequence, there is a continuing search for new and more robust optimization techniques capable of handling such problems. In the past decade we have seen an increasing interest in biologically motivated approaches to solving optimization problems, including neural networks (NNs), genetic algorithms (GAs), and evolutionary strategies (ESs). The initial success of these approaches has lead to a number of improvements in the techniques and a good deal of progress in understanding the kind of functions for which such techniques are more-suited.

There are three broadly similar avenues of investigation in simulated evolution: evolutionary strategies, evolutionary programming and genetic algorithms (with related efforts in genetic programming and classifier systems). When applied for practical problem solving each begins with a population of contending trial solutions brought to the task at hand. New solutions are created by randomly altering the existing solutions. An objective measure of performance is used to assess the fitness or error of each trial solution, and a selection mechanism determining which solutions should be maintained as parents for the subsequent generation. The differences between the procedures are characterized by the types of alterations that are imposed on solutions to create offspring, the methods employed for selecting new parents, and the structures used for representing the solutions. But these differences are minor in comparison to the similarities in approach. Evolutionary computation has become the standard term that encompasses all of these techniques.

The origins of evolutionary computation can be traced back to the late 1950's. Back, Hammel and Schwefel discuss in their work [11] the history of evolutionary computation and its current state. The field remained relatively unknown to the scientific community due to lack of available powerful computer processors at that time. The fundamental work of Holland [46] and L. Fogel [29] served to slowly change this picture during the 1970's. We currently observe a remarkable and steady increase in the number of publications and conferences in this field, a clear demonstration of the scientific as well as economic relevance of this subject matter.

But what are the benefits of evolutionary computation that may justify the effort invested in this area? The main argument in this emerging field is that the major advantage of using evolutionary search lies in the gain of flexibility and adaptability to the task at hand. These attributes are combined with robust performance (although it depends on the problem at hand) and global search characteristics. In fact, evolutionary computation should be understood as a general adaptable concept for solving difficult optimization problems, rather than a collection of related and ready-to-use algorithms.

The majority of current implementations of evolutionary algorithms descend from three strongly related but independently developed approaches that are strongly related which are: GAs, EPs and ESs. Genetic algorithms introduced by Holland [46] and subsequently studied by DeJong [23] and Goldberg [36] and others, have been originally proposed as a general model of adaptive processes, but by far the largest application of these techniques is optimization.

Evolutionary programming introduced by Fogel [52] was originally offered as an attempt to create artificial intelligence. The approach was to evolve finite state machine (FSM) to predict events on the basis of former observations. Any FSM is an abstract machine that transforms a sequence of input symbols into a sequence of output symbols. The transformation depends on a finite set of states and a finite set of transition rules. The performance of the FSM with respect to its environment might then be measured on the basis of the machine's prediction capability, i.e., by comparing each output symbol with the next input symbol and measuring the worth of a prediction by some payoff function.

Evolutionary strategies, introduced by Rechenberg [62] and Schwefel [66] were initially designed with the goal of solving difficult discrete and continuous, mainly experimental, parameter optimization problems.

Motivation

The research motivation behind this thesis work is to develop a hybrid intelligent system which combines the powers of an evolutionary method: genetic algorithms with a physical concept: simulated annealing. The aim of this hybridization is to combine the powers of both techniques and introduce a new technique that can be regarded as a novel approach for solving the problem of Most Probable Explanations (MPEs) on Bayesian Belief Networks. The MPE problem is one of the problems which are known to be NP-hard [5].

Organization

This thesis is organized into three major parts. The first part consists of chapters two and three. These chapters discuss the theoretical background of the search techniques used in development of the new hybrid intelligent technique. Chapter two is dedicated to the survey of genetic algorithms and their theoretical foundations. The chapter also includes a discussion of the attempts that have been performed in order to develop a hybrid genetic algorithm through the combination of local search heuristics.

Similar to chapter two, the third chapter is dedicated for the presentation of the theoretical foundations of simulated annealing. Having its origins from the physical concept of annealing, the third chapter presents a survey of how simulated annealing developed into an appealing optimization technique to solve *NP*-complete problems. Parallel simulated annealing which is a major enhancement to the original algorithm is also presented at the end of chapter three. This enhancement creates a link between the concept of population evolution and simulated annealing.

The second major part of the thesis lies in chapters four and five. The second part is considered the contribution of this thesis to the field of combining genetic algorithms and simulated annealing. Chapter four presents a survey of all the previous attempts to combine genetic algorithms and simulated annealing. These attempts have been collected and compiled into a categorization by the researcher and presented as an original part of this thesis. This chapter is of great importance to the reader since it illustrates the similarities and differences in the approaches found in the literature with respect to combining genetic algorithms and simulated annealing. In chapter five the second part of the contribution is presented which describes the technique that has been developed throughout this research. This hybrid technique is compared with other similar techniques that were introduced earlier in chapter four.

The third part of thesis constitutes of chapters: six, seven and eight. Chapter six represents an overview of the problem selected for experimentation, which is finding Most Probable Explanations on Bayesian belief networks. The problem definition is presented along with previous attempts for solving this problem using genetic algorithms and other heuristic techniques. In chapter seven the experiments that were performed on various sizes of the Bayesian belief networks are presented. After which an analysis of the results obtained for each experiment is presented. Finally, chapter eight concludes this thesis and presents some of the future directions for the continuation of this work.

CHAPTER II

GENETIC ALGORITHMS

General Overview

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. The initiator of research in genetic algorithms was John Holland. In the year 1975, he published the book *Adaptation in Natural and Artificial Systems* [46] which laid the basic principles and foundations of GAs. From there on some dissertations and papers began to be published by different researchers. Two fomal conferences on Genetic algorithms: The International Conference on Genetic Algorithms and their Applications, which is being held every two years since the year 1985, and the theoretical conference of Foundations of Genetic Algorithms and Classifier Systems. Both conferences have played a great role in the formulation of the field of genetic algorithms.

Genetic algorithms combine "survival of the fittest" among string structures with a structure yet randomize information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest individuals from the old population. Though randomized, genetic algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance.

This chapter focuses on the fundamentals of genetic algorithms, their definition, history and theories. In order to prepare readers for understanding the effectiveness of GAs in the world of *NP*-complete problems, advanced techniques for making GAs perform better are also presented. Enhancing the basic algorithm has lead to a high reputation for GAs as being an efficient and powerful optimization and search technique for difficult problems such as *NP*-complete problems, such as in [23], [27] and [30].

Definition

How are genetic algorithms different from other optimization and search methods? In his book [36], Goldberg defines what makes them different, these are four points :

- Genetic algorithms work with a coding of the parameter set, not the parameters themselves.
- Genetic algorithms search from a population of points, not the parameters themselves.
- Genetic algorithms use payoff (objective function) information, not derivations or other auxiliary knowledge.
- Genetic algorithms use probabilistic transition rules, not deterministic rules.

Outline of Genetic Algorithms

Genetic algorithms use a direct analogy of natural behavior. They work with a population of individuals, each representing a possible solution to a given problem. Each individual is assigned a "fitness score" according to how good a solution to the problem it is. The highly fit individuals are given opportunities to "reproduce" by "cross-breeding" with other individuals in the population. This produces new individuals as "offspring", which share some features taken from each "parent". The least fit members of the population are less likely to get selected for reproduction, and so they "die out". The outline of traditional genetic algorithms is as follows:

1. Initialize and encode a random population of chromosomes.
2. Decode and evaluate each chromosome's fitness in the population.

3. Reproduce a new generation by stochastically selecting current chromosomes as parents according to fitness to generate new children.
4. Apply crossover and mutation operators to the new chromosomes.
5. Repeat 2–4 until an adequate solution is found.

Different researchers may implement the algorithm in slightly different ways but the outline is almost the same. The difference between genetic algorithms and other search algorithms is that they separate domain-specific knowledge from domain independent knowledge. Unlike other algorithms, with GAs only the encoding, decoding method and evaluation function are domain dependent, whereas the search technique itself (crossover and mutation operators) is domain independent.

Basic Principles

Initialization and Coding

When using genetic algorithms, any potential solution to a problem should be represented as a set of parameters. These parameters, known as genes, are joined together to form a string of values. This string of values is referred to as a chromosome. In genetic terms, the set of parameters represented by a particular chromosome is referred to as a genotype. The genotype contains the information required to construct an organism; which is referred to as the phenotype. The phenotype is defined as the external appearance of the organism. The fitness of an individual depends on the performance of the phenotype. This can be inferred from the genotype i.e.: it can be computed from the chromosome, using the fitness function.

There are many ways to initialize and encode the initial generation. Binary vs. non-binary or fixed vs. variable length strings can be used. The traditional Holland's [46] encoding method is as a binary-fixed length string. At the initial stage of the evolution cycle, the genetic algorithm just randomly generates bits of chromosomes and encodes all the chromosomes as a population, then decodes and evaluates the chromosomes for use in reproduction and selection of the first generation.

Fitness Function

A fitness function must be devised for each problem to be solved. Given a particular chromosome, the fitness function returns a single numerical "fitness" or "figure of merit", which should be proportional to the "utility" or "ability" of the individual which that chromosome represents. For many problems, particularly function optimization, it is obvious what the fitness function should measure; it should just be the value of the function. But this is not always the case, for example with combinatorial optimization problems.

Reproduction

During the reproduction phase of the GA, individuals are selected from the population and recombined, producing offspring that will comprise the next generation. Parents are selected randomly from the population using a scheme that favors the more fit individuals. Good individuals could probably be selected several times in a generation, while poor ones may not be at all. One of the most common methods for parent selection is the Roulette Wheel Parent Selection mechanism defined in [36]. This mechanism is defined by selecting parents such that the probability of an individual being selected is directly proportional to its fitness. Although this selection procedure is based on a random mechanism, it does not imply that genetic algorithms are a directionless search methodology.

Having selected two parents, their chromosomes are recombined, typically using the mechanisms of crossover and mutation. The crossover mechanism is defined as the process yielding recombination of bit strings via an exchange of segments between pairs of chromosomes. Crossover takes two individuals, and cuts their chromosome strings at some randomly chosen position to produce two "head" segments, and two "tail" segments. The tail segments are then swapped over to produce two new full length chromosomes. Each one of the two offspring inherits some genes from each parent. This technique is known as single or one point crossover and is demonstrated in Fig. 2.

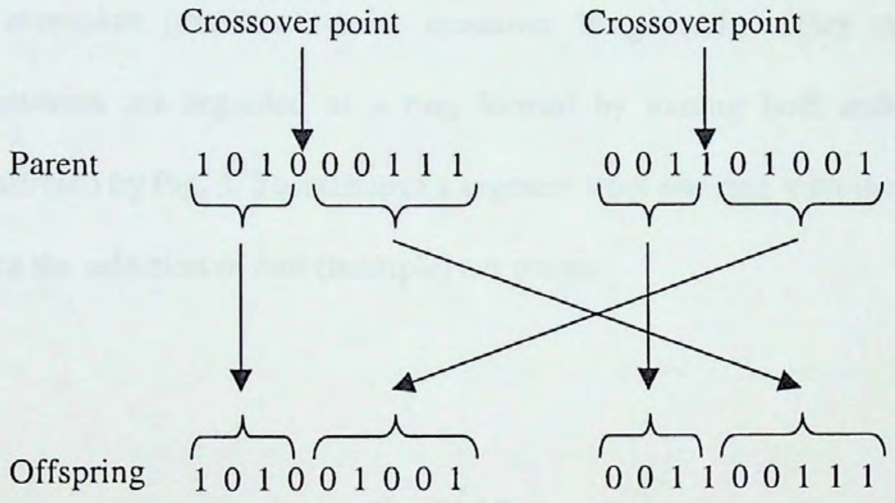

Figure 2: Schematic of One Point Crossover

This mechanism is not applied to all pairs of individuals selected for mating. A random choice is made, where the likelihood of crossover being applied is typically between 0.6 and 1.0. If crossover is not applied, duplicating the parents produces offspring. This gives each individual a chance of passing on its genes without the disruption of crossover.

Although the traditional one-point crossover has been widely used, many advanced crossover algorithms have been devised often involving more than one cut point. DeJong [23] investigated the effectiveness of multiple-point crossover. In two-point crossover (and multi-point crossover in general), rather than linear strings, chromosomes are regarded as a ring formed by joining both ends of the string as demonstrated by Fig. 3. To exchange a segment from one ring with that from another ring requires the selection of two (multiple) cut points.

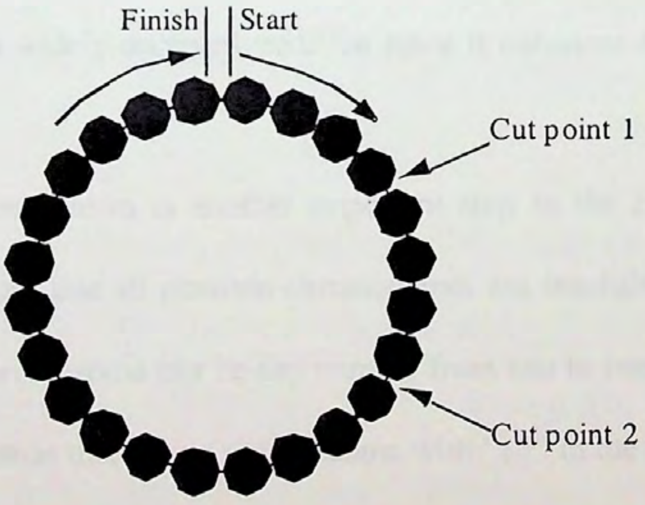

Figure 3: Schematic of Two Point Crossover

General acceptance for the two-point crossover is due to the fact that a chromosome considered as a ring will contain more building blocks -since they are able to "wrap around" at the end of the string. It was concluded in this research that the two-point crossover gives an improvement over the traditional one-point crossover. It was also determined that adding further crossover points reduces the performance of the GA. The problem with adding additional crossover points is that building blocks are more likely to be disrupted. Goldberg in [36] described a rather different crossover operator called the Partially Matched Crossover (PMX). He advocated the use of such an operator for order-based problems such as the Travelling Salesman Problem (TSP), since in order-based problems the gene values are fixed and the fitness depends on the order which they appear. In PMX it is not the values of the genes that are crossed, but the order in which they appear. Offspring have genes that inherit ordering information from each parent. This avoids the generation of offspring that violate problem constraints. Using domain specific knowledge, as Grefenstette shows in [39], in crossover or any of the genetic operators has become a widely accepted tradition since it enhances the performance of GAs.

The mutation mechanism is another important step in the reproduction phase. Mutation is used to insure that all possible chromosomes are reachable. For example, if the first position in a chromosome can be any number from one to twenty, it may happen that in the initial population there is no chromosome with "15" in the first position. With crossover only, it is impossible to generate such a chromosome. This ensures that the selection process does not get caught in a local minimum because it is possible that the use of crossover operator will only produce a set of bit strings that re better than all local neighbors but not optimal in a global sense. This can happen since crossover may not be able to produce undiscovered bits. The mutation operator can overcome this by simply randomly selecting any bit position in a string and changing it. Thus mutation is applied to each child individually after crossover. It randomly alters each gene with a very small probability (typically 0.0001). The simplest form of mutation is the flip mutation, demonstrated in Fig. 4, selects a point randomly and flips its value using a specific probability ratio. Mutation is seen as an operator responsible for re-introducing inadvertently "lost" gene values, preventing genetic drift, and providing an element of search in the vicinity of the population when it has largely converged.

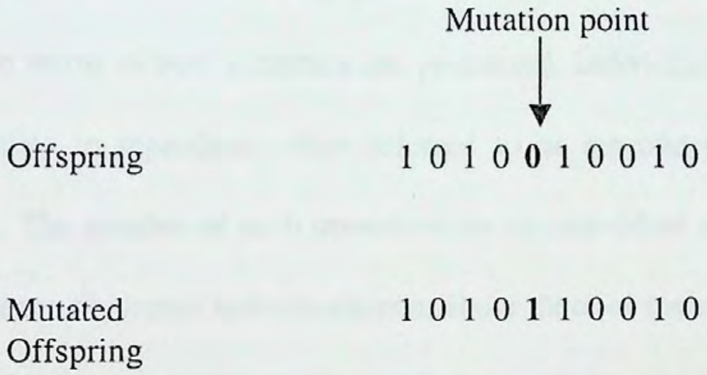

Figure 4: Schematic of Mutation Operator

Convergence

If the GA has been correctly implemented, the population will evolve over successive generations so that the fitness of the best and average individual in each generation increases towards the global optimum. Convergence is defined as the progression towards increasing uniformity. A gene is said to have converged when 95% of the population share the same value [23]. The population is said to have converged when all of the genes have converged.

Fundamental Theorem of GAs

In this section we describe the fundamental theorem of genetic algorithms, known as the Schema Theorem. The mathematical analysis of this theory provides a proof of why genetic algorithms are capable of finding near-optimal solutions that are better than those found by other techniques. The theoretical foundations of GAs exist in the Schemata Theroem and the Building Block Hypothesis, developed by David Goldberg and John Holland [36] & [46].

Schemata and the Schema Theorem

A schema as defined in [46] is a similarity template describing a subset of strings with similarities at certain string positions. The Schema theorem explains the power of the GA in terms of how schemata are processed. Individuals in the population are given opportunities to reproduce, often referred to as reproductive trials, and produce new offspring. The number of such opportunities an individual receives is in proportion to its fitness, hence the better individuals contribute more of their genes to the next generation. It is assumed that an individual's high fitness is due to the fact that it contains good schemata. By passing on more of these good schemata to the next generation, we increase the likelihood of finding even better solutions.

A schema is a string of the following format: # 1 0 1 # 0, where each position is either a 1, 0 or #[1]. The more # it has, the less specific it becomes. In a binary string, if there are n '#', then a schema can describe 2^n strings. In the above example, there are 2 '#', therefore it can describe four strings: 0 1 0 1 0 0, 0 1 0 1 1 0, 1 1 0 1 0 0 and 1 1 0 1 1 0.

[1] Define: the # is a don't care symbol, can represent a 1 or 0.

The total number of different schemata of length *n* is $(2+1)^n$, because each of the *n* positions can be 1, 0 or #. The total search space remains 2^n since the '#' symbols are not involved in the real string processing; they only serve as a tool to describe similarity templates. How many schemata are processed usefully in each generation? Despite the processing of only *c* chromosomes each generation, where *c* is the number of chromosomes in a generation, GAs process approximately c^3 schema. Holland has defined this as *implicit parallelism*. This observation is a major part of the explanation of the robustness of GAs. Even though we perform computation proportional to the size of the population, we get useful processing of something like c^3 schema in parallel with no special book keeping or memory other than the population itself.

The frequency $m(H, t)$ of a schema *H* at generation *t* will change at generation $t+1$ proportionally to the selection probability of strings for reproduction. There are two steps to consider. The first step is the selection step, in which we choose from the generation at time *t* the chromosomes that will be the parents for the generation at time $t+1$. Let $m^P(H, t)$ represent the frequency of schema *H* among the parents selected to produce generation $t+1$. In the variation step, which we will consider later, we crossover the parents with each other to produce the new generation $t+1$.

Let $f(H)$ be the average fitness of a string containing schema *H* in generation *t* and *F* represent the average fitness of the entire population. We can derive the following formula in the following equation:

$$m^P(H, t) = m(H, t) * f(H)/F \qquad (1)$$

This simply states that the number of instances of schema *H* which will be parents of the generation $t+1$ is equal to the number of instances of schema *H* in the generation t, multiplied by how much more fit schema *H* is than the average fitness. The equal sign does not accurately relate both sides of the equation, but since the selection step is a stochastic process we can say that the equals sign becomes more and more true as the size of the population gets larger. It is clear from this simple analysis that good schema, or building blocks become more and more frequent in later generations. In a similar manner, bad schema die away. Implicitly, a genetic algorithm processes about c^3 schemata of this sort all the time.

We must now look at the effect of the variation step, represented by the crossover and mutation operators, on the Schemata theorem. Without the variation step, we would only get an algorithm that continues to work, according to Equation (1), until all of the chromosomes are the same, which is the one that happened to be the best one in the first generation. Without crossover and mutation, we can never get new chromosomes. However, do these operators badly affect the property of the selection step? In other words, the selection step ensures that schemata with above average fitness increase in number. The crossover operator is necessary in order to have new possibly good schemata, and to combine together previous good schemata. However, does the variation step destroy too many good schemata, so that the effect of Equation (1) is decreased?

During crossover, when a schema *H* is selected to mate with another schema of the same category, this schema is preserved. When this happens, no disruption in the schema occurs and thus it will have no counter effect on the convergence of the population. We will therefore just consider how often it might happen that a schema *H* is mated with a schema different than H. This means that H is destroyed when the crossover position cuts the schema at an inappropriate position (e.g.: if 1 1 1 1 1 1 is crossed over with 0 0 0 0 0 0 at position 3, then the result is the chromosomes: 1 1 1 0 0 0 and 0 0 0 1 1 1. But if 11#### is a good schema, then we can see that this schema is not destroyed. However, the schema 1111## is destroyed, because the crossover point cuts into the schema). So we must find the probability that an instance of schema H will be crossed over with a chromosome that will destroy the schema. This is the combination of the probability of crossover, the probability of another chromosome not having an instance of schema H, and the probability that the crossover position cuts schema H.

If P_c is the probability of a crossover, and the defining length $\delta(H)$ is the distance between the first and the last specific schema positions of H, and l is the total length of a string then the probability of a schema being disrupted because of crossover is the following:

$$P_c * \delta(H)/(l-1) \qquad (2)$$

The probability that an instance of schema H is crossed with a schema different from H is described by the following equation:

$$1 - m^P(H, t)/c \qquad (3)$$

where c is the number of chromosomes in the population. If we multiply the probabilities from the last two equations, we get the probability that schema H will be destroyed by the crossover operation, which is given as:

$$(1 - m^P (H, t)/c) * P_c * \delta(H)/(l - 1) \qquad (4)$$

This is, however, usually a small number. E.g.: P_c will be typically about 0.7, while, with a good schema H, the number in the population which are not in this schema may be about 0.4, and the probability of cutting the schema will be typically about, say 0. So the number in Equation (4) might be usually a small number less than 0.1. Let's call this number ε. We can now show another equation that gives the number of instances of schema H that we expect in the new population:

$$m(H, t + 1) \geq (1 - \varepsilon) * m(H, t) * f(H)/F \qquad (5)$$

The greater-than-or-equal sign is there because we actually overestimate the amount of schemas H that will be destroyed. This is because it is still possible to produce instances of schema H by crossover of schema H with a chromosome, which has no schema H even if the crossover position cuts the schema. Also, it is possible to produce instances of schema H even if not one of the parent chromosomes contains schema H. As we can see, since $(1 - \varepsilon)$ is near to 1, the variation step does not harm the selection step too much.

In an analogous manner, we need to analyze the effect of mutation operator over the selection step. The mutation rate is very small in normal situations. If the probability of a mutation is P_m and the order of a schema H, denoted by $O(H)$, is simply the number of fixed positions present in the template, then the probability of a schema being disrupted because of mutation is:

$$O(H) * P_m \tag{6}$$

If we have two schemata, such that:

H1: # # 1 # 0 #

H2: 0 1 # 1 # 0

In the above example, $O(H1)$ is 2 and $O(H2)$ is 4. If we assume that the probability of mutation is $P_m = 0.001$, then there is 0.002 probability of disruption of schema *H1* and 0.004 for disruption of schema *H2*. Therefore we know that short order schemata are more likely to survive than long order ones. Now, we need to modify our schema formula

$$m(H, t + 1) \geq (1 - \varepsilon) * (1 - O(H) * P_m) * m(H, t) * f(H)/ F \tag{7}$$

So the effect of the selection step is only affected by the amount $(1 - \varepsilon)*(1 - O(H) * P_m)$ which is an overestimate of how much crossover and mutation destroy instances of schema H. We can conclude that highly fit, short defining length, lower order schemata are strongly propagated from generation to generation. This propagation of good schemata is driven by the stochastic selection according to high fitness. Moreover, the above equation means that we can crossover and mutate and produce new and possibly much better schemata with only small change to this effect (because the amount of destruction of good schema is usually very small).

The Schema Theorem indicates some necessary conditions for utilizing the implicit parallelism. For fully using the schema information, disruption of the above average fitness schemata should be minimized. Highly fit schemata of low defining length and low order are so important in the field of GAs that they have a special term: Building Blocks. Genetic algorithms seek near optimal performance through the evolution of short, low-order, high-performance schemata or building blocks.

Building Block Hypothesis

According to Goldberg [36], the power of GAs lies in it being able to find good building blocks. These are schemata of short defining length consisting of bits which work well together, and tend to lead to improved performance when incorporated into an individual. A successful coding scheme is one which encourages the formation of building blocks by insuring that:

1. Related genes are close together on the chromosome, while
2. There is little interaction between genes.

If the above rules are observed, then a GA will be as effective as predicted by the Schema Theorem. Unfortunately, conditions (1) and (2) are not always easy to meet. Genes may be related in ways that do not allow all closely related ones to be placed close together in one-dimensional string. In many cases, the exact nature of the relationship between the genes may not be known to the programmer, so even if there are only simple relationships, it may still be impossible to arrange the coding to reflect this.

Condition (2) is a precondition for (1). If the contribution to overall fitness of each gene were independent of all other genes, then it would be possible to solve the problem by hillclimbing on each gene in turn. If we can ensure that each gene only interacts with a small number of other genes and these can be placed together on the chromosome, the conditions (1) and (2) can be met. But if there is a lot of interaction between genes, then neither condition can be met. Clearly we should try to design coding schemes which fit the recommendations, since this will ensure that the GA will work as well as possible.

Interaction often referred to as epistasis between genes means that the contribution of a gene to the fitness depends on the value of other genes in the chromosome. In fact there is always some interaction between genes in multimodal fitness functions. This is significant because multimodal functions are the only sort of any real interest in GA research, since unimodal functions can be solved easily using simpler methods.

Exploration and Exploitation

Any efficient optimization algorithm must use two techniques to find a global maximum: exploration and exploitation. Exploration is used to investigate new and unknown areas in the search space, while exploitation is needed to make use of knowledge found at points previously visited to help find better points. These two requirements are contradictory, and a good search algorithm must find a tradeoff between the two.

A purely random search is good at exploration, but does no exploitation, while a purely hillclimbing method is good at exploitation but does little exploration. Combinations of these two strategies can be quite effective, but it is difficult to know where the best balance lies (i.e. how much exploitation should be performed before giving up and exploring further?).

Goldberg [36] and Holland [46] showed that a genetic algorithm combines both exploration and exploitation at the same time in an optimal way. However, although this may be theoretically true for a GA, there are inevitably problems in practice. These arise because Holland made certain simplifying assumptions, including:

1. The population size is infinite
2. The fitness function accurately reflects the utility of a solution, and
3. The genes in a chromosome do not interact significantly.

Assumption (1) can never be satisfied in practice. Because of this the performance of a GA will always be subject to stochastic errors. One such problem, which is also found in nature, is that of genetic drift.

Even in the absence of any selection pressure (i.e.: a constant fitness function), members of the population will still converge to some point in the solution space. This happens simply because of the accumulation of stochastic errors. If, by chance, a gene becomes predominant in the population, then it is just as likely to become more predominant in the next generation as it is to become less predominant. If an increase in predominance is sustained over several successive generations, and the population is finite, then a gene can spread to all members of the population. Once a gene has converged in this way, it is fixed; crossover cannot introduce new gene values. This produces a ratchet effect, so that as generations go by, each gene eventually becomes fixed.

The rate of genetic drift therefore provides a lower bound on the rate at which a GA can converge towards the correct solution. That is, if the GA is to exploit gradient information in the fitness function, the fitness function must provide a slope sufficiently large to counteract any genetic drift. The rate of genetic drift can be reduced by increasing the mutation rate. However, if the mutation rate is too high, the search becomes effectively random, so once again gradient information in the fitness function is not exploited. Assumptions (2) and (3) can be satisfied for well behaved laboratory test functions, but are more difficult to satisfy for real word problems.

Hybrid Genetic Algorithms

Genetic algorithms are optimization techniques that work well on a broad range of problems. They search the most promising areas of the space irrespective of local optima and without any specific knowledge about the problem other than the objective function. This is a very good characteristic of GAs and it is one of the reasons why they are so elegant, so robust and so easy to apply.

GAs do not require specific knowledge about the problem other than the objective function, thus it becomes very easy to hybridize them with other techniques. Typically the GA would find the most promising areas of search space and the more specialized scheme would polish off the solution to obtain an extra improvement. Another approach for performing hybrid GAs that has quite prevailed is to seed the initial population with a certain percentage of good solutions obtained by a heuristic method. This way the GA will have good building blocks right from the beginning. Incorporating knowledge about the problem in the genetic operators themselves has also been one of the attractive approaches to enhance genetic algorithms [39].

The objective of this section is to present the most relevant research on hybrid GAs. By far, the most important work in any field is usually the theoretical work that lays down its foundations. On the contrary, the field of hybrid GAs has not yet matured with respect to its theoretical foundations. Rather, most of the work that has been done focuses on creating hybrid GAs that are tailored to solve specific optimization problems. The field of hybrid GAs lacks a general framework for developing them similar to those existing in pure GAs.

Little work in the current literature has been targeted towards developing a micro-level theory for hybrid GAs. By micro-level we mean that practitioners focus on designing an optimal framework for the problem solved using the specific global and local methods. Developing such frameworks are still in their early stage of development and have not yet taken a role in the practical usage of hybird GAs. The rest of this section will start by discussing the existing attempts for developing micro-level theories for hybrid GAs. Following that will be a presentation of the practical methods that have been introduced in the literature for solving combinatorial optimization problems.

Micro-Level Theory

Theory of Hybrids

Little theory has been developed on how to design hybrid GAs and most of the work in this area is done empirically to tune the specific problem that is being solved. Theoretical work in the area of hybrid GA theory has focused mainly on combining the global search power of the GA with the local search power of a hillclimber. Two interesting lines of work have been studied: the Lamarkian Evolution and the Baldwin Effect, such as in the early works [8] and [45] followed by [72].

The Lamarkian evolution as defined in [72] forms a hybrid genetic algorithm by using the local search to improve the initial population as well as the strings produced by genetic recombination. This type of hybrid GAs creates an initial population that is described in genetic terms as "mature". In a traditional GA, the initial population is usually created randomly. In the Lamarkian evolution, the initial population is created by a local search heuristic that has a good reputation for reaching near-optimal results for the problem being solved. This local search heuristic is also used after each genetic recombination step in order to improve the resulting population. The resulting improvements are then coded onto the strings possessed by the genetic algorithm. Local search in this context can be thought of as being analogous to a kind of learning that occurs during the lifetime of an individual string.

On the other hand, the Baldwin effect as defined in [8] and [45] is a way of learning that allows a change in the individual's fitness without alterions in its genetic coding. In hybrid GAs that are based on the Baldwinian effect, local search heuristics are used in the same way as in the Lamarkian evolution. The major difference is that the learning phase enhances the fitness of the matured individual, thus introducing a modification in the fitness landscape of the problem.

Both approaches for hybrid GAs, use the metaphor that an individual learns (hillclimbs) during its lifetime (generation). In the Lamarakian case, the resulting individual, after hillclimbing is put back into the population. In the Baldwinian case only the fitness is changed and the genotype remains unchanged. The drawback of the Lamarkian approach is loss of diversity. On the contrary, the Baldwinian strategy maintains the diversity in the population and can be very useful in a search space that does not have nice hills.

In the work of Darrell Whitley e.t. al. [72], a comparison of the two forms of hybrid genetic algorithms is presented. It showed that advantages and disadvantages of each methodology and how one can outperform the other when combined with the simple genetic algorithm. The Lamarkian strategies have the advantage of being an extremely fast form of search. Their disadvantage lies in the fact that they are similar to pure GAs that tend to converge to locally optimum points in the solution space. The advantages of using the Baldiwian approach is highly advocated in Darrell Whitley's comparison. The work illustrates how local search can alter the fitness landscape. They show that taking *N* steps deepens the basin of attraction, thus making the function flatter around the local optimum. If local search is done to convergence, then the fitness function becomes flat in each basin of attraction. However, each basin of attraction has a potentially different evaluation corresponding to the evaluation of the local optimum. Changing the fitness landscape in the way introduced by the Baldwinian effect, creates the potential for impacting genetic search by increasing the likelihood of allocating more samples in certain basins of attractions.

Warehouse Scheduling with Hybrid GAs

In a work done by Whitley e.t al. in 1996 [71], the warehouse scheduling problem was selected for experimentation by 5 heuristic techniques. These techniques included: genetic algorithms, local search operators, heuristic rules, systematic search and hybrid approaches. The research used for the stochastic search: the genetic algorithm and two local search operators. It used as a heuristic rule the Limited Discrepancy Search (LDS) [43], which is also considered a systematic AI search method. For the hybrid approaches, the LDS heuristic and the stochastic methods were combined, such that the LDS methods was used to initialize the stochastic methods. The warehouse scheduling problem is defined as the problem of sequencing requests for products so as to minimize the average time required to fill an order and the amount of product left in the inventory. The problem is complex due to several factors. First, the search space is large: all possible sequences of all orders. Several hundred possible orders are considered for each schedule. Optimization involves several distinct performance measures, which can be inversely related. Evaluating a schedule is costly; a simulation is used to determine when enough of the needed product is available and how long it takes to move the product from its starting location to its transport. The aim of this research was to assess the trade-off between search and heuristic methods.

The techniques used provided: a globally motivated search through GAs, an improvement update through local search techniques and domain knowledge through the LDS heuristic technique. The LDS has been shown to perform well on benchmark scheduling test problems [43]. The researchers implemented their own sequencing heuristic that they used in two different contexts. The first context was a direct greedy construction of the schedule. The second was using it as the heuristic in the systematic AI search method called Limited Discrepancy Search.

The conclusion of this work was that methods combining stochastic search with heuristic initialization significantly outperform heuristic-based methods, given equal amount of computation time. Moreover, the globally motivated search (i.e. genetic algorithm) outperforms the local search methods. The involvement of heuristics as an initialization for the GA outperforms all other methods applied.

Genetic Hybrids for QAP

The Quadratic Assignment Problem (QAP) is a well-known *NP*-complete problem. It is defined as: $min_{\phi \in P(n)} \sum \sum a_{ij} b_{\phi(i)\,\phi(j)}$ where $A = (a_{ij})$ and $B = (b_{kl})$ are two n x n matrices and $P(n)$ is the set of all permutations of $\{1...n\}$. Matrix A is often referred to as a distance matrix between sites. The second matrix *B*, is defined as a flow matrix between objects. In most cases, matrices *A* and *B* are symmetrical with a null diagonal. A permutation may then be interpreted as an assignment of objects to sites with a quadratic cost associated to it. There are many applications that can be formulated as QAP. It has been particularly difficult to use exact methods to solve even relatively small instances of this problem.

In a research by Fleurent & Ferland in 1993 [27], a new hybrid procedure that combines genetic operators to existing heuristics was proposed to solve the QAP. Traditionally local search methods and tabu search were the popular methods for solving QAP [68]. In this research, the genetic algorithm was introduced into the local search methods. The main idea was to overcome the well-known drawback of local search methods, which is stopping at the first local minimum it reaches. This drawback was overcome by allowing the local search procedure to use new starting points. The research introduced a multi-start approach that can be improved by the genetic operators. The hybrid scheme consists of working with a population *P* of solutions that are a production of previous heuristic procedures.

In the GA proposed, there is no mutation operator since each individual is generated with a heuristic procedure. Alternatively, the heuristic used to generate the initial population and to enhance the individual created through crossover, was regarded as the mutation operator applied at a 100% rate. In this work the researchers' aim was to go beyond the natural analogy in which parents only pass their chromosomal material to their children. Indeed, they allow parents to improve on their genetic materials first and then use their enhanced chromosomes for reproduction, something that is only possible in nature with very rare and small mutations. Their experiments showed that the genetic hybrids outperform the local search method and the tabu search method.

Genetic Hybrids for TSP

The travelling salesman problem (TSP) is one of the most attractive *NP*-complete problems. The TSP has become a standard testbed for combinatorial optimization methods that attempt to find near optimum solutions to this *NP*-hard problem [31].

In 1996, Friesleben and Merz [30], introduced a genetic local search algorithm for solving symmetric and asymmetric travelling salesman problem. The symmetric TSP is one in which the distance between cities A and B is equal to the distance between cities B and A, while the asymmetric is one in which the previous statement is not true. The main idea was to allow the local search algorithm to find the local optima in a given TSP search space and then the genetic algorithms are used to search the space of local optima in order to find the local optimum. This idea was formulated by using a nearest-neighbor tour construction heuristic for creating the initial population of the GA. After that a local search algorithm is applied to all individuals of the initial population for producing a population of local optima. Finally, the GA operates on the search space of local optima to determine the global optimum point.

This general approach was applied to both the symmetric and asymmetric TSP, but the GA operators are adapted to the individual characteristics of each of the two problem classes. A new crossover operator was developed for enabling the GA to perform particular "jumps" within the search space of local minima. Its design is based on the observation that the solution space of a typical TSP has a "global convex" or "big valley" character, such that the optimal solution can be found near the center of a single valley of near-optimum solutions with approximately equal distance to each other. There was also a novel mutation operator that was developed for this experiment. This mutation operator worked by performing random jumps within the neighborhood of a local optimum.

The STSP-GA used symmetric TSPs from the TSPLIB [63] with cities of size 51, 100, 198, 532, 783 and 1577. The first step in the algorithm is to create an initial population of tours using the nearest neighbor heuristic. Each individual in that population is then exposed to Lin-Kernighan [52] tour improvement heuristic. The result of this step is a population of individuals that are locally optimal solutions. The GA starts operating by selecting two individuals randomly to be the parents of crossover operation. The crossover operator created for this problem was called distance preserving crossover (DPX). Since the offspring produced but the DPX operator are usually not local optimal, the LK heuristic is then applied to each offspring in the next step of the algorithm to again obtain a local optimal tour. The mutation operator selected is very similar to the logic of the crossover operator. A problem specific operator was developed called non-sequential-4-change. The mutation was followed by a LK heuristic to transform the offspring produced into a local minimum.

The STSP-GA found the optimal solution for all cases except the last that doesn't have a known optimal solution. The algorithm demonstrated its capability of producing a high quality solution for all TSP problems investigated. For all the instances used, the optimal solution was found within a relatively small number of generations.

On the other hand, the ATSP-GA used TSP problems for cities of size 43, 48, 70, 124 and 170. The optimal solution for all the TSP asymmetric instances were found. The ATSP is more difficult than the STSP and so needed modifications on the genetic operators created previously as well as using a heuristic approach other than the LK. The problem with LK approach is that it involves the 2-Opt method that introduces directional changes of tour fragments. In the case of ATSP, this might alter the tour length unpredictably. The selected heuristic for ATSP was the 3-Opt approach that takes a tour fragment and reinserts it at another position without reversing the order in which cities are visited. The LK heuristic used in all previous discussions of genetic operators as well as finding the initial local minimum population is replaced with the fast-3-Opt heuristic. Unlike the STSP, where the computation time increases with increasing problem size, the ATSP problem size is not the main factor behind computation time. The results showed that the 70 cities instance involved the highest computational time among all ATSP instances.

Parallel Genetic Algorithms

Genetic algorithms have been applied successfully to various problems in domains such as engineering, business and science as described in Goldberg's book [36]. As they are used for solving larger and harder problems, we find that an increase in the time required for finding adequate solutions also increases. Although GAs themselves possess the attribute of being implicitly parallel, there have been multiple efforts to make them faster. One of the major trends that have recently occupied researchers of this field is introducing explicit parallelism to the basic genetic algorithm.

The objective of this section is to present the most relevant work existing in the domain of parallel genetic algorithms. In order to facilitate the understanding of the parallelization techniques that have been introduced in the domain of GAs, the sub-sections will present a categorization of the parallel GAs that currently exist.

The basic idea behind many parallel programs is to introduce a division of task into parts that can be solved simultaneously using multiple processors. This is known as the divide-and-conquer approach and has lead to different methods of parallel GAs. Some methods change the behavior of the algorithm, others exploit the massively parallel computer architectures and others may render themselves to multicomputers with fewer and more powerful processing elements.

Global Parallelization

The first approach used in parallel GAs is called global parallelization. In this type of parallel GAs there is only one population as in the serial GA, but the evaluation of individuals and genetic operators are parallelized explicitly. Since there is one population, selection considers all the individuals and every individual has a chance to mate with any other. This means that the behavior remains unchanged from the basic GA. The evaluation of individuals is the most common operation that is parallelized. Evaluating the fitness of an individual is not dependent on any data communication between individuals of the population and thus renders itself as one of the operators suitable for parallelization. The most common implementation of the parallelization of evaluation functions is by assigning each processor a subset of the population to evaluate. Communication only occurs when each processor receives a new subset for evaluation and when it returns back the evaluated subset to the global processor. The global parallelization model does not assume anything about the underlying computer architecture, and it can be implemented efficiently on shared-memory and distributed-memory architectures. On shared-memory multiprocessors, the population is stored in the shared memory and each processor reads the individual assigned to it and writes the evaluation results back. Using the shared-memory approach, no conflicts can arise among processors in accessing the same memory locations and thus no synchronization is required in this part. On a distributed memory, the population is stored in one processor. This "master" processor is responsible for sending the individuals to the other processors, the "slaves", for evaluation, collecting the results and applying the genetic operators to produce the next generation. The schematic in Fig. 5 shows the master/slave relation between processors in global parallelization. In this approach the number of slaves is quite significant in the cost of communication. As the number of slaves increase, the communication cost increases too. Therefore a tradeoff must be made between computation cost and communication cost. Research by Cantu-Paz [15] has showed that there exists an optimal number of slaves which minimizes the execution time of global parallel GAs.

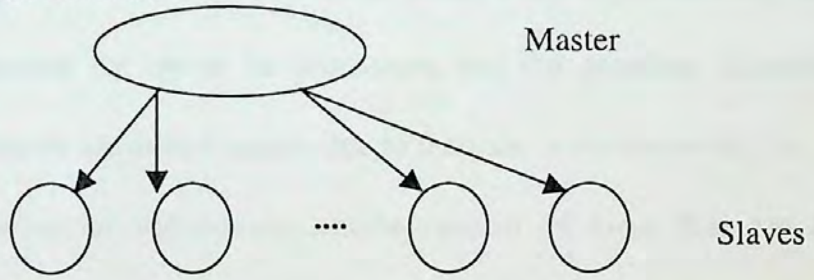

Figure 5: Schematic of a Global Parallel GA

One of the early researches done in the global parallelization domain is that by Fogarty and Huang [28] in which they analyzed the effect of different topologies on the performance. Their work's goal was to evolve a set of rules for a pole balancing application that takes a considerable large amount of time to evolve. They used a network of transputers that are microprocessors designed specifically for parallel computations. A transputer in their design, can connect directly to four transputers. When one of the transputers needs to communicate with another one all the intermediate nodes must receive and retransmit the message. Such a limitation caused an overhead in communications and in order to try to minimize the effect on the performance, they connected the transputers in different topologies. As a result of their analysis they found that the configuration of the network did not make a significant difference. They obtained an increase in speedup and identified the fast-growing communications overhead as an impediment for further improvements in speed.

Following that research in 1992, Abramson and Abela [7] implemented a global GA on a shared memory computer of 16 processors to find efficient timetables for schools. Unfortunately, they reported limited speedup and explained that a few parts of serial code on the critical path caused this seen limitation. Later in 1993, Abramson et. al.[6], added a distributed memory machine with 128 processors . They also changed the application to train timetables and modified the serial code. Their report showed that they found significant speedup for up to 16 processors, but the speedups degraded significantly as more processors are added mainly due to increase in communications.

Besides the evaluation of individuals, another aspect of GAs that can be paralleized is the way genetic operators are applied. Crossover and mutation could be parallelized using the same idea of partitioning the population and distributing work among multiple processors. However, these operators do not represent a high execution cost and thus sending individuals back and forth can cause time to overweigh performance gains. The communication overhead is also a problem when selection is parallelized because several forms of selection need information about the entire population and thus require some communications. In a recent study by Branke et. al. [13], they parallelized different types of selection operator on a 2-D grid of processors.

Their results showed that their algorithms were optimal for the topology of the interconnection network of the computer that they used.

Multi-Population Genetic Algorithms

In the second approach of parallel GAs, a fundamental change in the operation of the genetic algorithms is introduced. In this category of PGAs, the population is divided into multiple sub-populations called demes. Each deme is isolated from the other demes during evolution, but exchanges individuals occasionally. This exchange of individuals is called migration and is controlled by several parameters. Multi-population GAs are also known as Coarse-grained PGAs. Another name for this type of PGAs is distributed GAs since they are usually implemented on distributed memory architectures.

The most important characteristic of coarse-grained PGAs is the use of few relatively large demes along with the migration concept that relates these demes to each other. Coarse-grained PGAs are considered the most popular method of parallelization of GAs. This section presents the most influential work in this domain, and which had great impact on formulating the parameters affecting this methodology.

The first systematic study of coarse-grained was a dissertation by Grosso [40]. His objective was to simulate the interaction of several parallel sub-components of a population in evolution. Individuals in this GA were known as diploid individuals, which meant that there were 2 sub-components for each gene in the chromosome each carrying a value that affects the fitness evaluation. The population was divided into 5 demes and all of them were allowed to interact with each other at a fixed rate of migration. Through experiments and controlled parameters, Grosso found that the improvement of the average population fitness was faster in smaller size demes than in large panmictic population. This confirmed the hypothesis in population genetics that states that favorable change in the genetic composition spread faster to other individuals when the demes are of smaller size rather than large size populations. However, through observation, Grosso found that when the demes were isolated, the rapid increase in fitness value stopped at a lower value than with large single populations. This means that the quality of the solution found after convergence was worse in the isolated case than in the single population. When Grosso tried lower migration rate, the demes still behaved independently and explored different regions of the search space. The migrants did not have a significant effect on the receiving deme and the quality of the solution was similar to the case where the demes were isolated. On the other hand, when intermediate values of migration rates were used the demes found solutions similar to those found in the single population. Such observations lead to the belief that there is a critical migration rate below which the performance of the algorithm is obstructed by the isolation of demes. In addition, using a much higher migration rate than the critical rate leads to high communication costs and will probably find solutions similar to that found by a single population GA.

In a similar parallel implementation by Pettey et. al. [60], a copy of the best individual found in each deme was sent to all its neighbors after every generation. The purpose of this constant communication was to ensure and establish a good mixing of individuals in each deme. Like Grosso, the researchers in this work observed that parallel GAs with a high level of communication are equivalent to a sequential GA with a single panmictic population. These observations prompted more serious questions regarding the parameters involved in coarse-grained PGAs. Such questions were: what is the level of communication necessary to make a parallel GA behave like a single population GA?, what is the cost of this communication?, is the communication cost small enough to make this viable alternative for the design of PGAs?. Most research have started since then becoming more oriented towards understanding the effect of migration on the quality of the search in PGAs in order to answer such questions. In 1987, Tanese [70] proposed a parallel GA with the demes connected on a 4-D hypercube topology. In Tanese's implementation the migration of individuals among demes occured at fixed intervals. The algorithm also limited migration between neighboring processors along one dimension of the hypercube. The migrants were selected using traditional selection mechanisms in GAs to replace the worst individuals in the new demes. In the first set of experiments Tanese fixed the interval between one migration and the other to be 5 generations and varied the number of processors and the percentage of population that migrated. The results obtained show that the quality of solutions found are similar to that found by the serial counterpart of the algorithm. when using two different percentages for the migrated population. The other set of experiments performed in the same work studied the effect of exchange frequency on the quality of the search. The results were aligned with Grosso's findings that migrating too frequently or too infrequently degraded the performance of the algorithm. In 1989, Tanese [69] completed her work by using a very complex experimental study on the frequency of migrations and the number of individuals exchanged each time. In this research, the performance of a serial GA and parallel GAs with and without communication were compared. All the algorithms had the same total population size (256 individuals) and were executed for 500 generations. Tanese found that a coarse-grained PGA with no communication and *r* demes with *n* individuals, each was able to find at some point during the run solutions of the same quality as a single serial GA with a population of $r*n$ individuals. However, the average quality of the final population was much lower in the parallel isolated GA. When Tanese repeated the same experiment with communicating demes, the migration effect contributed to an increased final average quality.

A recent research by Belding in 1995 [12], confirmed Tanese's findings. The research sent migrants to random destinations rather than to neighbors on a hypercube topology. The author shows that results were very similar when the experiment was repeated on a hypercube topology. In most cases, the global optimum was found more often when migration was used rather than having totally isolated demes. One other research in 1987, [19] Cohoon et. al., studied the effect of connecting demes in different topologies on the performance of the PGA. The problem they used was the linear placement problem and they connected the demes using a mesh topology. Their conclusion was that the performance of PGAs is not affected significantly by the topology as long as the topology has "high connectivity and small diameter to ensure adequate mixing of individuals as time progresses". They later extended their work to a VLSI problem, the graph partitioning problem, on a 4-D hypercube topology in 1991 [20]. Like in the mesh topology, the nodes in the hypercube have 4 neighbors but the diameter of the hypercube is shorter than the mesh.

In the late 1980s and early 1990s most work started formalizing into theoretical studies of parallel GAs. This tendency emerged from the desire to formulate the parameters affecting PGAs' performance.

These parameters are:

(1) Topology which defines the connections between the subpopulations.

(2) Migration rates that control how many individuals migrate.

(3) Migration interval which affect how often migration occurs.

The need to understand factors that might limit the efficiency and accuracy of PGAs was the main drive for theoretical desire to design better parallel algorithms. The first attempt to provide some theoretical foundation of PGAs was by Pettey and Leuze [61]. In their work they presented a derivation of the schema theorem for parallel GAs where randomly selected individuals are broadcast every generation. Their calculations showed that the expected number of trials allocated to each schemata can be bounded by an exponential function, just as for serial GAs.

Theoretical work targeted towards migration policies did not mature until 1997 with the work of Cantu-Paz [15]. In it Cantu-Paz presented an analysis of PGAs with multiple populations (demes) and makes an explicit relationship between the probability of reaching a desired solution with the deme size, the migration rate and the degree of the connectivity graph. The research addresses the problem of scaleable communications and its goal was to find the configuration of multiple demes that can reach the desired solution in the shortest time possible. The models used in this work assumed that the demes evolve in isolation until each deme converges to a unique solution. At this point, the demes exchange an arbitrary number of individuals and restart their execution. Each convergence-migration event was called an epoch. The initial analysis was performed to describe the relation between the deme size, the migration rate, and the degree of the topology with the probability of success after two epochs. This analysis showed illustrated how to find the configuration that optimizes the execution time while reaching a predetermined target quality. These results were later generalized in the second phase of analysis to multiple epochs. After multiple epochs, the topology would seem to be an important factor in the solution quality because a deme receives indirect contributions from varying number of demes. Different topologies with the same degree reach almost identical solutions after any number of epochs. The small differences in the solution found in the results were attributed to the different sizes of the extended neighbors. The equivalence of the topologies with the same degree facilitated the derivation of a general model of solution quality. The quality model was transformed into an accurate deme sizing equation, which was used to find the degree and the number of epochs that minimize the execution time. Equations presented in this theoretical work enable users to optimize their algorithms to the particular hardware environment and problem domain and thus eliminates guessing or excessive experimentation to determine appropriate values for the deme sizes and topology of communications.

CHAPTER III

SIMULATED ANNEALING

General Overview

Simulated annealing (SA) has been known as a method for obtaining good solutions to difficult optimization problems, and has received great interest since 1983. The interest in SA began with the work of Kirkpatrick et al. in 1983 [49] and Cerny in 1985 [16]. Kirkpatrick [49] was the first to show how a model for simulated annealing of solids could be used for optimization problems, where the objective function to be minimized corresponds to the energy of the states of the solid. Since then, SA has been applied to many optimization problems occurring in areas such as computer design (VLSI), image processing, molecular physics and chemistry and job shop scheduling, such as in [17], [47] and [55]. There has also been much work on the theoretical results from a mathematical analysis of the method, as well as many computational experiments comparing the performance of SA with other methods for a range of problems.

Several combinatorial optimization problems have been shown to be *NP*-hard, which means that the running time for any algorithms currently known to guarantee an optimal solution is an exponential function of the size of the problem. Simulated annealing is one of many heuristic approaches designed to give a good though not necessarily optimal solution, within a reasonable computing time.

SA is categorized as being a type of local search algorithm. A simple form of local search (a descent algorithm) starts with an initial solution perhaps chosen at random. A neighbor of this solution is then generated by some suitable mechanism and the change in cost is calculated. If a reduction in cost is found, the generated neighbor replaces the current solution, otherwise the current solution is retained. The process is repeated until no further improvement can be found in the neighborhood of the current solution and so the descent algorithm terminates at a local minimum point.

Although a descent algorithm is simple and quick to execute, the disadvantage of the method is that the local minimum found may be far from any global minimum. One way of improving the solution is to run the descent algorithm several times starting from different initial solutions and take the best of the local minima found. In SA, instead of this strategy, the algorithm attempts to avoid becoming trapped in a local optimum by sometimes accepting a neighborhood move which increases the value of *f*. The acceptance or rejection of an uphill move is determined by a sequence of random numbers, but with a controlled probability. The probability of accepting a move which causes an increase δ in *f* is called the acceptance function and is normally set to exp $(-\delta/T)$ where *T* is a control parameter which corresponds to temperature in the analogy with physical annealing. The idea of an acceptance probability has been acquired from the physical process of annealing. Thus SA can be described as a local search mechanism which adopts natural processes, similar to genetic algorithms. The acceptance function implies that small increases in *f* are more likely to be accepted than large ones and that when *T* is high most moves will be accepted, but as *T* approaches zero most uphill moves will be rejected. So in SA, the algorithm is started with a relatively high value of *T*, to avoid being prematurely trapped in a local optimum point. The algorithm proceeds by attempting a certain number of neighborhood moves at each temperature, while the temperature parameter is gradually dropped.

The Physical Analogy

The motivation for the simulated annealing algorithm comes from an analogy between the physical annealing of solids and combinatorial problems. Physical annealing refers to the process of finding low energy states of a solid by initially melting the substance, and then lowering the temperature slowly, spending a long time at temperatures close to the freezing point. An example would be producing a crystal from the molten substance. In a liquid, the particles are arranged randomly, but the ground state of the solid, which corresponds to the minimum energy configuration, will have a particular structure such as seen in a crystal. If the cooling is not done slowly, the resulting solid will not attain the ground state, but will be frozen into a metastable, locally optimal structure, such as glass or a crystal with several defects in the structure [1]. In an analogy between physical annealing and simulated annealing, the different states of the substance correspond to the different feasible solutions to the combinatorial optimization problem, and the energy of the system corresponds to the function to be minimized.

Physical annealing has been successfully modeled as a Monte Carlo simulation. In the 1950s, Metropolis et al. [56] introduced a simple algorithm to simulate a collection of atoms at a given temperature. During each iteration an atom is given a small random displacement and the resulting change, δ, in the energy of the system is calculated. If δ is found to be less than zero then the resulting change is accepted, but if it is greater, the change is accepted with probability $\exp(-\delta/k_BT)$ where T is the temperature and k_B is a physical constant called the Boltzmann constant. If a large number of iterations are carried out at each temperature, the system attains thermal equilibrium at each temperature. At thermal equilibrium, the probability distribution of the system states follows a Boltzmann distribution where the probability of the system being in state i at temperature T is $(1/Z(T))*exp(-E_i/k_BT)$ where E_i is the energy of state i and $Z(T)$ is the partition function required for normalization. The acceptance function introduced by Metropolis et al. ensures that the system evolves into the required Boltzmann distribution.

So in analogy, the different states of the substance correspond to different feasible solutions to the combinatorial optimization problem and the energy of the system corresponds to the function to be minimized. The descent algorithm corresponds to 'rapid quenching' where the temperature is reduced quickly, such that moves which result in a reduction of the energy of the system are accepted. The determination of the initial temperature, the rate at which the temperature is reduced, the number of iterations at each temperature and the criterion used for stopping is known as the annealing or cooling schedule. The choice of annealing schedule has an important bearing on the performance of the algorithm.

The SA algorithm can be modeled using the theory of Markov chains [24]. If the temperature parameter T is kept constant, then the transition matrix P_{ij} representing the probability of moving from state i to state j is independent of the iteration number, which corresponds to a homogeneous Markov chain. The matrix shows that using it, it is possible to get from any state i to any other state j in a finite number of moves with non-zero probability. Such transitions generate a Markov chain corresponding to the SA algorithm and which has a unique stationary distribution $q(i)$, that does not depend on the initial state. The stationary distribution corresponds to the Boltzmann distribution at thermal equilibrium in the physical analogy. As a corollary of this result, the limit as $T \rightarrow 0$ of this stationary distribution is a uniform distribution over the set of optimal solutions, i.e. the SA algorithm converges asymptotically to the set of globally optimal solutions. Asymptotic convergence to the set of globally optimal solutions is an encouraging result, but it does not tell us anything about the number of iterations required for reaching convergence. Aarts and van Laarhoven [1] have shown that the stationary distribution is approximated, only if the number of iterations is at least quadratic in the size of the solution space. Since the size of the solution space is usually exponential in the size of the problem itself, this means that approximating the stationary distribution arbitrarily closely results in exponential running time for SA.

However, the SA algorithm can also be described as a sequence of homogeneous Markov chains of finite length, using decreasing values of the temperature parameter *T*. This can be considered as a single non-homogeneous Markov chain, as the transition probabilities are now not independent of the number of iterations. Several results have been derived giving sufficient conditions for asymptotic convergence to the set of global optima. Hajek [42] gives the strongest result in that he provides necessary and sufficient conditions for this result. He shows that if $T(k) = c/log(1+k)$ where *k* is the number of iterations, the condition for asymptotic convergence is that *c* be greater than or equal to the depth of the deepest local minimum which is not a global minimum. This temperature function represents a very slow cooling schedule. It has also been shown that attempting to approximate arbitrary closely to the uniform distribution on the set of optimal solutions, leads to a number of iterations which is larger than the size of the solution space, and so results in exponential running time for most problems. The theoretical results for the asymptotic convergence of SA modeled as a non-homogeneous Markov chain do not require the stationary distribution to be achieved at any non-zero temperature.

Implementing SA

In order to use SA for a particular combinatorial optimization problem, there are a number of decisions to be made. These decisions were identified and categorized into two groups by Johnson et al. in 1988 [48]. Decisions are divided into generic choices and problem specific choices. The basic simulated annealing algorithm is shown as:

```
Select an initial state.
Select an initial temperature $T > 0$.
Set a real value for gradient $\alpha$.
Repeat
Set n = 0; where n is the number of iterations per temperature.
    Repeat
    Generate new state j from current state.
     IF the new move is better than the current, it is accepted.
    ELSE the new move is accepted with a specific probability $P(T)$ such that $P(T_i)$
    $= \exp(-\delta/T_i)$; $\delta = E(S_{new}) - E(S_{current})$.
    n = n + 1
    Until n = N(T)
    $T_{i+1} = \alpha T_i$
Until stopping criterion
```

Figure 6: Simulated Annealing algorithm in Pseudo-code

Generic Choices

Generic choices are choices that need to be made for any implementation of SA. They are categorized as generic since their choice is problem independent. Generic choices in SA implementation mainly constitute the annealing or cooling schedule via:

i. The initial value of the temperature parameter *T*.

ii. A temperature function, *T(t)*, to determine how the temperature will be changed.

iii. The number of iterations, *N(t)*, that will be performed at each temperature value.

iv. A stopping criteria for terminating the algorithm.

A great variety of cooling schedules has been suggested. The earliest proposed cooling schedules were based on the analogy with physical annealing. So for example, Kirkpatrick et al. [49] set the initial temperature value of T to be large enough so that virtually all transitions are accepted, which corresponds to heating up a substance until all the particles are randomly arranged in a liquid. A proportional temperature function is used, i.e.:

$T(t+1) = \alpha T(t)$ where α is a constant. Typical values of α used in practice lie between 0.8 and 0.99. Such a function provides smaller decrements in T as zero temperature is approached. The number of iterations at each value of T, $N(t)$ is determined by a sufficient number of transitions being accepted subject to a constant upper bound. This is an attempt to bring the system to a state corresponding to thermal equilibrium in the physical analogy. Other simple schemes may keep $N(t)$ constant, or make it proportional to the size of the problem instance or the size of the neighborhood defined. The SA algorithm is stopped when the solution obtained at each temperature change is unaltered for a number of consecutive temperature changes. The final state is then said to correspond to the frozen state. Other cooling schedules make a more direct appeal to the theoretical results on asymptotic convergence. The number of iterations to be carried out at each value of T, $N(t)$, is chosen so that the system is sufficiently close to the distribution at that value of T. Aarts and van Laarhoven in 1985 [1] refer to this as 'quasi-equilibrium'. Different rules or heuristics for deciding when quasi-equilibrium has been reached lead to different cooling schedules.

However, as Hajek's results in [42] show, for asymptotic convergence to the set of optimal solutions, the condition is for the cooling to be carried out sufficiently slow, rather than there necessarily being any requirement to attain a state corresponding to thermal equilibrium at a succession of reducing temperatures. This result also suggests that if T is fixed for a number of iterations, then there will be a trade-off between large reductions in T and a small number of iterations at fixed T. So at one extreme there is, for example, the scheme proposed by Lundy and Mees [53], where the temperature is only a single iteration at each temperature. They use heuristic arguments to derive a temperature function of the form:

$$T(t + 1) = T(t)/(1 + B*T(t)) \qquad (8)$$

where B is a constant, and which is equivalent to:

$$T(t) = C_1/(1 + t.C_2) \qquad (9)$$

where C_1, C_2 are constants. As the number of iterations increases, this will represent slower cooling than a proportional temperature reduction with fixed values of α and N. At the other extreme, there is the heuristic scheme based on SA proposed by Connolly [21] in 1988. This scheme suggests that the majority of the iterations should be conducted at a suitably fixed temperature.

One broad distinction made among cooling schedules is to contrast them with and without feedback for $T(t)$ and $N(t)$. Simple schedules without feedback determine $T(t)$ and $N(t)$ at the start of any simulated annealing run. Other schedules allow feedback from the progression of the algorithm to affect the current values of $T(t)$ and $N(t)$. The schedule described by Kirkpatrick et al. [49] allows the number of accepted transitions to influence $N(t)$. Another example is the cooling schedule proposed by Aarts and van Laarhoven [1] where $N(t)$ is kept constant, but the temperature function depends on an estimate of the standard deviation of the distribution of objective function values at the current temperature, that needs monitoring during execution. Aarts and van Laarhoven have shown that their cooling schedule normally leads to a worst case running time that is a polynomial function of the size of the problem. Other researchers have suggested that alternative measurements, such as monitoring the specific heat of the system, may be a useful guide to a suitable rate of cooling.

Several users of SA have found that implementing it in the ways described above has led to either very long execution time or relatively poor solutions when the run time is limited. Thus many users have modified the SA approach in ways which take the algorithm away from the physical analogy on which it is based, but eventually lead to a more efficient algorithm.

Problem Specific Choices

Problem specific choices are choices that are problem dependent and vary from one problem to the other. When simulated annealing is applied to solving optimization problems, there are two choices that need to be made and are categorized as problem specific choices. These choices are the neighborhood move operator and the cost/energy function. The neighborhood of any solution is usually generated through a heuristic like method that randomly generates a new move from the current one. A function that evaluates the states is needed in order to determine the energy of the system and participate in the annealing schedule. An initial is generated using the neighborhood move generate operator.

Several researchers have shown that the efficiency of SA depends on the neighborhood structure that is used, such as in Cerny [16]. This is not surprising given Hajek's result in [42] which showed that the rate of cooling required for asymptotic convergence to the optimal set of solutions depends on the depth of the local minima, i.e.: to the topology that is imposed by the neighborhood structure. In general, a neighborhood structure imposing a 'smooth' topology such that the local optima are shallow is preferred to a 'bumpy' topology where there are many deep local minima. If the problem is constrained, a choice must be made between restricting the solution space to solutions that conform to the constraints, or allowing solutions that break the constraints in expense of a suitably defined penalty function. The second option is likely to lead to simpler neighborhood moves, and a smoother topology, particularly if the penalty weights can be kept small.

Modifications to Simulated Annealing

This section contains a categorization of the on going modifications to the basic SA algorithm. The major limitation in the pure simulated annealing algorithm is the long execution time required for reaching a globally optimal solution. Modifications to the basic algorithm have been proposed by several researchers and have been introduced to improve the effectiveness of the basic algorithm when used for solving difficult combinatorial optimization problems.

Storing Best Solution So Far

The SA algorithm sometimes accepts solutions worse than the current solution. It is therefore possible in any single SA run for the final solution to be worse than the best solution found during the run. It is not usually computationally expensive to store details of the best solution found so far, so better solutions may result from run times similar to those for the basic algorithm. In addition, Glover and Greenberg [34] argue that with this modification, there is less need for the SA algorithm to rely on a strong stabilizing effect over time. This idea is supported by Connolly's [21] modification of SA, where having found a suitable fixed temperature, all the remaining iterations are carried out at that temperature. In the final phase, a descent algorithm can be employed to find the local optimum containing the best solution encountered in the earlier phases.

Sampling the Neighborhood without Replacement

When the temperature parameter T gets low in the latter part of a simulated annealing run, the probability of accepting worse solutions than the current one becomes small. Thus in the region close to a local optimum, most of the computer time is spent rejecting worse solutions. If there are only one or two neighborhood moves that give improved solutions, the basic SA algorithm may take a lot of time to find them. Therefore some researchers have suggested that neighborhood moves should be generated in such a way that all possible moves in the neighborhood of a solution are attempted before repeating a move, unless a new solution is accepted.

Johnson et al. [48] implemented this idea by generating a new set of potential neighborhood moves every n iterations, where n is the size of the neighborhood. Within every block, each potential move is considered once and in a random order. Experiments with graph partitioning problems showed that this modification gave significantly better results than simply choosing neighborhood moves at random in a similar run time. Connolly [21] also presents experimental evidence of the benefit of sampling the neighborhood without replacement, though in this case it is implemented by simply searching through the neighborhood in a sequential rather than a random manner. Another advantage of this approach is that when the algorithm has been terminated and there has been no improvement in the solution for at least n moves, where n is the size of the neighborhood, then the final solution is guaranteed to be a local optimum.

Alternative Acceptance Probabilities

Some researchers have considered replacing the standard function, $exp(-\delta/T)$ for the probability of accepting a solution which increases the objective by δ, with some alternative function. Indeed there has been some theoretical work to show that asymptotic convergence holds for more general acceptance functions [57]. In applications of SA, where the calculation of $exp(-\delta/T)$ is a significant factor in the run time, Johnson et al. [48] proposed that the exponential function be approximated by a simple lookup scheme. Experiments on graph partitioning problems saved one third of the running time with no apparent effect on the quality of the solutions.

Complex Modifications to SA

The nature of more complex modifications might require more implementation modifications to the basic SA algorithm. They are usually more dependent on the problem under consideration, and as these ideas may result in a more powerful algorithm for a specific application, they cannot be easily transferred to different applications.

Hybrid Simulated Annealing

The first of these modifications is to combine SA with another method. There are two ways in which SA may be combined with an alternative method; either using the alternative method to provide a "good" initial solution which SA attempts to improve, or by using SA to provide a "good" initial solution as a starting point for the alternative method. The latter approach is exemplified by using SA as a way of obtaining a good initial solution for a branch and bound algorithm or an integer programming algorithm. This approach has not been of great attraction to researchers in simulated annealing, since heuristic methods that are problem specific tend to outperform SA in generating good initial solutions.

The first approach, which uses other local search mechanism to enhance the initial starting solution for SA has been the focus of several research, such as [44] and [21]. This mechanism is illustrated by Chams et. al. [44], when considering the graph colouring problem. Johnson et. al. [21] also provide some experimental results for the graph partitioning problem. They show that both quality of solution and running time may be improved by the use of a good starting solution. When a good initial solution is used, the initial temperature in the cooling schedule is reduced, otherwise the benefits of the good initial solution will be lost. They also show that starting solutions that take advantage of the special structure of the problem instance being considered seem preferable to those obtained by general heuristics.

Hybrid-CLO Technique

In 1994, [55] introduced a meta-heuristic which combined simulated annealing with local search methods for combinatorial optimization problems. The main idea was to embed deterministic local search techniques into simulated annealing and test it on two problems: the TSP and the graph partitioning problem. The heart of this meta-heuristic technique is to restrict the sampling of the Markov Chain in SA to the locally optimal configurations. Then, the bias that the Markov Chain provides would enable the sampling of the best locally optimal configurations more efficiently than local search repeated from random starts.

The algorithm proceeds as follows, suppose the configuration is currently locally optimal. Apply a perturbation to this state in order to reach another *intermediate* state.

Standard simulated annealing would impose the accept/reject procedure directly to this new state. Instead, this state is first improved by a local search mechanism and then the acceptance probability rule is applied to it. The iteration of this procedure was named "Chain Local Optimization (CLO)". This approach can be seen as a large-step Markov chain generalization of simulated annealing. It allows the accept/reject step to be applied only after any state is returned to a local minimum point.

To implement the above methodology for an arbitrary problem, two choices need to be carefully made: a good local search method and a good neighborhood move operator. When the CLO was applied to the TSP, it used the Lin-Kernighan [52]. For the graph partitioning problem, the Kernighan-Lin algorithm [52] was used. Both of these algortithms are known for being the best local search methods for these problems. For each problem, a specific neighborhood move was developed that best suits it. The TSP experiments used three instances existing in TSPLIB [63]. For each one of these instances, the CLO technique was able to find an optimum solution in less time than the Lin-Kernighan technique. For an instance of the TSP, that was 10,000 cities the CLO was able to enhance on the best solution found by Lin-Kernighan by 1.3% and over the 3-opt technique by 1.6%. For the graph partitioning problem, they used five geometric graphs as their benchmark for comparison between Kernighan-Lin and the CLO. The five graphs were randomly generated with sizes equal to 100, 250, 500 and 1000. They ran the Kernighan-Lin algorithm for 2000 runs and the CLO for 20 runs. For the small graphs of size 100, both algorithms found the same best solution that has been found by any previous method. As the graph size increased, it was observed that the CLO rapidly improved over Kernighan-Lin. For the 250 nodes graph, a 2000 runs of Kernighan-Lin was not able to find the best ever solution and the number of runs were increased in order to find it. As for the larger networks, increasing the number of runs with Kernighan-Lin was never able to find the best solution. Using CLO, the best solution ever found was reached for graphs of size 500 and 1000.

Approximate Calculations

Many users of SA adapt the basic approach to take advantage of special features of the problem being considered. In many applications, the bulk of the run time is taken by calculating δ, the change in the objective value at each transition. Grover [41] shows that significant speed-ups can be obtained by calculating δ approximately instead of exactly. These approximations are shown not to affect the quality of the solution significantly, provided the errors are less than a function of the temperature parameter, T. It has been suggested that the algorithm may be speeded up by using a quicker, approximate method in order to calculate δ. Several variations of the approximate method have been introduced. The "Poor Man's Swindle" presented in [41] is one of these variations which does not calculate δ every iteration, but rather it is employed with a certain probability. This probability is updated as the algorithm proceeds so that the resulting algorithm will simulate the results of the basic SA approach.

Another approach is the "Neighborhood Prejudice Swindle" [41] which is built on the idea of using problem-specific information to identify a promising subset of the neighborhood, and a generation mechanism that will give a greater probability of attempting moves into the promising subset. Although this approach has been used in extensive research in [41], it has not been demonstrated that it can perform significantly better than the schemes suggested for sampling from a neighborhood without replacement.

In the work by Greene and Supowit [37], they proposed a rejection-less method. This scheme also aims to reduce the amount of time spent in the latter part of a SA run when the majority of the moves are rejected. In this scheme, a probability distribution is constructed over the set of all moves to show the relative probability of a move being selected if it is chosen. A move is then selected at random according to this distribution and accepted automatically. It was shown that the result of this procedure is equivalent to the basic SA algorithm. To apply this approach to problems means finding an efficient way to update the probability distribution and this depends on problem-specific information.

Parallel Simulated Annealing

Simulated annealing has been recognized as an efficient and powerful optimization method for complex engineering problems such as standard cell placement, floor planning, circuit extraction which are all VLSI design problems. Although SA is a powerful technique it has been described as an algorithm that requires huge amounts of time to reach the optimal solution. In order to reduce execution time, researchers have started introducing parallel techniques to the basic simulated annealing algorithm. Since a new state contains modifications to the previous state, simulated annealing is often considered an inherently sequential process.

Most of the parallel techniques that have been developed try to eliminate some of the sequential dependencies existing in the basic SA algorithm. Important questions were posed with regards to parallel SA and have assisted in categorizing the parallel techniques. The problem in the basic algorithm is that it depends on perturbation of one state at a time. This enforces global knowledge of the current optimal state in order to evaluate the quality of the newly generated state with respect to the current one. This limitation in the basic sequential algorithm of simulated annealing has lead to several questions in the field of parallel SA. These questions are:

1. How is the state space divided among the processors?
2. Does the state generator for the parallel algorithm produce the same neighborhood as the sequential algorithm? How are states generated?
3. Can moves made by a single processor cause cost-function calculation errors in another processor? Are there mechanisms for controlling such errors?
4. What is the speedup? How does the final cost vary with the number of processors? How fast is the algorithm, when compared to an optimized sequential program?

In order to answer these questions researchers in the field of parallel SA have developed techniques that address these questions and attempt to find solutions for each one. This has lead to categorization of the parallel techniques that have been developed.

In [38] and [10] we find a well-defined categorization of the techniques that have been developed in simulated annealing. Although each one categorizes them from a different perspective we still reach the same understanding of the current parallel simulated annealing techniques. In [10] the emphasis is more on the theoretical analysis of the speed-up that the various techniques have introduced to the huge amount of execution time needed by the basic simulated annealing algorithm. While in [38], the emphasis is on the algorithmic differences between the available techniques. The latter also emphasizes the differences among the parallel techniques through the effect of communication between multiple processors.

In the following sections we present a categorization of the parallel simulated annealing techniques. This categorization is based on the algorithmic differences between the various alterations performed on the basic technique in order to develop parallel versions from it. In this categorization the advantages and disadvantages of each approach is presented in order to provide some guidelines on how to select an appropriate approach for the problem being solved.

Serial-Like Parallel SA Techniques

The parallel algorithms of this category are characterized by preserving the convergence properties of the sequential simulated annealing algorithm. There are three types of serial-like algorithms, these are: functional decomposition or accelerated moves, simple serializable set, and decision tree decomposition.

Accelerated Moves

In this approach, each individual move is evaluated faster by breaking up the task of evaluating a move into sub-tasks, such as selecting a feasible move, evaluating the cost changes or deciding on accepting or rejecting the new move.

Concurrency in this approach is obtained through delegation of sub-tasks to different processors. The division of tasks makes this type of algorithm also known as the functional decomposition approach. Algorithms belonging to this category exploit parallelism in the cost function. In problems where the evaluation of the objective function is complex operation, the functional decomposition methods are usually the best to use for parallel SA. It is also an advantage to use this parallelization approach when the cost function is easily decomposed into sub-tasks that have minimal dependency.

The major limitation when using this approach is on speedup. The functional decomposition algorithms need high communication and synchronization time. This is due to the fact that the sub-tasks involved in the evaluation function itself needs to synchronize with each other in order to have the latest cost calculations among the different processors. In order to accept or reject the new move that has been generated all processors need to communicate their latest calculations and synchronize in order for the processor responsible for the decision making to decide to accept or reject the new state. The main disadvantage of this type of parallelization approach is the limitation on large-scale parallelism. These approaches also tend to be restricted to implementation on shared memory architectures.

Simple Serializable Sets

The simple serializable set approach can be described as having a collection of moves that affect independent state variables and thus distinct processors can independently compute the cost without communicating with other processors. This is similar to spatial decomposition in the most ideal context where the problem being solved can be decomposed into totally independent state variables. This approach is called the serializable set because moves can be concluded in any order, and the result will be the same.

The simple serializable set exploits the property that at low temperature the acceptance ratio is often very low. If processors compete to generate one accepted state, most will generate rejected moves that can all be executed in parallel. This is mostly true at the beginning of the annealing schedule when temperature is still low. If the acceptance ratio at temperature T is $\propto(T)$, then the maximum speedup of this algorithm, ignoring communication and synchronization, is $1/\propto(T)$. At high temperature, where the acceptance ratio is close to 1, the algorithm provides little or no benefit. But since traditional annealing schedules spend a majority of time at low temperatures, the simple serializable set approach can improve overall performance.

Speculative Computation

The last category of serial-like parallel simulated annealing is called speculative computation. This approach attempts to predict the execution behavior of the simulated annealing schedule by speculatively executing future moves on parallel nodes. Although the speed-up is limited to the inverse of the acceptance rate, the speculative computation method has the advantage of retaining the exact execution profile of the sequential algorithm, and thus the convergence characteristics are maintained.

Altered Generation Parallel SA Techniques

The altered generation methods are characterized by modifying the state generation process in order to reduce the interprocesser communication costs. Algorithms belonging to this category of parallel SA techniques change the pattern of state space exploration, and thus change the expected solution quality and execution time. In this type of algorithms the state generation is modified with a reduction in communication costs but with retaining an accurate calculation of cost. A group of algorithms that are part of altered generation are the shared state space algorithms. The parallel moves algorithms are another kind of algorithms that also belong to this category.

Shared State Space

The different kinds of algorithms that belong to this category are known to be synchronous algorithms that tend to be implemented on shared memory architectures. The shared state space algorithms make simultaneous, independent moves on a shared memory state space but with no cost function errors because there is no outdated evaluations being performed. In this type of algorithms the entire state space exists on a shared memory accessible by all participating processors. There is no state variable decomposition when using this approach. Each processor is allowed to generate a move involving the entire state space. Shared state space algorithms rely on performing a lock on the state variables if there is need to make any evaluations. If there are no conflicts between a generated move on one processor and any other move in-progress on another processor, then the new move is broadcast to all processors. The conflict is detected through the locking of state variables. Once a conflict arises, the generated move is abandoned and another move is generated. The main disadvantage of this approach is the lock step that needs to be performed on the shared memory state variables. When several locks occur, this will affect the speedup of the parallel technique as well as preventing possibly good moves from being broadcast.

Parallel Moves

In this method, each processor generates and evaluates moves independently. In order to have parallel moves, the state variables involved in the state space are distributed among processors. This type of parallelization is also referred to as spatial decomposition. In order to avoid incorrect generation of states, this approach must either coordinate move generation among processors, or must prevent processors from generating moves that affect other processors' state variables. Doing so allows synchronization among processors containing different parts of the state space.

Two techniques have emerged due to the limitation of spatial decomposition, the first is cooperating processors and the second is the independent processors. The cooperating processors' algorithm disjointly partitions state variables over processors. Any processor that generates a new state notifies the master processor that usually keeps track of the latest optimal solution found. The master decides if the new move is better and thus starts to broadcast it to other processors. Before broadcasting, an evaluation from all other processors is requested in order to update their states. If any interference occurs between processors, the move is delayed from broadcasting. Interference might occur if processors change the same state variables.

The main disadvantage of synchronization is the communication overhead that might limit speedup. An ideal situation arises if the structure can be decomposed in such a way that each processor is allocated a set of variables that contributes independently to the structure's energy. This can be suitable for the second type of approach that is called the independent processors. In this method each processor generates state changes which affect only its own variables. Although this approach might seem good, it has several limitations. One of the limitations is that problems cannot easily have variables that are decomposed in a non-interacting manner. The second limitation is that this fixed variable assignment can lead to restricted state space exploration and thus requires periodic state variable redistribution.

Multiple Markov Chains

This approach calls for the concurrent execution of separate simulated annealing chains with periodic exchange of solutions. In the Multiple Markov Chains approach multiple random states are generated and distributed among existing processors. Each processor then starts annealing on the state received using the same temperature schedule. After a periodic time, processors evaluate their states and the best solution found is redistributed among processors in order to continue using it. Multiple Markov Chains do not promise a significant increase in speed-up since we need to give each processor the chance to freeze and reach the best possible solution based on its initial state.

Multiple Markov Chains has an advantage that each processor starting independently at a different initial state allows concurrent exploration of different areas of the search space and thus helping prevent falling into local optimum points through redistribution of the best solution found so far. This approach is particularly promising since it has the potential to use parallelism to increase the quality of the solution. This is the first algorithm presented through this categorization that benefits from parallelism an enhancement in the solution quality. All previous methods approached parallelism for its natural benefit, which is speedup.

Asynchronous Parallel SA Techniques

The category of asynchronous parallel simulated annealing can include any of the previously presented techniques that need synchronization among various processors. In this category of parallel simulated annealing algorithms there is not sufficient synchronization among processors. Processors can simultaneously read and alter dependent state variable, causing cost function calculation errors.

In these algorithms, processors operate on out-dated information about the state variables. Since simulated annealing randomly selects hill-climbing moves, it can tolerate a certain percentage of error, and the annealing algorithm can evaluate the cost under old state information but still converge to a reasonable solution. Error tolerances provide a great advantage in multiprocessing: when processors operate independently on different parts of the problem they need not synchronously update other processors.

One approach in these techniques, is where a processor can save several changes and then send a single block to all other processors. Through this mechanism, the processor sends less control information and compresses multiple changes to a state variable into one, thus reducing total communication traffic. In addition, if update operations can be done asynchronously, synchronization operations are reduced and the communication overhead incurred by their existence is eliminated. Asynchronous algorithms require a minimum synchronization which is needed in order to prevent any two processors working independently to provide a state that is inconsistent with the original problem definition.

The techniques carried out in the previous category, altered generation algorithms, also exist in this category. The asynchronous spatial decomposition algorithms are similar to the synchronous algorithms, except that each processor maintains read-only copies of the state variables from other partitions. When a processor in the asynchronous spatial decomposition method evaluates a new state, it uses only local copies of state variables.

In some implementations of this approach, when a new state is accepted it is immediately sent to other processors. While in other implementations, the processor would complete a fixed number of trials before transmitting the modifications. The longer the time taken before transmission is made, the more the execution-to-communication ratio increases and a gain in speedup is obtained. On the other hand, there is an increase in calculation errors which reduces the solution quality. A trade-off exists between solution quality and execution time when developing asynchronous parallel simulated annealing implementations.

Combinatorial Optimization Problems

Simulated annealing has proven to be a very powerful and general algorithm for solving combinatorial optimization problems, such as [37] and [31]. With the development of parallel simulated annealing techniques, the domain of *NP*-complete problems started to benefit from these new techniques. Parallel SA an approach that gave promise to enhancements in both quality of solution and execution times for difficult combinatorial optimization problems. In this section, some of the experimentation results that have been reached by applying parallel SA to *NP*-complete problems are reviewed.

The Job Shop Scheduling problem was used in [47] to analyze the effect of using a Multiple Markov Chain approach on the performance of simulated annealing. The algorithm developed in 1996 [47] was called: "Clustering Algorithm for SA". This algorithm takes advantage of the idea of having multiple good initial solutions that result in faster convergence. The parallel algorithm starts by having *n* nodes in the network each running SA. After a fixed number of iterations, the nodes exchange their partial results in order to get the best one. All the nodes accept the best partial solution and then start applying the SA technique for that best partial result. This algorithm is a Multiple Markov Chain parallel SA with synchronization among nodes. The performance of the Clustering Algorithm was compared with that of sequential SA for different problem sizes, ranging from 10 jobs on 10 machines to 20 jobs on 15 machines. Both types of annealing algorithms used the same initial temperature and the same annealing schedule. The performance of each algorithm was evaluated on the basis of the execution time of the algorithm and the cost of the solution. The Clustering Algorithm performed better in terms of both measures, and it also showed superlinear speedups.

VLSI Problems

One of the famous domains in which simulated annealing has proven its effectiveness is the VLSI domain. In the VLSI world integrated circuits are usually designed in a modular fashion with small logic functions designed as individual cells and combined to make an IC complete. Given the set of cells in the design and specifications as to how they should be connected, the optimization problem is to find the best way to place the cells on the chip. It is very important to solve the problem without violating any physical constraints while routing the cells. Several optimization problems emerge from having multiple constraints in the VLSI placement domain, e.g. minimizing the layout area, minimizing the estimated wire length or creating an even distribution of wires around the chip, and lately minimizing wire length delay. Simulated annealing has become the most popular approach for solving VLSI problems. With the rapid increase in complexity of circuit design in VLSI, parallel processing has started to emerge as an attractive solution to reduce the inordinate amount of time spent in design. As a consequence, most parallel simulated annealing techniques that have been developed have selected VLSI problems as the applications for their new techniques.

In 1997, a research [17] investigated three parallel simulated annealing techniques on the standard cell placement problem. The results were extremely favorable when compared to the results produced by the sequential simulated annealing. The three techniques involved in the experimentation belonged to three different categories of parallel simulated annealing techniques, which are: parallel moves, Multiple Markov

Chains and speculative computation. This work is important for both domains, the VLSI and the parallel simulated annealing. This importance results from the effectiveness shown by each kind of parallel technique in improving the speedup and quality of solution. It also demonstrates the usefulness of introducing the notion of parallelism to solve highly complex problems using simulated annealing.

The project involving three parallel simulated annealing techniques to solve the standard cell placement problem was part of a larger project called ProperCAD II project[2] and was called ProperPLACE. This research tries to eliminate the major drawbacks that are seen in previous work on parallel placement algorithms. These drawbacks are: each of the earlier proposed algorithms were architecture specific which makes it very difficult to run them on other architectures. Another drawback that this work has avoided is to develop the parallel algorithm based on rewriting the sequential algorithm which affects the performance of the algorithm on a single processor making it perform worse than the best sequential algorithm. The last drawback which was avoided is the error in the cost function evaluation in the parallel moves approach which lead to significant quality degradation as more processors are added in earlier works.

The implementation of parallel moves starts by a random input placement that is replicated on each available physical processor. The overall circuit placement is provided as a local copy of the overall structure so that the worker on each processor can maintain a coherent state of the current placement during annealing. The placement is then divided up topographically by rows, with the rows and its cells assigned to each processor, allowing the processor to modify only the row assigned to it. After partitioning, each slave on a processor proceeds with the annealing algorithm. A valid cell is selected for

[2] ProperCAD II: it is a suite of parallel applications that were developed to address the most significant tasks in VLSI design automation including circuit extraction, test generation and logic synthesis.

perturbation then a displacement or exchange is performed on that cell. Two categories of moves were used in this work, displacement and exchange. These represent the possible alternatives for moving cells in the approach of parallel moves. The cell displacement is defined as moving a cell to a new location, if the new location exists on the same processor it is known as an intra-cell displacement. Otherwise, if the new location is located on a different processor then it is called an inter-cell displacement. On the other hand, an exchange occurs if the cell is to be moved in a location where another cell already exists. An intra-exchange happens if the two cells exist on the same processor, while an inter-cell exchange occurs when each cell resides on a different processor. If an inter-cell exchange happens the slave on the processor which owns the cell to be moved send a request for permission to perform the exchange action. The second processor then evaluates the status of the cell belonging to it in order to grant a permission for the move or send a rejection. During the time when a request message is sent and an answer is received, the first processor does not remain idle. It starts annealing on other non-frozen cells. The implementation of this parallel moves approach has also introduced a prioritized message scheme in order to give higher priority to the exchange request message which takes most of the time. Finally, when a move is accepted the slave accepting the exchange sends the new move to the master which has the overall circuit structure in order to have a consistent cell position database. In order to amortize the startup cost of sending a message, position update messages are kept until a certain number of messages is accumulated. Although this scheme reduces the overall update messages communicated among processors, it increases the inaccuracy of the cost function evaluation since the overall structure is left out-of-date for longer periods. The proposed solution for this problem was to find a good size of the accumulated messages that would not be too large and thus degrade the quality of the evaluated function or is too small and would then cause a huge communication overhead. The naive approach of having a statically preset value for the size of the accumulated update message was rejected since there is no good mechanism of determining such a value ahead of time. In a dynamic approach proposed in this work, the size of the accumulated update messages is determined during the annealing process by monitoring the size of the error in the cost function in the system. If the size of the error increases, the size of the accumulated update messages will be reduced and vise versa. In order to do this dynamic change in the size of update messages, the algorithm estimates the error that each processor contributes by moving its own cells. The advantage of obtaining the error this way is that it can be calculated by each processor independently without any synchronization. The equation used to estimate the error on each processor takes into account the error that this processor might be contributing to other processors. Thus there are some probability calculations involved in the estimation of error which represent that probability of accepting a move which involves a move between two cells connected to each other through wires but do not exist on the same processor. The experimental evaluation of this technique was performed on several benchmark suites for circuit design, as well as on an industry circuit. The results show that using circuits of size ranging from 752 cells to 25,114 cells the placement quality was not affected by the new mechanism of priority messages or dynamic sizing. The overall run time increased in some cases but when they introduced a synchronization barrier in order to obtain better evaluation of the error, there was a very small increase in the quality of the solution against a huge increase in overall run time.

Using the same circuit designs, the project had an implementation of the Multiple Markov Chains method. The basic Multiple Markov Chains approach does not introduce any speedup since each processor is individually performing the same amount of work as the sequential algorithm. In order to achieve any speedup, the number of moves that are evaluated in each chain must be reduced by a factor of 1/N where N is the number of processors. Performing reduction alone is not the correct solution because it will affect the quality of solution produced. Thus, the implementation in this work introduced a factor of interaction among the various chains that are being created on multiple processors.

A synchronous Multiple Markov Chains with periodic exchange was the first implementation of this type of method used in this research. In this approach there was a periodic comparison of solution at fixed intervals of time. The method allowed each Markov Chain to update its local database with the best solution, and then continue on. This exchange point served as the end of a segment of computation and represented a synchronization barrier. In their implementation, [17] did not allow any assumptions of the underlying architecture, thus they had to perform the synchronization using message passing and not in a shared memory architecture context. When a slave processor reaches the synchronization barrier it propagates the solution value to the master, which then determines the best solution and redirects it to all other processors. The master usually finds the best solution then requests the solution state from the slave processor possessing it in order to broadcast it. This two-phase messaging scheme was selected due to the large sizes of the state in a cell placement problem. The synchronous method proposed was regarded as being costly due to the barriers inserted at various intervals of time.

An asynchronous Multiple Markov Chains method was proposed in the same work in order to avoid the communication overhead introduced by the synchronization barriers. In this approach, the master processor does not perform any computations. Instead, it serves as the location of the best available solution at any particular time. When a slave processor completes an iteration, it sends the solution value to the master and requests the best solution available. The master on receipt of this request will determine if it is better than the best local solution found so far. If the new solution is better, the master will request from the sender the actual solution state. If the new solution is found to be worse, the master will send to the slave processor the best local solution found so far. The experimental work was more concentrated on the asynchronous implementation of Multiple Markov Chains, since the research was advocating the asynchronous notion in all parallel simulated annealing techniques used for solving the cell placement problem.

Of the significant analysis results was that the quality of the solutions showed no degradation in quality as the number of processors increased. In fact, they sometimes showed improvements because of the periodic exchange of solutions. Communication time was not significant since the number of messages sent was few. It was concluded that the speedup and quality were much better than can be achieved with a pure parallel moves strategy. When this implementation was compared to another parallel cell placement algorithm which uses simulated annealing, TimberWolfSC, the solution quality was improved by an average of 9% at a cost of only 3% in runtime.

CHAPTER IV

HYBRID GENETIC ALGORITHMS & SIMULATED ANNEALING

Introduction

Simulated annealing and genetic algorithms are similar, naturally motivated, general purpose optimization procedures. Both techniques work reasonably well on a variety of problems, and require little problem specific knowledge other than fitness or energy (cost) information. The two techniques generate new points in the search space by applying operators to current points, and move probabilistically towards more optimal regions of the search space.

We can describe each by the following statements:

- Simulated annealing uses the notion of (1) a probabilistic acceptance rule, (2) exploration in the neighborhood of the current solution, and (3) the Boltzmann distribution and thermal equilibrium to guarantee asymptotic convergence to global optima in combinatorial optimization problems.
- Genetic algorithms rely on (1) probabilistic selection rules (reproduction or selection), (2) exploration in the neighborhood of the current solution (mutation), and (3) recombination of parental solutions (crossover) to promote their search.

SA and GAs possess some crucial but reconcilable differences. While in practice both may converge prematurely to sub-optimal points, only SA currently possesses a formal proof of convergence [33]. This proof depends on the cooling schedule of SA. By manipulating the cooling schedule, the SA practitioner can exercise control over convergence; slower cooling and more iterations lead to better final solutions. On the contrary, genetic algorithms do not employ any concept similar to cooling.

Smaller contrasts between simulated annealing and genetic algorithms have to do with loss of solutions, redundancy, and deception. Since SA maintains only one structure (solution) at a time, whenever it accepts a new structure, it must discard the old one; there is no redundancy and no history of past structures. The end result is that good structures and substructures (in the extreme case the global optimum) can be discarded, and, if cooling proceeds too quickly, may never be regained. SA compensates by increasingly sampling good solutions as temperature decreases.

Genetic algorithms are also prone to the loss of solutions and their substructures or bits, due to the disruptive effects of genetic operators. Upon disruption, the simple GA will not maintain an old, but better solution; it must accept any newly generated solution, regardless of fitness. However, the GA partially overcomes this, especially at larger population sizes, by exponentially increasing the sampling of above average regions of the search space, the schemata. Schemata act as a partial history of beneficial components of past solutions. However, for each above average schema duplicated, a competing, below average schema must be discarded. This can lead to trouble on hard or deceptive problems, where low order, low fitness schemata are needed for the construction of an optimal higher order schema. SA is similarly subject to deception, as the algorithm will have a difficult time paying extended visits to the high cost neighbors of a deceptive problem's global optimum.

Another important difference between GAs and SA is the ease with which each algorithm can be made to run in parallel. GAs are naturally parallel, while SA iterates a single point and thus not easy to run on parallel processors. While attempts have been made to parallelize SA a straightforward, satisfactory, general-purpose method has not yet been demonstrated. It would thus be desirable to transfer the parallel processing capabilities of GAs to SA. The parallelism mentioned here is the traditional notion of parallelism, which is known as explicit parallelism.

GAs as a standalone technique exhibit implicit parallelism, as described in [46] and [36]. SA is regarded as an inherently sequential algorithm that cannot be easily parallelized. That is the main reason why standard methods of parallelizing SA have achieved a limited explicit speedup. With the characteristic of explicit parallelism alone in SA, we would expect a parallelization method to introduce a speedup of less than *P*, given *P* processors. When adding the feature of implicit parallelism, assuming a constant number of population elements per processor, we would expect to achieve polynomial (superlinear) speedup, since the number of useful schemata increases polynomially. Superlinear speedup is what some research works have seen when combining GAs and SA, such as in [4]. This type of speedup has increased the interest in hybrid GA-SA as an efficient technique for solving difficult combinatorial optimization problems.

Theoretical Approaches

Boltzmann Tournament Selection

In the work of Goldberg [35], the notions of thermal equilibrium and the Boltzmann distributions were borrowed from simulated annealing and adapted to genetic algorithms' practice. Specifically, a Boltzmann tournament selection procedure was derived and implemented to give stable distributions within a population of structures that are provably near Boltzmann. It was thought that such a mechanism would be able to carry over to genetic algorithms the same guarantees of asymptotic convergence enjoyed by simulated annealing.

The essence leading to this work was the desire to find connections between simulated annealing and genetic algorithms to allow easier parallelization of simulated annealing. It was shown that this selection mechanism is suitable for asynchronous selection in parallel simulated annealing or for use in population-oriented genetic algorithms. The skeleton of the selection mechanism is easy:

- Choose an individual uniformly at random (with or without replacement) from current population.
- Choose another individual, with fitness different than the first with a threshold amount θ.
- Half of the time, choose a third individual with a fitness different than the earlier two individuals by threshold θ (this is called a strict choice). The other half of the time, choose the third to be only different from the first individual by the threshold amount θ (this is called a relaxed choice).
- Hold a secondary (anti-acceptance) competition between individuals two and three and keeping individual two with certain anti-acceptance probability equal to $exp^{(-E3/T)/[exp(-E2/T) + exp(-E3/T)]}$.
- Hold a primary (acceptance) competition between the winner from the above step and individual one, keeping one with the acceptance probability equal to: $\exp^{(-E1/T)/[\exp(-E1/T) + \exp(-Ewin/T)]}$; where E_{win} is the fitness value of the winner from the anti-acceptance competition.
- Repeat all previous steps until we have created a new full population.

The reason why it was chosen to select individuals of different function values, is that in serial simulated annealing there is no harm in comparing any two instances of the same individual during the acceptance step. In population approaches, it might occur erroneously that an individual is compared to a copy of itself or to another member of its equivalence class. If such error is not prevented, the Boltzmann distribution might not be stabilized and could ultimately lead to filling the population with copies of the best individual.

The anti-acceptance was considered in this approach as a major requirement in order to lead to a Boltzmann distribution. Holding the anti-acceptance competition was introduced as the way to create the population equivalent of generating a neighbor uniformly at random. If a uniform random sample is drawn from a Boltzmann-distributed population, it will be biased toward better individuals. To counteract this tendency, the competitor for the first individual was chosen by performing a competition based on the complement of the usual acceptance probability between two other individuals. Since the complimentary probability prefers poorer individuals, the distribution following anti-acceptance will be considerably flattened with respect to the original Boltzmann distribution.

A Simulated Annealing-Like Convergence Theory for GAs

In 1991, Davis and Principe [22] introduced a simulated annealing like convergence theory for the simple genetic algorithm. The aim of this theory was to extrapolate the convergence characteristics of simulated annealing on a simple Markov chain model for the simple genetic algorithm. The work of Davis and Principe [22] was not targeting a new hybrid methodology for combining GAs and simulated annealing, but rather a convergence theory for GAs similar to that in SA, this work gives deep insight on the similarity between GAs and SA which hybrid methods could benefit from.

[22] developed a non-stationary Markov chain simple genetic algorithm model. That model was accomplished for simple genetic algorithm variants implementing three combinations of the primary operators. The resulting model provided a mechanism for using the probability of mutation in a manner analogous to the absolute temperature control parameter of simulated annealing.

The essence of introducing the simulated annealing technique into the genetic algorithms was to make use of the local stochastic hillclimbing features existing in SA. In the domain of combining GAs and SA, the methods that have been developed which incorporate SA into the core of the GAs are known as the GA-oriented hybrid methods.

It is widely recognized that GAs are not well suited to performing finely tuned local search. Like natural systems, GAs progress by virtue of changing the distribution of high performance substructures in the overall population; individual structures are not the focus of attention. Once the high performance regions of the search space are identified by the GA, it may be useful to invoke a local search routine to optimize the members of the final population. Interest in this hybrid mechanism started in 1987 with [39], which emerged from the desire to incorporate domain specific knowledge into GAs to enhance its performance. Due to this great interest, several hybrid GA methods which incorporate heuristics in the basic simple GA where introduced, as described in an earlier chapter.

Incorporating simulated annealing as a local search heuristic into genetic algorithms was more promising than other problem-specific heuristics because SA as a technique has a theoretical proof of converging to the global optimal solution. In this section we focus mainly on the practical applications that made use of simulated annealing inside the simple genetic algorithm.

Parallel Genetic Heuristic

In 1989, Brown et. al, [14] introduced a Parallel Genetic Heuristic for the Quadratic Assignment Problem(QAP)[3]. The QAP has been of interest to researchers for a long time since it was first introduced to model the economics of facility location. Many

[3] QAP: is defined in the Hybrid GAs section in Chapter 2.

techniques for finding "good" solutions to the QAP have been developed. Optimal branch and bound techniques were developed for solving this well-known *NP*-complete problem. Of the sub-optimal, or heuristic, methods for the QAP the most common are based on the pair-wise interchange heuristic. Simulated annealing was used to make the genetic algorithm more sensitive to small cost changes by fine tuning the population.

The algorithm introduced by this research became known as the SAGA. Using Grefenstette's [39] "fine-tuning" concept, the Parallel Genetic Heuristic algorithm was conceived as follows:

1. Initialize the parameters of the GA.
2. Generate an initial population of solutions for the GA.
3. Use the GA to produce K "good" solutions.
4. For each of the K solutions, do the following:
   - Initialize the parameters of SA.
   - Improve the "good" solution using SA, and return it to the GA population.
5. Repeat steps 3 and 4 as needed.

The SAGA technique was defined as a parallel hybrid of customized simulated annealing and genetic algorithms which solves combinatorial optimization problems. The theoretical basis of the SAGA comes from having each offspring in the genetic population go through an annealing schedule. Each offspring in the population undergoes a maturity phase after going through the basic reproduction operations of mutation and crossover. After that, the individual goes into a simulated annealing phase that performs a complete SA until temperature freezes and the individual becomes mature. At the end of one generation, all individuals should have gone through the same annealing process and are now ready for selection in the GA. The phase of selecting the future population from the current one is now performing this GA operator on an enhanced version of the individuals.

The basic idea of the SAGA technique was originally investigated and implemented as a sequential algorithm in 1988 by Huntley as part of his dissertation. In 1989, Huntley's work was extended by applying a parallel version of the SAGA technique in [14]. The introduction of parallelization allowed performing the annealing process on the good individuals in parallel and thus causing a minor effect on the overall execution time.

In the parallel strategy proposed by SAGA, each of the offspring generated by the GA in a given generation is improved using SA. In most implementations of simulated annealing, the initial temperature is set to an arbitrary high value, encouraging a high degree of randomness in the initial stages of the search. Such a strategy was not appropriate for SAGA, since the initial solution given to the SA is usually quite good. To perform an annealing schedule on such a solution which starts at a high temperature would have almost destroyed the work done by the GA.

The simulated annealing parameters used in the parallel implementation of SAGA did not all conform to the standard implementations. The annealing schedule used a linear operator to update the temperatures, which is the "classic" standard implementation where temperature T is scaled by a constant $\propto$: $T \leftarrow \propto T$; $(0 < \propto < 1)$. However, the implementation used in the SAGA used the following heuristic to find the appropriate initial temperature:

1. Since the pair-wise interchange heuristic has been one of the most commonly used heuristics for solving the QAP, the approximated the expected change in cost incurred by random pair-wise interchange of the terms of the solution. They performed 100 pair-wise interchanges and the mean absolute deviation (MAD) of the cost was calculated.

2. The initial temperature was set to $T_o = \beta$ x MAD; where β is a user-defined probability of accepting an average cost change in the earlier stages of the search.
3. The decrement constant α used in the temperature scaling during the annealing schedule, was set to $\alpha = [T^*/T_o]^{1/\eta}$; where η is the number of temperatures in the schedule.

On the other hand, the GA incorporated into SAGA is also not standard. The main differences introduced in the SAGA made the GA greedier than the simple classic GA. The major differences from the standard GA are the crossover operator and the method used to select parents in the breeding procedure. The crossover operator developed by SAGA is similar to the PMX crossover operator used by Goldberg [36] and Grefenstette [39]. The new crossover operator creates a new offspring from two selected parents by performing the following:

1. Select two points for crossover, such that the parents and the new empty offspring strings are divided into 3 parts.
2. Copy the second section of the first parent into the corresponding section in the new offspring.
3. The first and third sections in the new offspring are filled by copying all terms from the corresponding positions in the second parent which have not been assigned yet.
4. All remaining empty positions in the new offspring are copied from the permutations of the remaining terms in both parents.

To demonstrate the crossover operator, assume P_1=1 2 3 4 5 6 7 8 and P_2=6 4 8 3 7 2 1 5, and crossover points at positions 3 and 5. Thus we have the first section with 2 positions, second with 3 positions and the third with 3 positions.

Second step will produce an offspring:	- - 3 4 5 - - -
Third step:	6 - 3 4 5 2 1 -
Fourth step:	6 8 3 4 5 2 1 7

The selection operator developed in the SAGA technique was also non-standard, as the crossover operator. The first parent is selected at random from among the best *s* solutions, where *s* is a user defined constant. The second parent is selected as in the simple GA, at random from the rest of the population. This type of select operation is based on a rank ordering of the costs.

Results of experiments using this approach have shown that the SAGA was capable of achieving superior solution qualities than that found by steepest descent pair-wise interchange. The runtime performance of SAGA was not as good as the simpler version of the GA or SA for small size problems. But the work [22] claims that for larger sized problems the situation is reversed. This opinion is advocated because for small sized problems it is easy and does not require huge computation times, while large size problems are intractable and difficult to be solved by the simple versions of GA or SA.

UFO Technique

A parallel hybrid technique that combines genetic algorithms and simulated annealing was proposed by [4] to solve MAPs. This method was known as the UFO algorithm. It was very similar to the SAGA algorithm [14]. The network used in this experiment was the 13 node network used in the GALGO research [64]. Abdelbar and Hedetniemi in [4] transformed the singly connected network with 13 nodes into a 14 node multiply connected network by adding a node and creating a cycle using the additional node to the network.

The hybrid method works as follows. It starts by assigning one processor to run the sequential GA. All other processors involved are used for running a simulated annealing. Each SA processor selects a chromosome from the global population and uses it as a starting point for a full annealing schedule. After the chromosome is improved it is inserted again into the population. This model can be seen as a single population that improves through natural selection. The improvement is performed periodically with UFO's that descend to the isolated population and take an individual for enhancing and then returns it back. The new individual now has its new genotype and phenotype in the population and can use its new genes for interaction with other individuals through the evolution process.

Although this algorithm inherits from the SAGA technique, it can be seen as a loosely coupled version of the SAGA. The loose coupling comes from the nature of the improvement process that is not performed sequentially on all individuals. Rather, improvement in the UFO technique is performed in a random and periodic manner because whenever a processor finishes simulated annealing, it takes another individual for a new annealing schedule.

The technique was developed using the PVM [32], which is a public domain package that allows a network of heterogeneous UNIX machines to be used as a single parallel computer. In the implementation of the UFO technique, the master processor had a GA running with 100 individuals representing the entire population. The number of slave processors varied in the experimentation phase. Nevertheless, each slave processor ran a simulated annealing until it has found a globally optimal solution or because the current state has not improved for 25 generations. Once the slave has reached either condition it sends the current state to the master. The master processor at the end of each generation performs a non-blocking receive to see if there are any pending messages from slave processors. If there is a chromosome pending to be admitted into the population, the master processor evaluates it and decides if it can displace one of the existing individuals or not. The master processor then sends another chromosome to the sender for another simulated annealing. The way the master and slave communicate helps in preventing the blocking in execution of either of the algorithms that is found in the SAGA. It is also presents an algorithm that is more suitable on distributed systems since the communications costs here are minimal. The communications cost involved is only limited to the send/receive message from the simulated annealing slave processor to the master GA processor, followed by the send/receive message of the new individual from the master to the slave.

The theoretical optimal solution for the MAP on the 14 node network found using an exhaustive search, was equal to 0.1082. The sequential GA alone was able to find the optimal solution with an average of 368.91 generations. The simulated annealing technique used the same set of parameters, initial temperature equal to 0.005 and reduction factor was equal to 0.1, that were used for the UFO technique and was able to find the optimal solution in 9% of the time. With the UFO implementation and using one processor as a slave, the average number of generations needed to find an optimal solution was 89.8, which represents a speedup of 4.11%. With four processors acting as slaves the speedup was observed to be equal to 16.95.

Adaptive SAGA

In 1994, Esbensen and Mazumder [26] introduced a modified version of the SAGA technique. Their unified genetic algorithm with simulated annealing was applied to the VLSI problem of macro-cell placement. The main characteristic in this new version of SAGA is the dynamic change in GA and SA performance during the optimization process. The dynamic change in the way the technique works makes the algorithm start by being a pure GA that gradually due to stagnation moves into being a pure SA with one individual in the population. This technique can be seen as a GA that has been modified in two major ways:

1. The mutations performed on an individual are accepted with a certain probability as in SA. Each individual has its own temperature, and during its lifetime, its temperature is decreased according to its own cooling schedule.
2. Initially, SAGA performs as a pure GA. As time progresses, the GA stagnates and SAGA gradually switches over to being a SA.

The algorithm starts by generating a random population of individuals. Each individual is assigned the same initial temperature that will start the annealing schedule. An initial evaluation of the fitness of each individual is performed in order to find the best score since it is necessary for the algorithm to keep track of the best individual found so far. At the beginning of each generation the algorithm checks to see if the GA has reached stagnation. It does that by checking if there have not been any improvements for a predefined number of generations. If the GA has not reached stagnation, the GA starts the evolutionary cycle by selecting mates for crossover. Then mutation is performed on the resulting individuals as in traditional GAs.

In this technique, a SA-controlled mutation is introduced which replaces the regular mutation operator. The SA-controlled mutation is the first feature introducing simulated annealing properties to the basic genetic algorithm. The hybrid method described here has replaced traditional mutation with the SA-controlled mutation. The SA-controlled mutation operator works exactly like a standard mutation operator; it gets a solution as input, mutates it and then returns the solution as an output. The difference in its operation lies in that internally, a SA operator exists and calls the evaluation function.

It then uses the cost value in order to decide whether to accept the new state, with a certain probability, or not.

This modified mutation operator works by setting a schedule for reducing the probability of accepting a cost-increasing mutation. The acceptance probability used in SA to generate Markov chains is applied in the same manner after each mutation operator on an individual. But rather than applying a full annealing schedule on each individual until the temperature freezes, the temperature of the individual is modified at the end of a mutation operation, given that the new mutation is accepted.

The second main feature of this SAGA technique is its adaptability to stagnation in the genetic algorithm. The switch towards SA was introduced in order to bring the fine tuning features to GAs once no improvement occurs for a defined number of generations to the value of the best solution found so far. A step towards SA involves reducing the population size and increasing the probability of mutation. This implies that the objective was to increase the number of SA-controlled mutations that are performed on a smaller number of individuals. Each time a reduction in the population needs to occur, the new population size is calculated using an equation controlled by a set of real valued parameters. Pure SA is reached when the population size is reduced until it reaches one. In this case, the reproduction step is equivalent to a mutation that is accepted if and only if it improves cost.

The experimental results produced by applying this adaptive SAGA approach to the macro cell placement problem showed an increase in the quality of solution in comparison with GAs alone. The technique was implemented and tested for a benchmark data used in an earlier work by Esbensen in 1992 [25]. The benchmarks were from the 1992 MCNC International Workshop on Placement and Routing which included Xerox and HP benchmarks. On both benchmarks, the SAGA system when compared to GAs alone and to other systems provided a better quality solution.

Genetic Simulated Annealing

Following these promising attempts to hybridize GAs and SA, a new method called the Genetic Simulated Annealing (GSA) started to evolve. This was introduced in 1996 with [50] that introduced the basic algorithm of GSA as another methodology for incorporating simulated annealing as a GA operator that acts in a manner similar to local hillclimbing techniques. The main idea behind this new hybridization methodology is to use the simulated annealing technique as a genetic operator during the evolution process.

The simulated annealing operator in the GSA technique replaces the basic mutation operator in the genetic evolutionary process. Thus the GA works with three genetic operators, which are the GA-based crossover operator, SA-based local search, and the selection operator for population update. The GSA algorithm starts by a randomly generated population of individuals. It then determines the best solution existing in the initial population and keeps it as the best solution found so far. The GSA then starts the evolutionary process of GA which goes through a predefined number of generations. At the beginning of each generation the temperature of the annealing schedule that will be used is set to a pre-selected initial temperature. Then the worst individual in the population is identified and replaced by a new individual. The new individual is created through a crossover between any two randomly selected individuals. That individual before being inserted into the population goes through an annealing schedule. The annealing process keeps on working until the temperature reaches zero or the system is frozen. At that point, an update of the best local individual found so far occurs and the old one is replaced with the annealed individual if its fitness is higher. The process keeps on going until all the generations are exhausted.

The GSA technique was applied to the non-slicing floorplan problem. For a given set of arbitrary shaped and fixed size modules, and connection information among these modules, the floorplan problem is defined as finding the minimum area placement and the shortest wire length for the placement of these modules. The experiments ran on benchmark data which had approximate optimal solutions using simulated annealing. The benchmark used in this experiment had 149 modules in the floorplan problem. The GSA and SA were applied for five runs with the different initial floorplans generated at random. Both methods were bound by the same amount of CPU time in order to complete their executions. When the results obtained through applying GSA was compared to SA results, the earlier technique was found to produce better solutions. The average reduction in chip area for the floorplanning problem using GSA was 12.4%. As well as the reduction in chip area, the hybrid method introduced a reduction in the total wire length by an average of 2.95%.

Parallel Genetic Simulated Annealing

A recent research in the domain of hybrid genetic algorithms and simulated annealing introduced a massively parallel version of the GSA, PGSA, in 1998 [18]. The first major characteristic of the new algorithm is that it is a massively parallel algorithm that is suited for implementation on Single Instruction-Multiple Data (SIMD) architectures. A major enhancement on the basic idea of GSA which was introduced through this algorithm is that each processing element runs a full annealing schedule with the use of crossover, mutation and selection as part of the annealing core algorithm. The mechanism of hybridizing GA and SA in this method relies on combining the recombinative powers of GA (via the crossover operator) and the annealing schedule.

The new algorithm could be seen as a genetic algorithm with each individual in the population residing on a different processing element. Each individual goes through a predefined number of generations, this number represents the length of the genetic evolutionary cycle. The PGSA algorithm introduced here does not require parallelization of any problem specific portions of a serial implementation, existing serial implementations are incorporated as is. The algorithm is defined by the following pseudo-code:

```
1. Set temperature on processor to be $T_o$.
2. Create a random solution(r).
3. for I = 1 to MaxIteration do
   a) Create a random direction.
   b) Create a random distance.
   c) Receive a new individual from another processor based on values of
      distance and direction.
   d) Perform mutation, then do crossover using resident individual and new
      individual in order to produce 2 new children.
   e) Perform a selection operation on three individuals: the resident individual
      and the two children using the current temperature.
   f) Decrement the current temperature according to annealing schedule.
```

In PGSA the inter-processor communication as defined above allow for implicit synchronization among processors. If this algorithm is implemented on a SIMD architecture where processors communicate through one thread of instruction then synchronization while receiving and sending is achieved automatically. Each processor would only communicate with the nearest *n* processors and would generate a random value for the distance and direction representing the specific processor to communicate with. During any iteration, all processors use the same random seed for generating the distance and direction of the other processor. This restriction enables coordination among processors without explicit communication and ensures that each processor chooses a unique pairing candidate. The benefit of implicit synchronization is apparent when each processor needs to perform data transfers. Since all processors follow the same single instruction stream, each processor knows from which processor it has received a message and to which processor it should transmit data. Thus, no explicit synchronization and identification protocols are needed for each inter-processor transfer.

Another major characteristic of the PGSA algorithm is the selection operation. The selection process chooses a single candidate solution that will replace the resident solution from the current resident solution and the newly generated children from crossover operation. The selection operation uses only locally available information to make its decision. No global coordination or ranking is required while performing selection. The process of selection in its new version represents a modification of the standard stochastic selection criterion in simulated annealing. The standard selection in SA is modified in the PGSA algorithm in two ways. First, a deterministic selection criterion is used, where a new candidate is accepted if the cost increase is less than or equal to the current temperature. This deterministic criterion defined in [58] has been found to be much more efficient to implement and has been empirically to perform equivalently to the more expensive stochastic criterion. The selection operator in this implementation is broken down into three steps: first a select occurs between resident solution and child one, then a select between resident solution and child two is performed, and finally a select between the two children is done. The result of the three step selection operation is based on the cost difference between any two competing candidates and whether this difference is higher than the current temperature or not.

The significant contribution of this method in the field of parallel hybrid GAs and SA, is that none of the problem-specific portions of the GA or SA require parallelization. The individual crossover and mutation operators as well as the fitness function evaluation are replicated on each processor as if they were independent serial algorithms working on parallel nodes. This allows the algorithm implementation to be easily modified to function for another application.

The research on PGSA in [18] was mainly concerned with determining the effectiveness of the method based on three criteria:

1. Whether or not, PGSA can take advantage of GA and SA to overcome their weaknesses.
2. Whether or not GSA can utilize massively parallel architectures with high efficiency.
3. Whether GSA can be effective across a diversity of different tasks.

In order to address the first criterion, a set of experiment were performed using two types of annealing schedules. To address the second one, another set of experiments were done that kept increasing the number of processors from 256 to 16K. The last criterion was tested by applying the pervious two sets of experiments to the TSP problem and the problem of a 24 word, 12 bit error correcting code (ECC).

The first set of experiments compared a Uniform Temperature schedule with a Random Temperature schedule. The Uniform Temperature approach set the initial temperature to a value such that the initial acceptance probability is 0.97. Final temperature is set similarly such that the final acceptance probability is 10^{-10}.

Temperature is then lowered with a certain step such that the final temperature is achieved upon completion of the maximum number of generations. The initial temperature, final temperature and the decrement step are replicated across all processing nodes. In comparison, the Random Temperature approach uses a set of random values for initial and final temperatures across processing nodes. In this cooling schedule, the initial temperature is selected on each node by random initial acceptance probabilities between 10^{-10} and 1.0. The final temperatures are selected in the same manner in the range of 0 to 10^{-10}. The comparison of the two approaches is performed for population sizes ranging from 256 to 16k. Using the TSP problem for the first set of experiments that compare annealing schedules, the Random Temperature schedule produced similar or better results than the Uniform Temperature schedule, regardless of the population size. The Random Temperature schedule was found to make steady progress immediately after the algorithm starts, whereas the Uniform Temperature schedule made little progress during the initial period due to the high temperature at the beginning of the run. In the Random Temperature schedule the main advantage is that the PGSA not only is storing a diversity of useful genetic material, but also a diversity of annealing schedules. This characteristic helps in maintaining a good balance between diversity and disruption in the population. Using the random approach, disruption is kept low by protecting good solutions from being destroyed. Distributing various annealing schedules among the processors allows processing nodes with a low temperature good solutions are not disrupted easily. At the same time, other processing nodes that might have a high temperature provide diversity in solutions.

The second set of experiments was concerned with answering the question of whether better performance or shorter execution time can be achieved by simply adding processors. The most striking characteristic observed was the inversely proportional relationship between the number of processing nodes and the time needed to reach a near optimal solution. This was more evident in the TSP problem than the ECC. In the TSP experiments, the constant of proportionality approaches one for finding a solution within 2% of the optimal. This means that we halve the execution time by doubling the number of processors. The characteristic of improved performance with increased population size appears unique to PGSA. It has been observed that other GA systems with hybrid methods of annealing, such as [54], demonstrate such performance. Rather in [54] it has been shown that when the population size exceeds a certain range the performance begins to decrease. The key element behind the good scale-up in behavior is not only recombination effect of GAs. Rather, the SA originated annealing schedule that PGSA utilizes contributes with the recombination power of GA to reach this increase in performance as the population is increased. In a traditional GA system, proportional selection selects solutions into a new population using the roulette wheel mechanism. Thus, each solution occupies a certain percentage of the wheel space based on its fitness. This percentage is independent of the population size. As a result, above average solutions replicate themselves in the new generation, and low fitness solutions gradually drop out. Diversity can be quickly lost using this selection process. Increasing the population size does not help keep diversity from being lost. In PGSA, a healthy diversity is maintained by using a SA-type annealing schedule in place of selection. This was particularly evident in the experiments with Random Temperature schedules, where a diversity of annealing schedules exist. Processing nodes with high temperatures accept new solutions at greater rate, and help maintain diversity. While nodes with lower temperatures, keep good solutions. Furthermore, PGSA was designed such that no visiting solution is allowed to replace the resident solution. The reason for that is to minimize the ability of an individual good solution to propagate and dominate the population.

Simulated Annealing Mutation & Recombination

In 1993 [9], a GA/SA hybrid technique was introduced which extrapolates the acceptance probability scheme from simulated annealing onto the mutation and recombination genetic operators. The basic idea of this hybridization is to use the SA stochastic acceptance function internally to limit adverse moves. The technique proposes the Simulated Annealing Mutation (SAM) and Recombination (SAR) operators to act in place of the traditional mutation and recombination operators existing in the simple GA.

The SAM operator works exactly like a regular mutation operator. It receives a candidate and performs any random or problem specific mutation operator on it. Traditionally, the mutation operator would have no further role in the selection of the new population. The SAM on the contrary, takes the new offspring and its parent to perform an acceptance trial between them. The acceptance trial works like the regular SA with a temperature schedule that contributes to the probability of acceptance for the new individual.

The Simulated Annealing Recombination operator works in a similar manner to that of the SAM operator. It takes two parents from a select operation and performs on them the regular crossover operation. After which, each of the two new offspring is compared to the best performer of the two parents. The winning two individuals from this acceptance trial get to live in the new population. From the point of view of the GA theory, the SAM and SAR operators work in the same way as their counterpart traditional operators in a simple GA. Thus, the Schemata Theory will still work correctly in terms of convergence analysis, since the theory does not assume any specific behavior on the genetic operators.

Not only does this hybrid technique present an enhancement to the GA, but rather from another point of view, [9] advocates its advantages for the simulated annealing algorithm as well. The hybrid technique is seen to extrapolate the notion of population onto the simulated annealing. So, instead of working with an individual solution a simulated annealing is presented that works with a population of individuals.

The SAM/SAR operators combined together represent a mechanism for introducing new solutions based on fitness selection mechanism. Having the possibility that each individual is operated on by the mutation and recombination operators several times, helps in creating neighborhoods around a specific individual which get explored through the genetic operators. The convergence properties of SA are also not affected by its integration within the GA. Each individual in any generation is the successor of another member in the previous generation, which maintains the Markov Chain model within GA domain. It is though possible to regard this technique as a parallel simulated annealing approach.

The performance of the hybrid GA-SA algorithm with SAM/SAR operators was compared on the problem of training a feed-forward neural network to predict the chaotic logistic map: $x_{t+1} = 4x_t (1 - x_t)$. The network had two inputs, x_t and x_{t-1}, one fully connected hidden layer of 10 sigmoidal units, and had one output x_{t+1}. This problem involves optimizing 50 weights. Backpropagation is able to optimize the weights in 450 iterations through 250 input vectors, achieving an out-of-sample R^2 of 0.98. The objective function when using the hybrid technique was the mean squared-error of running the network forward with the given weights, compared to the true output for the input sample. After running the learning phase, the neural network was run using an out-of-sample set to 750 vectors. The performance of the hybrid technique out performed pure GA by an order of magnitude. For 250 generations, the R^2 using pure GA was found to be equal to 0.08, while with the hybrid method it was 0.81.

Incorporating Genetic Algorithms into Simulated Annealing

Parallel Recombinative Simulated Annealing

Goldberg and Mahfoud in 1993 [54] introduced a parallel method of simulated annealing. The Parallel Recombinative SA (PRSA) method borrows from genetic algorithms to develop an effective combination of SA and GAs. The new algorithm strives to retain the desirable asymptotic convergence properties of SA [33], while adding the population approach and recombinative power of GAs. Goldberg's paper on Boltzmann tournament selection [35], introduced as a formal approach for hybrid GAs and SA, have motivated the research leading to the development of PRSA.

Parallel recombinative simulated as an algorithm can be regarded as simulated annealing with multiple versions running in parallel and possessing both mutation and crossover. The mutation is used as the neighborhood operator, and crossover recombining independent solutions. Both mutation and crossover in the GA sense represent the new neighborhood operator for the PRSA technique.

The PRSA algorithm extrapolates the convergence characteristic of simulated annealing through the tournament selection that it performs among the selected parents and generated children. In PRSA, this is called the Boltzmann trial. The Boltzmann trial works by performing a competition between two individuals *i* and *j*, where *i* wins with

Boltzmann probability equal to $1/(1+e^{(E_i\ -\ E_j)/T})$. Two possible competition forms were considered for the PRSA technique. The first, *double acceptance/rejection*, allows both parents to compete as a unit against both children. This means that the sum of the two parents' energies should be substituted for E_i in the Boltzmann probability equation and the sum of energies of the two children is used as E_j. The second method of acceptance is known as single acceptance/rejection. This mechanism holds two competitions, each time pitting one child against one parent, and accepting the parent with probability $1/(1+e^{(Eparent\text{-}Echild)/T})$. The problem with the second mechanism is that we need to identify which child will compete with which parent. In the experiments of [54] it was decided to select a parent for competition with the child formed from its own right-end and from the other parent's left end. The sequential Parallel Recombinative Simulated Annealing algorithm is described as:

```
I.    Set initial temperature.
II.   Initialize the population randomly.
III.  Generate a new population from the current one by:
   A.    Do for number of times equal to half the population size:
            1.    Select 2 parents at random from the entire population.
            2.    Generate 2 new children using recombination operator (crossover),
                  followed by a neighborhood operator (mutation).
            3.    Perform a Boltzmann trial between children and parents.
            4.    Overwrite the parents with the trial winners.
   B.    Periodically lower the temperature.
```

[54] presented two parallel algorithms, one synchronous and one asynchronous for the PRSA. The prototype synchronous algorithms employ a synchronization barrier represented by the end of the generation. The synchronous PRSA had a node processor representing the master. This master performed for *G -1* generations; where *G* is the total

number of generation, a random shuffling of the individuals in the population, then partition the population and send each host/slave processor its part in the overall population. The master would then wait until it receives the elements from slave processors in order to iterate for the next generation.

The host or slave processors each starts initially before performing any annealing or send/receive of individuals by setting the initial temperature. This temperature is the same value on each processor. The decrease in temperature value was performed after each receive/send of individuals which occurs for *G* times; where *G* is the total number of generations. For each generation, the slave processor applies crossover and mutation to all the individuals. It then performs a Boltzmann trial in order to find winners to be included in the next generation. The temperature is then reduced and the new sub-population is sent to the master processor. The slave waits until the master re-sends another set of individuals to constitute the new population.

The synchronization barrier is employed using the synchronous send/receive that the master processor performs when it waits for all slaves to send their newly generated population. Due to that, the asynchronous version of the PRSA was created by repeating the host/slave algorithm defined above on all processors; including the master. The send/receive of populations was thus employed to/from any processors. In [54] they used the same random number generators at each processor in order to arrive at destination nodes for each sub-population. The PRSA was applied as a prototype on a CM-5 (Connection Machine).

While it has been pointed out that SA can be viewed as a GA with population size one, the opposite claim, that a GA can be a special case of SA, seemed extraordinary at first. In the PRSA research, proof of convergence was given through such claim. This claim was based on a variation of the PRSA algorithm used, which says that self-mating is not allowed. If all strings in a PRSA population are concatenated in a side-by-side fashion so as to form one superstring, we will define the fitness of such string as the sum of the individual fitnesses of its component substrings. They introduced the cost as being the negated fitness of the superstring. The cost function will reach a global minimum only when each substring is identically at a global maximum. Thus, to optimize all elements of the former population, an algorithm can search for a global minimum for the cost function assigned to its superstring. Considering the superstring as our structure to be optimized, the crossover-plus-mutation neighborhood operator is applied to selected portions of the superstring to generate new superstrings. Crossover-plus-mutation's net effect as a population-level neighborhood operator is to swap two blocks of the superstring, and then probabilistically flip bits of these swapped blocks and of two other blocks.

Not allowing self-mating is what brings the convergence of SA into effect. There are two conditions on the neighborhood generation mechanism sufficient to guarantee asymptotic convergence. The first condition requires that it be able to move from any state to an optimal solution in a finite number of transitions. The presence of mutation satisfies this requirement. The second condition is symmetry. It requires that the probability at any temperature of generating state y from state x is the same as the probability of generating state x from state y.

The Annealing Genetic Approach

The annealing-genetic approach was introduced in [51]. The research presented a new method for hybrid GAs and SA. The hybrid method was intended to design an efficient annealing schedule for simulated annealing. The genetic algorithm's techniques were incorporated into the simulated annealing in order to design the annealing schedule. The key issues that represent critical factors in the design of simulated annealing schedules are:

1. The best values for the initial temperature of the annealing schedule.
2. How to determine an effective length of Markov chain at each temperature.
3. Detecting the equilibrium condition of the system at each temperature.
4. How to prevent the system from getting trapped into a local minimum.
5. How to know that the system has frozen.
6. Finding the most effective decrement ratio of the temperature.

The annealing genetic approach tried to provide good solutions for the above critical factors affecting simulated annealing overall performance using genetic algorithms' features such as:

1. GAs maintain a population of solution which reduces the probability of getting trapped into local minimum points.
2. Selecting parents based on their fitness values means that parents with a higher value always have a higher probability of contributing one or more offspring in the next generation. In contrast, parents with lower fitness values still have a chance of crossover and escaping from local minima is higher than in simulated annealing.

To start solving the key issues in simulated annealing, the research of [51] introduced the following solutions that incorporated the advantages of GAs:

- Initial Temperature Value: The annealing-genetic approach defined an equation for calculating the initial temperature for the annealing schedule. The equation was reached through a sequence of steps that represent a pre-processing phase of the overall algorithm. First, an initial random population is created P_0, then the regular genetic operators are applied on it once to produce an intermediate population. The second step is to take the intermediate population and apply on the worst individual the following. Take the worst individual, perform a Markov chain on it and if the resulting state is better insert it into a new population called P_1, then continue generating other states using the same Markov chain. If the new state is worse, then stop this Markov chain, select another individual from the intermediate population and start the process again. Finally, the initial pre-processing stage stops when a new population P_1 has been completely generated. The researchers in [51] set a predefined value for the acceptance probability equal to 0.6. They used the Metropolis criterion for acceptance, $P = exp(-\Delta C/T)$. Using the value of P to be 0,.6, the initial temperature can be defined as $T = -\Delta C/ln\ 0.6 \approx 2\Delta C$. Thus, the initial value of temperature is defined as the difference between the highest and lowest cost in the new population called P_1 divided by half the population size. $T=\Delta C/(pop\_size/2)$.

- The length of a Markov chain: the Markov chain length is bounded by the population size, in particular, the chain is generated from multiple states of the population. The sequence of states in the Markov chain are generated by starting from a state having the lowest fitness in the population, a next state is generated from that state using the regular neighborhood generation strategy. If the next state is better it is placed into the new population and it becomes the current state. Otherwise, the Markov chain is stopped by inserting the current state into the new population and selecting another starting state from the old population to begin another Markov chain.

- The quasi-equilibrium of the system: The quasi-equilibrium of the new population at each temperature is achieved by applying the regular genetic operators on the population created from the Markov chain step. Thus any population P_n becomes P_{n+1} after a full Markov chain followed by applying crossover and mutation operators.
- System frozen: Essential to the convergence proof for the system is the fact that the average cost of each generation is less than or equal to the last one. The state of the lowest cost in each generation is kept as the local best found so far. When 80% of the population in a certain generation having their costs equal to the best solution found so far, the frozen condition is signaled in order to indicate the annealing genetic algorithm must stop.
- Decrement ratio of the temperature: the ratio by which temperature is decremented was defined in an adaptive manner. The temperature is decremented by the following equation $T_{k+1} = T_k * \alpha$. The adaptive rule for the value of α is: (1) $\alpha = 0.5$ representing a fast annealing schedule as long as the difference of average cost between the last and the current generations is large, (2) $\alpha = 0.95$ for a slow annealing while the difference is small. (3) In the average case, α is set to 0.85.

Experiments of the annealing genetic algorithm on the TSP problem in [51] proved that it could find better solutions than simulated annealing applied alone. The solutions found by the annealing genetic approach reduced the gap between them and the best known solution in comparison with the gap in the solution values obtained by simulated annealing alone. In their results, the authors of the annealing genetic approach prove that their algorithm for the TSP problem is of $O(n^2)$. Their proof is not demonstrated in a mathematical fashion which makes it difficult to repeat on different applications. Rather they form their empirical analysis of the algorithm on the square root value of moves for each number of cities as well as the square root of the turnaround time of the algorithm. Although the experimental results do not show a greater enhancement in the solution quality when compared to the best known solution, they do exhibit a reduction in the total number of moves is achieved. This reduces the total amount of time needed to find an approximate solution.

CHAPTER V

A NEW HYBRID GA/SA TECHNIQUE

Simulated Annealing as a Genetic Operator

In the following section, the driving motivation for this research is presented. The technique that is introduced as the main contribution of this research can be described as a simulated annealing which acts as a genetic operator. The traditional techniques that have been reviewed earlier in combining genetic algorithms and simulated annealing had the tendency of either carrying the implicit parallelism of GAs into simulated annealing or taking the global convergence characteristics of SA into GAs. The analysis of the results of these combinations have shown that experiments were moving towards an area of great potential in solving difficult combinatorial optimization problems.

The technique developed through this research introduces simulated annealing to the core of the genetic algorithm as a new intelligent operator. In the standard genetic metaphor, random crossover and random mutation are meant to simulate a large number of generations of evolution. Mutation as a genetic operator plays a significant role in the diversity of the genetic population. The traditional mutation operator randomly generates new characteristics that would not be produced otherwise, by crossover for example. The advantage of random mutation is that it is very cheap computationally, which allows a large number of generations of slow evolution to be achieved.

The role which simulated annealing plays in the new technique, can be thought of it as an intelligent or directed mutation operator compared to the random (undirected) mutation in genetic algorithms. The simulated annealing operator in this new context, can be thought of as simulating "evolutionary spurts". Thus some elements of the population can be regarded as going through an infrequent maturity cycle relative to pure random mutation. Simulated annealing can be seen as carrying out a function similar to random mutation with survival of the fittest. If an application of the random mutation operator produces a genome that has lower fitness than the original, the new chromosome may still be admitted to the population. In the annealing case, an examined neighboring state may be admitted as the new current state even if its fitness is lower than the current state. Moreover, in evolution, as time passes and the population fitness increases, the population becomes more selective of the individuals contributing to the new generation. Thus it becomes more difficult for an unfit individual to gain admittance. In simulated annealing, as time passes and the temperature cools, the system becomes more selective in accepting new states.

The advantage of introducing an intelligent mutation such as simulated annealing, is that simulated annealing can be described as allowing a greater degree of non-conformity: a newly created state always has a non-zero chance of becoming the current state (since the exponential decay function which represents the probability of accepting a new state: $e^{-\Delta/T}$ never reaches zero in finite time). Whereas, with random mutation, if the newly produced chromosome has a fitness lower than that of the least fit member of the population it typically has no chance of being explored. Furthermore, this conformity pressure increases the more time that has passed since the start of the simulated evolution process, since we expect the overall population fitness to increase with time. The use of annealing offsets this conformity pressure because each application of the annealing operator starts with a high temperature even when the general population has reached a high level of fitness. This increases the likelihood of discovering a good solution that lies in a bad neighborhood of the search space.

Table 1: Intelligent mutation compared to other hybrid GA/SA methods.

Technique	Similarities	Differences
SAGA	• The entire simulated annealing algorithm exists inside the genetic algorithm without any alterations. • Initial temperature value and the cooling schedule are global among all individuals in any generation. • The crossover, mutation and selection operators are not altered in any way.	• In SAGA, the individuals must undergo an annealing schedule. This means that annealing is tightly coupled within the genetic operators. • In our hybrid technique, the simulated annealing does exist as a genetic operator but with a certain probability, which allows it to be loosely coupled. Thus, not all individuals in a generation go through an annealing schedule.

Technique	Similarities	Differences
Adaptive SAGA	• The crossover and selection operators are not altered in any way. • Both techniques have a modified version of the mutation operator, but the internal workings of this modification is specific to each one.	• In adaptive SAGA, the concept of acceptance probability is carried from SA into the mutation operator within the genetic algorithm. The mutation works in the traditional manner, but accepts the newly created offspring with a probability based on the individual's temperature schedule. • In the hybrid SA intelligent mutation, the mutation becomes a simulated annealing schedule with a specific defined probability. The traditional mutation when performed acts in the same manner as in a simple GA. • In adaptive SAGA, each individual is assigned an initial temperature value that might differ from other individuals. It is also assigned a cooling schedule different that other individuals. • In the hybrid SA intelligent mutation, all individuals when annealed start at the same initial temperature and go through the same annealing schedule.

		• In adaptive SAGA, the algorithm starts by a certain population size and gradually decreases this population each time the GA goes through a stagnation period. • In the hybrid SA intelligent mutation, the population size is kept constant for all generation runs and is not adaptive in any sense.

Technique	**Similarities**	**Differences**
GSA	• The entire simulated annealing algorithm exists inside the genetic algorithm without any alterations. • Initial temperature value and the cooling schedule are global among all individuals in any generation. • The crossover and selection operators are not altered in any way.	• In GSA, the mutation operator is removed and replaced with the local search technique of SA. In this technique, all individuals go through three genetic operations: crossover, SA, and finally selection. • In the hybrid SA intelligent mutation, mutation still exists in the genetic engine and we have four operators: crossover, mutation or intelligent mutation with SA, and finally selection. All individuals still go through mutation, but in reality it depends on the probability values which type of mutation is actually performed.

Technique	Similarities	Differences
PGSA	• The crossover and mutation operators are not altered in any way. • All individuals in the population start at the same initial temperature and go through the same annealing schedule.	• The PGSA is a massively parallel hybrid SA/GA technique. • The hybrid SA intelligent mutation technique is a purely sequential algorithm. • The PGSA, has an individual residing on each processor. • The hybrid SA intelligent mutation technique has all the population residing on one processor. • In PGSA, the acceptance probability in SA is carried to the selection operator. • In PGSA, the selection operator is performed after regular mutation and crossover are performed on a processor. The selection works with a certain acceptance probability among three competing individuals: the two newly created offspring and the second parent which did not initially reside on the local processor. The selection should produce one winner which will become the local individual residing on the processor.

Technique	Similarities	Differences
SAM/SAR	- The selection operator is not altered in any way. - Both techniques have a modified version of the mutation and crossover operators, but the internal workings of this modification is specific to each one. They are similar in that the basics of each operator are not modified. - Individuals in both techniques have the same initial temperature and go through the same annealing schedule.	- SAM/SAR technique carries the concept of acceptance probability inside SA to the mutation & crossover operators within the genetic algorithm. - In SAM/SAR both mutation and crossover work in the traditional manner, but accepts the newly created offspring with a probability based on the global temperature. - In SAM/SAR the crossover acceptance probability performs the competition between each one of the two newly generated offspring and the best of the two parents. - The hybrid SA intelligent mutation introduces a modified version of this operator. The mutation operator performs a complete simulated annealing schedule with a specific defined probability. When traditional mutation is performed it acts in the same manner as in a simple GA.

CHAPTER VI

BAYESIAN BELIEF NETWORKS

Uncertainty in Artificial Intelligence

Diagnostic reasoning, also known as abductive reasoning or explanation, is an important problem in artificial intelligence and has several applications such as natural language understanding, medical diagnosis, circuit fault diagnosis, common sense explanation and pattern recognition.

Artificial intelligence (AI) has been constantly occupied with solving problems that lie in the probabilistic domain. The major problem with this class of problems is that they do not provide all the data needed about their environment. This makes solving under uncertainty very difficult for AI agents which posses limited knowledge about the domain of interest.

Acting under uncertainty has developed into an important aspect in the field of AI and from it have emerged theories for handling uncertain knowledge. Agents working in problems with uncertainty cannot make rational decisions easily, because the decision will depend on both the relative importance of various goals as well as the likelihood, and degree, to which they will be achieved.

Handling uncertain knowledge is not easy with methodologies of first-order logic. The reason for that is they lack theoretical and practical information about the domain. The only resort would be to define a degree of belief in certain relationships and based on these beliefs come up with the best possible suggestions for acting under uncertainty. The main tool for dealing with degrees of belief will be the probability theory, which assigns a numerical degree of belief, between 0 and 1, to sentences defining relationships. Probability will provide a way of summarizing the uncertainty that comes from our laziness and ignorance of the complete domain. This probability could be derived from statistical data or from some general rules, or from a combination of evidence sources.

Probability statements are different from sentences developed in propositional or first-order logic. This difference comes from the fact that a sentence in the latter type is either true or false depending on the interpretation and the world; it is just true when the fact it refers to is the case. While in the probabilistic domain sentences do not have the same semantics, because the probability that an agent assigns to a proposition depends on the percepts it has received from the surrounding environment. This is usually referred to as the evidence. Thus, an assignment of probability to a proposition is analogous to saying whether or not a given logical sentence (or its negation) is entailed by the knowledge base, rather than whether or not it is true. Just as entailment status can change when ore sentences are added to the knowledge base, probabilities can change when more evidence is acquired.

All probability statements must therefore indicate the evidence with respect to which the probability is being assessed. As the agent receives new percepts, its probability assessments are updated to reflect the new evidence. Before evidence is obtained we know that it is unconditional probability, after the evidence is obtained we acquire conditional probability. In most cases, an agent will have some evidence from its percepts and will be interested in computing the conditional probabilities of the outcomes it cares about given the evidence it has. In some cases, it will also need to compute conditional probabilities with respect to the evidence it has plus the evidence it expects to obtain during the course of executing some sequence of actions.

The presence of uncertainty changes radically the way in which agents make decisions. A logical agent typically has a single (possible conjunctive) goal, and executes any plan that is guaranteed to achieve it. An action can be selected or rejected based on whether or not it achieves the goal, regardless of what other actions achieve. When uncertainty enters the picture, this is no longer the case. To make such decisions, an agent must first have preferences between the different possible outcomes of the various plans. We need to define a utility theory to represent and reason with preferences.

Along with basic probability statements are the axioms of probability that specify constraints on reasonable assignments of probabilities to propositions. Agents that violate these axioms will behave irrationally in some circumstances. The three basic axioms of probability are in fact sufficient to constrain the probability assignments that an agent can make to a set of propositions. Joint probability distributions complete the assignments we perform. The joint probability distribution specifies the probability of each complete assignment of values to random variables. Baye's rule allows unknown probabilities to be computed from known stable ones. In the general case, combining many pieces of evidence may require assessing a large number of conditional probabilities. Conditional independence brought about by direct casual relationships in the domain allows Bayesian updating to work effectively even with multiple pieces of evidence.

Any graphical representation of statistical or causal dependence constitutes an important component when probabilistic world knowledge is used. Three representations are of interest, these are Bayesian belief networks [59], Markov random fields, and a generalization of the latter, Markov networks. In all these cases, graphical representation makes the dependencies in our world knowledge explicit, as well as saving many orders of magnitude in the size of the representation. If our model has n random variables then representing the complete distribution might require space exponential in n. By using the structure (and independence assumptions) available in the graphical representation, and a very sparse graph, space linear in n is frequently sufficient.

Definition of Bayesian Belief Networks

Bayesian belief networks are a popular graphical representation for reasoning uncertainty. The main advantage of probabilistic reasoning over logical reasoning is that it allows the agent in the search space to reach rational decisions even when there is not enough information to prove that any given action will work. BBNs have been applied to various domains ranging from natural language processing to medical diagnosis and computer vision.

The main idea of this graph is based on the Bayes probability rules. The advantages of using Bayes conditional independence rules is that they simplify the computation of query results and greatly reduce the number of conditional probabilities that need to be specified. A belief network graph is one in which the following characteristics are satisfied:

1. A set of random variables make up the nodes of the network.
2. A set of directed links or arrows connects pairs of nodes. The intuitive meaning of an arrow from node X to node Y is that X has a direct influence on Y
3. Each node has a conditional probability table that quantifies the effects that the parents have on the node. The parents of a node are all the nodes that have arrows pointing to it.
4. The graph has no directed cycles (thus it is a directed acyclic graph, DAG).

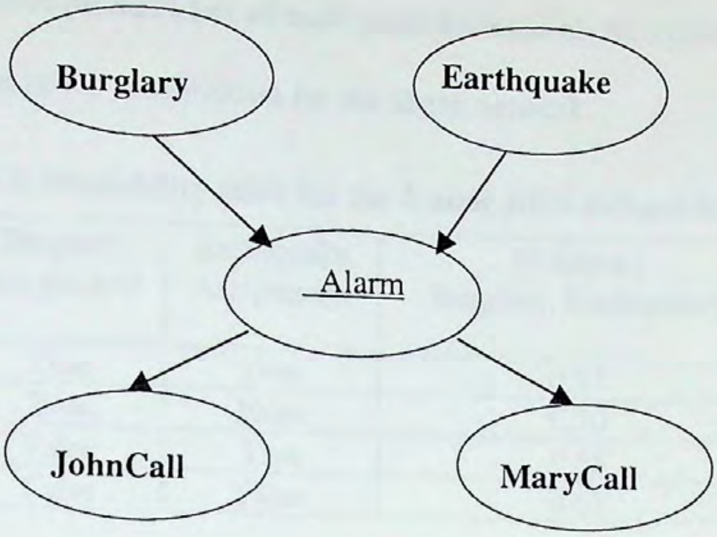

Figure 7: BBN diagram showing topology of a 4 nodes network.

In order to use BBNs for probabilistic reasoning we have to develop a topology for the network. This topology decides what direct conditional dependence relationships hold in the domain. We need only specify conditional probabilities for the nodes that participate in direct dependencies. The topology of the network can be thought of as an abstract knowledge base that holds in a wide variety of different settings, because it represents the general structure of the causal processes in the domain rather than any details of the population of individuals. Fig. 7 illustrates a simple topology of a 4 nodes network. Once we have specified the topology, we need to specify the conditional probability table for each node. Each row in the table contains the conditional probability of each node value for a conditioning case. A conditioning case is just a possible combination of values for the parent nodes. Each row in a conditional probability table must sum to 1, because the entries represent an exhaustive set of cases for the variable. Hence, only one of the two numbers in each row is independently specifiable.

In general, a table for a Boolean variable with n Boolean parents contains 2^n independently specifiable probabilities. A node with no parents has only one row, representing the prior probabilities of each possible value of the variable. The following table is an example of the probabilities for the above network.

Table 2: Probability table for the 4-node BBN defined in Fig. 7

Burglary Assignment	Earthquake Assignment	P(Alarm | Burglary, Earthquake)
True	True	0.95
True	False	0.70
False	True	0.55
False	False	0.65

In a formal and theoretical context we can define BBNs to be:

Let $V = v_1,\ldots\ldots,v_n$ be a finite set of random variables. Let P be some probability distribution over V and let X, Y and Z be three disjoint subsets of V. X is said to be conditionally independent of Y given Z if and only if for any given instantiations x, y and z of X, Y and Z respectively,

$$P(x \mid y, z) = P(x \mid z) \textit{ whenever } P(y, z) > 0 \tag{10}$$

Let $D = (V, E)$ be a DAG whose nodes are identified with the random variables $v_1,\ldots\ldots,v_n$. D is said to be a minimal dependency map of P if and only if every $v \in V$ is conditionally independent, given its parents $\pi(v)$, of all its non-descendants. Given a set V of random variables and a probability distribution P over V, a Bayesian belief network representation of P is a DAG (V, E), such that (V, E) is a minimal independency map of P, augmented with a set of conditional probability distributions $\{P_v\text{: } v \in V\}$ where each P_v is a local probability distribution which specifies the probability of each possible instantiation of v given every possible instantiation of its parents. For example, if node v

is identified with a binary valued random variable and has two parents, say u and w, which are identified with binary valued variables, then P_v must specify for each of the four possible instantiations of $\pi(v)$, (i.e., $\{u \leftarrow \mathrm{T}, w \leftarrow \mathrm{T}\}$, $\{u \leftarrow \mathrm{F}, w \leftarrow \mathrm{T}\}$, $\{u \leftarrow \mathrm{T}, w \leftarrow \mathrm{F}\}$, $\{u \leftarrow \mathrm{F}, w \leftarrow \mathrm{F}\}$) the two probabilities $P(v = T \mid \pi(v))$ and $P(v = \mathrm{F} \mid \pi(v))$. However, since for any given instantiation of $\pi(v)$, the probability of $v = \mathrm{T}$ and $v = \mathrm{F}$ must sum to 1, it is sufficient for P_v to specify one probability, conventionally the probability of true for binary valued variables, for each possible instantiation of $\pi(v)$. In general, if a variable is n-valued, P_v must specify $n{-}1$ probabilities for every possible instantiation of $\pi(v)$.

Thus for any given instantiation, $v_1,\ldots\ldots,v_n$, of a DAG representing a Bayesian belief network, the joint probability of that network can be computed through the following equation:

$$P(v_1,\ldots\ldots,v_n) = \Pi_{i=1}\, P(v_i \mid \pi(v_i)); \quad \forall\, 1 < i < n \qquad (11)$$

Most Probable Explanations

Two abductive reasoning problems exist on BBNs. These are probabilistic inference and maximum-a-posteriori (MAP) explanations, also known as most probable explanations (MPE). In probabilistic reasoning, the objective is to compute the probability of a given set of events conditioned on a given set of observations called the evidence. In other words, the objective is to find the highest probability assignment, or explanation, of A for a BBN B that is consistent with a set of observations or evidence $\mathcal{E}$.

For the special case of singly connected networks, exact polynomial time (linear) algorithms exist for solving MAPs. Singly connected belief networks are graphs that have the following property: between any two nodes in the graph there exists at most one directed path between them. In 1988, Pearl [59] developed for singly connected networks a message passing scheme for finding the optimal MAP on a BBN in linear time.

However, for multiply connected BBNs, the problem is *NP*-hard and even approximating it with a bounded degree of accuracy has proved to be *NP*-hard. In 1998, [5] proved that approximating MAP explanation is *NP*-hard.

Finding MAPs on BBNs using GAs

Several methods have been developed for finding approximate MAPs for multiply connected Bayesian belief networks. Research in this domain includes using genetic algorithms [64], simulated annealing [4] and lately integer programming [65] and neural networks [3].

The GALGO [64] genetic algorithm model developed by Rojas-Guzman and Kramer is the earliest and most recognized work for solving MAPs using GAs. The aim of the research was to explore the potential which GAs had for finding near-optimal solutions for large and multiply connected Belief networks. The GA algorithm developed was applied to four Belief networks: the first one modeled a chemical engineering plant and three others were artificially generated. The first network was a singly connected network, while the rest of the randomly generated networks were multiply connected networks.

The algorithm used was a simple sequential GA with two-chromosome mating through crossover, and single-chromosome mutation. The proposed algorithm extended the conventional GA by using a non-binary string representation of chromosomes and GA operators that were developed for a chromosome with a graph representation. The individuals in this population were represented as graphs. Selecting such a nonconventional representation was based on the notion of successful compact blocks being inherited through generations. Thus, they constructed the blocks in a way such that their elements were as closely related as possible. Since adjacency in graphs corresponds to what is known as semantic closeness, it was desirable in their opinion to have neighboring nodes in the problem definition become neighbors in the genotype string. The graphs representing the individual chromosomes were known as labeled graphs, since labels were defined on each node as being its current truth assignment.

The selection operation was implemented in three different techniques. In the first, the probability of being selected as a parent is proportional to the phenotype of the individual. In the second method, the probability of being selected is proportional to a monotonic function of the phenotype of the individual (i.e. *fitness* = *1/log*^{2} *fitness*). This function was used to control the sensitivity of the algorithm to the fitness values. In the last method the probability for each individual is the same for all elements of the breeding population. After each new individual is created, its phenotype or fitness is assessed and stored. In an iterative procedure, elements of each generation are stored to choose which will be replaced and which will be used as parents for the next generation.

Since a graph chromosome was being used to represent individuals in the population, it seemed that the regular crossover of linear strings would be inappropriate for use in this model. A new crossover operator was developed to suit the graph representation, called clustering. A cluster is a subset of the nodes in the network. When crossover is performed, clusters are interchanged. This was performed through three steps, the first was to make two new children copies of the parents. Second, a node was randomly selected to be the center of the cluster. Determining the cluster elements was performed through the constraint that all the nodes less than *N* links away from the center of the cluster become included. N was chosen to contain about half of the nodes in the cluster. In the last step, the clusters selected are interchanged in order to produce two new individuals.

The fitness function used consisted of evaluating, using Equation (11), the joint probability of the truth assignment embodied in the chromosome. The performance measures of the algorithm included: the presence or absence of an optimal solution in the evolving population and the distance between the best individual and the optimal one. The first network was a 13 node network and had 12,288 points in the search space. The optimal solution for this network was equal to 0.098 and was found by an exhaustive search method. The best solution found by GALGO for this network. For this singly connected network, the optimal solution was found in 95% of the 20 total runs performed using GALGO technique with the uniform select operator. The second network was a multiply connected network with 20 nodes and 5 undirected cycles. The search space consisted of 7,962,624 possible states. The optimal solution found through an exhaustive search was $5.98e^{-5}$. Using GALGO model with the uniform selection operator, the optimal solution was reached in 30% of the total number of runs which were 10. Network number 3, was a simplification of network 2. It had the same number of nodes, 20 nodes, but only one cycle instead of five. Its optimal solution using exhaustive search was found to be equal to $4.42e^{-5}$. The only type of GALGO model that could find the optimal solution for this network was the GALGO algorithm with the transformed select operator, where fitness = $1/\log^2$ fitness. The optimal solution was reached in 8% of 25 runs.

The preliminary results of experimentation with the GALGO algorithm have shown that GAs constitute a promising approach to perform inference in multiply connected complex systems. The GA yields sub-optimal and sometimes optimal solutions for the relatively large belief networks in tractable times. It also has the potential of avoiding the strong sensitivity to the number of undirected cycles which makes exact methods not feasible for large networks. However, the researchers indicate that the complexity growth of the algorithm needs to be observed with large amounts of experimentation involving larger networks. They also recognize the influence that the network connectivity might have on finding a near optimal solution. This is also one of the points which is needs further experimentation in order to come to a non-empirical result.

CHAPTER VII

IMPLEMENTATION and RESULTS' ANALYSIS

GALib

GALib is the name of the software package that was selected for use in the implementation of the technique described in chapter five. The GALib is a public domain library of GA classes developed by the MIT laboratory of Genetic Algorithms. For any genetic algorithm, there are three things you must do to solve the problem:

1. Define a representation
2. Define the genetic operators
3. Define the objective function

GALib helps with the first two items by providing many examples and pieces from which you can build your representation and operators. In many cases you can use the built-in representations and operators with little or no modification.

The objective function in GALib is completely up to the user. Once a representation has been decided on, along with genetic operators and an objective measure any type of genetic algorithm implemented by GALib or extended upon by the user can be applied to find better solutions for the problem.

Representation in GALib

When genetic algorithms are used to solve an optimization problem, you must be able to represent a single solution to your problem in a single data structure. The genetic algorithm will create a population of solutions based on a sample data structure that you provide. The genetic algorithm then operates on the population to evolve the best solution.

In GAlib, the sample data structure is called a GAGenome that represents the chromosome in genetic algorithms' concepts. The library contains four types of genomes: GAListGenome, GATreeGenome, GAArrayGenome, and GABinaryStringGenome. These classes are derived from the base GAGenome class and a data structure class as indicated by their names. For example, the GAListGenome is derived from the GAList class as well as the GAGenome class. Use a data structure that works with your problem definition. For example, if you are trying to optimize a function that depends on 5 real numbers, then use as your genome a 1-dimensional array of floats with 5 elements.

Genetic Algorithms in GALib

There are many different types of genetic algorithms. GAlib includes three basic types: 'simple', 'steady-state', and 'incremental'. These algorithms differ in the way that they create new individuals and replace old individuals during the course of an evolution. GAlib provides two primary mechanisms for extending the capabilities of built-in objects. First of all (and most preferred, from a C++ point of view), user specific classes can be derived along with defining new member functions. If minor adjustments are needed to the behavior of a GAlib class, a new function is defined and you tell the existing GAlib class to use it instead of the default one.

The genetic algorithm object determines which individuals should survive, which should reproduce, and which should die. It also records statistics and decides how long the evolution should continue. Typically, a genetic algorithm has no obvious stopping criterion. You must tell the algorithm when to stop. Often the number-of-generations is used as a stopping measure, but you can use goodness-of-best-solution, convergence-of-population, or any problem-specific criterion if you prefer.

The library contains four flavors of genetic algorithms. The first is the standard 'simple genetic algorithm' described by Goldberg [36]. This algorithm uses non-overlapping populations and optional elitism. Each generation the algorithm creates an entirely new population of individuals. The second is a 'steady-state genetic algorithm' that uses overlapping populations. In this variation, you can specify how much of the population should be replaced in each generation. The third variation is the 'incremental genetic algorithm', in which each generation consists of only one or two children. The incremental genetic algorithms allow custom replacement methods to define how the new generation should be integrated into the population. So, for example, a newly generated child could replace its parent, replace a random individual in the population, or replace an individual that is most like it. The fourth type is the 'deme' genetic algorithm. This algorithm evolves multiple populations in parallel using a steady-state algorithm. Each generation the algorithm migrates some of the individuals from each population to one of the other populations.

The base genetic algorithm class contains operators and data common to most flavors of genetic algorithms. When you derive your own genetic algorithm you can use these member data and functions to keep track of statistics and monitor performance. The genetic algorithm contains the statistics, replacement strategy, and parameters for running the algorithm. The population object, a container for genomes, also contains some statistics as well as selection and scaling operators.

The library has built in functions for specifying when the algorithm should terminate. These include terminate-upon-generation, in which you specify a certain number of generations for which the algorithm should run, and terminate-upon-convergence, in which you specify a value to which the best-of-generation score should converge. You can customize the termination function to use your own stopping criterion.

Implementation Details

The objective of the experiments performed through this research is to analyze the advantages in solution quality produced by using simulated annealing as a genetic operator. Therefore, the simulated annealing was chosen to be implemented as a special case of mutation. Our implementation includes with a sequential genetic algorithm with the regular genetic operators of crossover, mutation and selection. It also includes a pure simulated annealing algorithm that operates using its initial temperature and temperature cooling ratio. The remainder of this section will describe in details all implementation issues related to genetic algorithms and simulated annealing, as well as implementation decisions when combing them.

Genetic Algorithm

The GASteadyStateGA built-in genetic algorithm was selected for representing the GA performing evolution and carrying the population. This genetic algorithm is similar to the algorithms described by DeJong [23]. It uses overlapping populations with a user-specifiable amount of overlap.

The algorithm creates a population of individuals by cloning the genome or population that you pass when you create it. Each generation the algorithm creates a temporary population of individuals, adds these to the previous population, then removes the worst individuals in order to return the population to its original size. You can select the amount of overlap between generations by specifying the *pReplacement* parameter of the GA. This parameter represents the percentage of the population that should be replaced each generation.

Newly generated offspring are added to the population, then the worst individuals are destroyed (so the new offspring may or may not make it into the population, depending on whether they are better than the worst in the population). If you specify a replacement percentage, then that percentage of the population will be replaced each generation. Alternatively, you can specify a number of individuals (less than the number in the population) to replace each generation. You cannot specify both - in a parameter list containing both parameters, the latter is used.

Initialization

The initialization of the first population of individuals is done through an operator defined in the steady state GA selected called GASteadyStateGA. The initialization operator is UniformInitializer. This operator works by setting the bits of the genome to random values. The library's random bit function is used so that we don't have to worry about machine-specific stuff.

The operator UniformInitializer seeds the random number generator with an appropriate value. Both the random number generator and the random bit generator have seeds. Set the seed if a seed is not specified by the user of the library when calling this operator. If a seed is specified, then set the seed to the specified value and use it. This function remembers the seed so that multiple calls to this function with the same seed do not reset the generator. Subsequent calls to this function with a different seed will initialize the generator to the new seed. The operator UniformInitializer uses the function GARandomSeed, which is a random bit generator Method II from numerical recipes in C.

Random Number Generator

In this implementation there are several random number generators used through out the experiments. When mutation is performed in this hybrid technique, a random number generator is used in order to decide whether regular mutation or simulated annealing will be performed. A built-in function in the GALib library called GARandomFloat was used. The GARandomFloat function works by returning a number selected at random within the bounds low and high, inclusive. The low and high bounds are specified by the user of this function when calling it.

The internals of this function work by initializing the random function by using the current time as the seed. The process id as well as the time is used for the random seed in an attempt to get rid of some of the periodicity from the low bits when using only the time as the random seed. These routines use the random/srandom routines to generate random numbers.

Representation

The GA1DBinaryStringGenome class was selected as the type of chromosme representation for the Bayesian belief networks. For the problem being experimented, MAPs on BBNs, each node is assigned either a true or false value (i.e.: 1 or 0, respectively). Each individual representing a BBN is regarded as an evidence vector of true or false assignments. Thus, the GA1DBinaryStringGenome was an appropriate type of genome since it is a 1-Dimensional vector of binary values. The binary string genome is derived from the GABinaryString and GAGenome classes. It is a string of 1s and 0s whose length may be fixed or variable. The genes in this genome are bits and the alleles for each bit are 0 and 1.

Crossover

The crossover operator used in the experiments was one of the crossover operators defined in the GA1DBinaryStringGenome chromosome type. The operator selected was called OnePointCrossover. The one point crossover is one of the simplest mechanisms for mating between two individuals in a GA population. A single point is selected for crossing and the 2 individuals are swapped at that point.

Mutation

The mutation operator for the hybrid technique that was developed in this research is a mixture of the traditional mutation operator and the simulated annealing technique. In the hybrid technique, the regular mutation operator is selected as the FlipMutator operation in the GA1DBinaryStringGenome. The mutation performed by this function simply takes an individual and selects at random a gene as the mutation point, then flips the value of the allele in that gene. This mutation operator is called when no simulated annealing is performed on the individual, otherwise the simulated annealing technique uses the FlipMutator as a neighborhood generation mechanism inside its execution flow.

Selection

For all experimentations, the GALib built-in selection method called *GARouletteWheelSelector* was used. From its name, this method is based on the Roulette Wheel selection mechanism [23].

This selection method picks an individual based on the magnitude of the fitness score relative to the rest of the population. The higher the score, the more likely an individual will be selected. Any individual has a probability *p* of being chosen where *p* is equal to the fitness of the individual divided by the sum of the fitnesses of all the individuals in the population.

GA Parameters

Unlike traditional genetic algorithm implementation which use two probabilty values, one for crossover P_c and one for mutation P_m, here three probabilities are used. The probabilities used in the hybrid implementation of GAs and simulated annealing are probability of crossover, probability of mutation, and probability of simulated annealing. The mathematical analysis provided in the next section will present the relation between the probability of mutation and probability of simulated annealing.

Annealing Schedule

The simulated annealing parameters needed to define any annealing schedule are the initial temperature of the cooling schedule and the cooling rate. The values of both parameters have been selected to be global across all experiment runs. The initial temperature was given a value of 0.95 and the gradient or the rate of decreasing temperature was 0.005. Both values were used for all individuals when annealing was applied as part of the mutation operation.

Objective Function

The fitness evaluation was performed by defining an objective function for the GA which was selected from GALib library. This objective function calculated the fitness of each individual based on the evaluation of the chromosome by the following equation:

$$P(v_1,\ldots\ldots,v_n) = \Pi_{i=1} P(v_i \mid \pi(v_i)); \quad \forall\, 1 < i < n \qquad (12)$$

where $v_1, v_2 \ldots\ldots, v_n$ is the instantiaion of the network represented by the genome and its allele values of 0's and 1's.

Mathematical Analysis

In this section we demonstrate a mathematical analysis for finding the equations describing the probability of mutation while incorporating simulated annealing in it as part of the genetic operator. In the previous section, we defined the probability of simulated annealing as a genetic parameter that is set for each run. Now, that probability is not the actual probability with which simulated annealing works. Incorporating simulated annealing with mutation makes the call to a mutation operator either performs a real mutation or a simulated annealing. The experiments that will be described in later sections use the probability of mutation that was used in GA alone as the probability of real mutation.

The first stage of experiments was run using a traditional GA in order to find the optimal reached by this method alone without incorporating SA. In these experiments the values of P_m and P_c, represent the probability of mutation and probability of crossover respectively were defined as a grid of 20x20. This gave a total of 400 runs for each of the networks used in the experiments. The values of P_m and P_c were scaled on the 20x20 grid such that $P_m = 0, 0.05, \ldots, 1$ for each value of $P_c = 0, 0.05, \ldots, 1$.

The second stage of experiments used the P_m and P_c pairs that produced the top 5% scores for each network and ran each one of these pairs for 200 runs using the new technique. For this set of experiments, a new probability parameter was introduced representing the probability of simulated annealing. This parameter is referred to as *SAP*. *SAP* was scaled over 200 runs, thus each run incremented the value of *SAP* by 0.005, giving a sequence of values for $SAP = 0.0, 0.005, 0.01, \ldots\ldots, 1$. This parameter called *SAP* is a user defined parameter and does not represent the real value of probability of simulated annealing that will be actually used inside the GA mutation engine. In order to reach the real values of probability of simulated annealing and probability of mutation that are used when determining whether a mutation is to be performed using the traditional operator or using the intelligent one the following mathematical derivations are needed.

Let P_μ = probability of real mutation (obtained from earlier GA runs on the BBN i.e: P_m).

Let SAP = 0.0, 0.005, 0.01,..........,1.0.

Let P_{MUT} = probability of mutation in hybrid SA-GA runs.

Let P_{SA} = probability of simulated annealing in hybrid SA-GA runs.

Thus we can define the following two equations for P_μ and SAP, respectively:

$$P_\mu = P_{MUT} \, x \, (1 - P_{SA}) \tag{13}$$

$$SAP = P_{MUT} \, x \, P_{SA} \tag{14}$$

We need to use Equations (13) and (14) in order to derive new equations for *SAP* and P_{MUT}.

From (14):

$$P_{MUT} = P_\mu /(1 - P_{SA}) \tag{15}$$

and from (14):

$$P_{MUT} = SAP/ P_{SA} \tag{16}$$

Therefore,

$$SAP/ P_{SA} = P_\mu /(1 - P_{SA}) \tag{17}$$

and,

$$SAP \, x \, (1 - P_{SA}) = P_\mu \, x \, P_{SA} \tag{18}$$

Therefore,

$$SAP/P_{\mu} = P_{SA}/(1 - P_{SA}) \tag{19}$$

From Observation1 below, we can determine the numerical value of P_{SA} by the following:

$$P_{SA} = (SAP/P_{\mu})/(1 + SAP/P_{\mu}) \tag{20}$$

Once we have the value of P_{SA}, we can find the value of P_{MUT} using Equation (16) above.

Observation 1:

If we have $\alpha = \chi/1\text{-}\chi$ then:

$$\alpha - \alpha\chi = \chi \tag{21}$$

and,

$$\alpha = \chi + \alpha\chi \tag{22}$$

$$\alpha = \chi(1 + \alpha) \tag{23}$$

thus:

$$\chi = \alpha/(1 + \alpha) \tag{24}$$

Experiment Set A: 30-Nodes Networks

We have created four Bayesian belief networks of the size 30 nodes. Each one of these networks has a different topology and a different set of probability tables. This section will present each network with its topology and then the set of results produced by running the first and second stages of the experiments.

BBN_30A

The topology of the first 30 node network used in this research exists in Appendix E as figure BBN_30A. The results obtained from running the first stage of experiments where the genetic algorithm is run for 400 times are provided in the following table:

Table 3: Highest 5% Scores for BBN_30A (without SA).

SET #	P_M	P_C	SCORE
1	0.11	0.54	0.000239
2	0.16	0.49	0.000235
3	0.15	0.5	1.89E-04
4	0.16	0.53	0.000182
5	0.21	0.49	0.00018
6	0.15	0.925	1.76E-04
7	0.15	0.1	0.000168
8	0.2	0.46	0.000166
9	0.25	0.65	1.57E-04
10	0.35	0.8	1.56E-04

The best MAP value found for this 30 node network is equal to 0.000239.

Following is the table showing the best value obtained when each pair of the above 10 values of P_m and P_c were used with simulated annealing as an intelligent mutation operator. The last column in this table represents the percentage in which the best score was found over the 200 runs.

Table 4: Highest 10% scores for runs with intelligent SA for the ten genetic parameters in Table 3.

P_REALMUT	P_C	P_{MUT}	SAP	P_{SA}	SCORE	RATIO /200
0.11	0.54	0.155	0.045	0.290323	0.000239	2.50%
0.16	0.49	0.18	0.02	0.111111	0.000239	3%
0.15	0.5	0.2	0.05	0.25	0.000239	1.50%
0.16	0.53	0.325	0.165	0.507692	0.000239	1%
0.21	0.49	0.29	0.08	0.275862	0.000182	0.50%
0.15	0.925	0.155	0.005	0.032258	0.000239	1.50%
0.15	0.1	0.195	0.045	0.230769	0.000239	1%
0.2	0.46	0.32	0.12	0.375	0.000239	1%
0.25	0.65	0.56	0.31	0.553571	1.82E-04	1%
0.35	0.8	0.46	0.11	0.23913	0.000129	0.50%

The following chart shows the increase in score obtained by the hybrid intelligent simulated annealing and genetic algorithms.

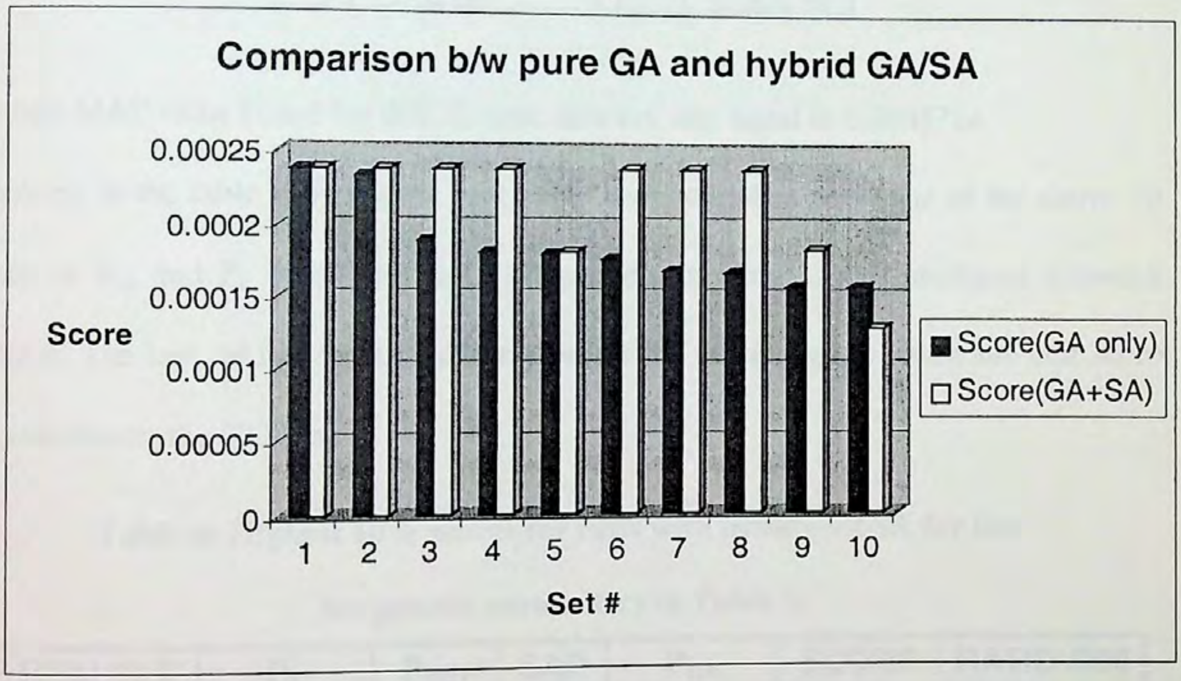

Figure 8: Performance difference for BBN_30A.

BBN_30B

The topology of BBN_30B can be seen in Appendix E. The results obtained from running the first stage of experiments where the genetic algorithm is run for 400 times are provided in the following table:

Table 5: Highest 5% Scores for BBN_30B (without SA).

SET #	P_M	P_C	SCORE
1	0.1	0.52	0.003784
2	0.2	0.6	0.002523
3	0.15	0.6143	2.30E-03
4	0.25	0.4	0.002212
5	0.1722	0.372	0.001781
6	0.25	0.25	0.001534
7	0.1833	0.4167	0.001483
8	0.25	0.6167	0.001474
9	0.1833	0.45	1.46E-03
10	0.05	0.1	0.00119

The best MAP value found for this 30 node network was equal to 0.0003784.

Following is the table showing the best value obtained when each pair of the above 10 values of P_m and P_c were used with simulated annealing as an intelligent mutation operator. The last column in this table represents the percentage in which the best score was found over the 200 runs.

Table 6: Highest 10% scores for runs with intelligent SA for the ten genetic parameters in Table 5.

P_REALMUT	P_C	P_{MUT}	SAP	P_{SA}	SCORE	RATIO /200
0.1	0.52	0.105	0.005	0.047619	0.003784	8.50%
0.2	0.6	0.215	0.015	0.069767	0.003784	0.50%
0.15	0.6143	0.19	0.04	0.210526	0.003784	1%
0.25	0.4	0.62	0.37	0.596774	0.0023	0.50%
0.1722	0.372	0.2422	0.07	0.289017	0.0023	0.50%
0.25	0.25	0.285	0.035	0.122807	0.001474	0.50%
0.1833	0.4167	0.2283	0.045	0.197109	0.003784	1%
0.25	0.6167	0.77	0.52	0.675325	2.52E-03	0.50%
0.1833	0.45	0.1983	0.015	0.075643	0.0023	1%
0.05	0.1	0.055	0.005	0.090909	0.003784	12%

The following chart shows the increase in score obtained by the hybrid intelligent simulated annealing and genetic algorithms.

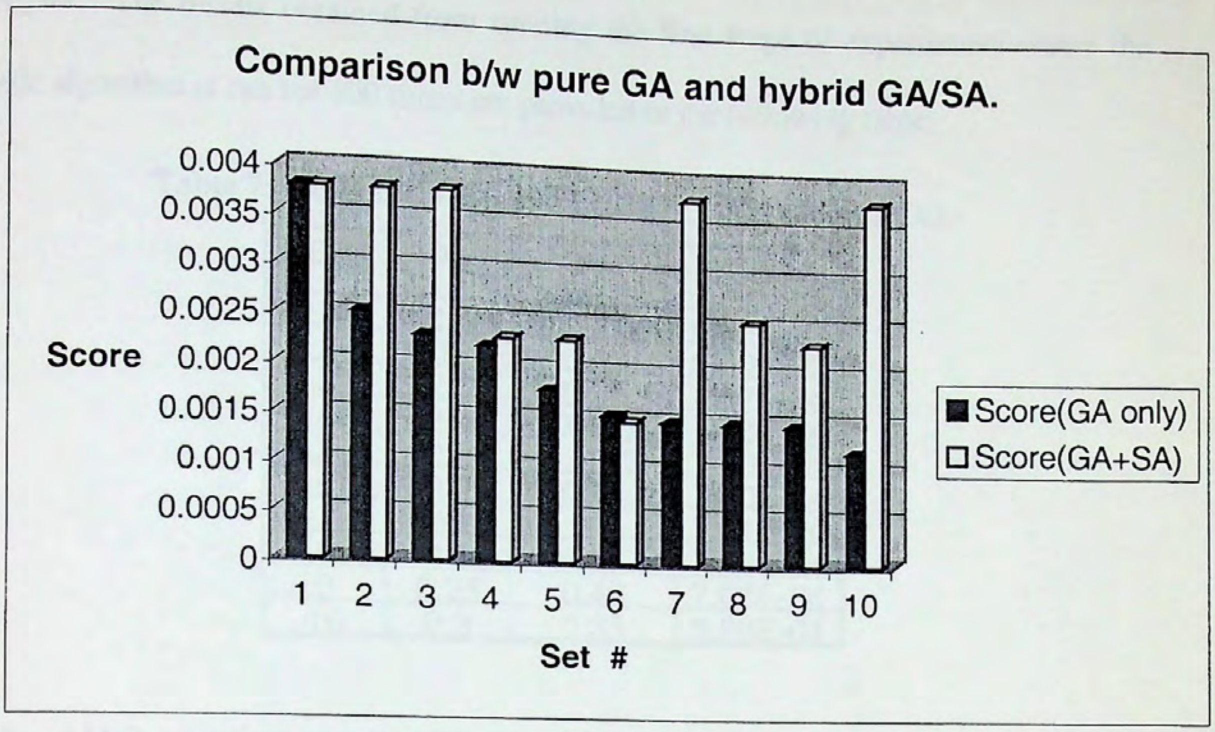

Figure 9: Performance difference for BBN_30B.

BBN_30C

The topology of the Bayesian belief network exists in Appendix E as figure BBN_30C. The results obtained from running the first stage of experiments where the genetic algorithm is run for 400 times are provided in the following table:

Table 7: Highest 5% Scores for BBN_30C (without SA).

SET #	P_M	P_C	SCORE
1	0.0971	0.5212	0.000931
2	0.183	0.43	0.000922
3	0.2125	0.25	8.87E-04
4	0.2125	0.825	8.78E-04
5	0.2	0.5875	7.88E-04
6	0.2	0.05	7.80E-04
7	0.225	0.5	7.51E-04
8	0.3	0.55	7.43E-04
9	0.25	0.46	7.03E-04
10	0.3	0.35	6.96E-04

The best MAP value found for this 30 node network is equal to 0.000931.

Following is the table showing the best value obtained when each pair of the above 10 values of P_m and P_c were used with simulated annealing as an intelligent mutation operator. The last column in this table represents the percentage in which the best score was found over the 200 runs.

Table 8: Highest 10% scores for runs with intelligent SA for the ten genetic parameters in Table 7.

P_REALMUT	P_C	P_{MUT}	SAP	P_{SA}	SCORE	RATIO /200
0.0971	0.5212	0.1021	0.005	0.048972	0.000931	10%
0.183	0.43	0.188	0.005	0.026596	0.000887	1.50%
0.2125	0.25	0.2225	0.01	0.044944	0.000788	1%
0.2125	0.825	1.1724	0.9599	0.818763	9.22E-04	0.50%
0.2	0.5875	0.395	0.195	0.493671	0.000931	0.50%
0.2	0.05	0.265	0.065	0.245283	0.000931	1%
0.225	0.5	0.86	0.635	0.738372	9.31E-04	0.50%
0.3	0.55	0.53	0.23	0.433962	7.51E-04	0.50%
0.25	0.46	0.455	0.205	0.450549	7.88E-04	0.50%
0.3	0.35	1.02	0.7199	0.705882	7.51E-04	0.50%

The following chart shows the increase in score obtained when using the hybrid intelligent simulated annealing and genetic algorithms:

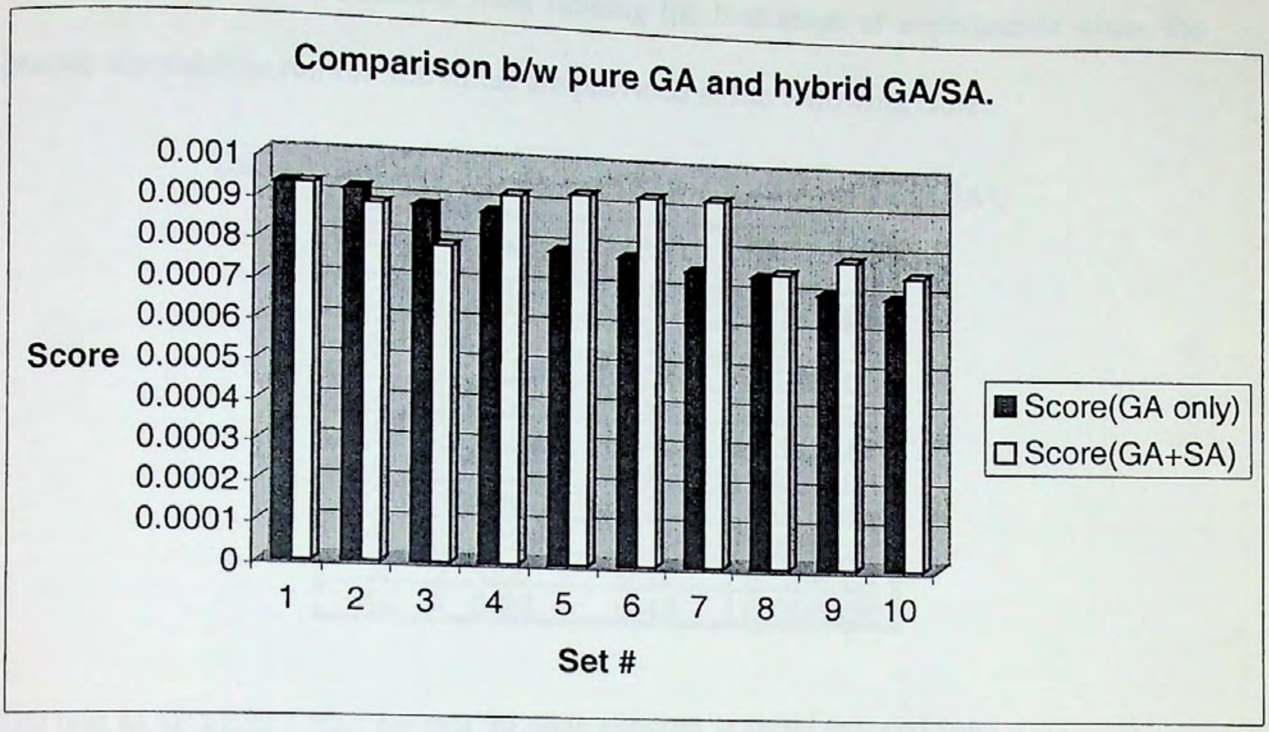

Figure 10: Performance difference for BBN_30C.

BBN_30D

The topology of the Bayesian belief network also exists in Appendix E as figure BBN_30D. The results obtained from running the first stage of experiments where the genetic algorithm is run for 400 times are provided in the following table:

Table 9: Highest 5% Scores for BBN_30D (without SA).

SET #	P_M	P_C	SCORE
1	0.12	0.53	0.001566
2	0.183	0.56	0.001514
3	0.2	0.57	1.28E-03
4	0.4	0.85	1.24E-03
5	0.2	0.45	0.00092
6	0.225	0.975	8.89E-04
7	0.2667	0.35	0.000887
8	0.2	0.4	8.63E-04
9	0.2	0.15	8.58E-04
10	0.25	0.45	0.000835

The best MAP value found for this 30 node network is equal to 0.001566.

Following is the table showing the best value obtained when each pair of the above 10 values of P_m and P_c were used with simulated annealing as an intelligent mutation operator. The last column in this table represents the percentage in which the best score was found over the 200 runs.

Table 10: Highest 10% scores for runs with intelligent SA for the ten genetic parameters in Table 9.

P_REALMUT	P_C	P_{MUT}	SAP	P_{SA}	SCORE	RATIO /200
0.12	0.53	0.125	0.005	0.04	0.001566	8.50%
0.183	0.56	0.208	0.025	0.120192	0.001566	1%
0.2	0.57	0.25	0.05	0.2	0.001566	1%
0.4	0.85	0.57	0.17	0.298246	0.000858	0.50%
0.2	0.45	0.26	0.06	0.230769	0.001566	1%
0.225	0.975	0.255	0.03	0.117647	0.001281	0.50%
0.2667	0.35	0.5567	0.29	0.520927	0.000887	0.50%
0.2	0.4	0.32	0.12	0.375	0.001566	1%
0.2	0.15	0.27	0.07	0.259259	0.001566	3%
0.25	0.45	0.285	0.035	0.122807	0.001281	0.50%

The following chart shows the increase in score obtained when using the hybrid intelligent simulated annealing and genetic algorithms:

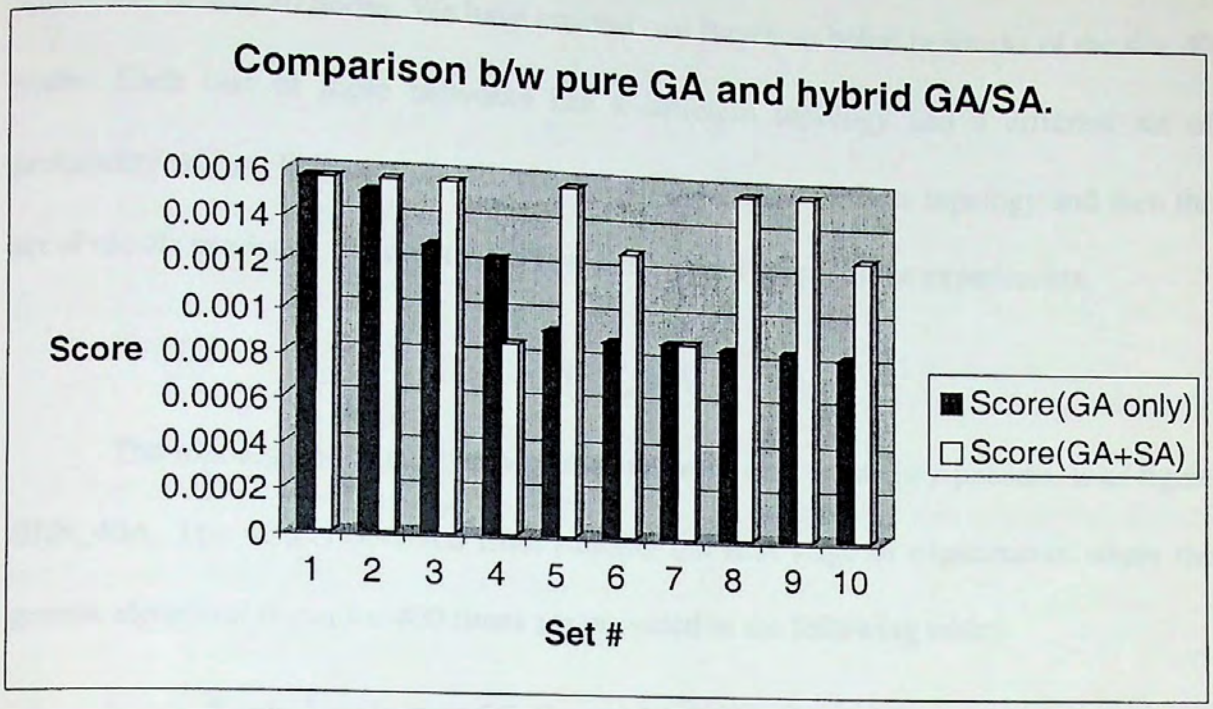

Figure 11: Performance difference for BBN_30D.

Experiment Set B: 40-Nodes Networks

The second set of experiments included a larger size Bayesian belief network, which was of size 40 nodes. We have created two Bayesian belief networks of the size 40 nodes. Each one of these networks has a different topology and a different set of probability tables. This section will present each network with its topology and then the set of results produced by running the first and second stages of the experiments.

BBN_40A

The topology of this Bayesian belief network also exists in Appendix E as figure BBN_40A. The results obtained from running the first stage of experiments where the genetic algorithm is run for 400 times are provided in the following table:

Table 11: Highest 5% Scores for BBN_40A (without SA).

SET #	P_M	P_C	SCORE
1	0.06	0.44	3.09E-05
2	0.06	0.47	3.03E-05
3	0.075	0.6	2.62E-05
4	0.1	0.5	2.56E-05
5	0.06	0.66	2.35E-05
6	0.125	0.3	2.34E-05
7	0.083	0.45	2.31E-05
8	0.05	0.8	2.00E-05
9	0.1	0.6	1.99E-05
10	0.1	0.66	1.91E-05

The best MAP value found for this 40 node network is equal to $3.09e^{-5}$.

Following is the table showing the best value obtained when each pair of the above 10 values of P_m and P_c were used with simulated annealing as an intelligent mutation operator. The last column in this table represents the percentage in which the best score was found over the 200 runs.

Table 12: Highest 10% scores for runs with intelligent SA for the ten genetic parameters in Table 11.

P_REALMUT	P_C	P_{MUT}	SAP	P_{SA}	SCORE	RATIO /200
0.06	0.44	0.065	0.005	0.076923	3.09E-05	0.50%
0.06	0.47	1.075	0.635	0.590698	8.03E-06	0.50%
0.075	0.6	0.115	0.04	0.347826	3.09E-05	0.50%
0.1	0.5	0.375	0.275	0.733333	1.83E-05	0.50%
0.06	0.66	0.065	0.005	0.076923	2.62E-05	0.50%
0.125	0.3	0.155	0.03	0.193548	1.75E-05	0.50%
0.083	0.45	0.143	0.06	0.41958	1.89E-05	0.50%
0.05	0.8	0.055	0.005	0.090909	3.09E-05	2.0%
0.1	0.6	0.11	0.01	0.090909	2.34E-05	0.50%
0.1	0.66	0.135	0.035	0.259259	2.31E-05	0.50%

The following chart shows the increase in score obtained when using the hybrid intelligent simulated annealing and genetic algorithms.

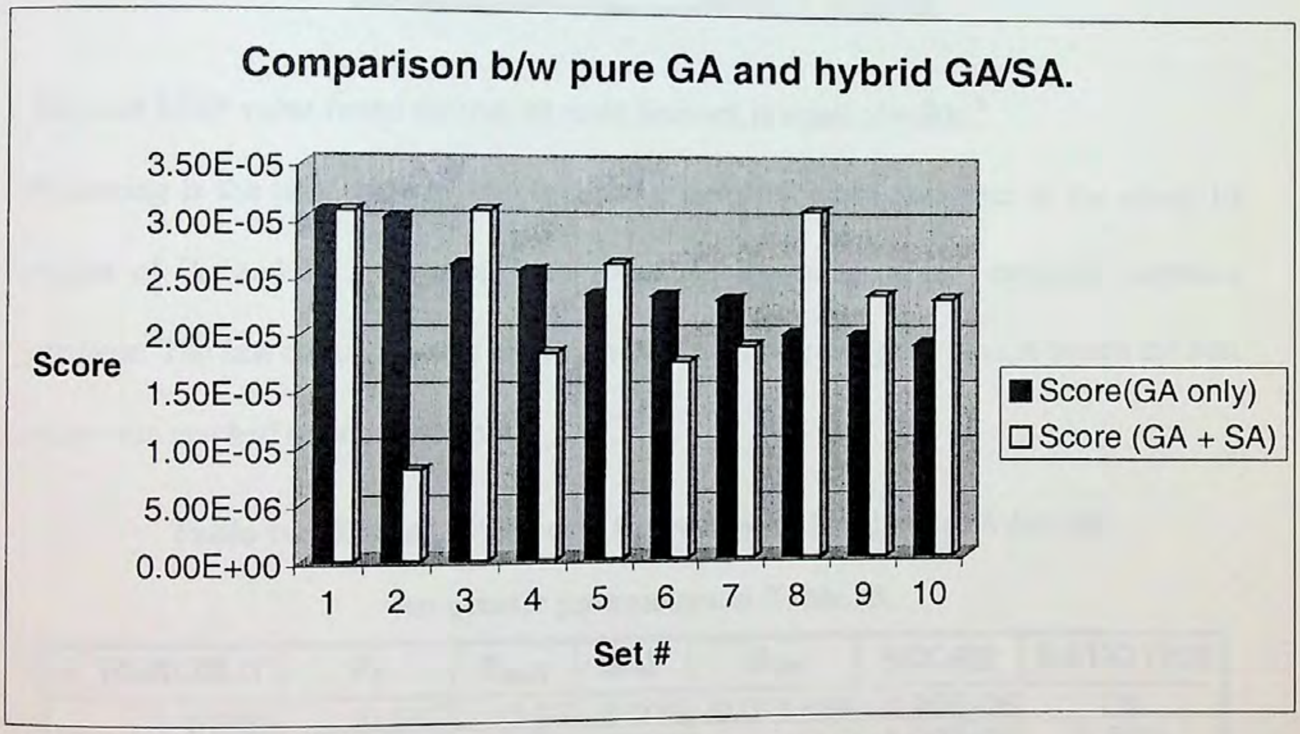

Figure 12: Performance difference for BBN_40A.

BBN_40B

The topology of this Bayesian belief network exists in Appendix E as figure BBN_40B. The results obtained from running the first stage of experiments where the genetic algorithm is run for 400 times are provided in the following table:

Table 13: Highest 5% scores for BBN_40B (without SA).

SET #	P_M	P_C	SCORE
1	0.065	0.517	4.90E-05
2	0.1	0.3556	3.95E-05
3	0.1125	0.8125	3.27E-05
4	0.1167	0.8833	3.18E-05
5	0.1	0.3	2.73E-05
6	0.15	0.3	2.63E-05
7	0.2	0.25	2.32E-05
8	0.15	0.55	2.26E-05
9	0.2	1	2.18E-05
10	0.15	0.65	2.12E-05

The best MAP value found for this 40 node network is equal to 4.90e^{-5}.

Following is the table showing the best value obtained when each pair of the above 10 values of P_m and P_c were used with simulated annealing as an intelligent mutation operator. The last column in this table represents the percentage of runs in which the best score was reached over the 200 runs.

Table 14: Highest 10% scores for runs with intelligent SA for the ten genetic parameters in Table 13.

P_REALMUT	P_C	P_{MUT}	SAP	P_{SA}	SCORE	RATIO /200
0.065	0.517	0.07	0.005	0.071429	4.90E-05	2%
0.1	0.3556	0.2	0.1	0.5	4.90E-05	0.50%
0.1125	0.1167	0.1175	0.005	0.042553	3.95E-05	1%
0.8125	0.8833	1.4674	0.655	0.446337	1.20E-06	0.50%
0.1	0.3	0.25	0.15	0.6	4.90E-05	0.50%
0.15	0.3	0.165	0.015	0.090909	2.12E-05	0.50%
0.2	0.25	0.33	0.13	0.393939	1.47E-05	1%
0.15	0.55	0.415	0.265	0.638554	2.02E-05	0.50%
0.2	1	0.21	0.01	0.047619	1.28E-05	0.50%
0.15	0.65	0.295	0.145	0.491525	3.95E-05	0.50%

The following chart shows the increase in score obtained when using the hybrid intelligent simulated annealing and genetic algorithms.

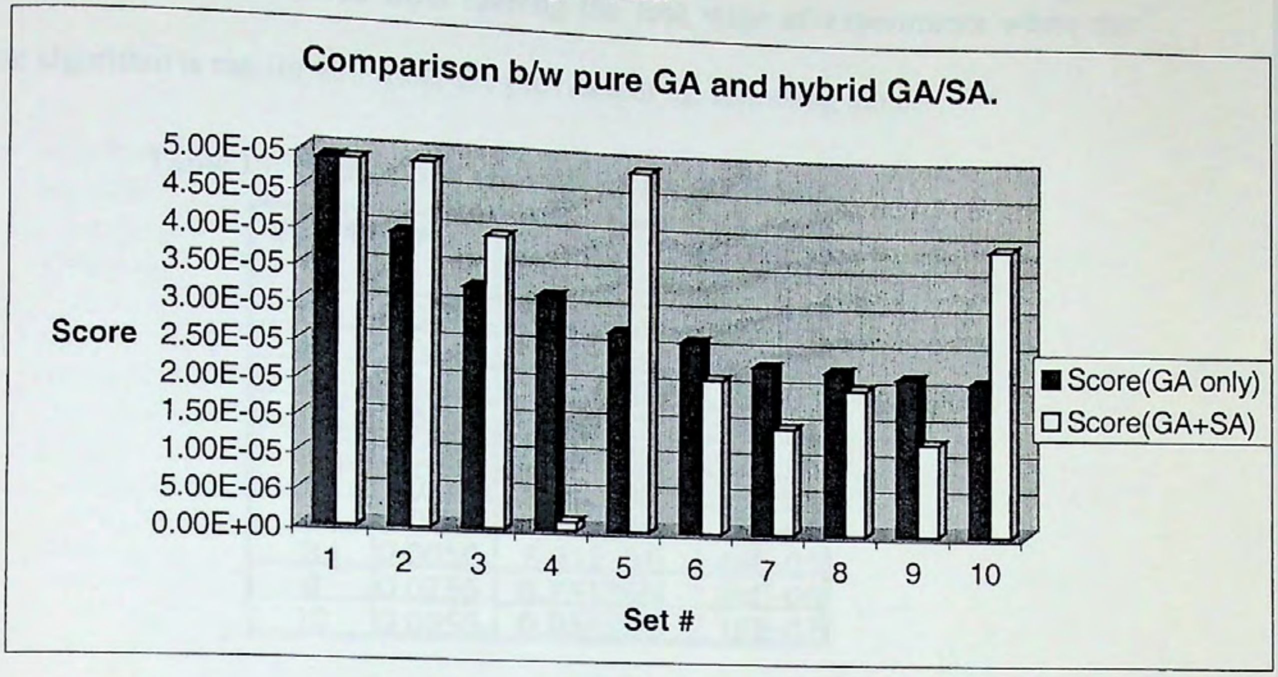

Figure 13: Performance difference for BBN_40B.

Experiment Set C: 50-Nodes Network

The topology of this Bayesian belief network also exists in Appendix E as figure BBN_50. The results obtained from running the first stage of experiments where the genetic algorithm is run for 400 times are provided in the following table:

Table 15: Highest 5% scores for BBN_50 (without SA).

SET #	P_M	P_C	SCORE
1	0.0319	0.549768	9.91E-05
2	0.0143	0.42809	1.10E-05
3	0.0305	0.18787	6.19E-05
4	0.0101	0.062623	2.53E-05
5	0.0203	0.125247	4.07E-05
6	0.059	0.995	3.77E-05
7	0.025	0.8885	1.75E-05
8	0.0050	5.31E-01	1.49E-05
9	0.0755	0.751255	1.38E-05
10	0.0955	0.955555	2.18E-07

The best MAP value found for the 50 node network is equal to $9.91e^{-5}$. This network is the largest in size that has been in the experiments.

Following is the table showing the best value obtained when each pair of the above 10 values of P_m and P_c were used with simulated annealing as an intelligent mutation operator. The last column in this table represents the percentage of runs in which the best score was reached over the 200 runs.

Table 16: Highest 10% scores for runs with intelligent SA for the ten genetic parameters in Table 15.

P_REALMUT	P_C	P_{MUT}	SAP	P_{SA}	SCORE	RATIO /200
0.031918	0.549768	0.0295	0.075	0.002394	1.42E-04	1%
0.014346	0.42809	0.0104	0.275	0.003945	1.75E-04	0.50%
0.030529	0.18787	0.0242	0.205	0.006258	1.33E-04	0.50%
0.010176	0.062623	0.0054	0.46	0.004681	1.75E-04	0.50%
0.020352	0.125247	0.0177	0.13	0.002646	1.08E-04	1.50%
0.059	0.995	5.52E-	0.065	0.003835	7.89E-05	0.50%
0.025	0.8885	0.0227	0.09	0.00225	1.75E-05	1%
0.005088	5.31E-01	0.0022	0.56	0.002849	1.75E-04	1%
0.07553	0.751255	0.0668	0.115	0.008686	3.07E-05	0.50%
0.09555	0.955555	0.0855	0.105	0.010033	3.19E-05	0.50%

The following chart shows the increase in score obtained when using the hybrid intelligent simulated annealing and genetic algorithms.

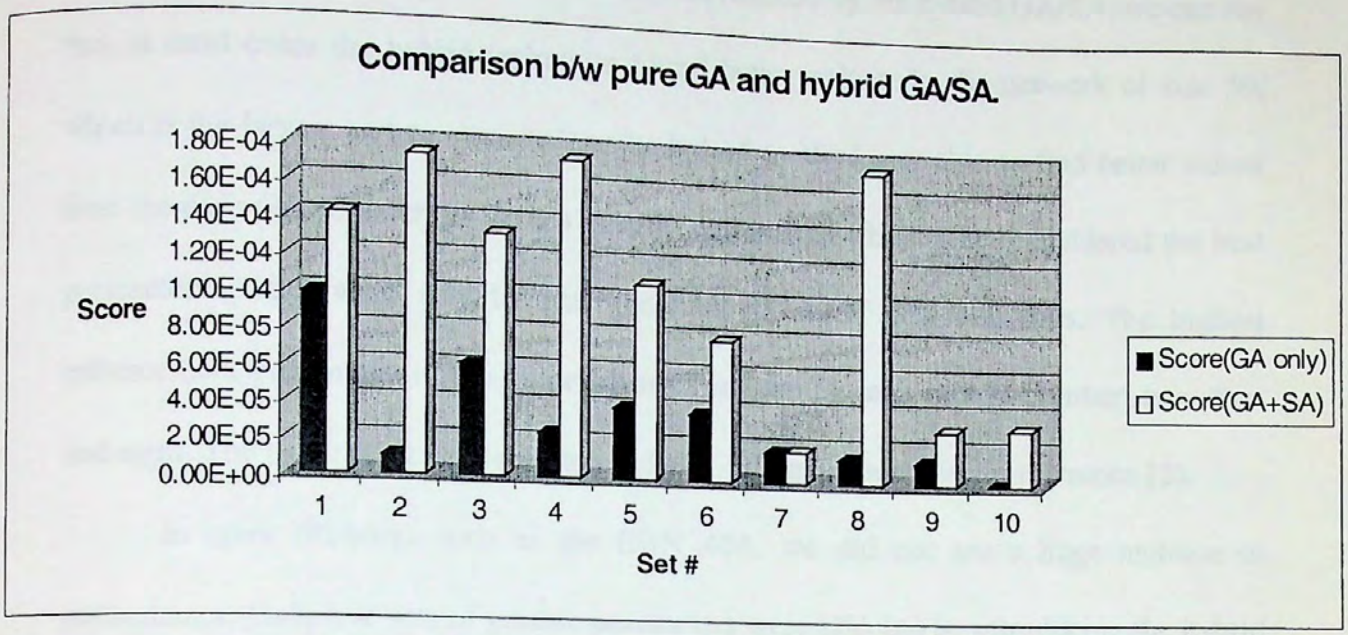

Figure 14: Performance difference for BBN_50.

Analysis

From the figures presented in the previous sections which compare between the best score values reached by pure GA and those reached by the hybrid GA/SA, we can see that in most cases the hybrid technique found better values. In the network of size 50, which is the largest and most complex, the hybrid method was able to find better values than the pure GA for each of the ten genetic parameters which were considered the best parameters which were able to find good MPE values over 400 runs. The highest enhancement percentage in the 50 nodes network can be seen in sets number: two, four and eight. The results that were reached in BBN_50 are present also in reference [2].

In other networks such as the BBN_40A, we did not see a huge increase in performance. Only five sets of genetic parameters were affected by introducing the hybrid technique. These were sets: three, five, eight, nine and ten. From the results of BBN_40B we can see a similar phenomena where only five sets found a higher value for its MPE using the hybrid technique. Examining the results of the 30 node networks we find that in all of the four networks, the hybrid technique was able to find a better solution than the GA in nine or eight of the times out of ten different genetic parameter sets. All the data produced from experiments with pure GA and hybrid GA/SA are provided as a zip file at the following link: http://www.cs.aucegypt.edu/bnnrg/sattia/data.zip.

CHAPTER VIII

CONCLUSION and FUTURE WORK DIRECTIONS

In this research, we used simulated annealing as a type of mutation operator: an "intelligent" or directed, mutation as opposed to the random, or undirected, conventional mutation operator. In our experiments, we demonstrated how the hybrid technique was able to reach better solutions for the MPE problem on Bayesian belief networks. The results shown by the experiments give the hybrid technique advantages over the pure GA technique. The problem selected for experimentation has not been given much attention for those instances with large size networks. In most complex domains where abductive reasoning plays a great role, graphs with size 50 are still smaller than the typical size. Nevertheless, most attempts for solving MAPs have not tackled large network sizes which comprise more than 30 nodes. The emphasis of this research on large size networks of BBNs is an addition to solving the MAP problem. The hybrid technique that was developed has generated better values in comparison with pure GAs. Exhibiting how well the technique can perform on large size BBNs introduces a new technique that can handle larger sizes as well as providing potential for hybrid techniques to perform better in solving the *NP*-complete problem of MAPs.

The MPE problem is an important problem for BBNs and the investigation of genetic methods for this problem is still in its early stages. In particular, there are two issues that seem especially worthy of further investigation:

1. The effect of BBN topology and probability distribution properties on the performance of genetic algorithms. Rojas-Guzman and Kramer [64] have suggested that the empirical performance of genetic algorithms did not seem dependent on the

connectivity of BBNs; this would indeed be a very significant result if it were borne out of further analysis.

2. The efficacy of specialized genetic operators, especially topology aware crossover operator such as cluster-based crossover [64], for BBNs.

The use of simulated annealing as a genetic operator also has important implications for parallelization potential [4]. With very minimal communication overhead, each call to the simulated annealing mutation operator can be carried out by a satelite processor. Once a simulated annealing processor receives the genome which would be the start state for the annealing search, it need not communicate with the central genetic algorithm processor until the end of the annealing process when it need only return the result of the search. Therefore, it is especially attractive for parallelization on architectures with high communication cost.

LITERATURE CITED

1. Aarts, E. and van Laarhoven, P. "Statistical Cooling: A General Approach to Combinatorial Optimization Problems", Journal of Research, vol 40 pp. 193-226, 1985.
2. Abdelbar, A.M. and Attia, S. "Finding Most Probable Explanations Using Simulated Annealing as a Geneticc Operator", *Proceedings Third Internationaal Conference on Systemics, Cybernetics and Informatics*, Vol. 8, pp. 131, Orlando, Florida 1999.
3. Abdelbar, A.M. and Assaggaf, M. "Recurrent High-Order Networks for Probabilistic Explanation", *Proceedings IEEE International Joint Conference on Neural Networks*, 1999.
4. Abdelbar, A.M. and Hedetniemi, S.M. "A Parallel Hybrid Genetic Algorithm Simulated Annealing Approach to Finding Probable Explanations on Bayesian Belief Networks", *Proceedings IEEE International Conference on Neural Networks*, **I**, 450, 1997.
5. Abdelbar, A.M. and Hedetniemi, S.M. "Approximating MAPs for Belief Networks is NP-hard and other theorems", Artificial Intelligence, 102, pp. 21-, 1998.
6. Abramson, D., Mills, G. and Perkins, S. " Parallelisation of a Genetic Algorithm for the Computation of Efficient Train Schedules", *Proceedings of the 1993 Parallel Computing and Transputers Conference*, pp. 139-149, 1993.
7. Abramson, D. and Abela, J. "A Parallel Genetic Algorithm for Solving the School Timetabeling Problem", *Proceedings 15th Australian Computer Science Conference*, vol 18, pp.1-11, 1992.
8. Ackley, D.H. and Littman, M. "Interactions between learning and evolution", *Proceedings 2nd International Conference on Artificial Life*, C.G. Langton, ed., Addison-Wesley, 1991.
9. Adler, D. "Genetic Algorithms and Simulated Annealing: A Marriage Proposal", *IEEE International Conference on Neural Networks*, Vol. 2, pp. 1104-1109, 1993.
10. Azencott, R. ed., Simulated Annealing: Parallelization Techniques, John Wiley & Sons, New York, 1992.

11. Back, T., Hammel, U. and Schwefel, H-P. "Evolutionary Computation : Comments on the History and Current State", *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, 1997

12. Belding, T.C. "The Distributed Genetic Algorithm revisited", In L. Eschelman (Ed.), *Proceedings 5th International Conference on Genetic Algorithm*, pp. 155-162, 1995.

13. Branke, J., Anderson, H.C. and Schmeck, H. "Parallelising Global Selection in Evolutionary Algorithms", Journal of Parallel and Distributed Computing, 1997.

14. Brown, D.E., Huntley, C.L. and Spillane, A.R. "A Parallel Genetic Heuristic for the Quadratic Assignment Problem", *Proceedings 3rd International Conference on Genetic Algorithms*, pp. 406-415, 1989.

15. Cantu-Paz, E. "Designing Efficient Master-Slave Parallel Genetic Algorithms", IlliGAL Report No. 97004. Urbana, IL: University of Illinois at Urbana-Champaign, 1997.

16. Cerny, V. "Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm", Journal of Optimization Theory and Applications, Vol. 45, pp. 41-51, 1985.

17. Chandy, J., Kim, S., Ramkumar, B., Parkes, S. and Banerjee, P. "An Evaluation of Parallel Simulated Annealing Strategies with Application to the Standard Cell Placement", *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 16, pp. 398-410, 1997.

18. Chen, H., Flann, N.S. and Watson, D. "Parallel Genetic Simulated Annealing: A Massively Parallel SIMD Algorithm", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, pp. 126-136, 1998.

19. Cohoon, J.P., Hedge, S.U., Martin W.N. and Richards, D. "Punctuated Equilibria: A Parallel Genetic Algorithm", In J.J. Grefenstette(Ed.), *Proceedings 2nd International Conference on Genetic Algorithm*, pp.148-154, 1987.

20. Cohoon, J.P., Martin, W.N. and Richards, D. "Genetic Algorithms and Punctuated Equilibria in VLSI", In H.P. Schwefel and R. Manner (Eds.), Parallel Problem Solving from Nature, pp. 134-144, Berlin: Springer-Verlag, 1991.

21. Connolly, D.T. "An Improved Annealing Scheme for the QAP", European Journal of Operational Research, vol 46 pp. 93-100, 1987.

22. Davis, T.E. and Principe, J.C. "A Simulated Annealing Like Convergence Theory for the Simple Genetic Algorithm", *Proceedings 4th International Conference on Genetic Algorithms*, pp. 174-181, 1991.

23. De Jong, K.A. "An Annalysis of the Behavior of a Class of Genetic Adaptive Systems." Diss. University of Michigan, Ann Arbor, 1975.

24. Eglese, R.W. "Simulated Annealing: A Tool for Operational Research", European Journal of Operational Research, vol 46 pp. 271-281, 1990.

25. Esbensen, H. "A Genetic Algorithm for Macro Cell Placement", *Proceedings European Design Automation Conference*, pp 52-57, 1992.

26. Esbensen, H. and Mazumder, P. "SAGA: A Unification of the Genetic Algorithm with Simulated Annealing and its Application to the Macro-Cell Placement", *Proceedings 7th International Conference on VLSI Design*, pp 211-214, 1994.

27. Fleurent, C. and Ferland, J.A. "Genetic Hybrids for the Quadratic Assignment Problem", DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1993.

28. Fogarty, T.C. and Huang, R. "Implementing the Genetic Algorithm on Transputer based Parallel Processing Systems", Parallel Problem Solving from Nature, pp. 145-149, 1991.

29. Fogel, L.J. "Autonomous Automata", *Independent Research*, Vol. 4, pp. 14-19, 1962.

30. Freisleben, B. and Merz, P. "A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problem", *International Conference on Evolutionary Computation*, pp. 44-50, 1996.

31. Garey, M.R. and Johnson, D.S. Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, 1979.

32. Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Mancheck, R. and Sunderam, V.B. *PVM 3 User's Guide and Reference Manual*. Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1994.

33. Geman, S. and Geman, D. "Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images", *IEEE Transaction. Pattern Analysis and Machine Intelligence*, Vol. 6, pp. 721-741, 1984.

34. Glover, F. and Greenberg, H.J. "New Approaches for Heuristic Search: A Bilateral Linkage with Artificial Intelligence", European Journal of Operational Research, Vol. 39, pp. 119-130, 1989.

35. Goldberg, D.E. "A note on Boltzmann Tournament Selection for Genetic Algorithms and Population Oriented Simulated Annealing", Complex systems, pp. 445-460, 1990.

36. Goldberg, D.E.. Genetic Algorithms in Search, Optimization and Machine Learning. New York: Addison-Wesley, 1989.

37. Greene, J. and Supowit, K. " Simulated Annealing without Rejected Moves", *IEEE Transactions on Computer Aided Design*, Vol. 5, pp. 221-228, 1986.

38. Greening, D.R. "Parallel Simulated Annealing Techniques", Physica D, Vol. 42, pp. 293-306, 1990.

39. Grefenstette, J. Incorporating Problem Specific Knowledge into Genetic Algorithms. In Genetic Algorithms and Simulated Annealing, pp. 42-60, L. Davis(ed.). Morgan Kauffman Publishers Inc., Los Altos - CA, 1987.

40. Grosso, P.B. Computer Simulations of Genetic Adaptation: Parallel Sub-component interaction in a multilocus model. Unpublished doctoral dissertation, The University of Michigan, 1985.

41. Grover, L.K. "A New Simulated Annealing Algorithm for the Standard Cell Placement", *Proceedings IEEE International Conference on Computer Aided Design*, pp. 378-380, 1986.

42. Hajek, B. "Cooling Schedules for Optimal Annealing", Mathematics of Operations Research, vol 13 pp. 311-329, 1988.

43. Harvey, W. D. and Ginsberg, M.L. "Limited Discrepency Search", *Proceedings 14th International Joint Conference in Artificial Intelligence*, 1995.

44. Hertz, A., de Werra, D. and Chams, M. "Some Experiments with Simulated Annealing for Colouring Graphs", European Journal of Operational Research, vol 32 pp. 260-266, 1987.

45. Hinton, G.E. and Nolan, S.J. "How Learning can guide evolution", Complex Systems, 1:495-502, 1987.

46. Holland, J.. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI, 1975.

47. Janaki Ram, D., Sreenivas, T.H. and Ganapathy Subramaniam, K. "Parallel Simulated Annealing Algorithms", Journal of Parallel and Distributed Computing, Vol. 37, pp. 207-212, 1996.

48. Johnson, D.S., Aargon, C. R., McGeogh, L. and Schevon, C. "Optimization by Simulated Annealing: An Experimental Evaluation, Part II", *Technical Report : AT&T Bell Laboratories*, Murray-Hill, NJ, 1988.

49. Kirpatrick, S., Gelatt, C.D. and Vecchi, M. P. "Optimization by Simulated Annealing", Science 220, pp. 671-680, 1983.

50. Koakutsu, S., Kang, M. and Dai, W.W. "Genetic Simulated Annealing and Application to Non-Slicing Floorplan Design", *Proceedings 5th ACM/SIGDA Pysical Design Workshop*, pp. 134-141, 1996.

51. Lin, F. T., Kao, C. Y. and Hsu, C.C. "Incorporating Genetic Algorithms into Simulated Annealing", *Proceedings 4th International Symposium on Artificial Intelligence*, pp. 290-297, 1991.

52. Lin, S. and Kernighan, B. "An Effective Heursitic Algorithm for the Traveling Salesman Problem", Journal of Operations Research, 21:498, 1973.

53. Lundy, M. and Mees, A. "Convergence of an Anealing Algorithm", Mathematical Programming, Vol. 34, pp. 111-124, 1986.

54. Mahfoud, S.W. and Goldberg, D. "Parallel Recombinative Simulated Annealing: A Genetic Algorithm", *IlliGAL Report No. 9200*, 1993.

55. Martin, O.C. and Otto, S.W. Combining Simulated Annealing with Local Search Heuristics. Metaheuristics in Combinatorial Optimization, edited by G. Laporte and I. Osman, 1994.

56. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A. H. and Teller, E. "Equation of State Calculations by Fast Computing Machines", Journal of Chemical Physics, vol 21(6) pp. 1087-1092, 1953.

57. Mitra, D., Romeo, F. and Sangiovanni-Vincentelli, A.L. "Convergence of Finite Time Behavior of Simulated Annealing", Advances in Applied Probability, vol 18 pp. 742-771, 1986.

58. Moscato, P. and Fontanari, J. F. "Stochastic versus Deterministic Update in Simulated Annealing", Physics Letters A, vol 146 no. 4, pp 204-208, 1990.

59. Pearl,J. Probabilistic Reasoning in Intelligence Systems: Networks of Plausible Inference. Morgan-Kauffman, San Mateo, 1988.

60. Pettey, C.B., Leuze, M.R. and Grefenstette, J.J. "A Parallel Genetic Algorithm", In J.J. Grefenstette(Ed.), *Proceedings 2nd International Conference on Genetic Algorithm*, pp. 155-161, 1987.

61. Pettey, C.B. and Leuze, M.R. "A Theoretical Investigation of a Parallel Genetic Algorithm", In J.D. Schaffer(Ed.), *Proceedings 3rd International Conference on Genetic Algorithm*, pp.398-405, 1989.

62. Rechenberg, I. Evolutionsstrategie : Optimierung technisher Systeme nach Prinzipien der biologischen Evolution. Stuttgart, Germany : Fromann-Holzboog, 1973.

63. Reinelt, G. "TSPLIB - A Traveling Salesman Problem Library", ORSA Journal on Computing, vol 3 no. 4, pp. 376-384, 1991.

64. Rojas-Guzman, C. and Kramer, M.A. GALGO: A Genetic ALGOrithm Decision Support Tool for Complex Uncertain Systems Modeled with Bayesian Belief Networks", *Proceedings 9th Annual Conference on Uncertainty in Artificial Intelligence*, 1993.

65. Santos, E. "A Fast Hill-Climbing Approach without an Energy Function for Probabilistic Resoning", *Proceedings IEEE International Conference on Tools with Artificial Intelligence*, 1993.

66. Schwefel, H.-P. Evolutionsstrategie und numerische Optimierung Dissertation, Technische Universitat Berlin, Germany, 1975.

67. Shen, C.Y., Pao, Y.H. and Yip, P. "Scheduling Multiple Job Problems with Guided Evolutionary Simulated Annealing Approach", *Proceedings 1st IEEE Conference on Evolutionary Computation*, Vol. 2, pp. 700-706, Piscataway, NJ: IEEE Service Center, 1994.

68. Skorin-Kapov, J. "Tabu Search Applied to the Quadratic Assignment Problem", ORSA Journal on Computing, vol 2 pp. 33-45, 1990.

69. Tanese, R. "Distributed Genetic Algorithms", In J.D. Schaffer(Ed.), *Proceedings 3rd International Conference on Genetic Algorithm*, pp.434-439, 1989.

70. Tanese, R. "Parallel Genetic Algorithm for a Hypercube", In J.J. Grefenstette(Ed.), *Proceedings 2nd International Conference on Genetic Algorithm*, pp.177-183, 1987.

71. Whitley, D., Howe, A.E., Rana, S., Watson, J.P. and Barbulescu, L. "Comparing Heuristic Search Methods and Genetic Algorithms for Warehouse Scheduling", Technical Report published in Colorado State University, 1996.

72. Whitley, D., Gordon, V. S. and Mathias, K. E. "Lamarkian Evolution, The Baldwin Effect and Function Optimization", *International Conference on Evolutionary Computation*: PPSN 3 H-P Schwefel and R. Maenner, eds. Springer-Verlag, 1994.

73. Whitley, D. and Kauth, J. "GENITOR: A Different Genetic Algorithm", *Proceedings Rocky Mountain Conference on Artificial Intelligence*, pp. 118-130, 1988.

APPENDIX A: Source code for GA engine (without SA).

```
/* ------------------------------------------------------------------
-------
 DESCRIPTION:
   Program for the SimpleGA class and 1DBinaryStringGenome class.
   This program tries to implement Bayesian Belief Networks using pure
GA.
---------------------------------------------------------------------
---- */
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <math.h>
#include <time.h>

#include <GASStateGA.h> // we're going to use the simple GA
#include <GA1DBinStrGenome.h> // and the 1D binary string genome
#include <garandom.h>

//Define Constants used by BBN problem
#define MAXPARENTS       3
#define MAXTRUTH         12
#define NODES            30
#define MAX_ADJ          5
#define NET_FILE  "30_A_nodes.net"
#define POP_SIZE  200
#define MAX_GENERATION  400

//Define a struct to act as a data structure that contains
// global information about the network
struct {
      int   arity;
      int   parents;
      int   parent[MAXPARENTS];
      double      prob[MAXTRUTH];
      int   adj_count;
      int   adj[MAX_ADJ];
      } net [NODES];

//Define prototype for functions that are problem specific.
void read_net();
void compute_adjacent();

// This is the declaration of our obj function.
// and the mutation operator that incorporates Simulated Annealing
float Objective(GAGenome &);


int
main(int argc, char **argv)
{
//This program implements BBNs using 1D-Binary Strings and is a Steady
// State GA. See if we've been given a seed to use (for testing
// purposes).  When you specify a random seed, the evolution will be
// exactly the same each time you use that seed number.
```

```
  unsigned int seed = 0;
  for(int ii=1; ii<argc; ii++) {
    if(strcmp(argv[ii++],"seed") == 0) {
      seed = atoi(argv[ii]);
    }
  }


// Declare variables for the GA parameters and set them to
// some default values.

unsigned int iterator1;
unsigned int iterator2;
int length;
int ngen;
float pmut = 0.05;
float pcross = 0.05;

for(iterator1=0; iterator1<20; iterator1++)
{

   length   = NODES;
   ngen     = MAX_GENERATION;

   for(iterator2=0; iterator2<20; iterator2++)
   {
      read_net();

// Now create the GA and run it.  First we create a genome of the type
// that we want to use in the GA.  The ga doesn't operate on this genome
// in the optimization - it just uses it to clone a population
// of  genomes.

      GA1DBinaryStringGenome genome(length, Objective);

// Now that we have the genome, we create the genetic algorithm and set
// its parameters - number of generations, mutation probability, and
// crossover probability.  And finally we tell it to evolve itself.

      GASteadyStateGA ga(genome);
      ga.pReplacement(1.0);
      ga.nBestGenomes(10);

      ga.populationSize(POP_SIZE);
      ga.nGenerations(ngen);
      ga.pMutation(pmut);
      ga.pCrossover(pcross);

      ga.selectScores(GAStatistics::AllScores);
      ga.parameters(argc, argv);

      ga.initialize(seed);
      while(!ga.done())
      {
            ga.step();
      }

      // Now we print out the best genome that the GA found.

      genome = ga.statistics().bestIndividual();

      cout <<"Pmut="<<pmut<<"\n";
      cout <<"Pcross="<<pcross<<"\n";
```

```
        cout <<"Best_Indiv=" <<genome<<"\n"; cout.flush();
        cout <<"Best_score="<< genome.score()<<"\n"; cout.flush();

        pmut = pmut + 0.05;

    }//end of inner loop which increments P_cross

        pcross = pcross + 0.05;
        pmut     = 0.05;

} //end of outer for loop which increments P_mut

        // That's it!
        return 0;
}


//Here are the functions specific to this problem.
void read_net()
{
int node, truth, parent, i;

        ifstream in(NET_FILE);
        if(!in) {
                cerr << "could not read data filefile " << NET_FILE << "\n";
                exit(1);
        }

        for(node = 0;node <NODES; node++)
                in >> net[node].arity;

        for(node = 0;node <NODES; node++)
        {
                in >> net[node].parents;
                truth = net[node].arity;

                for (parent = 0 ; parent < net[node].parents ; parent ++)
                {
                        in >> net[node].parent[parent];
                        net[node].parent[parent]-- ;
                        truth = truth * net[net[node].parent[parent]].arity;
                }

                for (i = 0 ; i < truth ; i ++ )
                        in >> net[node].prob[i];
        }

        compute_adjacent();
        in.close();
}

void compute_adjacent()
{
int node, other, i, parent;

        for(node = 0;node <NODES; node++)
          {
                net [node].adj_count = net[node].parents;
                for (i = 0 ; i < net[node].parents ; i ++ )
                        net[node].adj[i] = net[node].parent [i];

                for (other = 0 ; other < NODES ;other ++ )
                {
                        if (other==node)
                        continue;
                  for (parent=0; parent < net [other].parents; parent++)
                  {
                        if (net [other].parent[parent]==node)
                              net  [node].adj  [net[node].adj_count++]  =
other;
                  }
            }
      }
}

// This is the objective function.
float
Objective(GAGenome& g) {
  GA1DBinaryStringGenome & genome = (GA1DBinaryStringGenome &)g;
  float score=1.0;
  int count=0;
  int node = 0;
  int parent, p, index, push;

      for(node=0; node<genome.length(); node++)
      {
            index = genome.gene(node);
            push  = net[node].arity;

            for(parent=0; parent < net[node].parents; parent++)
            {
                  p = net[node].parent[parent];
                  index += genome.gene(p) * push;
                  push  *= net[p].arity;
            }
            score *= net[node].prob[index];
      }
  return score;
}
```

APPENDIX B: Source code for GA engine with intelligent SA mutation.

```
/* ----------------------------------------------------------------------
-------
 DESCRIPTION:
   Program for the SimpleGA class and 1DBinaryStringGenome class.
   This program tries to implement Bayesian Belief Networks using pure
GA.
-----------------------------------------------------------------------
---- */
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <math.h>
#include <time.h>        //included for the calculation of CPU time used

#include <GASStateGA.h> // we're going to use the simple GA
#include <GA1DBinStrGenome.h> // and the 1D binary string genome
#include <garandom.h>

//Define Constants used by BBN problem
#define MAXPARENTS      3
#define MAXTRUTH        12
#define NODES           40
#define MAX_ADJ         5
#define NET_FILE  "40_B_nodes.net"
#define POP_SIZE  200

//Define simulated annealing parameters
#define GRADIENT        0.005
#define INITIAL_T       0.95
#define MAX_STAGNATION  30
#define MAX_GENERATION  200

//Define a struct to act as a data structure that contains global
//information about the network
struct {
      int   arity;
      int   parents;
      int   parent[MAXPARENTS];
      double      prob[MAXTRUTH];
      int   adj_count;
      int   adj[MAX_ADJ];
      } net [NODES];

//Define prototype for functions that are problem specific.
void read_net();
void compute_adjacent();

// This is the declaration of our obj function.
// and the mutation operator that incorporates Simulated Annealing
float pSim_Anneal = 0;
float Objective(GAGenome &);
int   Mutator(GAGenome&, float);
int   mutate_regular(GAGenome&, float);
```

int

```
main(int argc, char **argv)
{
// See if we've been given a seed to use (for testing purposes).
// When you specify a random seed, the evolution will be exactly the
same each time you use that seed number.

  unsigned int seed = 0;
  for(int ii=1; ii<argc; ii++) {
    if(strcmp(argv[ii++],"seed") == 0) {
      seed = atoi(argv[ii]);
    }
  }

// Declare variables for the GA parameters and set them to some
// default values.

unsigned int iterator1;
unsigned int iterator2;

float SAP = 0.0;
float p_real_mut = 0.0;
float pcross = 0.0;
float alpha = 0.0;
float pmut = 0.0;
float MUT_ARRAY[2] = {0.065, 0.1};
float CROSS_ARRAY[2]= {0.517, 0.3556};


for(iterator1=0;iterator1<2;iterator1++)
{
p_real_mut = MUT_ARRAY[iterator1];
pcross     = CROSS_ARRAY[iterator1];
SAP      = 0.005;

for(iterator2=0; iterator2<MAX_GENERATION; iterator2++)
{

  int length   = NODES;
  int ngen     = MAX_GENERATION;

  alpha = SAP/p_real_mut;
  pSim_Anneal = alpha/(1 + alpha);
  pmut = p_real_mut/(1 - pSim_Anneal);

   read_net();

// Now create the GA and run it.  First we create a genome of the type
// that we want to use in the GA.  The ga doesn't operate on this genome
// in the optimization - it just uses it to clone a population
// of genomes.

  GA1DBinaryStringGenome genome(length, Objective);
  genome.mutator(::Mutator);

// Now that we have the genome, we create the genetic algorithm and set
// its parameters - number of generations, mutation probability, and
// crossover probability.  And finally we tell it to evolve itself.


  GASteadyStateGA ga(genome);
  ga.pReplacement(1.0);
  ga.nBestGenomes(10);

  ga.populationSize(POP_SIZE);
  ga.nGenerations(ngen);
  ga.pMutation(p_real_mut);
  ga.pCrossover(pcross);

  ga.selectScores(GAStatistics::AllScores);
  ga.parameters(argc, argv);

  ga.initialize(seed);
  while(!ga.done())
  {
      ga.step();
  }

// Now we print out the best genome that the GA found.

 genome = ga.statistics().bestIndividual();

  printf("PSA=%f\n",pSim_Anneal);

  printf("PrM=%f\n",p_real_mut);

  printf("PC=%f\n",pcross);

  printf("PSAP=%f\n",SAP);

  printf("PM=%f\n",pmut);


  cout <<"Score="<< genome.score()<<"\n"; cout.flush();

  SAP = SAP + 0.005;

} //end of for loop

}
// That's it!
      return 0;
}


//Here are the functions specific to this problem.
void read_net()
{
int node, truth, parent, i;

      ifstream in(NET_FILE);
      if(!in) {
            cerr << "could not read data filefile " << NET_FILE << "\n";
            exit(1);
      }

      for(node = 0;node <NODES; node++)
            in >> net[node].arity;

      for(node = 0;node <NODES; node++)
      {
            in >> net[node].parents;
            truth = net[node].arity;

            for (parent = 0 ; parent < net[node].parents ; parent ++)
            {
                  in >> net[node].parent[parent];
                  net[node].parent[parent]-- ;
                  truth = truth * net[net[node].parent[parent]].arity;
            }
            for (i = 0 ; i < truth ; i ++ )
                  in >> net[node].prob[i];
      }

      compute_adjacent();
      in.close();
}

void compute_adjacent()
{
int node, other, i, parent;

      for(node = 0;node <NODES; node++)
        {
            net [node].adj_count = net[node].parents;
            for (i = 0 ; i < net[node].parents ; i ++ )
                  net[node].adj[i] = net[node].parent [i];

            for (other = 0 ; other < NODES ;other ++ )
            {
                  if (other==node)
                        continue;
                  for (parent=0; parent < net [other].parents; parent++)
                  {
                        if (net [other].parent[parent]==node)
                              net [node].adj [net[node].adj_count++] =
other;
                  }
            }
      }
}

int mutate_regular(GAGenome& g, float pmut)
{
  GA1DBinaryStringGenome &child=(GA1DBinaryStringGenome &)g;
  register int n, i;

  if(pmut <= 0.0)
      return(0);

  float nMut = pmut * (float)(child.length());

  if(nMut < 1.0){                   // we have to do a flip test on each bit
    nMut = 0;
    for(i=child.length()-1; i>=0; i--){
      if(GAFlipCoin(pmut)){
        child.gene(i, ((child.gene(i) == 0) ? 1 : 0));
        nMut++;
      }
    }
  }
  else{                             // only flip the number of bits we need
to flip
    for(n=0; n<nMut; n++){
      i = GARandomInt(0,child.length()-1); //the index of bit to flip
      child.gene(i, ((child.gene(i) == 0) ? 1 : 0));
    }
  }
  return((int)nMut);
}

int
Mutator(GAGenome& g, float pmut)
{
register int n, i;
```

```
float champion, challenger, temp, odds, diff, p;
int stagnation = 0;
int generation = 0;

        if(GARandomFloat() < pSim_Anneal)
        {
                //DO Simulated Annealing

                champion = Objective(g);

                temp = INITIAL_T;
                do
                {
                        stagnation++;
                        generation++;

                        //Call regular mutation operator
                        mutate_regular(g,pmut);
                        challenger = Objective(g);

                        if(challenger > champion)
                        {
                                champion = challenger;
                                stagnation = 0;
                        }
                        else
                        {
                                diff = challenger - champion;
                                odds = exp (diff/temp);
                                p = drand48();
                                if (p < odds)
                                {
                                        champion = challenger;
                                        stagnation = 0;
                                }
                        }
                        temp=temp*GRADIENT;
                }while( (stagnation < MAX_STAGNATION) && (generation <
MAX_GENERATION) );
        return 1;
        }
        else
                //DO Regular Mutation without any Simulated Annealing
                return(mutate_regular(g,pmut));
}
// This is the objective function.
float
Objective(GAGenome& g) {
  GA1DBinaryStringGenome & genome = (GA1DBinaryStringGenome &)g;
  float score=1.0;
  int count=0;
  int node = 0;
  int parent, p, index, push;

      for(node=0; node<genome.length(); node++)
      {
            index = genome.gene(node);
            push  = net[node].arity;

            for(parent=0; parent < net[node].parents; parent++)
            {
                  p = net[node].parent[parent];
                  index += genome.gene(p) * push;
                  push  *= net[p].arity;
            }
            score *= net[node].prob[index];
      }
  return score;
}
```

APPENDIX C: Sample data for pure GA runs (without SA)on BBN_30A.

GA Runs for BBN_30A

Pmut	Pcross	Score
0.05	0.05	1.82E-04
0.05	0.1	2.39E-04
0.05	0.15	2.39E-04
0.05	0.2	2.39E-04
0.05	0.25	2.39E-04
0.05	0.3	2.39E-04
0.05	0.35	2.39E-04
0.05	0.4	2.39E-04
0.05	0.45	2.39E-04
0.05	0.5	2.39E-04
0.05	0.55	2.39E-04
0.05	0.6	2.39E-04
0.05	0.65	2.39E-04
0.05	0.7	2.39E-04
0.05	0.75	2.39E-04
0.05	0.8	2.35E-04
0.05	0.85	2.39E-04
0.05	0.9	2.39E-04
0.05	0.95	2.39E-04
0.05	1	2.39E-04
0.1	0.05	2.35E-04
0.1	0.1	2.39E-04
0.1	0.15	2.39E-04
0.1	0.2	2.39E-04
0.1	0.25	2.39E-04
0.1	0.3	2.39E-04
0.1	0.35	2.39E-04
0.1	0.4	2.39E-04
0.1	0.45	2.39E-04
0.1	0.5	2.39E-04
0.1	0.55	2.39E-04
0.1	0.6	2.39E-04
0.1	0.65	2.39E-04
0.1	0.7	2.39E-04
0.1	0.75	2.39E-04
0.1	0.8	2.35E-04
0.1	0.85	2.39E-04
0.1	0.9	2.39E-04
0.1	0.95	2.39E-04
0.1	1	2.39E-04
0.15	0.05	2.39E-04
0.15	0.15	2.39E-04
0.15	0.2	2.39E-04
0.15	0.35	2.39E-04
0.15	0.45	2.39E-04
0.15	0.55	2.39E-04
0.15	0.6	2.39E-04
0.15	0.7	2.39E-04
0.15	0.45	2.39E-04
0.15	0.5	2.35E-04
0.15	0.55	2.39E-04
0.15	0.6	2.39E-04
0.15	0.65	1.89E-04
0.15	0.7	2.39E-04
0.15	0.75	1.82E-04
0.15	0.8	1.80E-04
0.15	0.85	1.80E-04
0.15	0.9	1.76E-04
0.15	0.95	1.76E-04
0.15	1	1.68E-04
0.2	0.05	2.39E-04
0.2	0.1	2.39E-04
0.2	0.15	2.39E-04
0.2	0.2	2.39E-04
0.2	0.25	2.35E-04
0.2	0.3	1.82E-04
0.2	0.35	1.82E-04
0.2	0.4	1.82E-04
0.2	0.45	1.80E-04
0.2	0.5	1.80E-04
0.2	0.55	1.66E-04
0.2	0.6	1.66E-04
0.2	0.65	1.66E-04
0.2	0.7	1.57E-04
0.2	0.75	1.45E-04
0.2	0.8	1.45E-04
0.2	0.85	1.44E-04
0.2	0.9	1.26E-04
0.2	0.95	1.17E-04
0.2	1	1.08E-04
0.25	0.05	2.35E-04
0.25	0.1	1.48E-04
0.25	0.15	1.45E-04
0.25	0.2	1.38E-04
0.25	0.25	8.88E-05
0.25	0.3	1.38E-04
0.25	0.35	1.31E-04
0.25	0.4	1.29E-04
0.25	0.45	8.09E-05
0.25	0.5	1.25E-04
0.25	0.55	1.17E-04
0.25	0.6	7.97E-05
0.25	0.65	1.10E-04
0.25	0.7	1.08E-04
0.25	0.75	1.08E-04
0.25	0.8	9.68E-05
0.25	0.85	9.68E-05
0.25	0.9	9.67E-05
0.25	0.95	9.46E-05
0.25	1	6.37E-05
0.3	0.05	2.35E-04
0.3	0.1	1.57E-04
0.3	0.15	1.31E-04
0.3	0.2	1.26E-04
0.3	0.25	1.18E-04
0.3	0.3	1.15E-04
0.3	0.35	1.11E-04
0.3	0.4	1.11E-04
0.3	0.45	1.11E-04
0.3	0.5	1.08E-04
0.3	0.55	1.01E-04
0.3	0.6	9.52E-05
0.3	0.65	9.52E-05
0.3	0.7	8.96E-05
0.3	0.75	8.88E-05
0.3	0.8	7.74E-05
0.3	0.85	7.73E-05
0.3	0.9	7.64E-05
0.3	0.95	7.23E-05
0.3	1	6.17E-05
0.35	0.05	2.39E-04
0.35	0.1	2.39E-04
0.35	0.15	1.80E-04
0.35	0.2	1.56E-04
0.35	0.25	1.31E-04
0.35	0.3	1.26E-04
0.35	0.35	1.10E-04
0.35	0.4	1.08E-04
0.35	0.45	1.08E-04
0.35	0.5	9.52E-05
0.35	0.55	8.54E-05
0.35	0.6	7.85E-05
0.35	0.65	7.08E-05
0.35	0.7	6.48E-05
0.35	0.75	6.22E-05
0.35	0.8	5.99E-05
0.35	0.85	5.67E-05
0.35	0.9	5.13E-05
0.35	0.95	4.82E-05
0.35	1	4.08E-05
0.4	0.05	5.89E-05
0.4	0.1	8.35E-05
0.4	0.15	5.11E-05
0.4	0.2	6.42E-05
0.4	0.25	1.26E-04
0.4	0.3	3.12E-05
0.4	0.35	9.62E-05
0.4	0.4	7.20E-05
0.4	0.45	6.48E-05
0.4	0.5	2.40E-05
0.4	0.55	4.46E-05
0.4	0.6	9.57E-05
0.4	0.65	9.48E-05
0.4	0.7	7.64E-05
0.4	0.75	2.62E-05
0.4	0.8	3.22E-05
0.4	0.85	8.35E-05
0.4	0.9	2.54E-05
0.4	0.95	5.09E-05
0.4	1	4.66E-05
0.45	0.05	2.08E-05
0.45	0.1	4.36E-05
0.45	0.15	5.93E-05
0.45	0.2	5.86E-05
0.45	0.25	6.48E-05
0.45	0.3	2.29E-05
0.45	0.35	1.74E-05
0.45	0.4	3.09E-05
0.45	0.45	1.90E-05
0.45	0.5	4.11E-05
0.45	0.55	2.77E-05
0.45	0.6	4.81E-05
0.45	0.65	8.28E-05
0.45	0.7	7.64E-05
0.45	0.75	3.31E-05
0.45	0.8	6.73E-05
0.45	0.85	3.09E-05
0.45	0.9	1.67E-05
0.45	0.95	6.67E-05
0.45	1	5.46E-05
0.5	0.05	1.53E-05
0.5	0.1	1.17E-05
0.5	0.15	3.02E-05
0.5	0.2	1.34E-05
0.5	0.25	4.92E-05
0.5	0.3	7.03E-05
0.5	0.35	7.80E-05
0.5	0.4	7.32E-05
0.5	0.45	3.24E-05
0.5	0.5	3.55E-05
0.5	0.55	1.29E-04
0.5	0.6	2.02E-05

Pmut	Pcross	Score
0.5	0.65	6.80E-05
0.5	0.7	3.34E-05
0.5	0.75	2.63E-05
0.5	0.8	1.50E-05
0.5	0.85	2.77E-05
0.5	0.9	2.09E-05
0.5	0.95	2.88E-05
0.5	1	5.18E-05
0.55	0.05	2.28E-05
0.55	0.1	1.38E-05
0.55	0.15	4.50E-05
0.55	0.2	1.18E-05
0.55	0.25	1.75E-05
0.55	0.3	1.18E-05
0.55	0.35	9.88E-06
0.55	0.4	1.18E-05
0.55	0.45	1.01E-04
0.55	0.5	1.47E-05
0.55	0.55	3.96E-05
0.55	0.6	7.80E-05
0.55	0.65	1.62E-05
0.55	0.7	3.26E-05
0.55	0.75	1.53E-05
0.55	0.8	4.48E-05
0.55	0.85	7.64E-05
0.55	0.9	1.18E-04
0.55	0.95	6.27E-05
0.55	1	3.32E-05
0.6	0.05	1.27E-04
0.6	0.1	2.39E-05
0.6	0.15	2.07E-05
0.6	0.2	1.63E-05
0.6	0.25	2.34E-05
0.6	0.3	2.85E-05
0.6	0.35	5.70E-05
0.6	0.4	1.74E-05
0.6	0.45	2.10E-05
0.6	0.5	4.61E-05
0.6	0.55	3.96E-05
0.6	0.6	1.28E-05
0.6	0.65	1.58E-05
0.6	0.7	2.67E-05
0.6	0.75	2.63E-05
0.6	0.8	2.09E-05
0.6	0.85	2.09E-05
0.6	0.9	4.14E-05
0.6	0.95	4.50E-05
0.6	1	1.71E-05
0.65	0.05	1.18E-05
0.65	0.1	2.69E-05
0.65	0.15	2.15E-05
0.65	0.2	1.37E-05
0.65	0.25	9.72E-06
0.65	0.3	1.14E-05
0.65	0.35	2.89E-05
0.65	0.4	4.01E-05
0.65	0.45	1.24E-05
0.65	0.5	7.47E-06
0.65	0.55	6.22E-05
0.65	0.6	1.21E-05
0.65	0.65	2.64E-05
0.65	0.7	4.93E-05
0.65	0.75	1.24E-05
0.65	0.8	3.56E-05
0.65	0.85	4.58E-05
0.65	0.9	3.05E-05
0.65	0.95	1.26E-05
0.65	1	1.23E-05
0.7	0.05	2.87E-05
0.7	0.1	1.32E-05
0.7	0.15	5.93E-05
0.7	0.2	1.49E-05
0.7	0.25	4.16E-05
0.7	0.3	9.83E-06
0.7	0.35	1.40E-05
0.7	0.4	1.47E-05
0.7	0.45	1.94E-05
0.7	0.5	1.34E-05
0.7	0.55	5.53E-05
0.7	0.6	6.86E-05
0.7	0.65	2.14E-05
0.7	0.7	7.55E-06
0.7	0.75	4.57E-05
0.7	0.8	1.04E-05
0.7	0.85	1.73E-05
0.7	0.9	1.17E-05
0.7	0.95	1.11E-05
0.7	1	1.23E-05
0.75	0.05	3.32E-05
0.75	0.1	2.67E-05
0.75	0.15	5.00E-06
0.75	0.2	2.72E-05
0.75	0.25	2.29E-05
0.75	0.3	2.52E-05
0.75	0.35	1.57E-05
0.75	0.4	7.85E-06
0.75	0.45	1.36E-05
0.75	0.5	4.14E-05
0.75	0.55	8.28E-05
0.75	0.6	1.88E-05
0.75	0.65	1.10E-05
0.75	0.7	1.32E-05
0.75	0.75	1.26E-04
0.75	0.8	1.18E-05
0.75	0.85	3.47E-06
0.75	0.9	1.75E-05
0.75	0.95	9.20E-06
0.75	1	1.04E-05
0.8	0.05	5.23E-06
0.8	0.1	1.85E-05
0.8	0.15	4.43E-05
0.8	0.2	8.73E-06
0.8	0.25	5.99E-05
0.8	0.3	5.13E-05
0.8	0.35	1.08E-05
0.8	0.4	6.38E-05
0.8	0.45	1.01E-05
0.8	0.5	2.68E-05
0.8	0.55	2.55E-05
0.8	0.6	1.91E-05
0.8	0.65	1.56E-05
0.8	0.7	1.05E-05
0.8	0.75	1.00E-05
0.8	0.8	7.38E-06
0.8	0.85	1.64E-05
0.8	0.9	4.99E-06
0.8	0.95	3.30E-05
0.8	1	2.28E-05
0.85	0.05	2.40E-05
0.85	0.1	4.16E-05
0.85	0.15	5.59E-06
0.85	0.2	1.91E-05
0.85	0.25	1.07E-05
0.85	0.3	8.50E-06
0.85	0.35	2.86E-05
0.85	0.4	1.17E-05
0.85	0.45	9.23E-06
0.85	0.5	3.49E-05
0.85	0.55	6.23E-06
0.85	0.6	5.59E-05
0.85	0.65	1.94E-05
0.85	0.7	8.65E-06
0.85	0.75	1.17E-04
0.85	0.8	5.39E-06
0.85	0.85	6.00E-05
0.85	0.9	9.95E-06
0.85	0.95	8.91E-06
0.85	1	4.75E-05
0.9	0.05	7.56E-06
0.9	0.1	5.18E-06
0.9	0.15	2.57E-05
0.9	0.2	1.23E-05
0.9	0.25	1.05E-05
0.9	0.3	1.27E-05
0.9	0.35	6.13E-06
0.9	0.4	4.07E-05
0.9	0.45	2.12E-05
0.9	0.5	1.98E-05
0.9	0.55	2.94E-05
0.9	0.6	4.66E-05
0.9	0.65	1.33E-05
0.9	0.7	2.06E-05
0.9	0.75	3.13E-05
0.9	0.8	2.97E-05
0.9	0.85	1.47E-05
0.9	0.9	9.68E-06
0.9	0.95	7.39E-06
0.9	1	3.86E-05
0.95	0.05	1.13E-05
0.95	0.1	4.86E-06
0.95	0.15	8.35E-06
0.95	0.2	1.70E-05
0.95	0.25	1.00E-05
0.95	0.3	1.22E-05
0.95	0.35	9.74E-06
0.95	0.4	4.91E-06
0.95	0.45	6.63E-06
0.95	0.5	1.43E-05
0.95	0.55	1.10E-05
0.95	0.6	4.90E-05
0.95	0.65	6.91E-06
0.95	0.7	2.04E-05
0.95	0.75	5.41E-06
0.95	0.8	5.27E-06
0.95	0.85	6.91E-06
0.95	0.9	2.91E-05
0.95	0.95	2.67E-05
0.95	1	4.41E-05
1	0.05	8.03E-06
1	0.1	1.07E-05
1	0.15	6.73E-06
1	0.2	1.84E-05
1	0.25	2.14E-05
1	0.3	1.24E-05
1	0.35	9.70E-06
1	0.4	1.05E-05
1	0.45	2.67E-05
1	0.5	1.71E-05
1	0.55	1.42E-05
1	0.6	8.43E-06
1	0.65	2.70E-05
1	0.7	7.30E-06
1	0.75	1.34E-05
1	0.8	8.45E-06
1	0.85	1.08E-05
1	0.9	2.99E-05
1	0.95	8.08E-06
1	1	3.43E-05

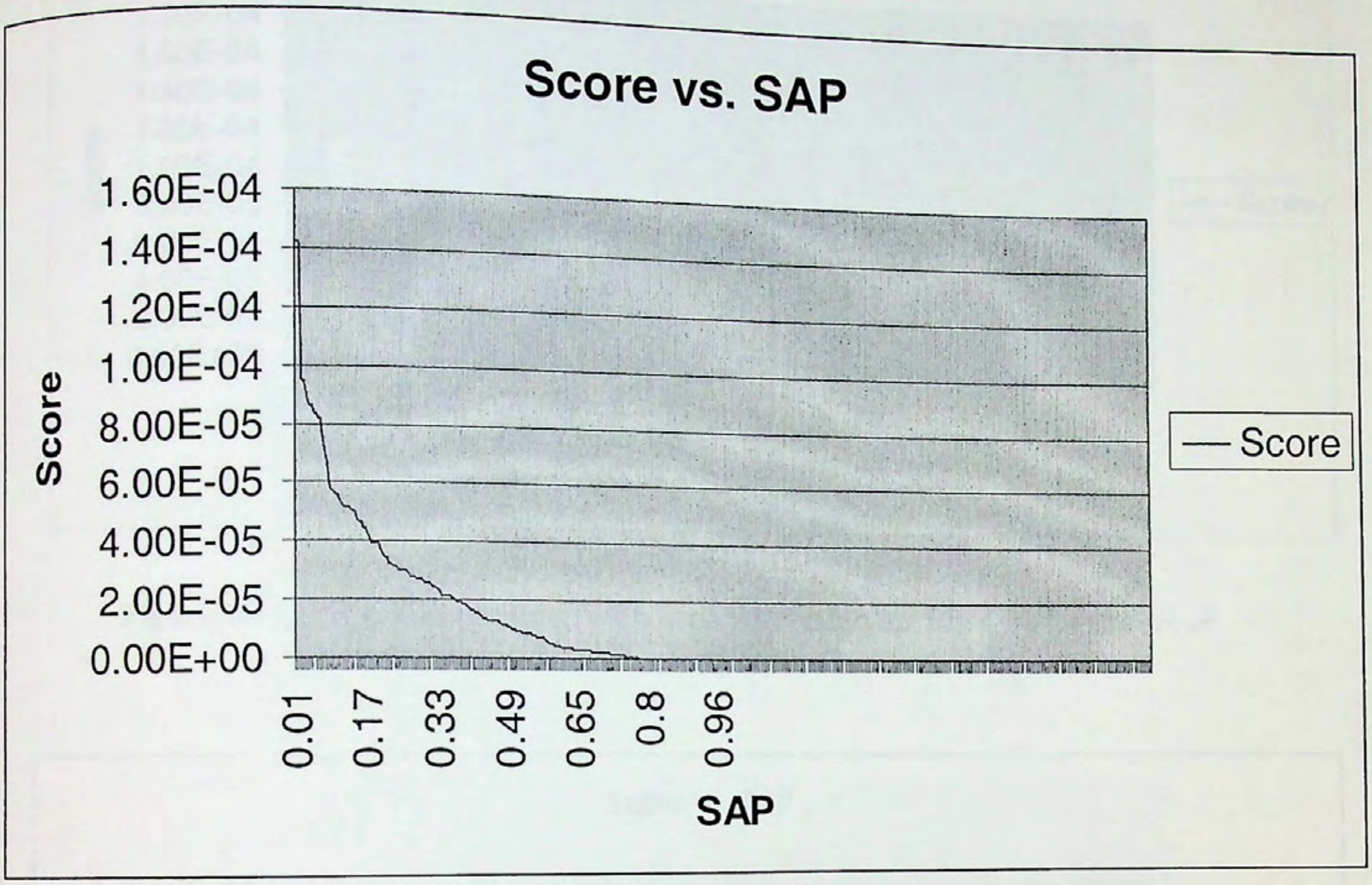

Figure 15: MPE values obtained with each SAP value for BBN_50 for the set of genetic parameters ($P_c = 0.49768$ & $P_m = 0.031918$).

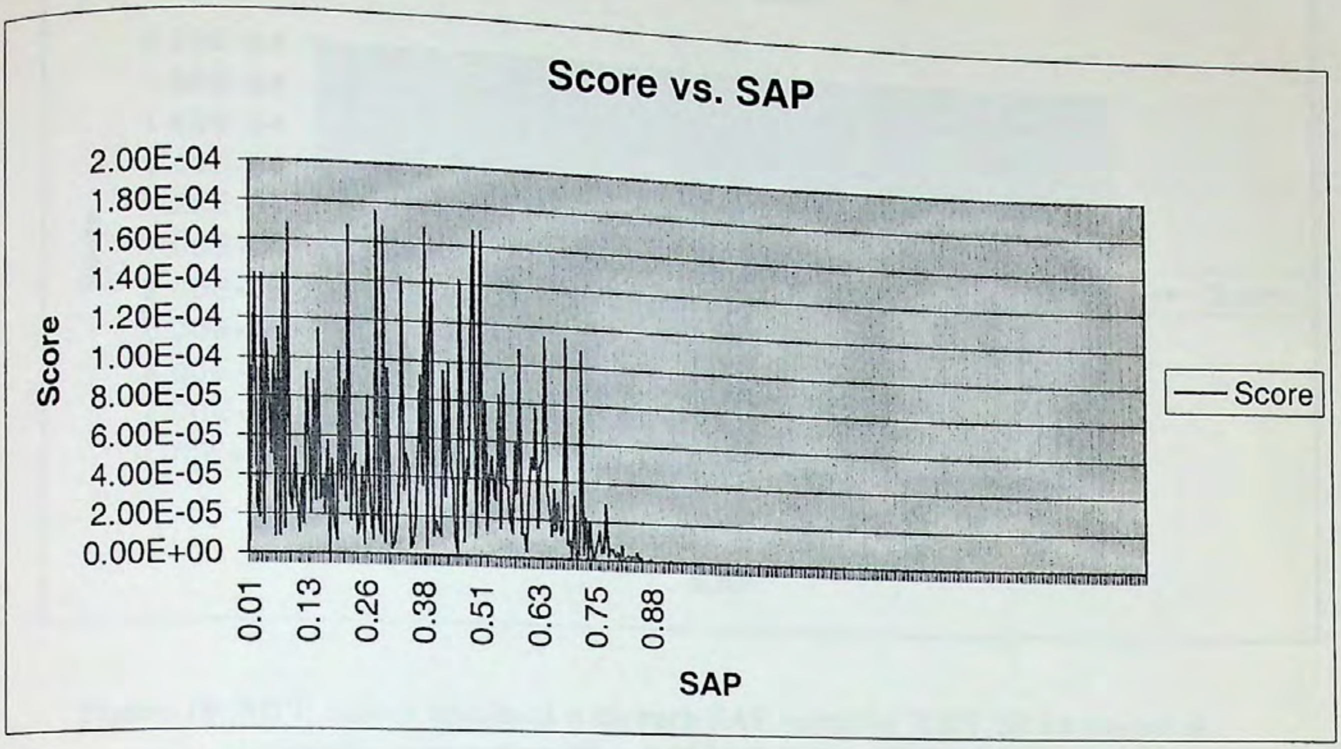

Figure 16: MPE values obtained with each SAP value for BBN_50 for the set of genetic parameters ($P_c = 0.42809$ & $P_m = 0.014346$).

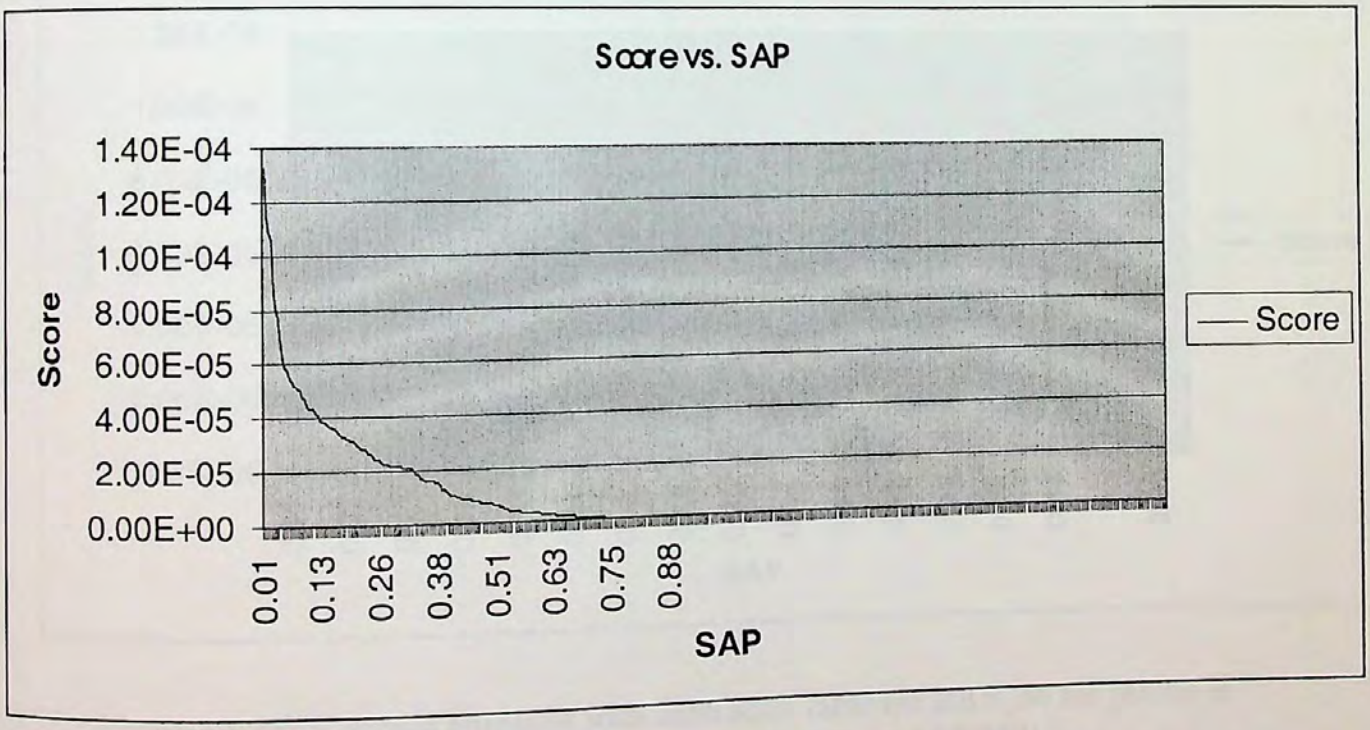

Figure 17: MPE values obtained with each SAP value for BBN_50 for the set of genetic parameters ($P_c = 0.18787$ & $P_m = 0.030529$).

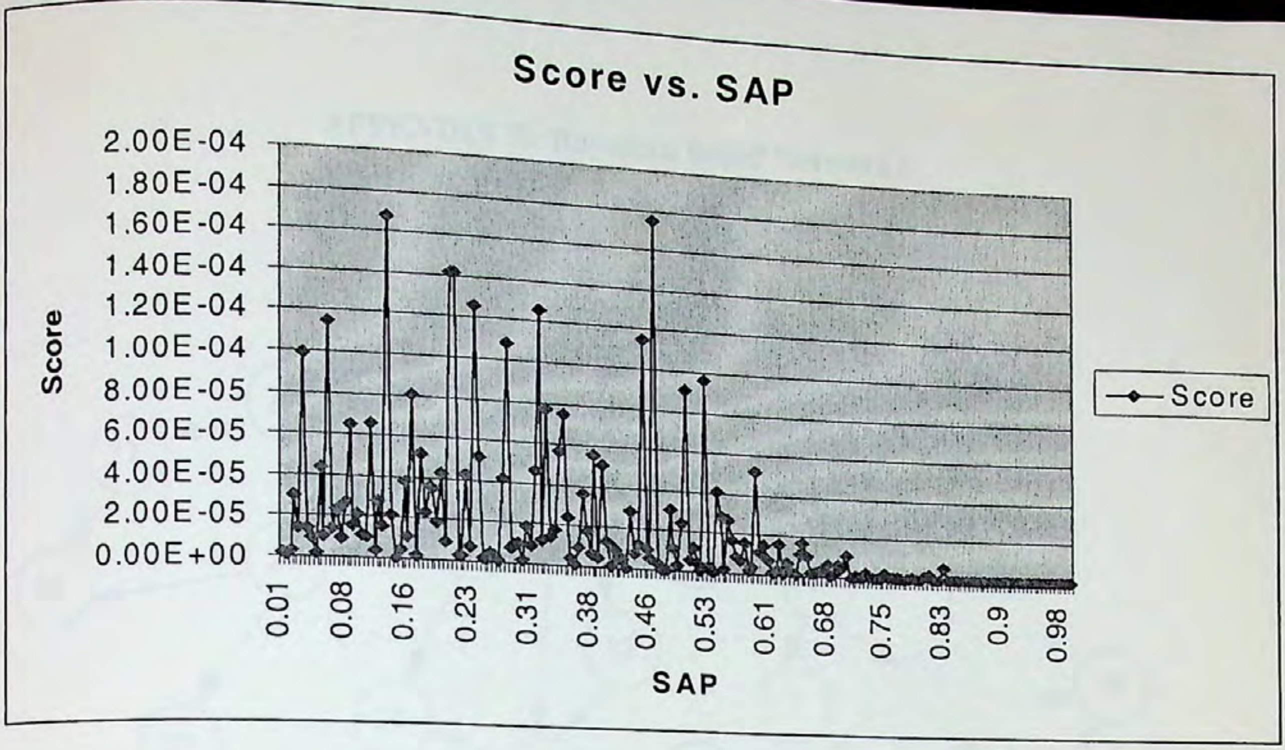

Figure 18: MPE values obtained with each SAP value for BBN_50 for the set of genetic parameters ($P_c = 0.062623$ & $P_m = 0.010176$).

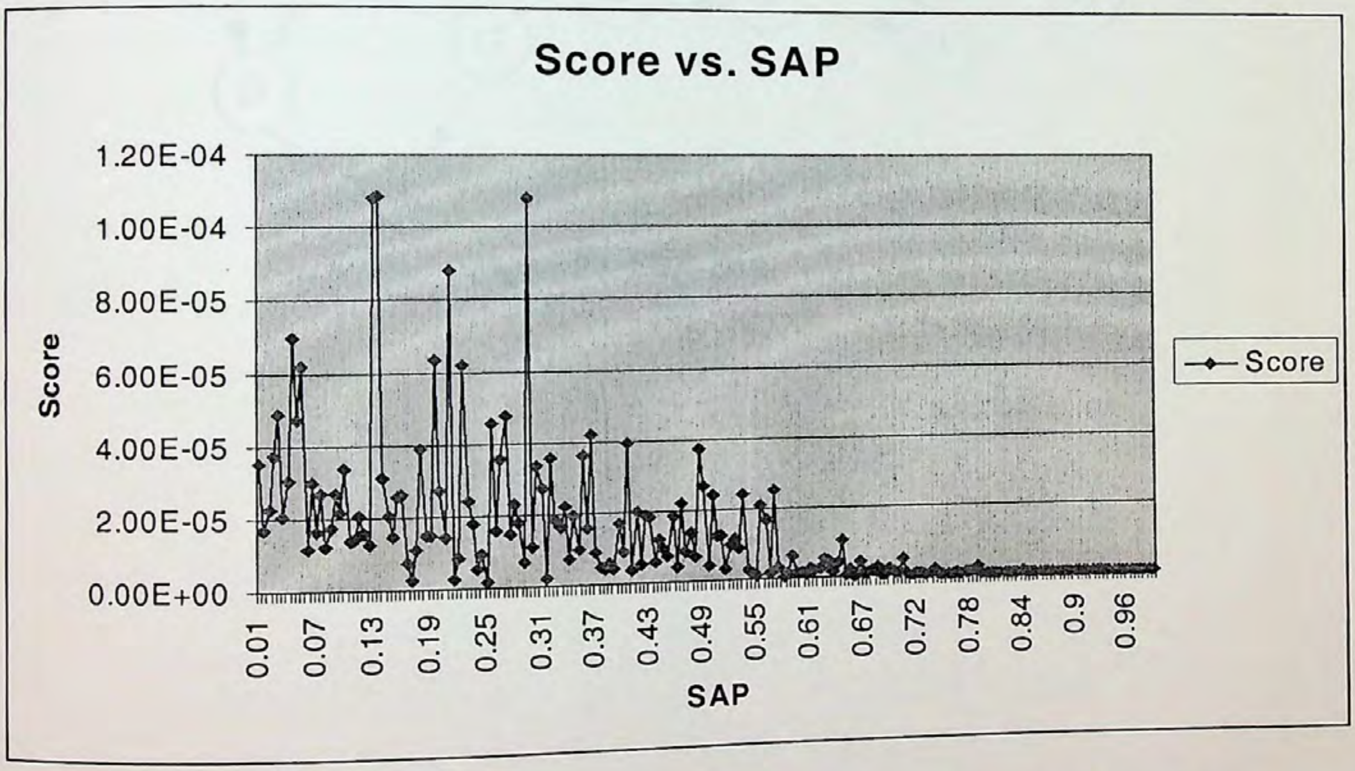

Figure 19: MPE values obtained with each SAP value for BBN_50 for the set of genetic parameters ($P_c = 0.125247$ & $P_m = 0.020352$).

APPENDIX E: Bayesian Belief Networks

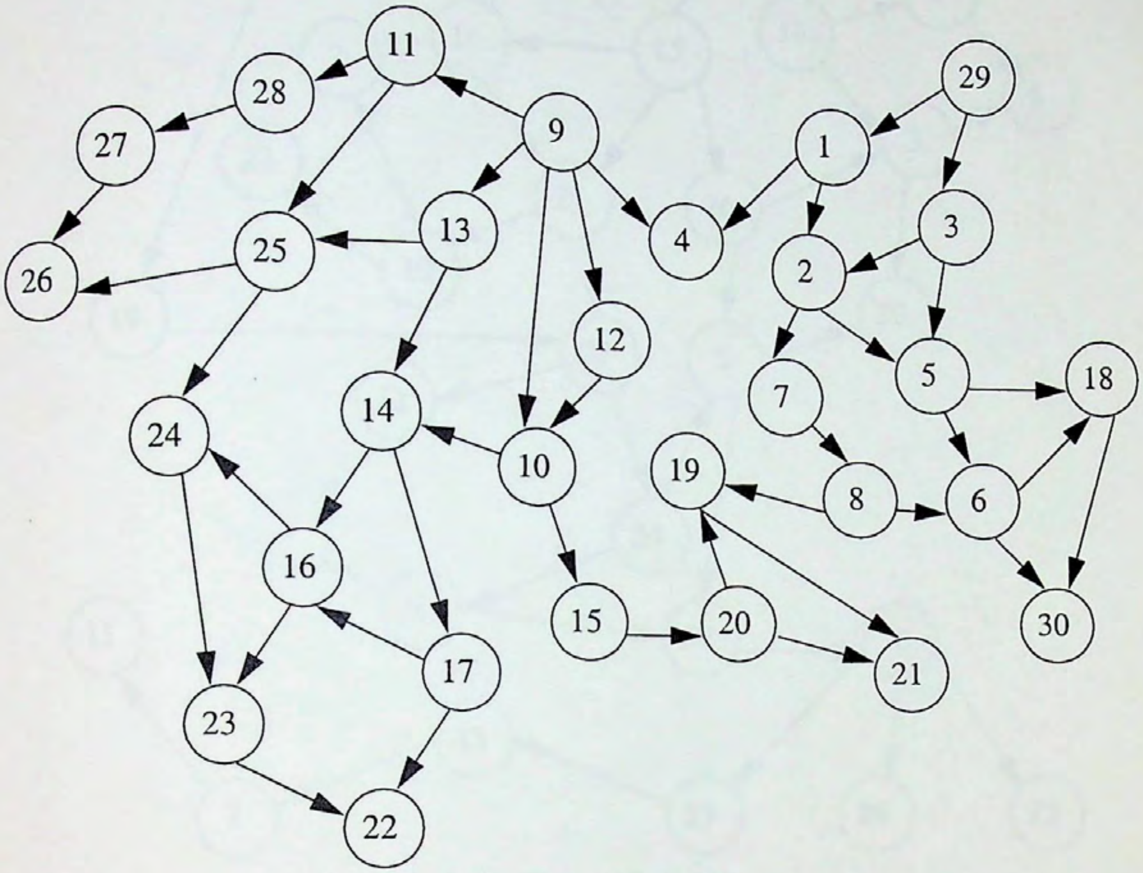

Figure 20: BBN_30A

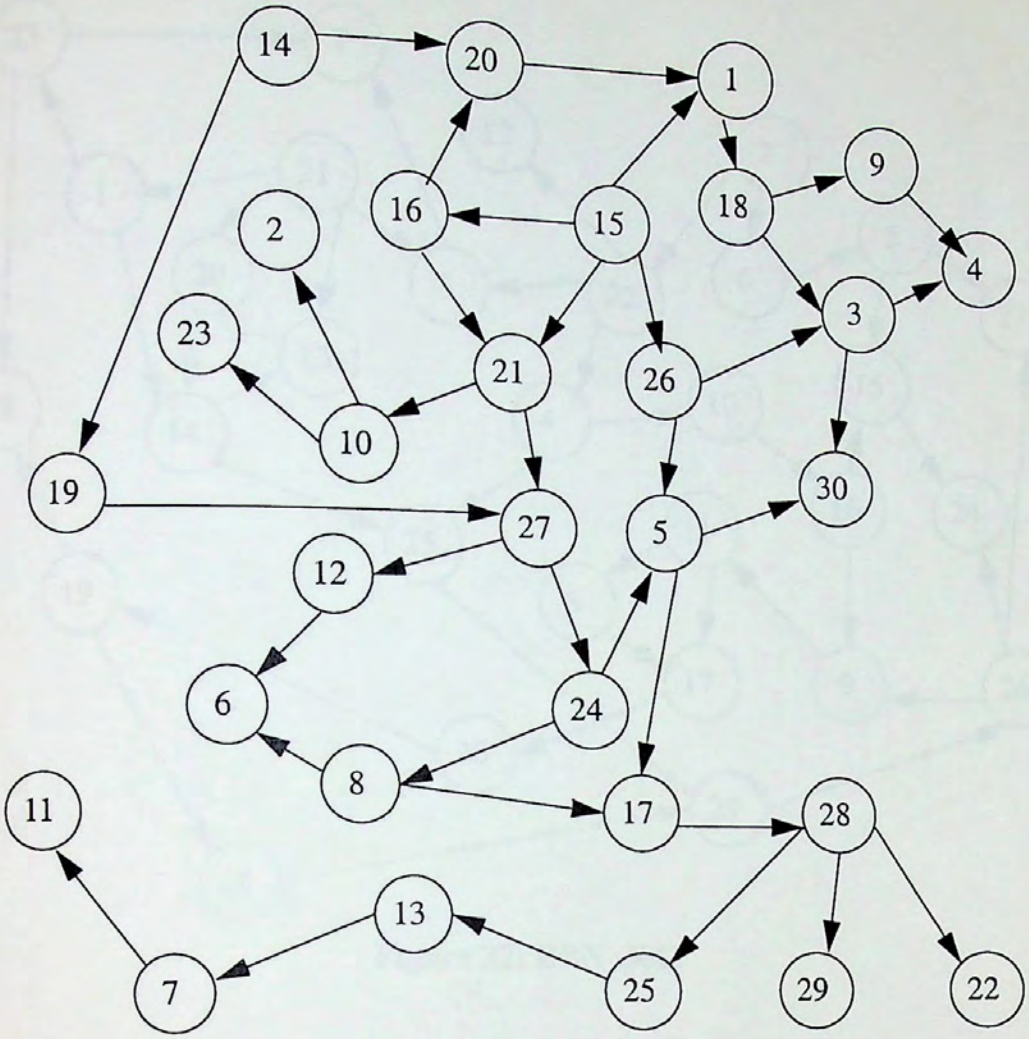

Figure 21: BBN_30B

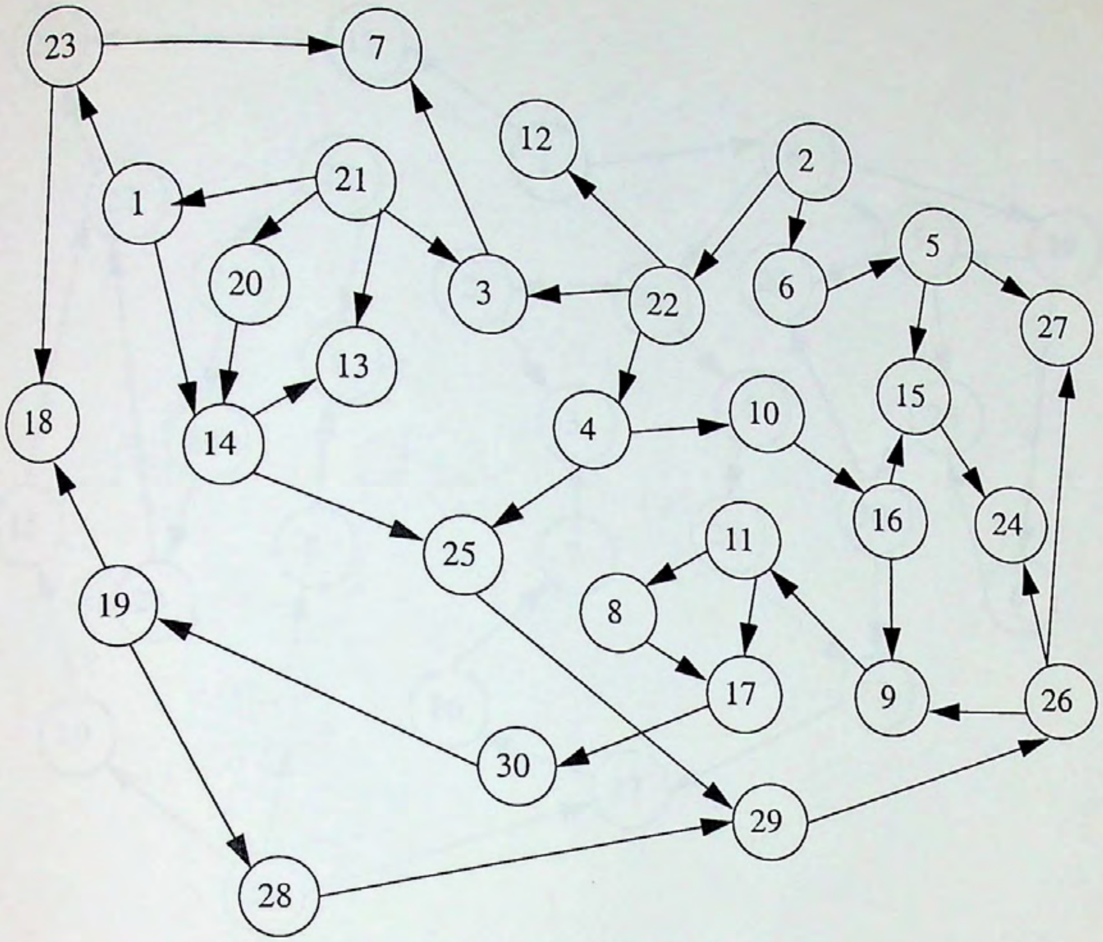

Figure 22: BBN_30C

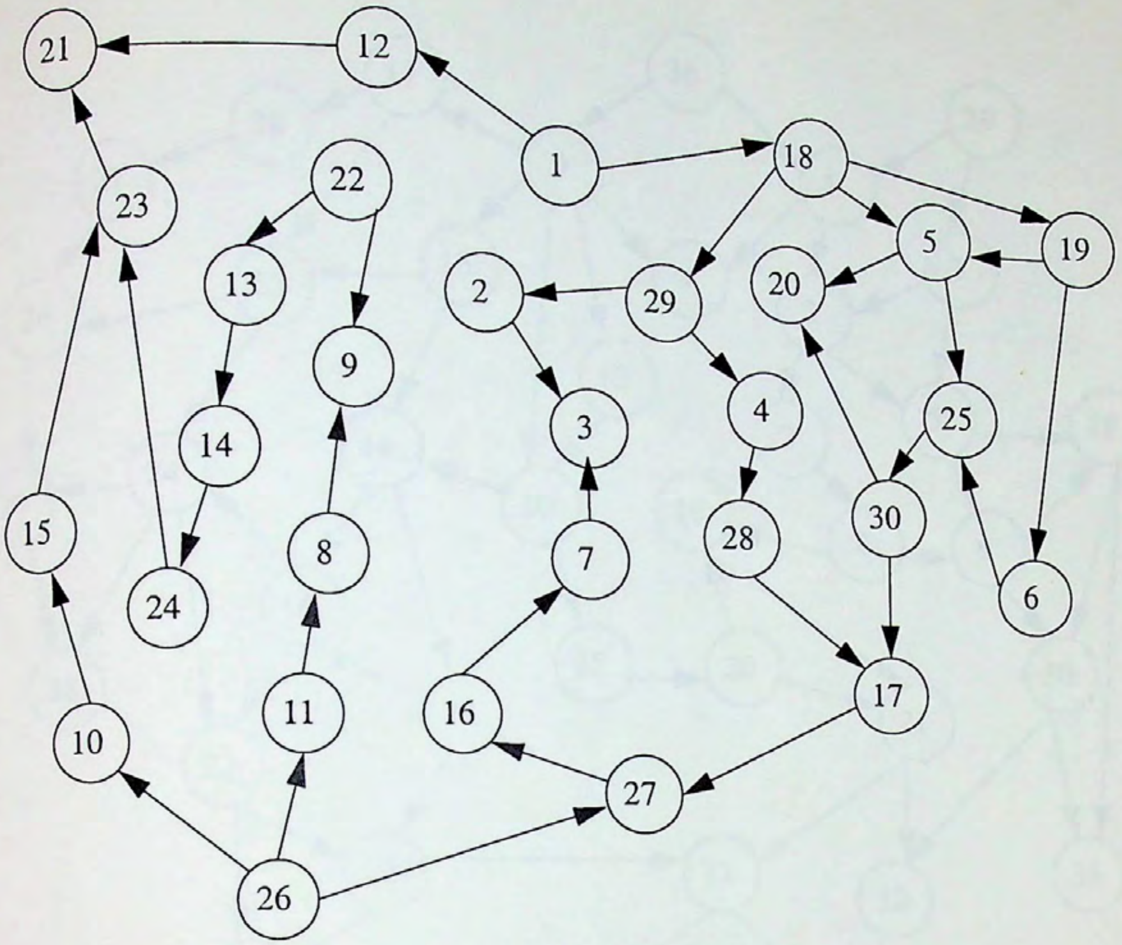

Figure 23: BBN_30D

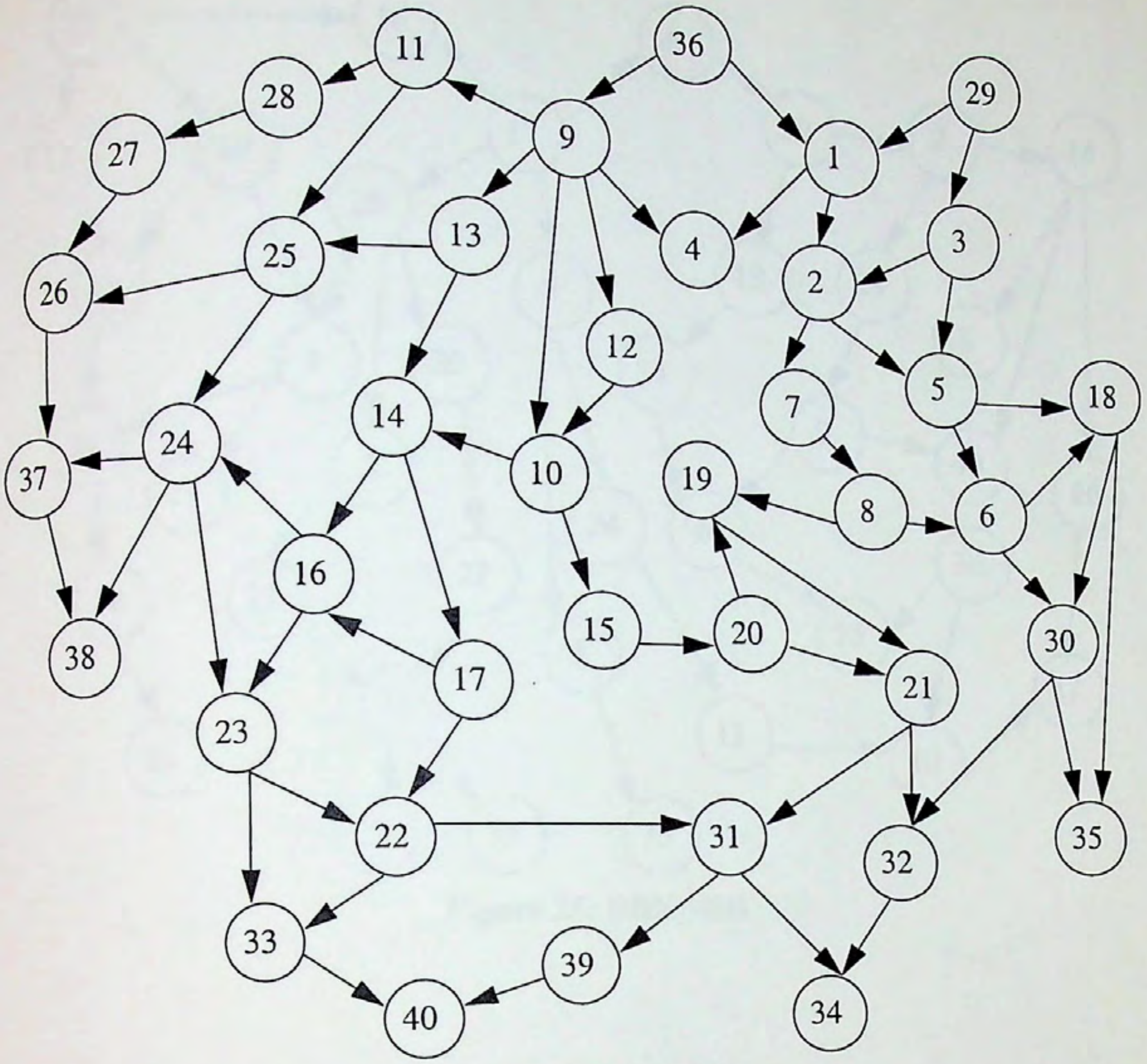

Figure 24: BBN_40A

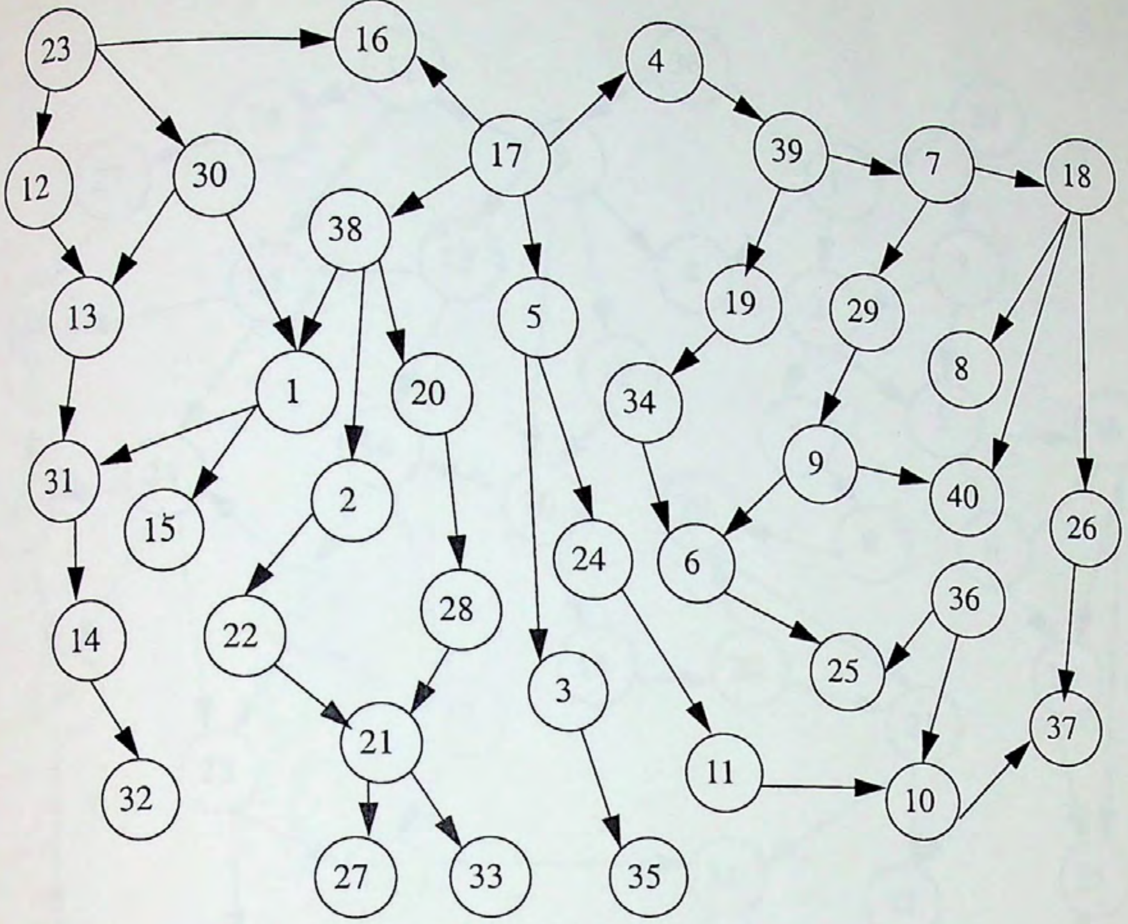

Figure 25: BBN_40B

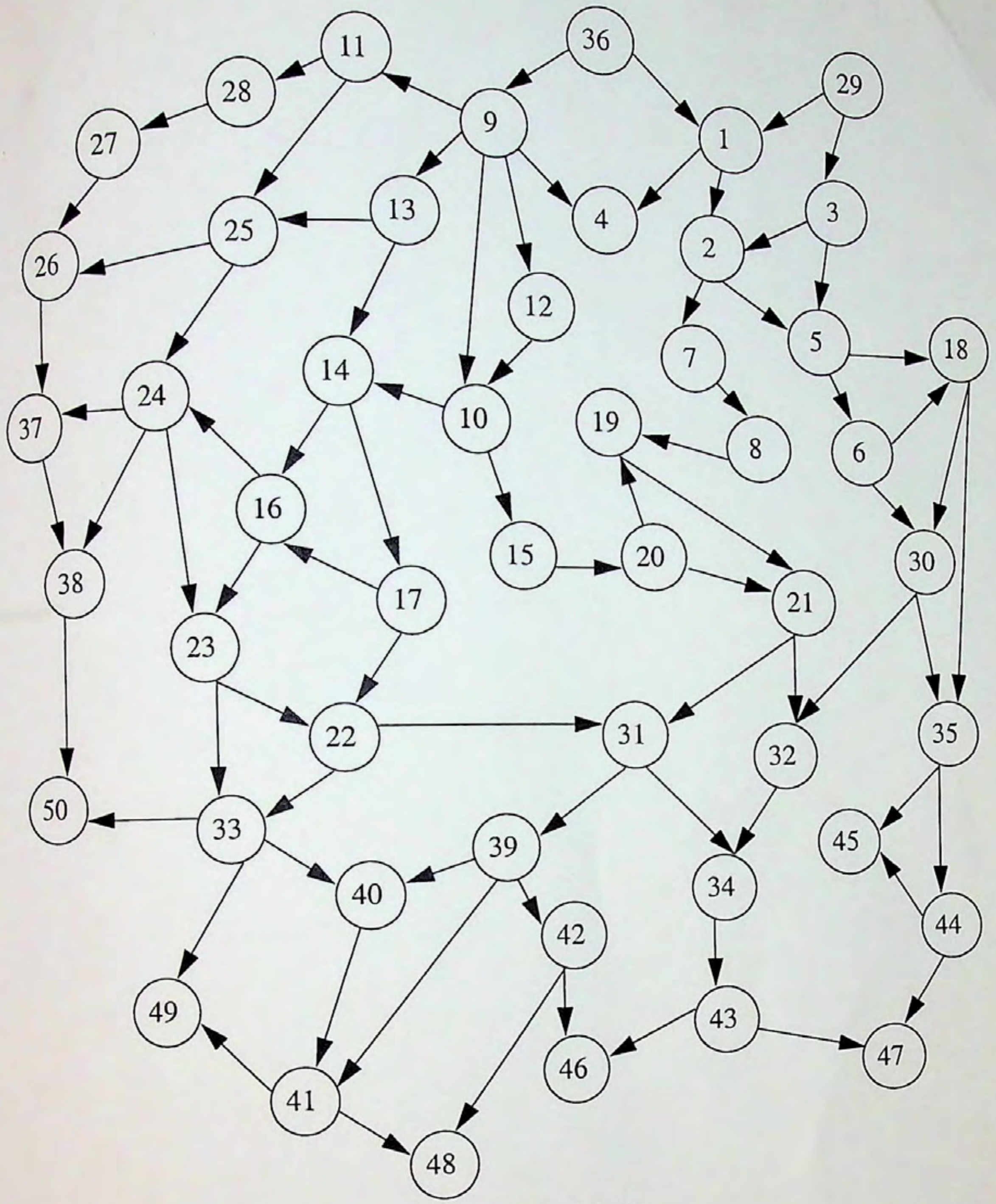

Figure 26: BBN_50