6-1-2005

Parallel versus iterated: comparing population oriented and chained sequential simulated annealing approaches to cost-based abduction

Heba Amer

The American University in Cairo

School of Science and Engineering

Parallel Versus Iterated: Comparing Population Oriented and Chained Sequential Simulated Annealing Approaches to Cost-Based Abduction

A Thesis Submitted to

The Department of Computer Science

in partial fulfillment of the requirement for

the degree of Masters of Science

by

Heba Abdallah Amer

B. Sc. In Computer Science, AUC, Feb, 1997

Under the supervision of **Dr. Ashraf Abdelbar**

July, 2004

DEDICATION

To my husband. Thank you for everything.

ACKNOWLEDGEMENTS

First of all, I would like to say "الحمد لله " for everything.

I would like to express my gratitude to my supervisor Dr. Ashraf Abdelbar for his continuous support and guidance throughout my thesis work. I would also like to thank my examiners Dr. Ahmed Rafea, Dr. Awad Khalil and the external examiner Dr. Ismail Amr Ismail for their time and effort to evaluate this thesis and their helpful comments.

In addition, I thank all my friends without whom I could have never passed through all this. Thanks for your continuous support and encouragement. Thank you, Hoda Khalil, for believing in me, even when I didn't believe in myself. Thank you, AUC Freshmen Advisors, for your encouragement and kindness every morning. Thank you ITWorx people for your emotional support and encouragement. Thank you my colleagues, computer science graduate students for your help and support and wish you the best of luck.

I also would like to thank ITWorx Egypt for allowing me to do the testing using their machines. I thank the engineers at the AUC labs for their help and support during the testing and results phase.

Lastly, I would like to thank my family. I would like to thank my mom for her continuous support and pushing. Thank you my husband for all the efforts you did and for supporting me in every way possible. I would like to thank my daughter for always trying to cheer me up. Finally, I would like to thank all my big family, brothers and sisters and relatives, for all the help and support they consistently give.

ABSTRACT

Stochastic search techniques are used to solve NP-hard combinatorial optimization problems. Simulated annealing, genetic algorithms and hybridization of both, all attempt to find the best solution with minimal cost and time. Guided Evolutionary Simulated Annealing is one technique of such hybridization. It is based on evolutionary programming where a number of simulated annealing chains are working in a generation to find the optimum solution for a problem. Abduction is the problem of finding the best explanation to a given set of observations. In AI, this has been modeled by a set of hypotheses that need to be assumed to prove the observation or goal. Cost-Based Abduction (CBA) associates a cost to each hypothesis. It is an example of an NP-hard problem, where the objective is to minimize the cost of the assumed hypotheses to prove the goal. Analyzing the search space of a problem is one way of understanding its nature and categorizing it into straightforward, misleading or difficult for genetic algorithms. Fitness-Distance Correlation and Fitness-Distance plots are helpful tools in such analysis. This thesis examines solving the CBA problem using Simulated Annealing and Guided Evolutionary Simulated Annealing and analyses the Fitness-Distance landscape of some Cost-Based abduction problem instances.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

AI	Artificial Intelligence
CBA	Cost-Based Abduction
EA	Evolutionary Algorithm
FDC	Fitness Distance Correlation
GA	Genetic Algorithm
GESA	Guided Evolutionary Simulated Annealing
ILP	Integer Linear Programming
NP-hard	Non-deterministic polynomial time hard
RLD	Run Length Distribution
RSA	Repeated Simulated Annealing
SA	Simulated Annealing
TSP	Traveling Salesman Problem
WAODAG	Weighted AND/OR Directed Acyclic Graph

Chapter 1 INTRODUCTION

Simulated Annealing (SA) and Evolutionary Algorithms (EA) are both nature-based stochastic computational techniques. SA is based on thermodynamics [LKH91] while EA is based on the idea of population genetics of nature [GZ01]. Evolutionary Programming (EP), Genetic Algorithms (GA), and Evolution Strategies (ES), all fall under the subtitle of evolutionary algorithms [BSW02]. Those methods, along with simulated annealing, are used for solving NP-hard (Non-deterministic polynomial time hard) combinatorial optimization problems. In such problems, a solution would be to maximize or minimize the value of a certain function that depends on many independent variables, usually called the cost, fitness or the objective function, which represents the quality of that solution [KGV83]. For NP-complete problems, no method with exact solution with the complexity bounded by the size of the problem has been found, but if such a solution were found, it could be mapped to solve all problems belonging to this category [KGV83].

SA and GA work reasonably well on different sets of problems, and require nearly no specific problem information other than fitness or cost information [MG95]. In SA, a single state representing the solution is maintained. In every iteration, a neighboring state is generated at random and its objective function is evaluated. If it has a better value for the objective function than the current state, it becomes the new current state. Otherwise, the new state replaces the current state with probability $e^{(-\Delta/T)}$ where Δ is the positive difference in the objective function and T is the temperature.

On the other hand, in Evolutionary Algorithms a problem's solution is viewed as a point in the finite space of solutions, where every solution has a fitness value that represents its quality. A population of states representing candidate solutions is maintained. Periodically, random members of the population are selected and some genetic operators are applied to them, such as mutation and recombination. Those generated members compete for a place with the parent members in the population according to their fitness or value of objective function [GZ01], [BBM93].

Several approaches have been taken to hybridize genetic algorithms and simulated annealing. One approach is to have parallel SA chains that work in a population framework. They may or may not share solutions and can influence the control parameters in one another.

Guided Evolutionary Simulated Annealing (GESA) is an example of such an approach. In GESA, a set of independent SA chains is maintained, with no recombination and no sharing of solutions. For each chain, the current state, called the parent state, generates a number of neighboring child states, where the most fit child states replaces the parent with a logistic probability just like in SA. The number of child states generated in each iteration for each chain depends on the value of the objective function of the child states generated in the previous iteration [YP95]. A variation on GESA introduces the heritage factor, which makes the number of child states a parent is allowed to generate dependent on the previous history iterations and not only the most recent one, thus giving a good family or chain a chance to recover from the effect of a single bad generation [Abd01].

Cost-Based Abduction (CBA) is an example of an NP-complete problem. In abduction, the problem is to find the best explanation for a given set of observations [San94]. In Artificial Intelligence (AI), abduction is modeled by trying to prove the observation by assuming some set of hypotheses. CBA associates a cost to each hypothesis. The best solution to the problem is the one with the minimum cost that proves the goal [San94].

Search space analysis of combinatorial optimization problems helps to understand the nature of such problems. Fitness landscape is considered an important criterion for the effectiveness of adaptive multi-start algorithms [SH00]. If we imagine the fitness landscape as a mountainous region with hills, craters and valleys, the effectiveness of a search strategy would depend on the ruggedness of the landscape, the distribution of the valleys and craters and their overall numbers [SH00].

Fitness Distance Correlation (FDC) shows the correlation between solution fitness and the distance to optimal solution. It gives an analysis about the distribution of local minima of a search space and their relative location with respect to global optima and is considered as a measure of search difficulty [JF95].

1.1 Thesis Objective and Approach

Combinatorial optimization problems are complex and hard to solve, hence many approaches have been proposed to reach a sub-optimal solution for them [YP95]. The objective of this thesis is to determine the effectiveness of using Guided Evolutionary Simulated Annealing in solving Cost-Based Abduction problems as an example of NP-hard problems. This is done by applying a heuristic repair function to solve the problem of penalty cost of unfeasible states. The GESA will be compared against pure iterated Simulated Annealing for solving the same problems. Introducing the heritage factor will be examined on a small scale to determine its effect on the generated number of children each generation and the quality of the reached solution. In addition, landscape analysis of the CBA problem is performed to form a general view about the difficulty of CBA problems.

1.2 Thesis Layout

In the next chapter, a description of current methods in literature is first given about SA and GA with their historical origins, along with a discussion about some hybridization techniques, followed by a formal definition of CBA and other attempts to solve it, and the definition of fitness landscape analysis and its application to some other well-known NP-hard problems. The proposed method and description of algorithms are given in chapter 3, with the problem representation and design issues involved in solving the problem. Chapter 4 discusses the preliminary results followed by conclusion and future enhancements in chapter 5.

Chapter 2 OVERVIEW OF CURRENT METHODOLOGIES

2.1 Deterministic and Stochastic Search Techniques

There are two types of search techniques used to find the optimum value of the criterion function of a certain problem, which are deterministic and stochastic. In deterministic search techniques, an exhaustive enumeration is performed to guarantee the optimal solution. On the other hand, stochastic search techniques generate a near-optimal partition reasonably quickly, and guarantee convergence to optimal partition asymptotically. Evolutionary search algorithms and simulated annealing fall under the latter technique [JMF99].

While deterministic techniques are typically greedy descent approaches, stochastic approaches permit perturbation to the solution in non-locally optimal directions also with nonzero probabilities. Stochastic search techniques are either sequential or parallel. Evolutionary approaches are inherently parallel, whereas the simulated annealing is a sequential stochastic search technique [JMF99].

Evolutionary Programming, Genetic Algorithms and Evolution Strategies work in a similar general frame. It can be simply summarized "by a loop over partially randomized variation and selection operators steering exploration and exploitation (or chance and necessity) and, in contrast to traditional optimization procedures, acting upon a set of search points in the decision variable space" [BSW02]. Following is an overview of both SA and GA.

2.1.1 Simulated Annealing (SA)

Simulated Annealing from the computational view is modeled as the physical process of annealing. In this process, physical substances such as metals are melted; i.e. raised to high energy levels, and then gradually cooled until some solid state is reached, which is a minimal-energy final state [RK91]. The physical substances usually move from higher energy configurations to lower ones. However, there is a probability that a transition to a higher energy state would happen. This probability is given by the functions

$$p = e^{(-\Delta E/kT)} \tag{2.1}$$

where ΔE is the positive change in the energy level, T is the temperature, and k is Boltzmann's constant [RK91]. Such transition is more likely to happen during the beginning of the process when the temperature is high, and becomes less likely as the temperature gets lower. From the SA computational point of view, the annealing process is the search space, where we start with a high cost state and search for a lower cost next state, but sometimes we accept higher cost states with probability analogous to the one given above so that the search doesn't get trapped in a local minimum.

The annealing schedule, which is a sensitive factor for the annealing process, is the rate at which the system is cooled. If the cooling schedule is too rapid, stable regions of high energy will form; i.e. the search reaches a local minimum not a global one. On the other hand, if the annealing schedule is slower for the physical substance, a uniform crystalline structure is more likely to develop, which corresponds to minimum energy or global minimum. The optimal annealing schedule for each particular annealing problem must usually be discovered empirically [RK91].

For Simulated Annealing, ΔE is generalized to represent the change in the value of the objective function that the SA is trying to minimize. The analogy for kT in equation (2.1) is only T where k is incorporated into it. The values for T should be selected to produce desirable behavior for the algorithm. Thus the analogous probability formula is

$$p' = e^{(-\Delta E/T)} \tag{2.2}$$

where ΔE is the positive change in the value of the objective function and T is the temperature [RK91].

Kirkpatrick suggested using SA for solving combinatorial optimization problems in 1983. Although the algorithm converges slowly, his paper attracted many researchers for studying SA [KGV83], [YP95]. The asymptotic convergence of an SA process to the global optimal solution can be guaranteed if the temperature is decreased at a rate that is no faster than logarithmic. However, this is considered too slow in practice and the most commonly used schedule is to reduce the temperature periodically by a fixed fraction [AA03].

2.1.2 Genetic Algorithms (GA)

Genetic Algorithms is based on the concept of “survival of the fittest”. It is analogous to the natural behavior. The basic principles of GA were first laid down by Holland in 1975. He also introduced the schema theorem that explains the power of GA. The system has a population of individuals representing candidate solutions to the problem in hand. Each member in the population has a fitness that represents how good it is. The members with high fitness are given more chance to “reproduce” by “cross breeding” or reproduction and mutation. The new members produced, known as “offspring” compete to get to the new generation according to their fitness.

Members with good characteristics are more likely to be in the new generation, where members with low fitness “die out” moving from one generation to another. This way of favoring the more fit individuals allows for exploring the best areas of the search space, and with a good design for the problem, the algorithm converges to the optimal solution [BBM93].

There are some basic principles that should be defined before the system can run; the problem should be coded in a representative way, the fitness function should be well defined, and a method of selecting members for reproduction and recombination should also be defined during the run of the system [BBM93].

Crossover and mutation are the main functions for reproduction. Random members of the population are selected in a way favoring the more fit ones, and then crossover takes place to produce the offspring. Mutation is then applied to each individual. Crossover and mutations have some variations according to the type of problem. Usually they are not done for the whole population, but to a percentage of the selected members [BBM93].

Convergence of GA means that the population shares the same values for their genes. When the system is correctly implemented, the global optimum is reached as the fitness of the best and average individuals increase in each generation [BBM93]. The schema theorem explained by Holland shows that giving the more fit members more opportunity to reproduce will eventually lead to finding better solutions [BBM93].

2.2 Hybridization of GA and SA

GA and SA are both methodologies for solving combinatorial optimization problems. Unlike GA, in SA only one candidate solution is kept, so there is no overall picture of the search space. Previous moves are not saved to guide for the selection of new ones [BBM93]. On the other hand, Esbensen and Mazumder [EM94] report that one of the problems of GA is the way it converges. The rate of getting a better fitness is high at the beginning but then drops and it becomes hard to get further improvement, which wastes most of the runtime at the later phase of the process.

Neither of the two algorithms is preferred for all types of problems, hence the hybridization was introduced. Esbensen and Mazumder show that a mixture of both algorithms leads to higher layout quality than pure GA [EM94].

Several attempts have been proposed to hybridize both techniques. One approach is to use multiple instances of SA chains working in parallel in a generation-like GA environment, where Guided Evolutionary Simulated Annealing falls in. Following is a review of some methodologies of this hybridization.

2.2.1 Parallel Recombinative Simulated Annealing

In 1995, Mahfoud and Goldberg introduced Parallel Recombinative Simulated Annealing (PRSA) as a method for hybridizing SA and GA. Their algorithm keeps the desirable asymptotic convergence properties of SA and at the same time adds the power of GA of using the population approach and recombination [MG95]. It is based on Goldberg's Boltzmann tournament selection [Gol90]. "PRSA closely follows simulated annealing, if one imagines several copies of SA running in parallel, with mutation as the neighboring operator, and crossover recombining independent solutions" [MG95].

PRSA can be viewed from both the GA and SA environment. Taking it from the GA framework, in every generation, two members are randomly selected and used to generate two children by recombination and mutation. Then the children and the parents hold a Boltzmann trial where the winners replace the parents. In the

Boltzmann trial, a child *i* replaces a parent *j* with logistic probability $1/(1 + e^{(Ei-Ej)/T})$ where E_i and E_j are the fitness of the child and parent respectively and T is the temperature. In a variation, the fitness of both children is compared against the fitness of both parents. The temperature is lowered periodically [MG95].

From the SA point of view, PRSA can be viewed as simultaneous multiple independent SA chains, with a global annealing schedule. Boltzmann distribution will be approached for each independent application of SA. Crossover and mutation can be viewed as an extension to the neighboring operator of SA [MG95].

Although most hybridization techniques of GA and SA lack formal proofs of their convergence, Mahfoud and Goldberg provided a formal proof of convergence under certain easily observed conditions [MG95].

2.2.2 Annealing-Genetic Algorithm

Lin et al. [LKH93] reported some observations on using SA from the performance analysis of their empirical results:

- The system has to tradeoff between the execution time and quality of final solution obtained.
- The cooling schedule might get the system trapped in a local minimum if the temperature drops too sharply.
- Detecting the equilibrium of the system at each temperature is not an easy task.
- The total number of iterations depends on the initial temperature.
- And, a good solution might be discarded if the number of iterations at low temperature is not large enough.

They proposed this method of Annealing-Genetic Algorithm to reach an efficient annealing schedule. They aimed at reaching a near optimal solution with error not more than 3% and to reach an execution time that is bounded by a polynomial function of the problem size.

Their algorithm starts with a GA environment. For each generation, the standard genetic operators are applied to create the new generation. Then, some members are selected to start an SA search that does not go through a full annealing schedule; instead it uses a fixed temperature and stops when reaching a no-change state. The temperature is decreased at the end of the generation and used by all SA chains in the next generation. Their approach can be viewed as SA with population-based state transition and with genetic-operator-based quasi-equilibrium control. It can also be viewed as a GA with Boltzmann selection operator [LKH93].

In another technique of hybridizing SA and GA, several chains of SA searches are kept running in parallel. Each chain keeps its own current state and starts from a separate starting state. They do not share best solutions, instead they influence each other's controlling parameters. New Population Oriented Simulated Annealing and Guided Evolutionary Simulated Annealing are two examples of this approach.

2.2.3 Population Oriented Simulated Annealing

In 1998, Cho et al. proposed a "New" Population-Oriented Simulated Annealing (NPOSA) technique based on local temperature concept. Their technique runs multiple SA chains in parallel, each with its local temperature. These chains do not share the best states, but they influence the control parameters of each other. "Each individual in the population can intelligently plan its own annealing schedule in an adaptive fashion to the given problem at hand". This speeds the search and leads to a near optimal global solution [COC98].

In this approach, each chain has its own temperature, which is adjusted according to its rank among the population. The rank is adjusted according to the cost of the solution of each chain. The algorithm starts with a number of SA chains each starts with a random starting state. In each iteration, each chain finds its rank according to the value of its objective function, where the best chain gets the highest rank. The temperature of each chain is then set to α^{ri} T_{ref} where T_{ref} called reference temperature is a constant defined by the user at the beginning of the system and does not decrease, and r_i is the rank of chain i and α is a user define constant called the distribution factor in the range $0 < \alpha < 1$. Then each chain generates its offspring and accepts it with the SA acceptance functions using its own temperature [COC98].

This approach of setting the local temperature allows a low ranked chain to get more chance to go uphill on the search surface by getting a higher temperature. On the other hand, a high ranked chain is given less chance to move uphill to enhance the solution accuracy [COC98].

NPOSA was tested on the TSP problem against the standard SA and proved to be more efficient. While in SA, no further progress is made towards the end of the system because it gets frozen as the temperature gets very low, NPOSA can still go on with the search because of the concept of local temperature adjustment [COC98].

2.2.4 Guided Evolutionary Simulated Annealing (GESA)

Yip proposed the algorithm GESA in 1995, which incorporates SA into the selection process of simulated evolution, adding a new level of competition to make the search regionally guided [YP95]. GESA starts with a number of SA chains, N, each having its own current state, called parent. All these chains have a single global temperature. In the first iteration, each parent is allowed to generate a number of neighboring or child states, say *M*.

There are two levels of competition among states in GESA. The first one is local to each chain. The child state with the best objective function is determined and compared with the parent state. If it has a better fitness, it becomes the current state for the next iteration, otherwise, it replaces the parent with the logistic probability used for SA. The second level of competition among states determines the number of child states generated in the next iteration for each chain [YP95]. Let $m_i^{(t)}$ be the number of child states chain *i* is allowed to sample at iteration *t*. In the first iteration, $m_i^{(1)}$ is set to *M* for all chains. For the next iteration, the number of child states for each chain *i* is made proportional to its acceptance number A_i, which is determined by comparing the fitness of each child with the parent state of chain *i*. For each child, A_i is incremented if the fitness of the child is better than the parent. Otherwise, A_i is incremented with the probability $e^{(-\Delta/T)}$ where Δ is the positive difference in fitness between the current child and the best fitness found ever. The number of child states of the next iteration $t+1$ is set equal to

$$m_i^{(t+1)} = M\,N\,A_i\,/\,S \tag{2.3}$$

where

$$S = \Sigma_j\,A_j \tag{2.4}$$

thus the total number of child states generated for all chains in each iteration is kept constant and equal to *MN* [Abd01].

Yip and Pao show that eventually one chain survives which is usually the best family. The acceptance number, which shows the second level of competition among chains, gives a measure of the regional in formation. A higher acceptance number indicates that better candidate solutions were found, which gives the search more attention to that region to generate more children [YP95].

The GESA algorithm can also be explained as parallel simulated annealing with competition. Each chain can be viewed as a multiple-trial-parallel simulated annealing machine with the children contributing the trials in parallel. The *N* chains can be viewed as *N* parallel simulated annealing machines competing with each other, where the better machine is getting more trials [YP95].

In their paper, Yip and Pao tested GESA on the Traveling Salesman Problem (TSP) which is known to be NP-hard [YP95]. They compared their results against a version of simulated evolution. Their results outperformed that version of simulated evolution. They also showed that the GESA approach is less complex than simulated evolution when the total number of individuals in each generation is very large.

GESA was applied in a study to investigate the effect of varying optimization parameters on the proposed optimum of a tablet coating formulation requiring minimization of crack velocity and maximization of film opacity. It was used to optimize an artificial neural network to identify the formulation that satisfied and exceeded the looser targets, when the stringency of the performance criteria were reduced to a crack velocity of > 0 ms^{-1} and film opacity of $< 100\%$. Under these conditions, starting GESA from different locations within model space resulted in the proposal of different optima [PR+03]. GESA was also used by Dean et al. [DZ+01] to optimize Gamma Knife radiosurgery treatment planning. It was used to maximize the therapeutic benefit through a probability model that dissects a patient volume image into three components: normal, critical normal, and tumor tissue. They compared GESA algorithm to other manual methods using two clinical examples where GESA optimization showed therapeutic advantage over the treatment team's manual effort.

GESA was in a comparison with GA and SA to solve the problem of file and task placements in distributed systems in 2002 [CC02]. The authors found that pure SA had better performance solving this problem and related that to the fact of a non-continues search space of the file and task placement problem. They suggested that having jumps within the search space would be more effective than using a generation-like search method.

2.2.5 Extending GESA with Heritage Factor

A variation on GESA is proposed in [Abd01] introducing the heritage factor. GESA has the deficiency of dramatically decreasing the number of children of one parent if this parent has a poor fitness, although the number of children depends on the relative fitness of children to their parent's and not on their absolute fitness. This was the motivation for introducing the heritage factor. It allows the recovery of one bad generation in a good family.

The heritage factor makes the acceptance number of a chain not only dependent on the last iteration's results, but also the previous iterations, i.e. the genetic line of the chain or family. In GESA, if the current parent of one chain has very poor children in one generation, the number of children allocated to this chain is tremendously affected, in spite that it is determined on the relative fitness of the children relative to the parent. The heritage factor considers the past history of the family and not just the last generation. The acceptance number is calculated in the same way like in GESA, but the number of child states $m_i^{(t+1)}$ generated in the next generation $t+1$ for chain i is set to

$$m_i^{(t+1)} = MN\,[\,A_i + \alpha\, m_i^{(t)}\,]\,/\,S \tag{2.5}$$

and

$$S = \Sigma_j\,[\,A_j + \alpha\, m_j^{(t)}\,] \tag{2.6}$$

where α is called the heritage factor and $0 \leq \alpha < 1$. The heritage factor is manually determined and the model reduces to pure GESA when $\alpha = 0$ [Abd01].

2.3 Cost-Based Abduction (CBA)

This section gives an overview of the term Abduction. It shows the meaning of abduction in inference logic, along with other inference types, and discusses the problem of abduction from the AI point of view. Finally it shows some other attempts to solve CBA problems in literature.

2.3.1 Abduction, Deduction and Induction

Abduction, as a type of inference, is the process that generates explanations [CM86]. Pen and Reggia [PR90] give an informal definition to abduction; "In informal terms, *abduction* or *abductive inference* is generally taken to mean *inferring the best or most plausible explanations for a given set of facts*". They define three fundamental logics of scientific inquiry as deduction, induction and abduction.

In deductive reasoning, given a general rule, and a specific case, a specific result can be deduced [PR90]:

Given Rule	– All the balls in the box are black
+ Case	– These balls are from the box
Conclude Result	– These balls are black

In inductive reasoning, given a specific case and a specific result, a general rule can be hypothesized [PR90]:

Given Case	– These balls are from the box
+ Result	– These balls are black
Hypothesize Rule	– All balls in the box are black

In contrast to deduction and induction, in abductive reasoning, given a general rule and a specific result, a specific case can be hypothesized [PR90]:

Given Rule	– All balls in the box are black
+ Result	– These balls are black
Hypothesize Case	– These balls are from the box

Looking at induction and abduction, both involve making and testing hypotheses. However, in induction, the general rule is what is being hypothesized, while in abduction it is the specific case. Moreover, in induction a large number of situations are used to hypothesize the rule supporting its plausibility, while in abduction a single situation can be used to conduct the hypothesized case [PR90].

As for deduction and abduction, both use a specific case to produce the result. However, in deduction, the result is a logical outcome of the general rule and the true case in hand, while the inferred specific case of abduction is only a possibility even though the general rule and the specific result are true [PR90].

Both abduction and deduction require relevant facts to infer a new fact, but abduction can get more than one answer. So it requires an extra step to decide which answer is best. Deduction is viewed as "legal inference" as it draws true inferences given true axioms. On the other hand, abduction is not a legal inference as it can give false conclusions. In spite of this fact, abduction is still an important and necessary way of inference. It is very useful in medical diagnosis, story understanding, vision, and natural language understanding [CM86].

2.3.2 Abduction

The nomological theory of explanation has it that "an explanation is a proof of what is to be explained from knowledge of the world plus a set of assumptions" [CS94]. Finding an explanation is characterized by the following [CS94]:

- Knowledge of the world is usually given in the form of rules and observed facts.
- Certain assumptions have to be made so that the evidence could be predicted or proved.
- The selection of the explanation should be optimal in some sense.

This theory has a problem from the AI point of view, which is having many possible sets of assumptions that can prove the desired fact. In Cost-Based Abduction (CBA), a minimal cost proof is to be found that best explains the facts to be proven [CS94].

2.3.3 Weighted Abduction

Weighted abduction was introduced by Hobbs et al. for The Abductive Commonsense Inference Text Understanding System (TACITUS) project at SRI [HS+88]. It was introduced as an explanation scheme to evaluate potential explanations using cumulative cost of assumptions. It simplified the conceptualization of the problem of interpreting text. The project was intended for processing messages and other texts for many purposes, including message routing and prioritizing, problem monitoring and database entry. It aimed at investigating how knowledge is used in the interpretation of discourse.

Text interpretation means to prove the logical form of the text from what is already mutually known, while using coercions, merging redundancies where possible, and making assumptions when necessary. In abduction, many possible explanations could be used to prove the evidence, hence some criteria is used to choose among the possibilities. Consistency of the evidence with the knowledge is one criterion. Simplicity and consilience are another criterion, where the evidence should be as small as possible and the goal as big as possible [Hef01].

Weighted abduction solved the problem of minimal set abduction. It also avoided the over-specification problem where irrelevant hypotheses are assigned values. On the other hand, it had a major deficiency, which is its lack of semantics [CS94].

2.3.4 Cost-Based Abduction Definition (CBA)

Cost-Based Abduction (CBA) was formally introduced by Charniack and Shimony in 1990 [CS94]. In their paper, they provided a definition of a CBA system as follows: The system has the rules of the form

$$R : (p_1 \wedge p_2 \wedge \ldots \wedge p_n)\, {}^{C}{}_{R} \rightarrow q$$

where q and all the p_i are ground atomic formulas or hypotheses, and C_R is the cost for applying the rule. Each conjunct has a cost $c(p_i)$, which is the same in all rules where the conjunct appears on the left hand side. Conjuncts that appear on the left hand side are called *antecedents* while the one on the right hand side is called the *consequent*. The cost of proving q using this rule is the cost of all the conjuncts assumed plus the cost of the rule. Without loss of generality, they assumed that all rule costs are 0 and added a p_0, that appears nowhere else to the left hand side, with a cost $c(p_0) = C_R$. Now the objective is to find a minimal cost proof for some fact set ε, which is the evidence.

Charniack and Shimony modeled the CBA problem as a minimization problem on a Weighted AND/OR Directed Acyclic Graph (WAODAG). Then a Boolean belief network was created based on the WAODAG and the semantics for complete models were constructed. They presented the problem of finding the Maximum A posteriori Probability (MAP) assignments to belief networks, and applied a transformation to allow their algorithm to work on the resulting graph [CS94].

2.3.5 CBA NP-Completeness

Problems that are defined as NP-complete must satisfy certain conditions. They have to belong to the complexity class of languages NP, which means they can be verified by a polynomial-time algorithm. Moreover, they have to be NP-hard, which means they can be polynomial-time reduced to other problems that are known to belong to the NP class [CLR90].

Charniak and Shimony [CS94] proved that finding the least cost proof is NP-hard, and that the given cost selection problem is NP-complete. They provided their proof in a form that has some restrictions. They used the Vertex Cover Problem (VC), which is a known NP-complete problem, to reduce their proof from. Recently, approximating least-cost proofs has also been shown to be NP-hard [Abd04].

2.3.6 Linear Constraint Satisfaction Approach for CBA

According to Santos Jr. [San94], having many possible explanations available in abductive reasoning is the basic problem. Several attempts were made to solve this problem based on some preferential ordering of the hypotheses. Minimizing the necessary number of hypothesis proved to be inadequate. Weighted abduction added costs to hypothesis and avoided the over-specification problem, but it had a major deficiency, which is its lack of semantics [CS94].

Santos Jr. presented an approach of modeling abductive reasoning by using linear constraints to represent causal relationships [San94]. The nodes on the WAODAG were considered variables and constraints were applied between nodes represented by linear inequalities. The problem was modeled with 1 and 0 instead of true and false, and linear programming techniques were applied to minimize the cost associated with the graph. This approach needed to be augmented since sometimes straight linear programming would not reach a proper solution, and thus the information provided by the solution could be used in an incremental branch and bound search that would guarantee minimal cost solution. This augmentation was rarely needed as shown by the experimental results, and the technique showed expected-case polynomial growth rate on typical problems.

2.3.7 Polynomial Solvability of CBA

Santos showed that many CBA problems could be solved efficiently with better performance using linear program relaxation of integer program formulation, where linear programming is known to be solvable in polynomial time [SS96].

To enhance the results of the linear constraint satisfaction approach of CBA, Santos and Santos studied those results and determined conditions for solving CBA problems in polynomial time [SS96]. In their approach, they used the concept of total unimodularity from network flow analysis and tied it to their CBA problem. They concluded that parity-balance guarantees polynomial solvability by using their transformation to integer linear programming.

Santos and Santos also provided a new heuristic for the problems that needed branching and bounding. It provided a tighter upper bound on the worst-case performance, which potentially reduced the total number of branches needed to be explored [SS96].

2.3.8 Parallel CBA Reasoning for Distributed Memory Systems

Kato et al. introduced a search control technique of A* into abductive reasoning mechanism in 1994. In their paper [KSI96], they proposed a parallel version for distributed memory systems. Parallel best-first search was used as a search control technique into abductive reasoning mechanism which resulted in more efficient results. An informal analysis of their PARallel Cost-based Abductive Reasoning system (PARCAR) was given, and the system was implemented on an MIMD distributed memory parallel computer. The main aim of the paper was to define the way to divide the search space on parallel processors and to construct a search tree on each one.

2.3.9 Networked Bubble Propagation for CBA

Ohsawa and Ishizuka proposed a near-optimal solution method for solving the CBA problem in polynomial time [OI97]. In a previous paper for Ishizuka, a CBA problem would be translated into an equivalent 0-1 integer programming and then the method of Pivot and Complement (PC) is applied to obtain a near-minimal cost quickly. The PC method takes time $O(N^4)$, where N is the number of variables or hypotheses. They suggested using a knowledge network that represents the knowledge structure with a mechanism similar to approximate 0-1 integer programming, which could achieve inference even faster than the original PC, in time $O(N^2)$ or less. They represented propositions as nodes in the network, and defined truth values of propositions as real numbers in the range of [0, 1], and introduced bubble propagation to propagate truth values along the arcs of the network. Their method solved the CBA problem in polynomial time, but their solution is near-optimal [OI97].

2.3.10 Slide-down and Lift-up (SL) Method

Slide-down and Lift-up (SL) method was introduced by Ishizuka and Matsuo in 1998 [IM98]. In SL, linear programming techniques are used to determine an initial search point for the problem and non-linear programming is used to find a near-optimal 0-1 solution. They also developed a local handler to escape the search from local optimal solutions. This method solves the CBA problem in polynomial time with respect to the problem size.

Like the networked bubble propagation method, the SL method also gets near-optimal solutions in polynomial time [IM98]. Although its performance is lower than the networked bubble propagation, the authors find it simpler and more understandable.

2.3.11 Binary Decision Diagrams for CBA

Kato et al. suggested using Binary Decision Diagrams (BDD) for solving CBA [KO+99]. BBD represent and manipulate Boolean functions efficiently and easily and can be applied to many problems. They proposed a specialized BDD and its operation suitable for abductive reasoning: PBDD (Partial BDD) and GPC (Graft and Pruning Construction). PBDD was used to keep the reasoning goal-directed and GPC was introduced to avoid generating irrelevant parts of the BDD to the most preferable solution.

2.3.12 Linear-Programming Based Admissible Heuristic for CBA

In Hefny's paper [Hef01], he introduced a new admissible heuristic for CBA. The A* algorithm was used with a heuristic estimating the cost bounded by the actual cost, which is the simplex method. The CBA instance was converted to a linear-programming instance that was used by the A* framework. The heuristic admissibility was proved theoretically and its efficiency was proved experimentally. He also defined a measure for the difficulty of CBA problems, where the larger number of hypotheses, the larger maximum number of antecedents, and the smaller percentage if AND nodes would lead to a more difficult CBA instance.

Hefny’s work in [Hef01] was compared to Santos work in [San94]. The results in Hefny’s work shows that his method is admissible and outperforms the other one. He also concluded that if the percentage of assumable hypotheses is less than or equal to 25%, his method would guarantee better results.

2.4 Fitness Distance Correlation (FDC)

Fitness Distance Correlation (FDC) was introduced by Jones and Forrest [JF95] in 1995 as a measure of search difficulty for GA. It was used to predict the performance of GA on problems with known global maxima. FDC was the result of investigating the connection between GA and heuristic search.

2.4.1 FDC Definition

Stützle and Hoos give an intuitive definition for *fitness landscape* [SH00]. They described it as a mountainous region with hills, craters, and valleys, where the search algorithm is viewed as a wanderer performing a biased walk in this landscape. The goal is to find the lowest point in this landscape for minimization problem.

Formally, they define fitness landscape as:

1. The set of all possible solutions S
2. An objective function that assigns a fitness value $f(s)$ to every $s \in S$
3. A neighborhood structure $N \subseteq S \times S$

The fitness landscape draws the shape of the search space as met by a local search algorithm. The neighborhood structure brings a distance metric on the set of solutions; the distance $d(s, s')$ between two solutions s and s' can be defined as the minimum number of moves that have to be performed to transform s to s' [SH00].

Correlation between solution fitness and distance to global optima is called in the genetic algorithms literature as fitness-distance correlation (FDC). It can be captured by the correlation coefficient, defined by:

$$\rho(F, D) = \mathbf{Cov}\,(F,D) \,/\, ((\mathbf{Var}(\mathrm{F}))^{½} * (\mathbf{Var}(\mathrm{D}))^{½}) \qquad (2.7)$$

where **Cov**(F,D) is the covariance between the random variables F and D which probabilistically describe the fitness and the distance of local optima to a global optimum, while **Var** denotes the variance.

For minimization problem, a high positive FDC indicates that the smaller the solution cost, the closer the solutions, on average, to a global optimum. Thus, algorithms combining adaptive solution generation and local search may be expected to perform well [SH00].

2.4.2 FDC as Measure of Search Difficulty

Jones and Forrest introduced FDC as a measure of search difficulty in 1995 [JF95]. Their work was to examine one aspect of the correspondence between evolutionary algorithms and heuristic state space search, which is the relationship between the fitness function of GA and heuristic functions. Deception and Ruggedness of fitness landscape are other methods of measuring the difficulty of GA. Jones and Forrest showed conflicting opinions of researchers about those methods. Some opinions believed that deception is the only thing that makes a problem difficult for GA, while others saw that it is neither necessary nor sufficient to make a problem hard for GA. Ruggedness also was not sufficient or necessary to make a problem difficult for GA [JF95]. The authors saw that the existing methods of measuring difficulty of GA algorithms are not definitive and did not explain some surprising results of GA. They suggested that the relationship between fitness and distance to the goal is very important for GA search. They used scatter plots of fitness versus distance and the correlation between fitness and distance to indicate problem difficulty.

Although Jones and Forrest pinpointed that using the actual operators of the GA would provide better predictions, they used Hamming distance as a simple first approximation to distance under the actual operators of GA. They also emphasized on the fact that FDC is only one of the possible ways of examining the relation between fitness and distance, and showed the importance of examining a scatter plot of fitness versus distance for understanding that relationship when it cannot be detected by the correlation [JF95].

The results of their work predicted GA behavior of some well-studied problems, and also predicted the results that were seen as surprising by other methods, and detected differences in coding and representation of problems. In their work, they used maximization problems, and classified problems in three groups according to the FDC results [JF95];

1. Misleading problems with correlation coefficient $\rho \geq 0.15$, where fitness tends to increase with distance from the global optimum

2. Straightforward problems with $\rho \leq -0.15$, where fitness tends to increase as distance decrease to global optimum.

3. Difficult problems with $-0.15 < \rho < 0.15$, where there is very little correlation between fitness and distance to global optimum. If the fitness-distance scatter plot shows no relation between fitness and distance, then the problem is GA

difficult, but if a structure appears in the scatter plot, it can indicate if the problem is misleading or straightforward as the case may be.

2.4.3 FDC Counterexample

In 1997, Altenberg criticized the work of Jones and Forrest for using the Hamming distance as the distance measure for calculating FDC and classifying problem difficulty accordingly [Alt97]. He constructed a counterexample to the third condition of Jones and Forrest classification of problem difficulty according to the FDC coefficient, where he produced a GA-easy fitness function that showed no relation between Hamming distance and fitness, and had a FDC coefficient of zero, and thus the problem would be classified as difficult, according to Jones and Forrest.

His results proved that FDC analysis using Hamming distance wrongly predicted the difficulty of his constructed test function, where other methods were better predictors, among which was FDC analysis with distance derived from the genetic operators. He illustrated this point using crossover-based distance measure for FDC analysis and mutation-based distance measure for FDC analysis. He proved that it was easily optimized by GA using single-point crossover and roulette wheel selection and that the efficiency of the GA increased with the size of the search space [Alt97].

2.4.4 FDC Analysis of Some NP-hard Problems

The FDC analysis was used by many researchers to analyze some famous NP-hard problems, like the Traveling Salesman Problem, Quadratic Assignment Problem, Set Covering Problem and Linear Ordering Problem.

The Traveling Salesman Problem (TSP) and Quadratic Assignment Problem (QAP) were studied by Stützle and Hoos [SH00] in 2000 as application domains for their MAX-MIN Ant System (MMAS) algorithm. The MMAS algorithm is an Ant Colony Optimization algorithm derived from Ant System. They showed that their system is among the best performing algorithms for these problems.

The TSP can be represented by a complete graph $G = (N, A)$ with N being the set of nodes, also called cities, and A being the set of arcs fully connecting the nodes. Each arc $(i, j) \in A$ is assigned a value c_{ij} which represents the distance cost between cities i and j. The problem is to find the least cost closed tour visiting each of the $n = |N|$ nodes of G exactly once [SH00]. In symmetric TSP, which was used by the authors, the distance cost between cities are independent of the direction of traversing the arcs; that is, $c_{ij} = c_{ji}$ for every pair of nodes.

The authors gave the distance measure for FDC between two tours s and s' by the number of different arcs, that is $d(s, s') = n - |\{(i, j) : (i, j) \in s \wedge (i, j) \in s'\}|$ (where n is the number of cities). They used a 3-opt local search algorithm with a number of standard speed-up techniques. Their results showed a strong positive correlation between solution cost and the distance from the closest optimal solution. They also indicated that locally optimal tours are concentrated around a small region of the whole search space [SH00].

The Quadratic Assignment Problem (QAP) is the problem of assigning a set of facilities to a set of locations with given distances between the locations and given flows between the facilities [SH00]. The goal is to place the facilities on locations in such a way that the sum of the products between flows and distances is minimal. It is considered one of the hardest optimization problems. The authors used benchmark

QAP instances to analyze the FDC coefficient. They classified those instances into four classes;

i. Unstructured randomly generated instances, which are considered the hardest to solve

ii. Grid-based distance matrix, which have multiple global optima

iii. Real-life instances, which have the flow matrices having many zero entries and the remaining entries not uniformly distributed

iv. Real-life-like instances, which are generated with matrix entries resembling the distributions found for real-life problems

They used a 2-opt algorithm which examines all possible exchanges of pairs of facilities. The distance between solutions was measured as the number of items placed on different locations.

The FDC analysis of QAP showed clear differences between the different problem classes. For class (i), the FDC coefficient was almost zero for all instances, indicating that the solution quality gave little guidance to the search algorithm. For the other classes, much higher correlation was found. They concluded that on average, better solutions are closer to an optimal solution in real-life QAP instances and also in those of classes (ii) and (iv), which indicates the potential usefulness of an Ant Colony approach to the QAP [SH00]. They also compared the FDC analysis for TSP and QAP, concluding that the local minima in the QAP appear to be spread over large parts of the QAP search space, while for the TSP they are concentrated on a relatively small subspace. The solution quality of QAP local minima does not give as much guidance as the TSP, which indicates that the QAP is more difficult to solve than the

TSP. Also, they suggested that stronger search space exploration is needed for QAP with effective algorithms as the local minima in the QAP search space are more scattered [SH00].

The Set Covering Problem (SCP) is a NP-hard combinatorial optimization problem. It consists of finding a subset of columns of a zero-one m × n matrix such that it covers all the rows of the matrix at minimum cost. Many different applications can be modeled as SCP; from crew scheduling in airlines to scheduling and production planning in several industries. Finger et al. presented an analysis of FDC for SCP [FSL02]. As there was no straightforward distance measure for the SCP, the authors used the *closeness* between solutions to define their distance measure. If s and $s' \in S$ were two feasible solutions for an SCP instance, then they defined the closeness $n(., .)$ and distance $d(., .)$ between s and s' as

$n(s, s')$ = number of same columns in s and s'

$d(s, s') = \max(|s|, |s'|) - n(s, s')$

They used two sets of benchmark SCP instances with different parameters. Their results showed very strong differences among the search space characteristics between different types of instances. Hence, they proposed new ways of generating core problems, which is a much smaller SCP containing a subset of the columns that are most likely to appear in optimal solutions. Their results showed significant speed-up of the SA algorithm working on core problems with a very minor loss in solution quality. They suggested that by increasing the number of local optima for generating the core problems, more best-known solutions would be found [FSL02].

The Linear Ordering Problem (LOP) is an NP-hard combinatorial optimization problem. Given an $n \times n$ matrix C, the LOP is the problem of finding a permutation of the columns of matrix C such that the sum of the elements in the upper right triangle is maximized. For the FDC analysis, Schiavinotto and Stützle used *insert* local search and *CK* algorithm. The distance between two solutions or permutations was defined as the minimum number of applications of the basic local search operation to pass from a permutation to another. But as it was not possible for them to obtain the number of minimum *insert* application to go from one solution to another, they used a surrogate distance that evaluates the difference of two permutations just considering the position of the elements in both, which they called *precedence metric* [SS03].

Their analysis was based on two benchmark libraries of LOP instances. One was real-world instances, that were considered rather small, and the other one was randomly generated instances that were meant to have large number of solutions with costs close to the optimal value to be considered hard instances. The results indicated significant differences between the two groups. They suggested that the second group instances should be, when adjusting for the difference in size, easier to solve for metaheuristics. They suggested future work to be done on the search space analysis of LOP to try to distinguish the features that most affect the hardness of the problem more directly [SS03].

In 2003, Vanneschi and Tomassini presented research studying the pros and cons of FDC in genetic programming [VT03]. Their work was done for tree-based genetic programming, and was the first attempt to quantify the difficulty of problems with multiple optima by the FDC in EAs. They concluded that FDC is a reasonable index of difficulty for a number of problems. A counterexample was constructed and showed FDC to be a not infallible measure for problem difficulty. They planned to look for an alternative measure of difficulty in GP that should not depend on knowing the global optima, which is the strongest limitation of FDC.

Chapter 3 METHODOLOGY

3.1 SA Algorithm

The SA algorithm starts with a randomly generated state and its cost is calculated. If it is the goal state then the search terminates. Other wise, the starting state is considered the current state, the starting temperature is assigned and the algorithm starts to loop until the goal state is reach, no more operators to be applied to the current state, or for a certain number of iterations. In each iteration, a neighboring state is generated and its cost is evaluated. If it is the goal state, the search terminates and the state is retuned as the solution state. If it is not the goal state, it is compared to the current state. If it is better than the current state, it replaces it, other wise, it replaces the old current state with the probability $p' = e^{(-\Delta E/T)}$ defined in equation (2.2), where ΔE is the positive difference in cost and T is the current temperature. The temperature is lowered according to the annealing schedule and the loop continues. When the loop ends, the best state found is returned as the solution state. The authors in [RK91] recommend maintaining the best state found so far in addition to the current state. This helps if the final state is worse than the best state found so far (because of bad luck in accepting moves to worse states), then the best state would still be available.

The SA algorithm implemented is illustrated in Figure 1 [RK91]:

Step 1	Evaluate the initial state. If it is a goal state, then return it and quit. Otherwise, continue with the initial state as the current state
Step 2	Initialize *BEST-SO-FAR* to the current state, where *BEST-SO-FAR* is the best state found so far
Step 3	Initialize *T* according to the annealing schedule, where *T* is the temperature
Step 4	Loop until a solution is found or until there are no new operators left to be

```
        applied in the current state:
            a) Select an operator that has not yet been applied to the current state
               and apply it to produce a new state
            b) Evaluate the new state. Compute
                       $\Delta E$ = (value of current) – (value of new state)
                   • If the new state is a goal state, then return it and quit
                   • If it is not a goal state but is better than the current state, then
                     make it the current state. Also set BEST-SO-FAR to this new
                     state
                   • If it is not better than the current state, then make it the current
                     state with probability $p' = e^{(-\Delta E/T)}$ as defined in equation (2.2).
                     This step is usually implemented by invoking a random
                     number generator to produce a number in the range [0,1]. If
                     that number is less than $p'$, then the move is accepted.
                     Otherwise, do nothing
            c) Revise $T$ as necessary according to the annealing schedule
Step 5  Return BEST-SO-FAR as the answer
```

Figure 1: SA Algorithm

3.2 GESA Algorithm

The GESA algorithm uses a population of SA chains instead of keeping only one state at a time. It starts by assigning the starting temperature and generating a number of states *N*. For each starting state, a number of children are generated, *M,* by applying a neighboring operator to the starting state or parent. The first level of competition starts within each chain. The best child of the generated children is determined and tested to see if it is the goal state or not. If it is not the goal state, it is compared to the parent. If it is better than the parent it becomes the new parent for the next iteration. Other wise, it replaces the parent with the probability $p' = e^{(-\Delta E/T)}$ defined in equation (2.2), where ΔE is the positive difference in cost and T is the current temperature. The number of children to be generated for the next generated depend on the acceptance number of the chain. For each chain, each child's cost is compared to the parent. If it is better than the parent, then the acceptance number is incremented. Otherwise, it is compared to the best cost found ever, and the acceptance number is incremented with the probability $p' = e^{(-\Delta E/T)}$ where ΔE is the positive difference in cost between the child and the best cost found ever, and T is the current temperature. The annealing temperature is then lowered according to the annealing schedule and the loop continues by allowing each chain to generated the new number of children as defined in equations (2.3) and (2.4), which are illustrated in *****. The algorithm terminates when the goal state is found or a certain number of iterations is reached. Figure 2 illustrates the GESA algorithm [YP95]:

Step 1	Set initial temperature T.
Step 2	Randomly select N parents.
Step 3	Generate children from the parents.
Step 4	Find the best child for each parent (1st level competition or local competition).
Step 5	Find the parents for the next generation (selection). For each family, we accept the best child as the parent for the next generation if $y_1 < y_2$ or $e^{-(y1-y2)/T} > \rho$ where y_1 is the objective value of the best child, y_2 is the objective value of its parent, T is the temperature coefficient, and ρ is a random number uniformly distributed between 0 and 1.
Step 6	Find the number of children that will be generated from the parents of the next generation (second level competition). The details of this step are given in Figure 3.
Step 7	Decrease the temperature.
Step 8	Repeat step 3 to step 7 until an acceptable solution has been found or until a certain number of iterations has been reached.

Figure 2: GESA Algorithm

The procedure of determining the number of children for each parent in the next generation is illustrated in Figure 3:

```
Step 1  Repeat step 2 to step 5 for each family; Goto step 6.
Step 2  *count* = 0.
Step 3  Repeat step 4 for each child; Goto step 5
Step 4  If the objective value of the child is less than that of its parent, increase *count*
        by 1, if not, then we increase *count* by 1 if
                $e^{-(y1-y2)/T} > \rho$
        where $y_1$ is the objective value of the child, $y_2$ is the lowest objective value
        ever found, $T$ is the temperature coefficient, and $\rho$ is a random number
        uniformly distributed between 0 and 1.
Step 5  Acceptance number of the family is equal to *count*.
Step 6  Sum up the acceptance numbers of all the families
Step 7  For each family $i$, the number of children generated can be calculated
        according to the following formula
            $m_i = U A_i / S$                                   (3.1)
        where
            $S = \Sigma_j A_j$                                  (3.2)
        where $m_i$ is the number of children that will be generated for that family $i$ in
        the next iteration, $U$ is the total number of points, which is the initial number
        of families $N$ multiplied by the initial number of children per family $M$, $A_i$ is
        the acceptance number for that family, $S$ is the sum of the acceptance
        numbers.
```

Figure 3: Calculation of Acceptance Number and Number of Children

3.3 Extending GESA with the Heritage Factor

In [Abd01], Abdelbar introduced the heritage factor to make the number of children of the next iteration not only dependent on the last iteration, but on the pervious history of the family. This allows the decay of a family to occur more smoothly and gives a good family a better chance of recovering one bad generation.

Introducing the heritage factor changes only the equations (3.1) and (3.2) in the GESA algorithm. The number of children m_i for of the family i in the iteration $t+1$ would be

$$m_i^{(t+1)} = U\,[\,A_i + \alpha\, m_i^{(t)}\,]\,/\,S \qquad (3.3)$$

and

$$S = \Sigma_j\,[\,A_j \; + \alpha\, m_j^{(t)}\,] \qquad (3.4)$$

where $0 \leq \alpha < 1$ is a manually-tuned parameter that the author called the *heritage factor,* U is the total number of points, which is the initial number of families N multiplied by the initial number of children per family M, A_i is the acceptance number for that family, and S is the sum of the acceptance number. When $\alpha = 0$, the model reduces to pure GESA.

3.4 CBA Instances

As there are no standard benchmark instances for CBA problems, a set of difficult random instances had to be generated. The problems had to be non-trivial and still manageable for the algorithm to reach the optimal solution. The problems used here for testing are the same used in [AGA04]. They were generated using Santos' ILP (Integer Linear Programming) method for CBA as a benchmark.

The CBA generator takes five parameters; number of hypotheses, number of rules, maximum number of antecedents in any rule, percentage of hypotheses which should be sub-goals, and a lower-bound on the number of assumable hypotheses [AGA04].

The generated CBA instances have the following characteristics [AGA04]:

- The hypotheses are either assumable or provable. Assumable hypotheses have non-infinite cost and do not appear as consequents of any rules. Provable hypotheses are consequents of rules and have infinite cost
- Hypotheses are serially numbered and no hypotheses are allowed to be an antecedent in a rule whose consequent has a lower serial number. This is due to the fact that logical cyclicity is not allowed in hypotheses, where hypothesis *a* is an antecedent of a rule that has hypothesis *b* as a consequent and vise-versa
- All assumable hypotheses are randomly assigned an integer cost in the range 1 to 1000

Table 1 shows a small example of a CBA instance. It shows the .cba file format and the meaning of it. This is only for illustration; the costs are not uniformly distributed. Appendix B is one of the problems used for testing illustrated in Table 2 which is qab030.cba.

Table 1: Small CBA example showing the .cba format used and what it means

.cba Format	What it means
1	Signature
7	Number of Hypothesis
4	h_1 assumable and its cost = 4
5	h_2 assumable and its cost = 5
6	h_3 assumable and its cost = 6
0	h_4 provable and its cost is infinity
0	h_5 provable and its cost is infinity
0	h_6 provable and its cost is infinity
0	h_7 provable and its cost is infinity
6	Number of Rules

2 1 3 4	2 antecedents in this rule: $h_1 \wedge h_3 \rightarrow h_4$
2 3 2 4	2 antecedents in this rule: $h_2 \wedge h_3 \rightarrow h_4$
2 2 4 5	2 antecedents in this rule: $h_2 \wedge h_4 \rightarrow h_5$
2 5 1 6	2 antecedents in this rule: $h_5 \wedge h_1 \rightarrow h_6$
2 3 4 6	2 antecedents in this rule: $h_3 \wedge h_4 \rightarrow h_6$
3 6 2 5 7	3 antecedents in this rule: $h_6 \wedge h_2 \wedge h_5 \rightarrow h_7$
7	Goal to be proved h_7

Fixing the number of hypotheses at 300 and the number of rules at 900, 425 instances were generated using the CBA generator while varying the number of assumable hypotheses from 40 to 200 in steps of 10. Three instances were picked as the most difficult according to the following criterion. Each of the 425 instances was solved as a linear program without the integrality constraints, and the ratio of the number of non-integral variables in the solution to the total number of variables in the linear program was taken as a measure of difficulty of solving the problems. This process was repeated with fixing the number of hypotheses at 100 and the number of rules at 300, and varying the number of assumable hypotheses from 30 to 70 in steps of 10, generating 25 instances at each step. The same criterion was used to determine the three most difficult instances to be used for testing [AGA04].

Table 2 describes the six instances, showing the optimal solution obtained using lp-solve as the ILP engine of Santos' ILP method, the number of hypotheses in each problem, the number of assumable hypotheses, the number of rules, the CPU time taken by lp-solve to reach the optimal solution, and the depth and number of nodes of the branch and bound tree processed by lp-solve. For all the six instances, the global optimum is unique [AGA04]. Theses instances, along with the CBA generator are available at http://www.cs.aucegypt.edu/abdelbar/CBAlib/

Table 2: CBA Instances with their optimal cost, number of hypotheses, number of assumed hypotheses, number of rules, time used by ILP to reach optimal solution, depth of the branch-and-bound tree used by lp-solve and the number of nodes in the tree

Name	*Optimal Cost*	*# of Hyp.*	*# of Assum.*	*# of Rules*	*Integer Linear Programming*		
					Time (sec.)	*Depth*	*Nodes*
raa180	10,821	300	180	900	88,835.1	41	178,313
caa200	7,678	300	200	900	7,604.6	31	6,033
oaa110	6,856	300	112	900	1,792.1	35	6,675
lab070	5,423	100	70	300	108.0	19	727
rab050	2,644	100	50	300	3.9	10	45
qab030	3,830	100	30	300	208.2	21	833

3.5 Problem Representation

The CBA problem is represented by the following structures:

- *Hypotheses Cost Array*, which is a one dimensional array of size equal to the number of hypotheses. It represents all hypotheses, those that can be assumed with their actual cost and those that cannot be assumed with cost equal zero
- *Rules Array*, which is a two dimensional bit array with the size of the number of hypotheses by the number of rules. Each row represents a rule, and each column represents a hypothesis. If the bit *b[i,j]* is set then hypothesis *i* is an antecedent for rule *j*, otherwise, it is not.
- *Results Array*, which is a one dimensional array with the size of the number of rules. It represents the consequent of the rules.
- *Goal*, which is the goal hypothesis that needs to be proved.

For example, if we want to represent the problem described in Table 1, here are the structures used:

Hypotheses Cost Array

4	5	6	0	0	0	0

Rules Array

		h_1	h_2	h_3	h_4	h_5	h_6	h_7	*Results Array*
$h_1 \wedge h_3 \rightarrow h_4$	r_1	1	-1	1	-1	-1	-1	-1	4
$h_2 \wedge h_3 \rightarrow h_4$	r_2	-1	1	1	-1	-1	-1	-1	4
$h_2 \wedge h_4 \rightarrow h_5$	r_3	-1	1	-1	1	-1	-1	-1	5
$h_5 \wedge h_1 \rightarrow h_6$	r_4	1	-1	-1	-1	1	-1	-1	6
$h_3 \wedge h_4 \rightarrow h_6$	r_5	-1	-1	1	1	-1	-1	-1	6
$h_6 \wedge h_2 \wedge h_5 \rightarrow h_7$	r_6	-1	1	-1	-1	1	1	-1	7

Goal = h_7

The state used by the algorithm is represented mainly by a *hypotheses vector*. This vector has the size of the number of hypotheses and each element in it has one of three states:

- -1 for hypotheses that cannot be assumed
- 0 for hypotheses that can be assumed but not currently assumed in this state
- 1 for hypotheses that can be assumed and are currently assumed

For example, this could be a state instance of the problem defined in Table 1 :

	h_1	h_2	h_3	h_4	h_5	h_6	h_7
s1	1	0	1	-1	-1	-1	-1

3.5.1 Starting State

The starting state is a randomly generated state where the hypotheses vector is initialized all to -1, then we iterate on the can-be-assumed hypotheses and assign them 0 or 1 randomly. Its cost is then evaluated according to the cost function used and state is returned as the starting state.

3.5.2 Neighboring Operator

In both SA and GESA algorithms, going from one state to another means toggling the state of a can-be-assumed hypothesis from being assumed to de-assumed or vise versa. With a small probability, usually 0.2, another hypothesis is chosen to be flipped from assumed to be de-assumed or vise versa. In GESA, changing one hypothesis to generate many children for one parent sometimes produces duplicate children. Hence another hypothesis would be chosen to change state to get to the neighbor of the neighbor of that parent, and so on.

3.5.3 Distance Measure

The distance measure is needed when calculating the FDC for the CBA instances. It is used here as the number of assumed hypotheses that are not assumed in the global minimum, plus the number of un-assumed hypotheses that are assumed in the global minimum.

3.5.4 Cost Function

The actual cost of a CBA state is the pure summation of the costs of all assumed hypotheses in the current state. However, not all states of a CBA instance are feasible states, meaning that not all representations of assumed hypotheses would prove the goal. Accordingly, an unfeasible state might have an actual cost less than the best state. Hence, at first, a penalty cost was added to unfeasible states to have a much higher cost than all other states, which was the case used in [AA03]. But this way makes the algorithm accepts unfeasible states rarely as they have a very high cost, which could prevent it from reaching a better state if the only way to get to it was to go through some unfeasible states, and so be trapped in a local minimum. This led to the generation of the *heuristic repair* functions. Each gives the cost of the current unfeasible state when it is converted to a feasible state. And so, the heuristic repair functions are never less than the best cost state of the problem.

3.6 Heuristic Repair Functions

Two heuristic repair functions were implemented to solve the problem of adding a penalty cost to unfeasible states; *Worst-Cost* function and *P-Cost* function. Following is a description of both functions.

3.6.1 Worst-Cost Function

The Worst-Cost function was designed to give an upper-bound value of the cost of the current unfeasible state when converted to a feasible state. It was used in the implementation and results in [AGA04]. The idea was to find the cost of a set of assumable hypotheses that when added to the hypotheses already assumed by the current unfeasible state would be sufficient to prove the goal. This cost is not necessarily the minimum; it was used as a pessimistic approximation of the cost of the unfeasible state.

The Worst-Cost function is based on assigning a pessimistic cost to each hypothesis. At first, the rules are sequentially reordered by the consequent hypothesis number. All hypotheses, as mentioned in the CBA Instances section, are already generated serially numbered in such a way that in any rule, the hypotheses numbers of all antecedents are less than the hypothesis number of the consequent. Starting from an unfeasible state *s*, we first examine its set of assumed hypotheses. We assign a pessimistic value of zero to all the assumable hypotheses that are assumed by *s*. The can-be-assumed hypotheses but not currently assumed are assigned a pessimistic cost that is equal to the actual cost of assuming each one of them. For example, if we consider the example shown in Table 1, suppose the current state is assuming only h_2, so we assign a worst-cost to h_2 to be zero, and assign worst costs to h_1 and h_3 to be their actual costs which are 4 and 6 respectively.

For the other set of hypotheses that cannot be assumed and has to be proved, we start examining the rules one by one starting with the lowest numbered consequent hypothesis and ending with the goal hypothesis rule. If a hypothesis appears as the consequent of only one rule, we assign it a pessimistic cost that is equal to the sum of the pessimistic costs of the antecedents of that rule. As the rules are sorted sequentially, the pessimistic costs of the antecedents are already calculated in a previous step. If a hypothesis appears as the consequent of more than one rule, then the sum of the pessimistic costs of the antecedents of each rule is computed, and the consequent hypothesis is assigned a pessimistic cost that is equal to the minimum of the sums of the antecedent pessimistic costs.

Following our example, applying the rule r_1 ($h_1 \wedge h_3 \rightarrow h_4$) gives a worst-cost for h_4 equal to 10, then applying r_2 ($h_2 \wedge h_3 \rightarrow h_4$) gives also h_4 a worst-cost of 6, which is better than 10, so we keep the new value 6. Then we move to r_3 ($h_2 \wedge h_4 \rightarrow h_5$) which gives a worst-cost 6 for h_5 , then r_4 ($h_5 \wedge h_1 \rightarrow h_6$) which gives a worst-cost 10 for h_6, then r_5 ($h_3 \wedge h_4 \rightarrow h_6$) that gives a worst-cost 12 for h_6, but the previous value of 10 is better, so we keep h_6 equal to 10. Then we have the final rule r_6 ($h_6 \wedge h_2 \wedge h_5 \rightarrow h_7$) that gives a worst-cost value of 16 to h_7.

At the end of the process, the state *s* is assigned its fitness to be equal to the sum of the assumability costs of the hypotheses assumed by *s* plus the pessimistic cost of the goal. In our example, that will be original cost of h_2 plus worst-cost of h_7, which is equal to 21. Note that the pessimistic cost of the goal of a feasible state would be equal to zero because no other hypotheses are needed to be assumed to prove the goal in a feasible state.

The assumed hypotheses of state *s* are not changed, but we keep another vector that has the indexes of the hypotheses needed to be additionally assumed to prove the goal in case we need to convert this unfeasible state *s* to a feasible one. The state representation is not changed to allow the search to continue from the same point it reached, because if the state representation changes, it will create a jump in the search space to another location to start from there.

3.6.2 P-Cost Function

As the Worst-Cost function gets an upper bound of the cost of an unfeasible state, the P-Cost function is designed to calculate the actual cost of an unfeasible state when converted to a feasible state. The process of the P-Cost function is almost the same as in the Worst-Cost function, with a minor change in examining each rule.

In this process also we use the fact that all hypotheses are already generated serially numbered in such a way that in any rule, the hypotheses numbers of all antecedents are less than the hypothesis number of the consequent. The rules are sequentially reordered by the consequent hypothesis number.

Now we start examining each hypothesis in the current state *s*. A bit vector of size equal to the number of hypotheses is associated to each hypothesis. This vector represents the dependent-on hypotheses; i.e. for hypothesis *h,* the bits that are *one* in its bit vector represent the hypotheses needed to be assumed to prove *h*.

For illustration, we will use the same example of Table 1, with the unfeasible state *s* where we assume only h_2. Starting from *s*, we go through the hypotheses one by one:

- The currently assumed hypotheses are assigned bit vectors equal to *zeros*, because they do not depend on any other hypotheses and they are already assumed so there is no extra cost in using them again. In our example, this means that h_2 has a bit vector of 0000000 and its cost is zero.
- The can-be-assumed hypotheses but not currently assumed by *s* are assigned dependent bit vector of *zeros* with *one* set only at the index of the hypothesis; i.e. if hypothesis h_i can be assumed but not assumed in the current state, its bit vector is all *zeros* except at index *i* where there is a *one*. In our example, this means that h_1 has a bit vector of 1000000 with cost 4 and h_3 has a bit vector of 0010000 with cost 6.
- The need-to-prove hypotheses appear as consequents of rules. Taking the rules one by one, each consequent hypothesis is assigned a bit vector that is the result of a bit-wise OR of all its antecedents' bit vectors. Note that the antecedents' bit vectors are already done because the rules are sorted sequentially and the consequents have higher numbers than the antecedents. The cost of proving this consequent is then calculated by summing the costs of the hypotheses with *one* bits in the bit vector.
- If the same hypothesis appears as a consequent of more than one rule, its cost and dependent bit vectors are calculated for each rule and the

hypothesis is assigned the bit vector of the lowest cost as its dependent bit vector.

Following our example, applying the rule r_1 ($h_1 \wedge h_3 \rightarrow h_4$) gives a bit vector for h_4 equal to 1010000 with p-cost equal to 10, then applying r_2 ($h_2 \wedge h_3 \rightarrow h_4$) gives also h_4 a bit vector 0010000 with p-cost of 6, which is better than 10, so we keep the new bit vector and the new value 6. Then we move to r_3 ($h_2 \wedge h_4 \rightarrow h_5$) which gives a bit vector 0010000 and p-cost 6 for h_5, then r_4 ($h_5 \wedge h_1 \rightarrow h_6$) which gives a bit vector 1010000 and p-cost 10 for h_6, then r_5 ($h_3 \wedge h_4 \rightarrow h_6$) that gives a bit vector 0010000 and p-cost 6 for h_6, which is better than the previous value of 10, so we keep the new values for h_6. Then we have the final rule r_6 ($h_6 \wedge h_2 \wedge h_5 \rightarrow h_7$) that gives a bit vector 0010000 and a p-cost value of 6 to h_7.

At the end of the loop, we will have the bit vector of the goal hypothesis with its cost. The fitness of state *s* is equal to the cost of the already assumed hypotheses of *s* plus the cost of the goal, which is in our example the original cost of h_2 plus the p-cost of h_7 which is equal to 11.

Note that the P-Cost function returns the exact cost of converting the current state to a feasible one using the current assumable hypotheses. When the P-Cost function is applied to a feasible state, the goal hypothesis bit vector will be all zeros and the cost returned will be the exact cost of the assumed hypotheses.

Both the Worst-Cost and the P-Cost functions were implemented and tested on the problem raa180.cba using pure SA. The P-Cost function showed improvement in the calculated cost of the current state in each iteration by 9% but was slower by 11%. It also made the search saturates in less number of iterations. As it is seen to be a better representative of the search space, it was used in the rest of testing.

3.7 Design Issues

There are many parameters involved in the design of the algorithms. For example, there is the starting temperature of the algorithms, the annealing schedule and the stopping conditions. Here are the decisions taken to determine these parameters.

3.7.1 Starting Temperature

The starting temperature *T* for the SA and GESA algorithms is an important parameter that has to be determined according to the problem being solved. For the CBA problem instances, *T* was determined based on the average cost of the assumable hypotheses. Going from one state to a neighboring state toggles one assumable hypothesis from being assumed to being de-assumed or vise versa. At the beginning of the search, a worse state would be accepted with the probability 0.5. This means that

$$e^{(-\Delta E/T)} = 0.5$$

where ΔE is the positive change in cost, which is ≈ 500 according to the generated CBA instances. This means that

$$T = -500 / \ln(0.5) \approx 720$$

Therefore, the decision was made to make the initial temperature for the search to be 720.

3.7.2 Annealing Schedule

After experimenting on some small CBA instances, decrementing the temperature every iteration by a factor of 0.9995 was found to produce best results.

3.7.3 Saturation

In the SA program, saturation means that the current state did not change for a certain number of iterations defined as an input parameter by the user. By this change we mean that the state representation did not change, or the cost did not change. For the GESA, each chain saturates when its current state does not change for a certain number of iterations defined by the user, and the program saturates when all the chains running reach their saturation state.

3.7.4 Stopping Criteria

In general, reaching the best solution could not be used as the only stopping criterion for the algorithms as it is not usually known. Hence, two additional methods, besides finding the best solution if known were used in the experimentation; number of iterations or saturation.

When using the number of iterations as a stopping condition, the user would define a maximum number of iterations for the program to go. It will stop either when the program is saturated according to the saturation criterion defined above or when reaching the maximum number of iterations defined by the user. On the other hand, the program would stop when it saturates, no matter how many iterations it had completed.

3.7.5 Acceptance Number

In the original GESA algorithm, when calculating the acceptance number of a chain, the best child is compared against the parent, if it has a better fitness, the acceptance number is incremented. If it is not better, it is compared against the best fitness found ever, and the acceptance number is incremented with the logistic probability used by SA [YP95]. This causes all chains but one to die eventually, having acceptance number of zero and producing no children. In the implementations, this would happen so early in iteration 10 or 11. Thus, the current implementation made the following change; instead of comparing against the best fitness found ever, the best child is compared against its parent, which is almost the best value found for the current chain, and the acceptance number is incremented if it is better than the parent or using the logistic probability of SA. Thus, all chains but one will die eventually, but at some later stage in the run.

Chapter 4 RESULTS AND ANALYSIS

The SA and GESA algorithms were implemented using C++ language. Experiments were done to fine-tune the parameters of SA, where the starting temperature was set as mentioned in section 3.7.1 and the annealing schedule fixed as mentioned in section 3.7.2 for both SA and GESA. Experiments are divided into three groups; Repeated Simulated Annealing (RSA) versus Guided Evolutionary Simulated Annealing (GESA), Fitness-Distance Landscape analysis for CBA using the SA algorithm, and experimentation within the GESA, varying the number of chains and the Heritage Factor.

4.1 RSA versus GESA

To test SA versus GESA, we used Run Length Distribution (RLD) as a measure. As the experiments were done on different kinds of machines with different configurations, Run Time Distribution (RTD) could not be used as a measure to compare among the results; instead we are using RLD for comparison.

RLD was introduced by Stützle and Hoos to model runtime behavior of randomized methods. Here is how it is calculated:

- A full run for RLD is considered the run ended by finding the optimal solution of the problem. So if a run has saturated on a local minimum, a new run starting from a new random state will then follow where the number of iterations of the first run is added to the number of iterations of the new run, and so on till one run finds the optimal solution and stops.
- A table is formed with one column showing the number of iterations of a full run and the other column showing how many times these iterations

reached the optimal solution. The table is sorted assendingly by the number of iterations.

- The number of times the runs reached the optimal solution is accumulated for the iterations. i.e., if the global was reached twice in five iterations and was reached three times in ten iterations, then we consider that by ten iterations, the algorithm reached the optimal solution five times.
- The accumulated number of times of finding the optimal solution in each iteration entry in the table is then divided by the total number of full runs, to get the percentage of finding the optimal solution.

For Repeated SA (RSA), we consider the runs of SA as repeated runs, where a whole run of RSA is terminated by finding the global optimum of the current problem instance. If a run of SA saturates on a local minimum, we start another run from a random state with a new search. The number of iterations of different runs is accumulated until the global minimum is found where the RSA is considered to terminate.

GESA was also treated in the same way, where different runs would add up the number of iterations gone through until the global optimum is found. But for a fair comparison between RSA and GESA, the number of iterations of GESA is multiplied by the number of chains N multiplied by the initial number of children M. In one iteration using SA, only one state is generated as the child of the parent. However, in GESA, the total number of children generated each iteration is equal to $N * M$, and hence we use this as an indicator that the run length of one loop using SA is equivalent to $N * M$ run length of GESA. Table 3 shows an example of the RLD entries of the problem qab030 using GESA with 5 chains, 20 children and saturation 100. It shows the number of iterations in each full run, the number of iterations multiplied by 100 which is *N*M*, how many times it reached the optimal solution in each iteration entry, the accumulated summation of how many times the optimal solution was found and the percentage of finding it.

Table 3 : RLD example of qab030 using GESA (5,20), showing the number of iterations in each full run, the number of iterations multiplied by 100 which is *N*M*, how many times it reached the optimal solution in each iteration entry, the accumulated summation of how many times the optimal solution was found and the percentage of finding it

Num Iterations	*Num Iterations * 100*	*Reached Solution*	*Accumulated Sum Solutions*	*Percentage of Finding Optimal Solution*
3	300	2	2	0.1
4	400	20	22	1.1
5	500	26	48	2.4
6	600	68	116	5.8
7	700	134	250	12.5
8	800	209	459	22.95
9	900	256	715	35.75
10	1000	232	947	47.35
11	1100	220	1167	58.35
12	1200	185	1352	67.6
13	1300	147	1499	74.95
14	1400	92	1591	79.55
15	1500	76	1667	83.35
16	1600	94	1761	88.05
17	1700	43	1804	90.2
18	1800	34	1838	91.9
19	1900	22	1860	93
20	2000	29	1889	94.45
21	2100	29	1918	95.9
22	2200	17	1935	96.75
23	2300	11	1946	97.3
24	2400	13	1959	97.95
25	2500	6	1965	98.25
26	2600	9	1974	98.7
27	2700	2	1976	98.8
28	2800	2	1978	98.9
29	2900	2	1980	99
30	3000	3	1983	99.15
32	3200	4	1987	99.35
33	3300	2	1989	99.45
35	3500	2	1991	99.55
36	3600	1	1992	99.6
38	3800	2	1994	99.7
39	3900	2	1996	99.8
45	4500	2	1998	99.9
54	5400	2	2000	100

With SA, about 10,000 runs were done for each CBA problem instance described in section 3.4, with the stopping criteria of either finding the optimal solution or looping for 100 iterations with no-change in state or cost. The same stopping criteria were used for GESA, with $N = 5$ chains, $M = 20$ initial children for each chain, and about 2000 runs for each problem. Table 4 shows the number of runs of each problem, the maximum number of iterations reached to find the global optimum, and the number of times the global optimum was found.

Table 4 : RLD Analysis for RSA and GESA, showing the total number of runs, maximum number of iterations found to reach the global optimum, and the number of times the global optimum was reached.

	RSA			*GESA*		
Instance	*# Runs*	*Max # Iterations*	*Reached Global*	*# Runs*	*Max # Iterations * N * M*	*Reached Global*
raa180.cba	10,000	8,602,558	44	2,000	44,034 * 100	76
caa200.cba	10,000	1,698,665	175	2,000	4,060 * 100	901
oaa110.cba	10,000	241,254	1,946	2,000	3,742 * 100	1,120
lab070.cba	8,758	72,223	3,161	2,000	2,848 * 100	1,245
rab050.cba	10,000	13,118	9,821	2,000	57 * 100	2,000
qab030.cba	10,000	12,975	9,612	2,000	54 * 100	2,000

Figure 4 to Figure 9 show a comparison between RSA and GESA of the RLD analysis for the six CBA instances.

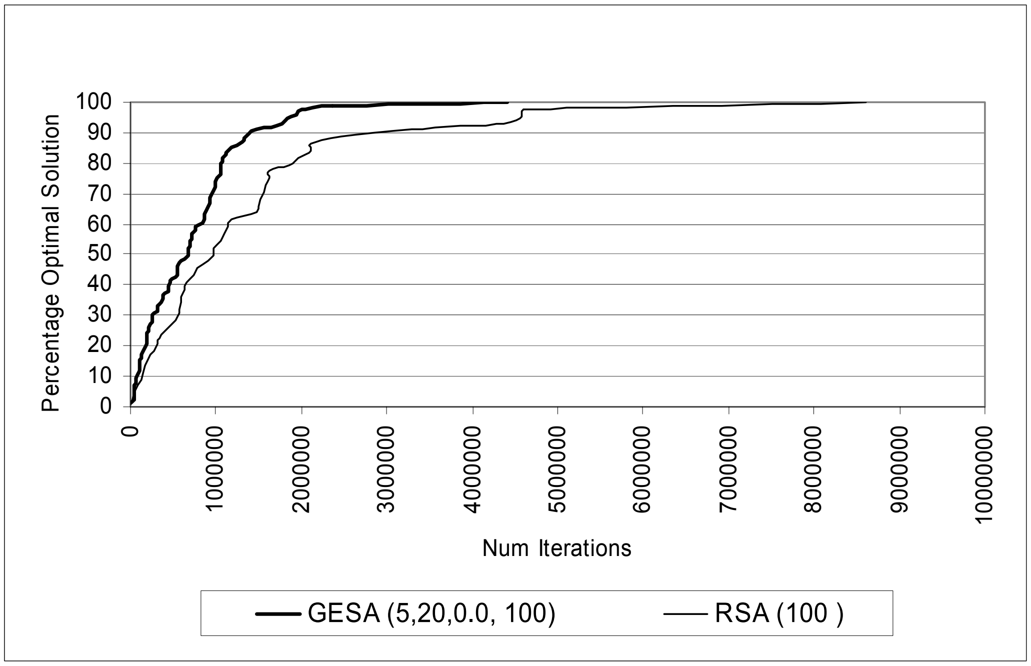

Figure 4 : RLD Analysis for raa180

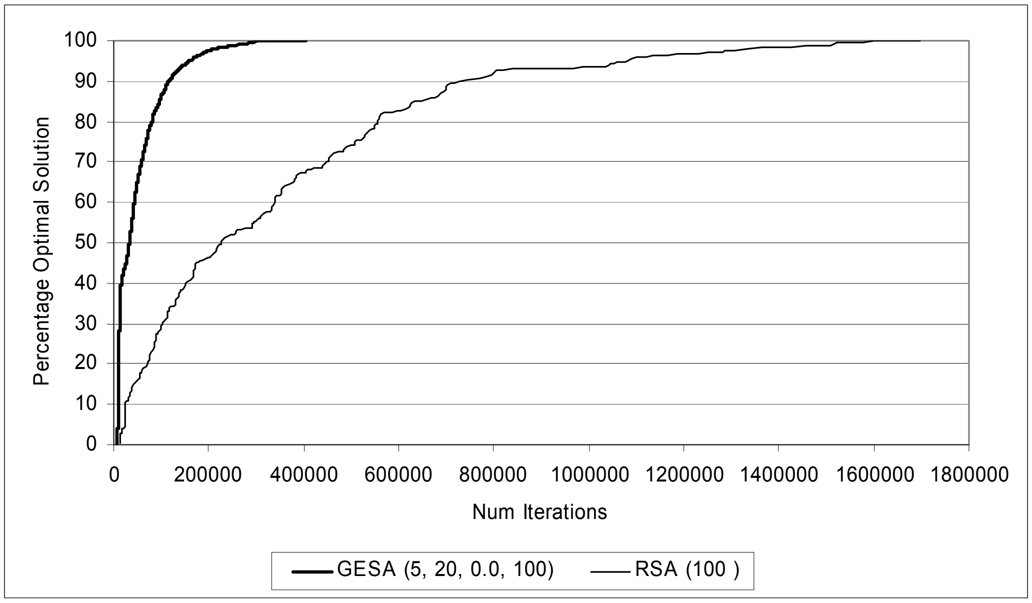

Figure 5 : RLD Analysis for caa200

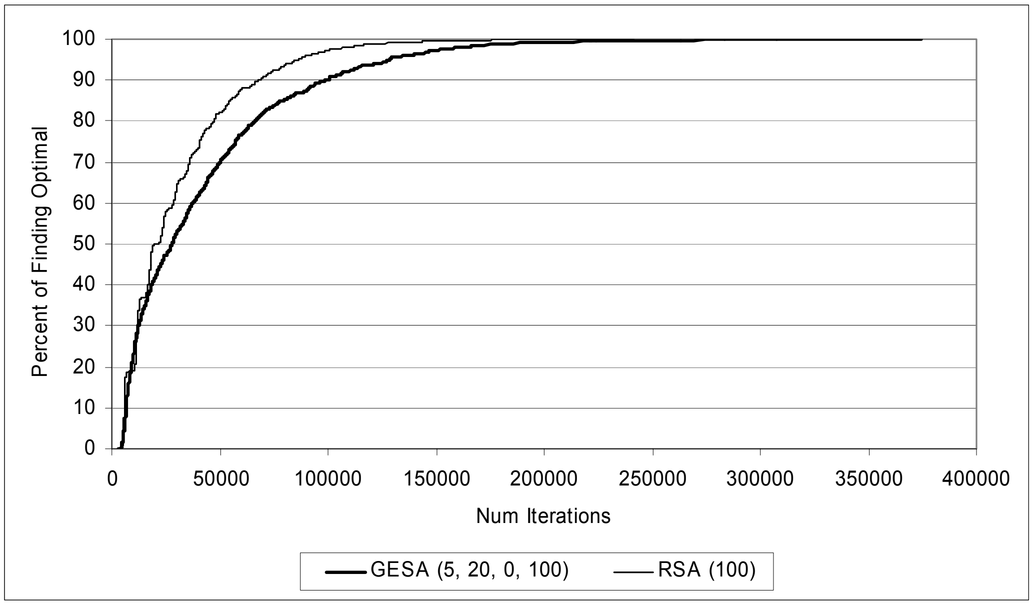

Figure 6 : RLD Analysis for oaa110

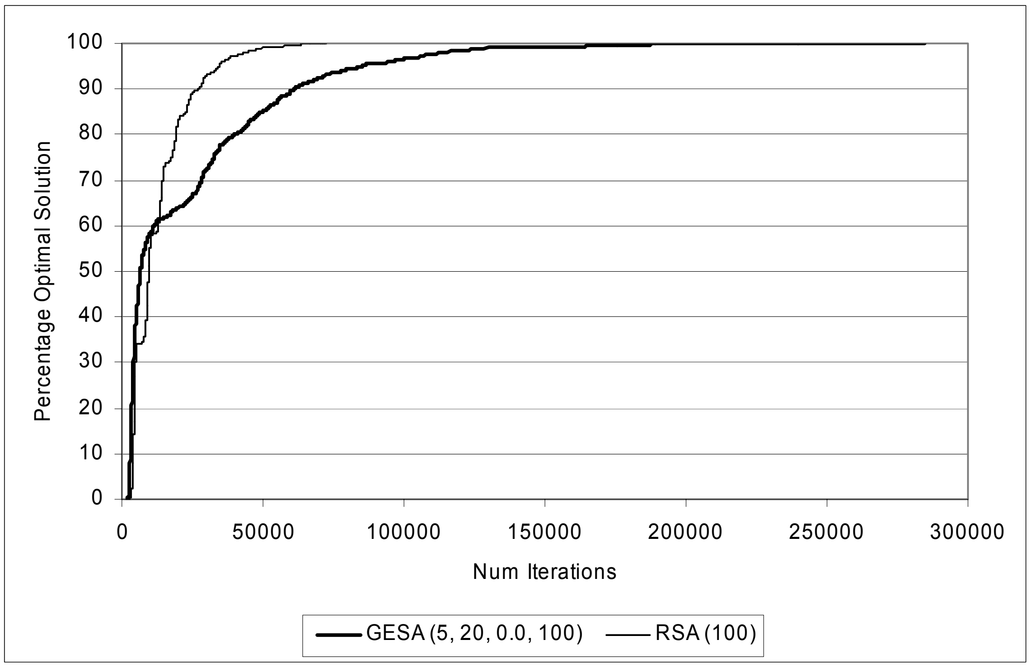

Figure 7 : RLD Analysis for lab070

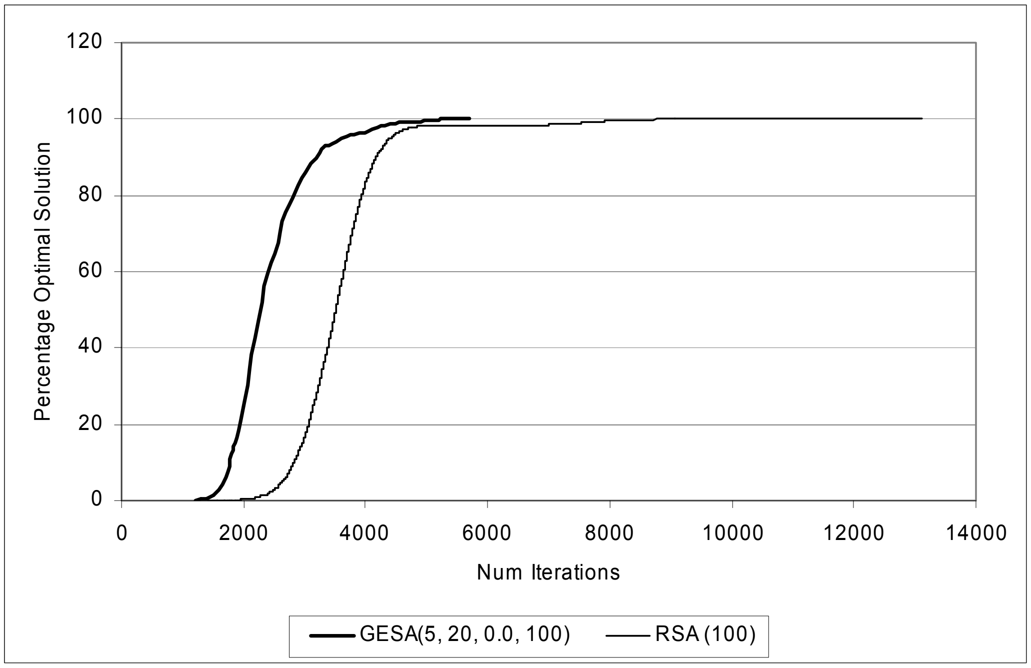

Figure 8 : RLD Analysis for rab050

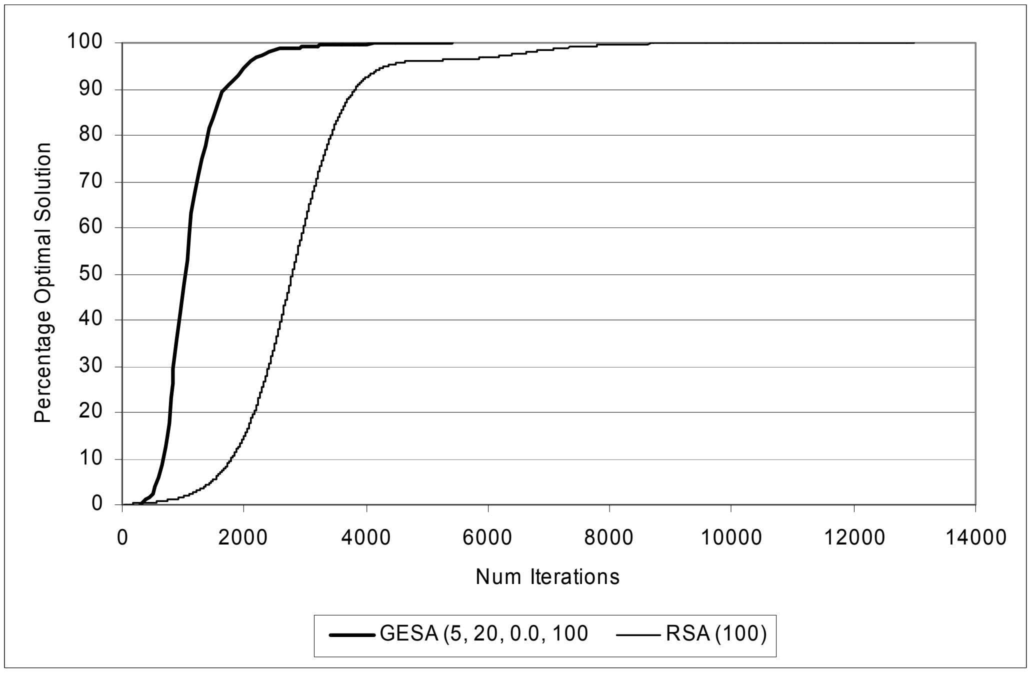

Figure 9 : RLD Analysis for qab030

The RLD analysis shows that GESA outperforms RSA in four of the six CBA problem instances used for testing. In the big problems raa180 and caa200 GESA clearly finds the optimal solution in less number of iterations. In the small problems rab050 and qab030, GESA found the global optimum in all runs and with much less number of iterations than RSA.

For the oaa110 problem, the two algorithms are almost overlapping until 40% of finding the optimal, and then SA outperforms GESA, and with lab070, GESA is taking the lead till 60% of finding the optimum and then SA outperforms it with a big difference. With closer look into the results of oaa110 runs, many local minima were saturated upon many times. For example, the local minimum of 6891 was the final solution 287 times out of the 2000 runs and it is 8 steps distant from the global optimum. Interestingly enough, in the paper [AGA04], the six instances of CBA used here were also solved using Iterated Local Search (ILS), which always found the optimum solution for caa200, rab050 and qab030, but sometimes failed to find the optimal solution and needed a phase of RSA in raa180, oaa110 and lab070.

The explanation given in [CC02] to justify the worse performance of GESA against SA in the distributed file and task placement problem might be valid here also. According to the authors, the search space for the distributed file and task placement problem is not continuous; hence the search for solutions must be carried out by looking through more possible solutions, so performance will be bettered if more generations are involved in the search. For CBA problems, the search space is also not a continuous one; there are many points that could be unfeasible states, which makes it easier for SA that jumps from one solution to another than for GESA that explores one generation at a time. But it is not clear why this is valid for only some of the instances and not all the cases.

4.2 Fitness-Distance Landscape Analysis

For the six CBA instances used in testing, a large number of local minima were needed along with the distance from the global optimum to calculate the fitness distance correlation coefficient and plot the fitness-distance landscape. Three sets of 10,000 runs were made for each CBA problem instance to produce the local minima. The first set used the stopping criteria of finding the best solution or saturation with 10; to produce as many different local minima as possible for the small problems. The second set was to saturate at 30, and the third was to saturate at 100, to try to come closer to the global optimum. Table 5 shows the results obtained using all the local minima found. It shows the number of local minima N_{ls}, the average percentage deviation $avg_{\%}$ of the cost of the generated local minima over the optimum cost, the average distance $avg_{d\text{-}ls}$ between the generated local minima, the ratio of $avg_{d\text{-}ls}$ to the number of assumable hypotheses $|H^A|$, the average distance $avg_{d\text{-}opt}$ to the optimum from the generated local minima, the ratio of $avg_{d\text{-}opt}$ to $|H^A|$, and the fitness-distance correlation coefficient ρ_{ls}.

Table 5 : Results of the FDC analysis for the six CBA instances

Instance	N_{ls}	$avg_{\%}$	$avg_{d\text{-}ls}$	$avg_{d\text{-}ls} / \lvert H^A \rvert$	$avg_{d\text{-}opt}$	$avg_{d\text{-}opt} / \lvert H^A \rvert$	ρ_{ls}
raa180.cba	24,497	0.5293	46.6862	0.2594	36.8030	0.2045	0.890
caa200.cba	24,498	0.9628	44.7875	0.2239	31.2962	0.1565	0.951
oaa110.cba	22,155	0.5131	28.2311	0.2521	22.8439	0.2040	0.837
lab070.cba	15,687	0.4653	19.8283	0.2833	16.2722	0.2325	0.546
rab050.cba	10,976	0.9199	11.4917	0.2298	7.8753	0.1575	0.784
qab030.cba	2,046	0.2998	8.9348	0.2978	7.1311	0.2377	0.460

Figure 10 to Figure 15 show the Fitness-Distance plots of the six CBA instances. Each figure is a plot of the distance to global optimum versus the percentage of deviation of fitness from global optimum.

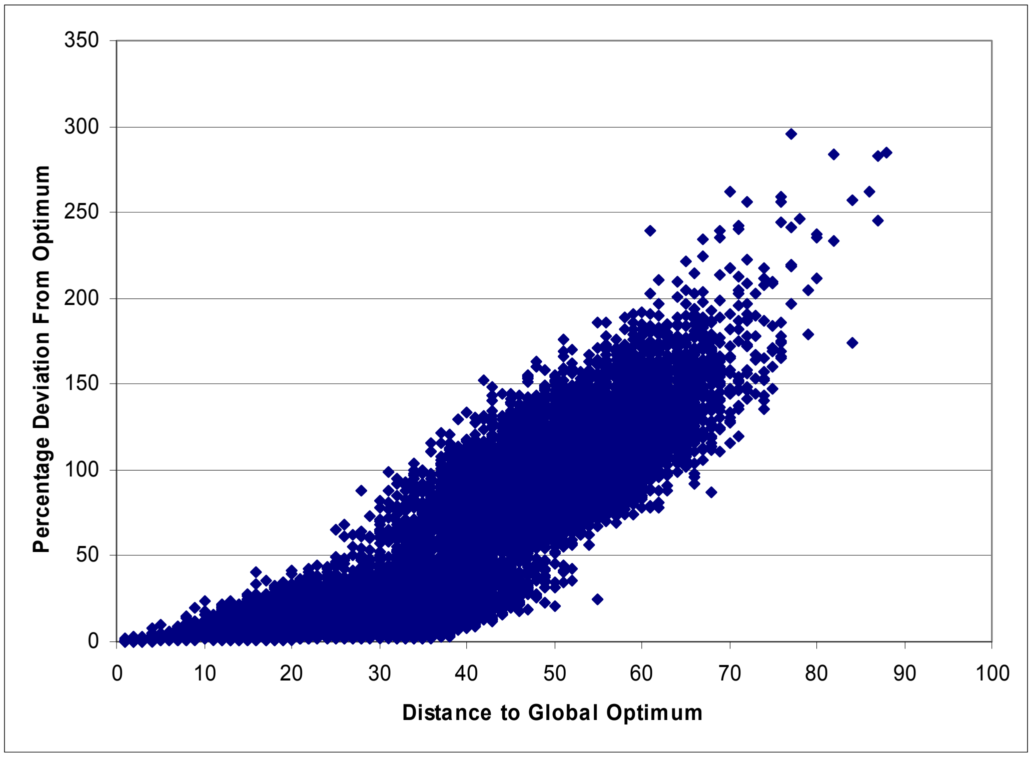

Figure 10: Fitness-Distance Plot of raa180

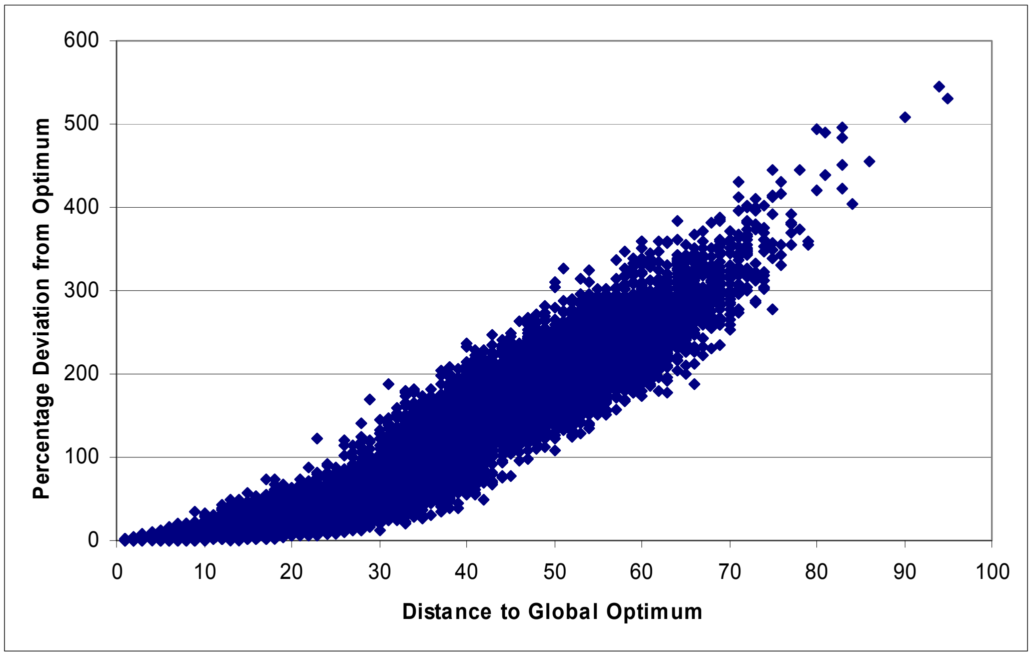

Figure 11 : Fitness-Distance Plot of caa200

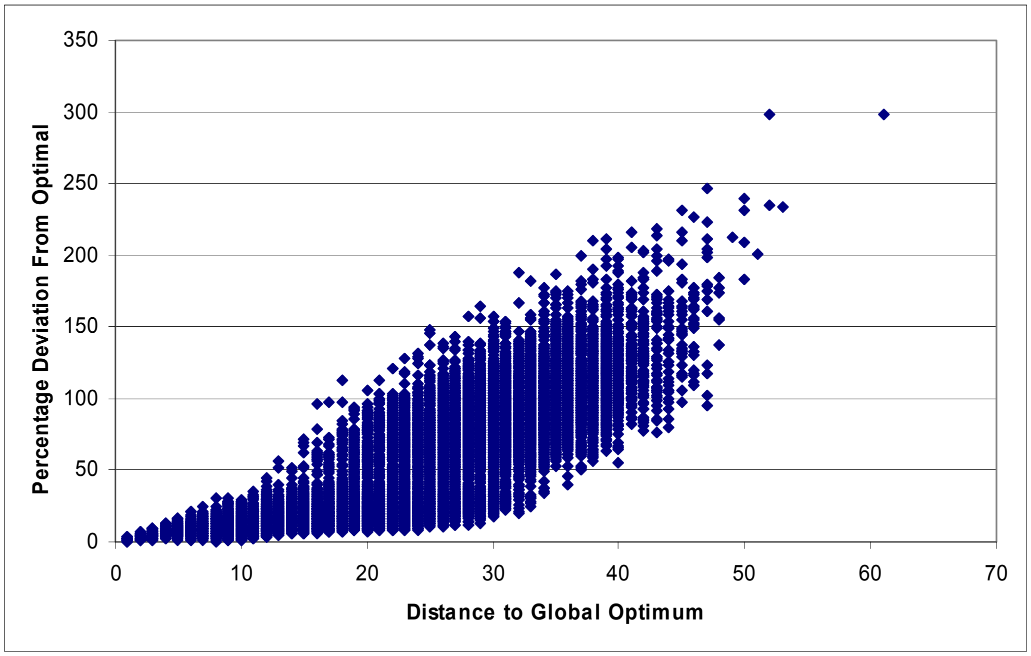

Figure 12: Fitness-Distance Plot of oaa110

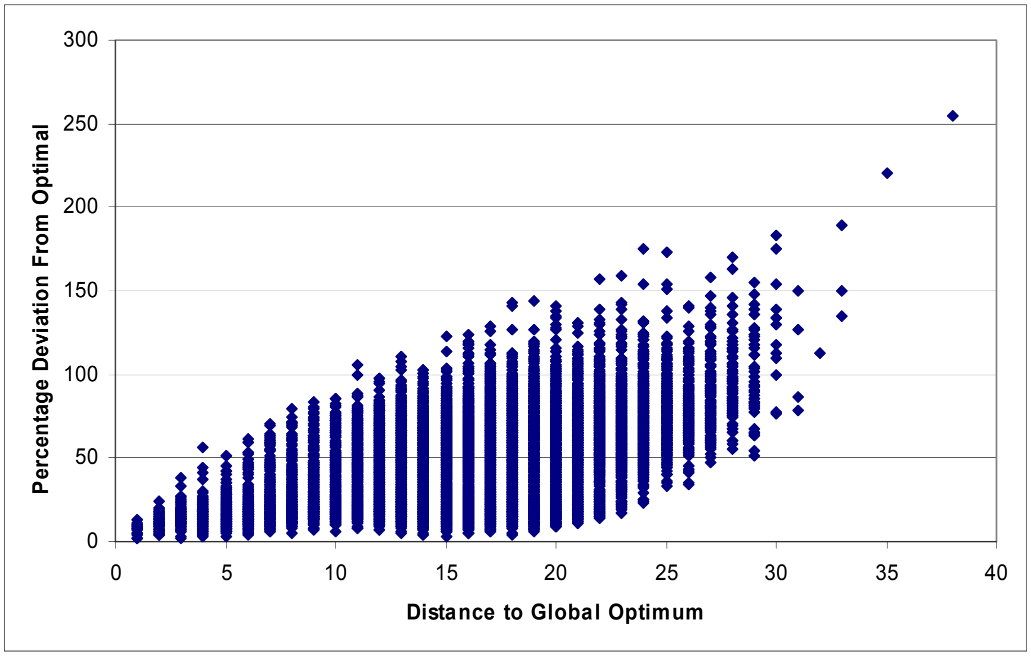

Figure 13: Fitness-Distance Plot of lab070

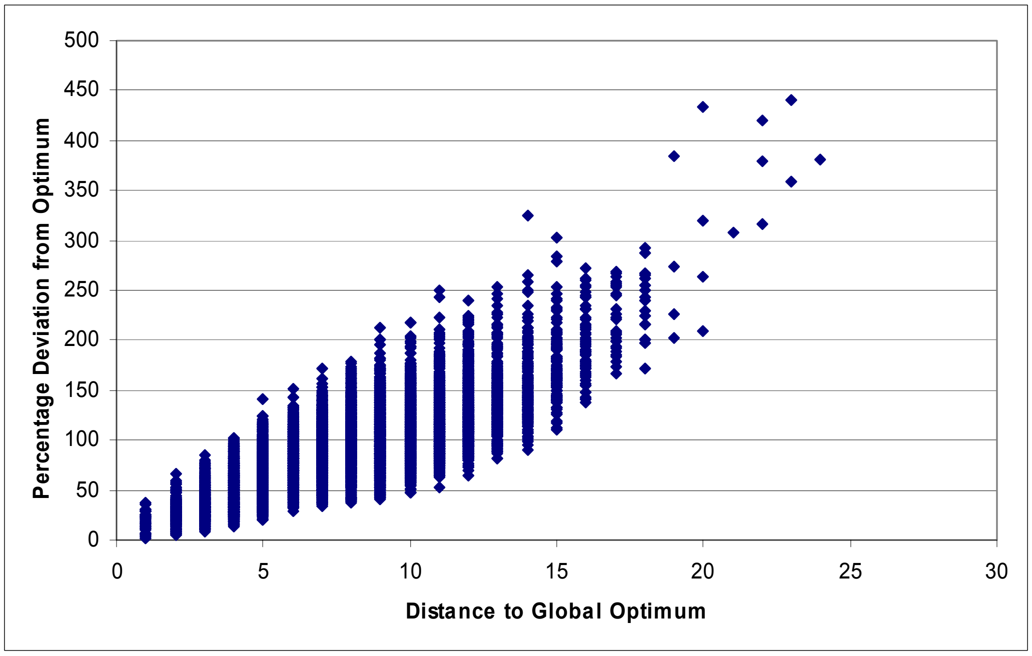

Figure 14: Fitness-Distance Plot of rab050

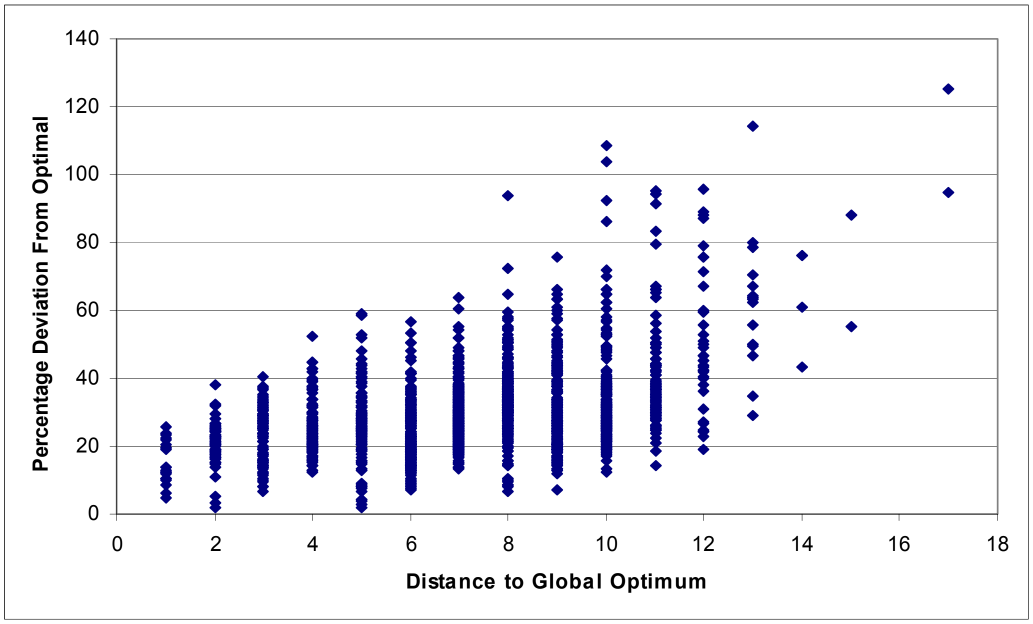

Figure 15: Fitness-Distance Plot of qab030

According to the classification of problem difficulty given in [JF95], the six CBA instances are classified as straightforward for genetic algorithms, because there is a strong correlation between fitness of a local minimum and its distance from the global optimum. This does not say that the problems are trivial, because for example, although raa180 has a much higher FDC Coefficient than qab030, still the global optimum is found much fewer times than qab030. Being straightforward is an indication that by getting lower fitness, the global optimum will be eventually reached. The Fitness-Distance plots of the six problems show that the farther the local minimum is from the global, the higher fitness it has in most cases. In raa180, oaa110, and lab070 there are many local minima that are so close in fitness to the global optimum but with a big distance from it, which might indicate that to saturate on the global optimum of those problems is harder than the other problems.

The values of the FDC coefficient found are considered high because the sample of local minima is really huge and so different. Classifying the local minima according to fitness and according to distance and taking the FDC coefficient accordingly gave the following interesting results.

4.2.1 FDC vs. Percentage of Deviation from Global Optimum

The results show variations of the value of the FDC coefficient ρ_{ls} that differs according to the number of local minima taken into consideration and their fitness with respect to the global optimum. Table 6 shows the different values of ρ_{ls} when fixing the percentage of deviation from the global optimum dev_{opt} to 5%, 10% and 100% respectively.

Table 6: FDC Coefficient vs. Percentage of Deviation From Optimum

	dev_{opt} = 5%		dev_{opt} = 10%		dev_{opt} = 100%	
Instance	N_{ls}	ρ_{ls}	N_{ls}	ρ_{ls}	N_{ls}	ρ_{ls}
raa180.cba	2609	0.420	4918	0.394	19133	0.822
caa200.cba	2598	0.310	4226	0.583	14509	0.838
oaa110.cba	712	0.195	2521	0.533	18785	0.799
qabo30.cba	8	-0.294	34	0.306	2042	0.453
lab070.cba	45	0.181	537	0.185	15222	0.521
rab050.cba	6	0.472	20	0.582	6640	0.691

From Table 6, the FDC coefficient value for oaa110 with 5% deviation from the optimal had the lowest value among the three big problems. This explains its behavior with GESA that RSA performed better as it has a low FDC coefficient near the optimal solution, indicating that it is harder than the other two big problems near the optimal solution.

Figure 16 to Figure 33 show the plots of FDC coefficient versus the percentage of deviation from global optimum with values 5%, 10% and 100% for the six CBA instances.

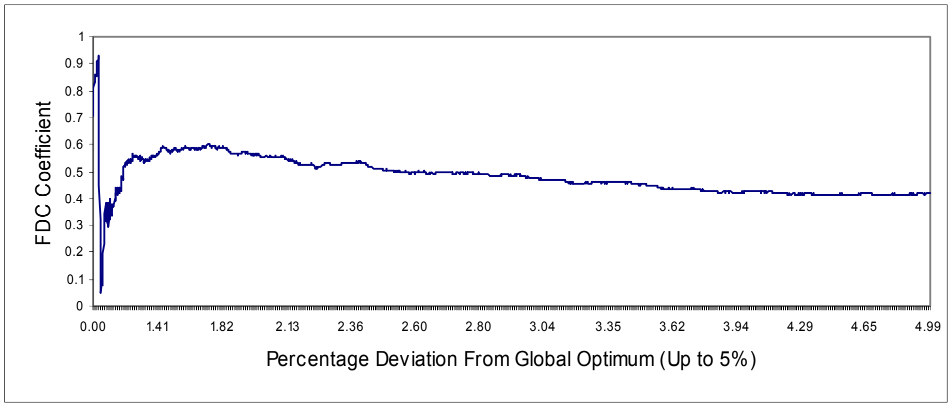

Figure 16 : FDC Coefficient vs. 5% Deviation from Global Optimum for raa180

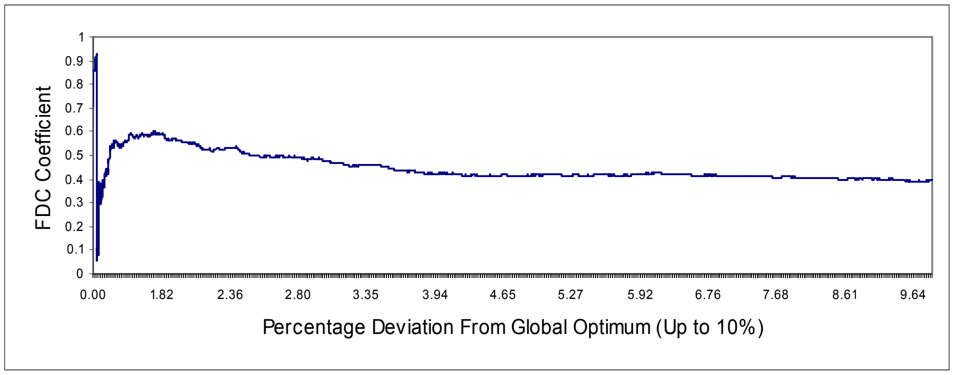

Figure 17 : FDC Coefficient vs. 10% Deviation from Global Optimum for raa180

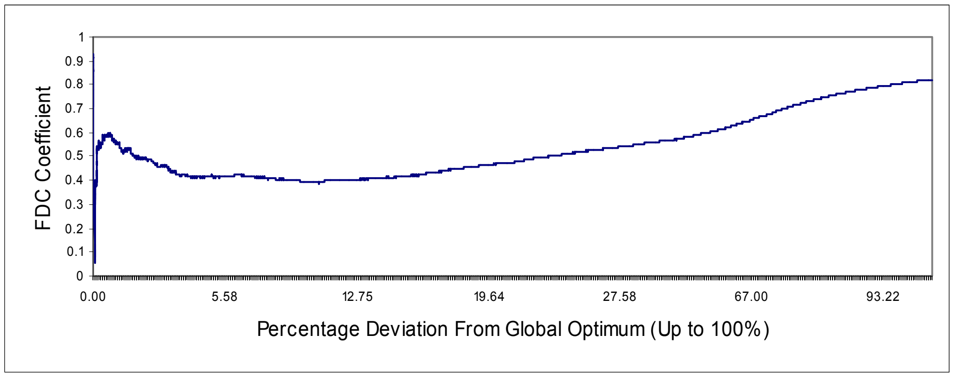

Figure 18 : FDC Coefficient vs. 100% Deviation from Global Optimum for raa180

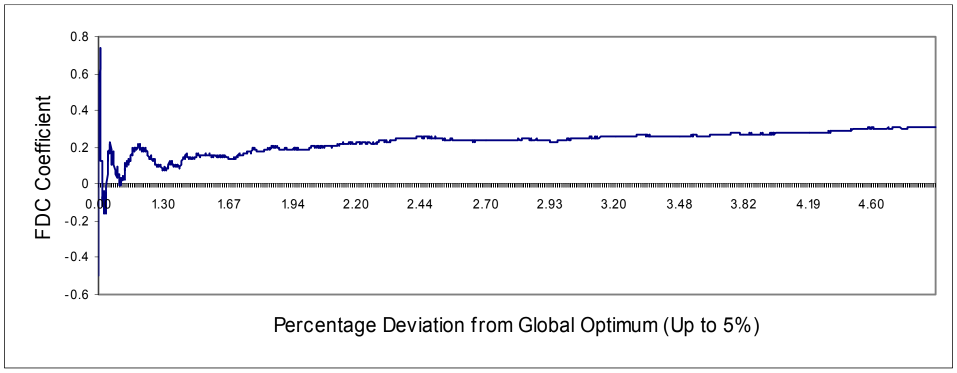

Figure 19 : FDC Coefficient vs. 5% Deviation from Global Optimum for caa200

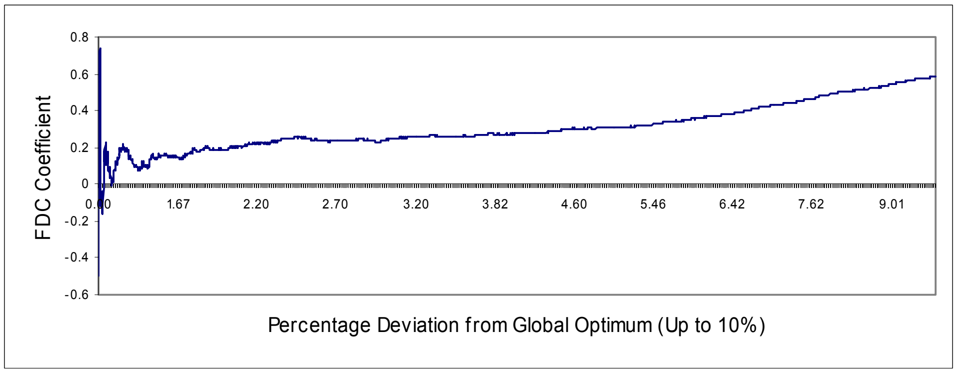

Figure 20 : FDC Coefficient vs. 10% Deviation from Global Optimum for caa200

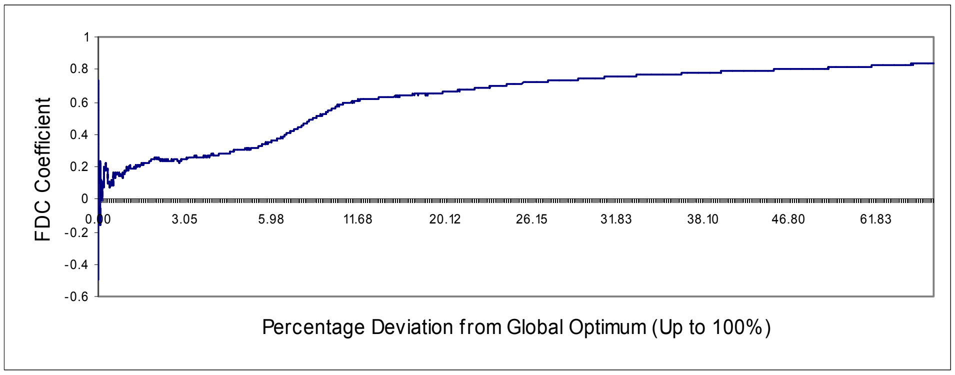

Figure 21 : FDC Coefficient vs. 100% Deviation from Global Optimum for caa200

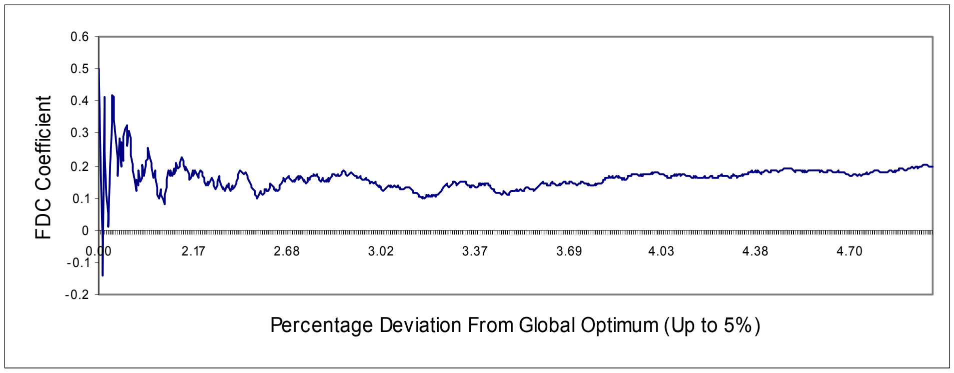

Figure 22 : FDC Coefficient vs. 5% Deviation from Global Optimum for oaa110

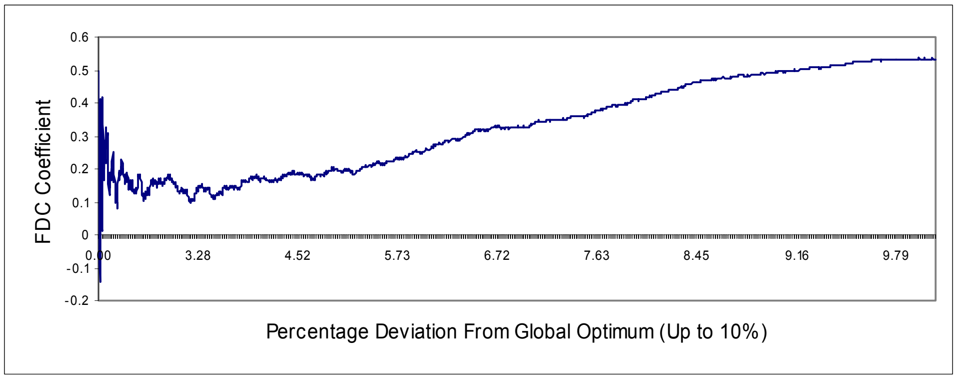

Figure 23 : FDC Coefficient vs. 10% Deviation from Global Optimum for oaa110

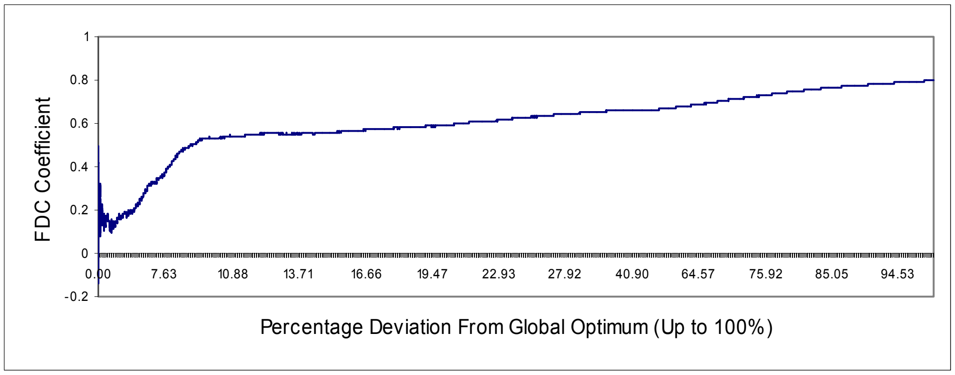

Figure 24 : FDC Coefficient vs. 100% Deviation from Global Optimum for oaa110

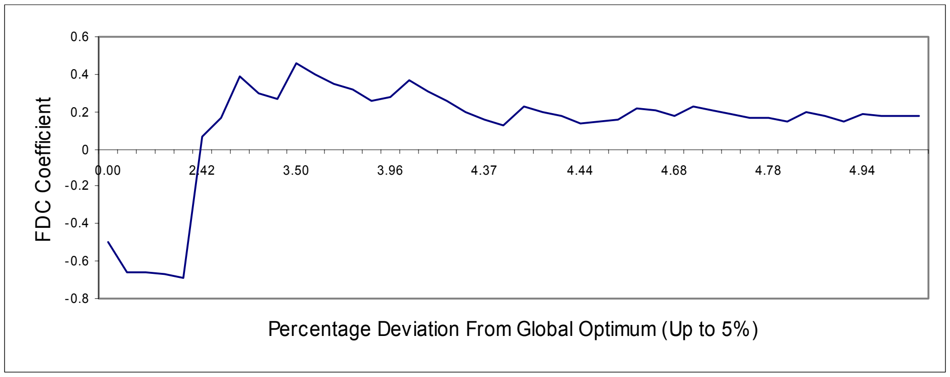

Figure 25 : FDC Coefficient vs. 5% Deviation from Global Optimum for lab070

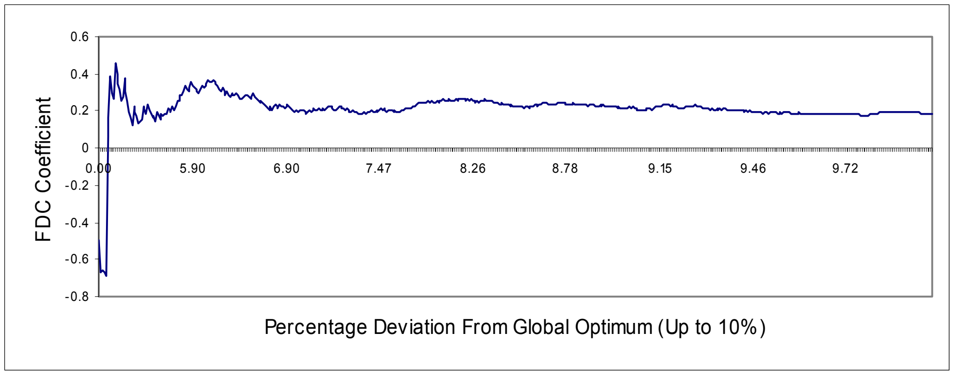

Figure 26 : FDC Coefficient vs. 10% Deviation from Global Optimum for lab070

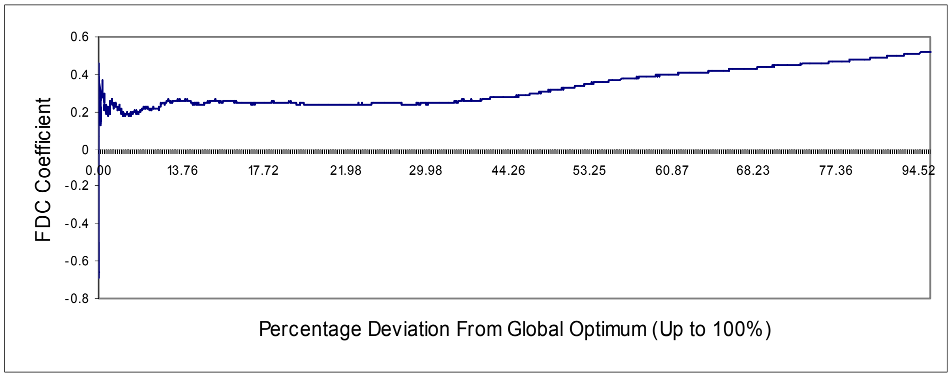

Figure 27 : FDC Coefficient vs. 100% Deviation from Global Optimum for lab070

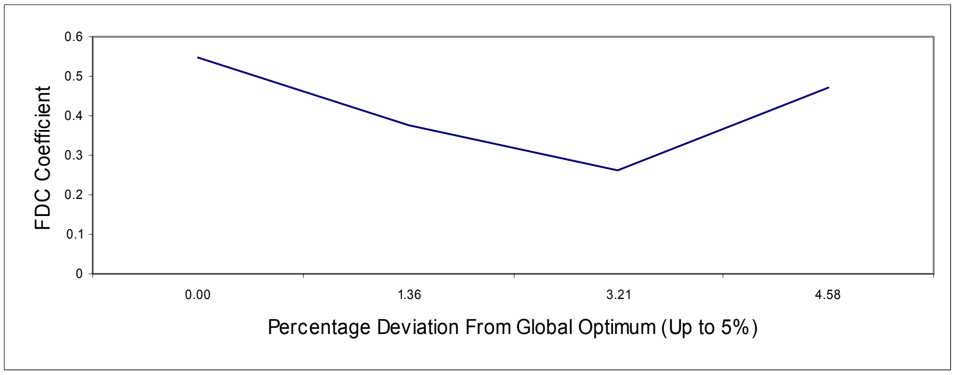

Figure 28 : FDC Coefficient vs. 5% Deviation from Global Optimum for rab050

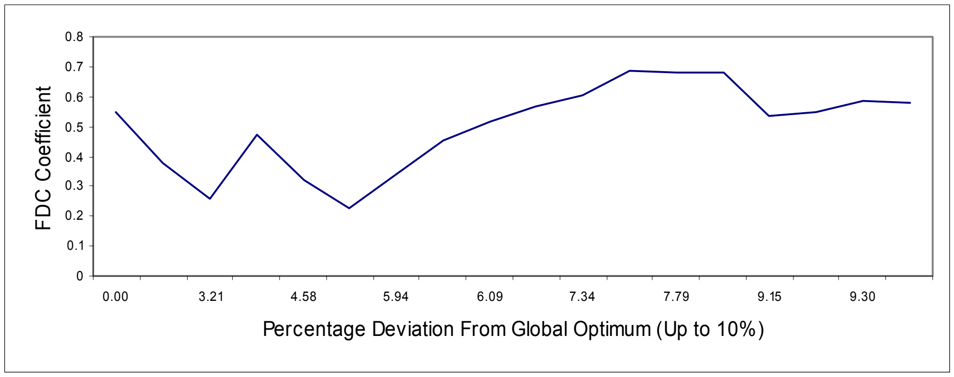

Figure 29 : FDC Coefficient vs. 10% Deviation from Global Optimum for rab050

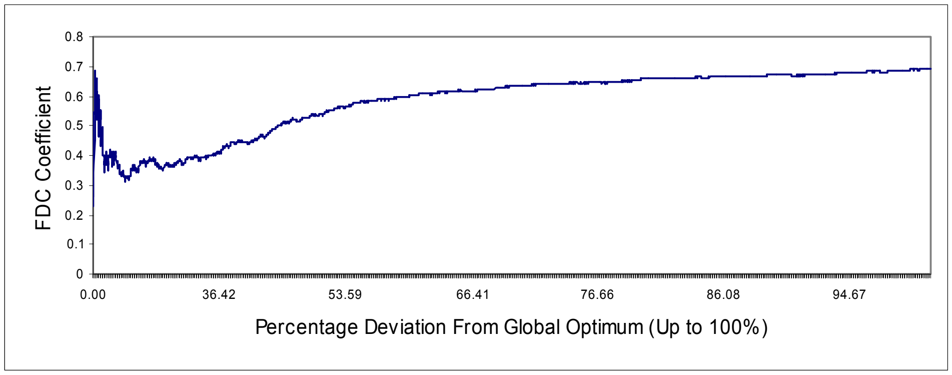

Figure 30 : FDC Coefficient vs. 100% Deviation from Global Optimum for rab050

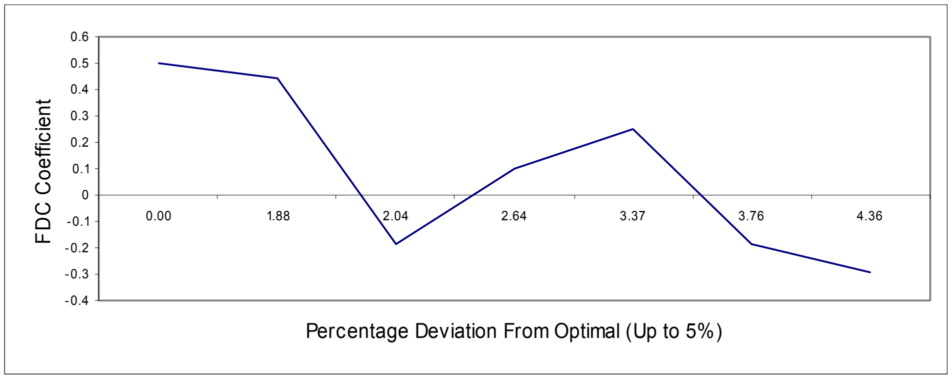

Figure 31 : FDC Coefficient vs. 5% Deviation from Global Optimum for qab030

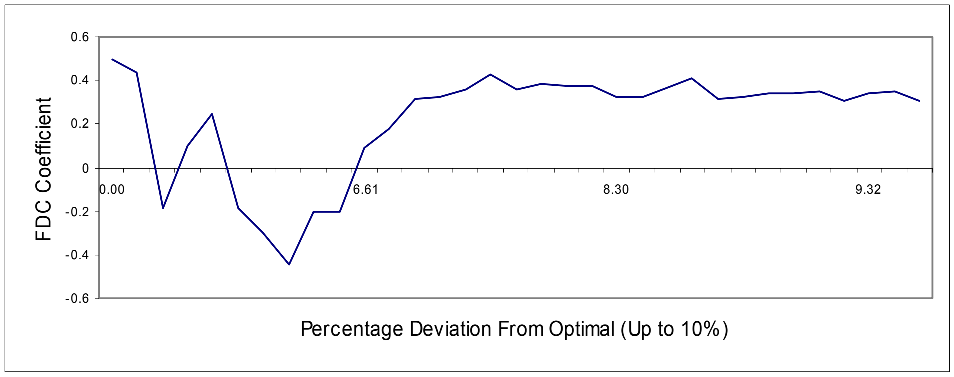

Figure 32 : FDC Coefficient vs. 10% Deviation from Global Optimum for qab030

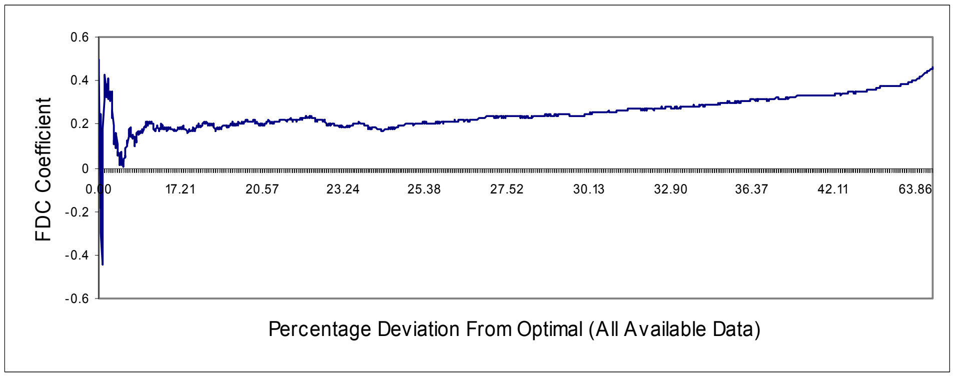

Figure 33 : FDC Coefficient vs. 100% Deviation from Global Optimum for qab030

Those figures show clearly that taking only the close-in-fitness local minima is not sufficient to calculate the FDC coefficient, and that a large number of local minima, close in fitness and far, are needed to get a clearer view about FDC coefficient.

4.2.2 FDC vs. Distance from Global Optimum

Variations of the value of the FDC coefficient ρ_{ls} also appear when changing the number of local minima taken into consideration and their distance from the global optimum. Figure 34 to Figure 39 show different values of FDC coefficient versus the distance to global optimum for the six CBA problem instances. The FDC coefficient ρ_{ls} is calculated after sorting the local minima with respect to their distance from optimal and at each point, ρ_{ls} is calculated from the nearest local optimum to the current.

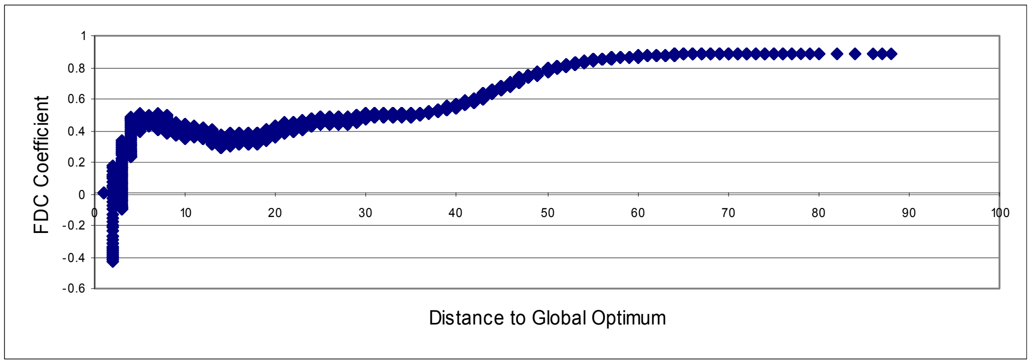

Figure 34 : FDC Coefficient vs. Distance to Global Optimum of raa180

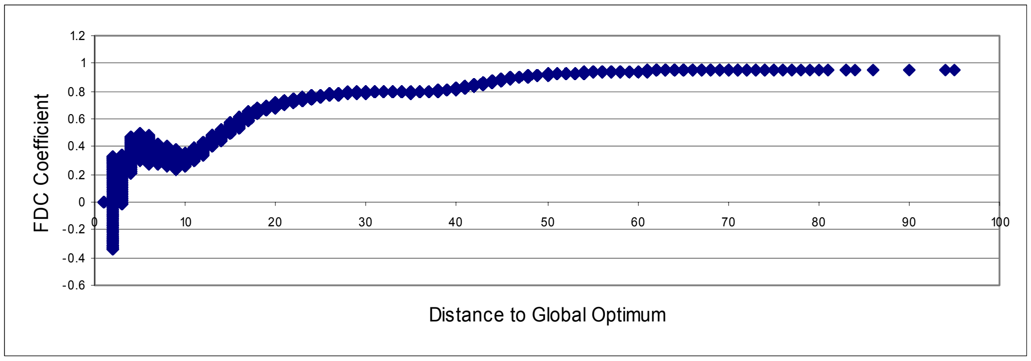

Figure 35 : FDC Coefficient vs. Distance to Global Optimum of caa200

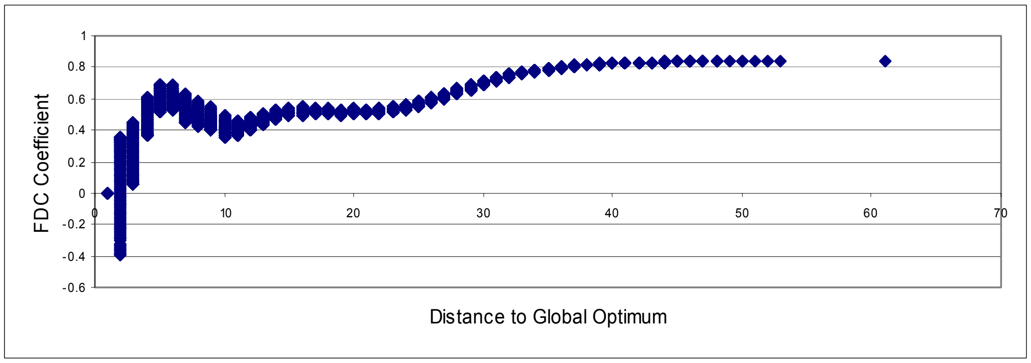

Figure 36 : FDC Coefficient vs. Distance to Global Optimum of oaa110

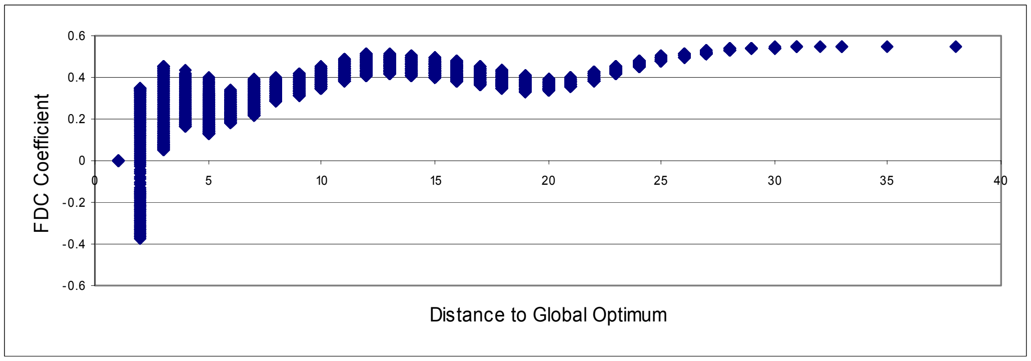

Figure 37 : FDC Coefficient vs. Distance to Global Optimum of lab070

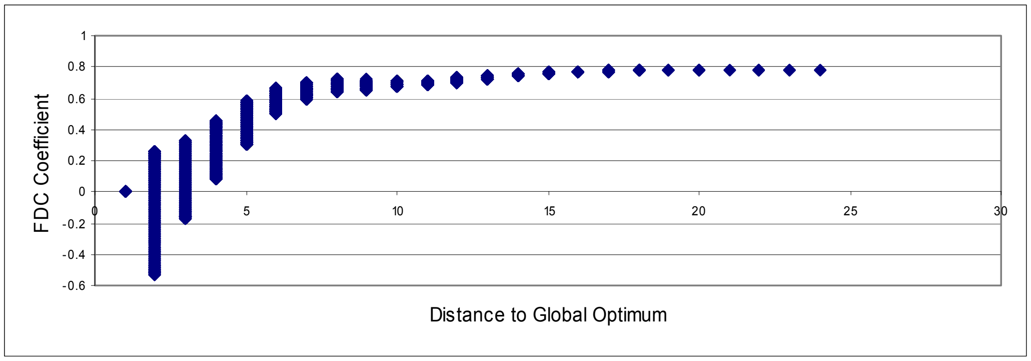

Figure 38 : FDC Coefficient vs. Distance to Global Optimum of rab050

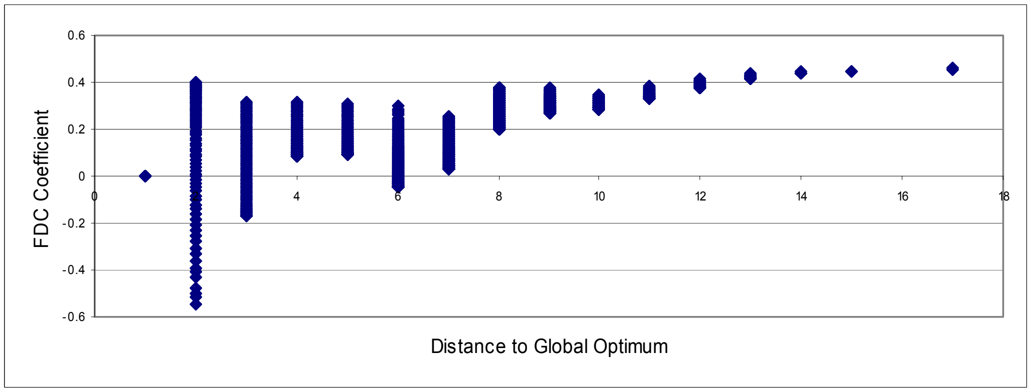

Figure 39 : FDC Coefficient vs. Distance to Global Optimum of qab030

Taking only the close in distance local minima is very misleading to the FDC coefficient, where its values range from being negative (misleading problems) to positive (straightforward). Also in the cases of the big CBA instances used, going from close to far local optima, the FDC starts with different values for the same distance, then it moves higher to become positive, then there is a drop in the value of FDC coefficient before it starts to move up again and saturate at a certain point. This looks like an interesting observation if it is common to all CBA instances and even other NP-hard problems.

4.3 Heritage Factor and GESA Parameters

In our analysis of using the heritage factor, different values of the heritage factor were examined, from 0.00 (pure GESA) to 0.50 with increments of 0.05 with 5 chains 20 children, 10 chains 20 children and 20 chains 20 children for each CBA problem instance. For each combination, the experiments were done 100 times and the following points were monitored:

- The average final cost reached
- The best-cost-loop-index which is the first time the algorithm reached the best cost that it saturated upon later
- The average of total number of iterations gone through before saturation
- The number of times all but one chain of the GESA chains died (had acceptance number = 0 and hence produced no children).
- The average duration of time taken by the runs

Table 7 to Table 15 show these values for the different combinations of GESA chains and children for the three big CBA Instances. From these results, it is clear that the average final cost gets better when using the heritage factor. But this is on the account of the total number of iterations which increases, causing the duration of execution to increase also. The number of chains that die except one does not change when using 10 and 20 chains, but with 5 chains, the program terminates with less number of times all chains but one have died. This means that the generation stays alive when using the heritage factor more than without using it, which allows for wider search.

There is no clear cut of a specific value that would be best when using the heritage factor. For example, when solving raa180 using GESA(5,20), the best average final cost was at heritage factor of 0.3, but with caa200, it was as heritage factor 0.45. The fine tuning of the heritage factor value should be done empirically.

Table 7 : Heritage Factor with GESA (5 Chains, 20 Children) for raa180

H Factor	*Avg. Final Cost*	*Avg. Best Cost Loop Index*	*Avg.Num Iterations*	*Num of times All Died But One*	*Avg. Duration (sec.)*
0.00	11194.51	129.79	212.93	35	87.8386
0.05	11168.31	128.47	209.68	36	83.1401
0.10	11122.61	117.17	203.48	28	80.3088
0.15	11165.18	130.19	223.20	26	86.3826
0.20	11121.31	131.95	218.07	20	84.1813
0.25	11127.01	132.11	222.44	21	88.8886
0.30	11114.16	122.19	211.87	22	84.1574
0.35	11148.20	130.87	225.39	17	87.2165
0.40	11140.92	126.02	220.00	10	85.6996
0.45	11160.26	135.10	226.54	10	88.4893
0.50	11115.38	138.07	238.09	15	92.9840

Table 8 : Heritage Factor with GESA (10 Chains, 20 Children) for raa180

H Factor	*Avg. Final Cost*	*Avg. Best Cost Loop Index*	*Avg.Num Iterations*	*Num of times All Died But One*	*Avg. Duration (sec.)*
0.00	11067.23	130.82	221.96	0	182.6413
0.05	11009.36	135.49	216.96	0	175.1532
0.10	11072.25	131.20	230.81	0	188.7816
0.15	11012.56	130.41	225.90	0	183.9818
0.20	11052.91	134.08	230.75	0	185.5177
0.25	11019.26	131.08	227.75	0	194.5921
0.30	11038.77	128.32	224.38	0	187.7322
0.35	11045.52	126.08	240.32	0	196.0005
0.40	11004.10	133.36	238.91	0	195.6499
0.45	11023.34	129.14	240.19	0	192.2563
0.50	11012.24	140.02	244.62	0	195.7536

Table 9 : Heritage Factor with GESA (20Chains, 20 Children) for raa180

H Factor	*Avg. Final Cost*	*Avg. Best Cost Loop Index*	*Avg.Num Iterations*	*Num of times All Died But One*	*Avg. Duration (sec.)*
0.00	10958.11	148.54	315.80	0	541.4275
0.05	10959.31	142.17	329.56	0	591.3636
0.10	10936.82	148.73	331.53	0	594.3821
0.15	10942.46	141.85	322.05	0	573.6850
0.20	10940.51	156.07	337.98	0	575.8763
0.25	10951.82	149.02	341.59	0	612.4336
0.30	10941.68	154.71	352.80	0	639.4532
0.35	10937.60	158.99	340.94	0	576.9604
0.40	10938.31	141.91	338.08	0	570.2164
0.45	10939.78	146.52	356.00	0	628.2091
0.50	10943.65	145.42	352.76	0	594.3343

Table 10: Heritage Factor with GESA (5 Chains, 20 Children) for caa200

H Factor	*Avg. Final Cost*	*Avg. Best Cost Loop Index*	*Avg.Num Iterations*	*Num of times All Died But One*	*Avg. Duration (sec.)*
0.00	7775.49	116.75	170.00	32	68.4275
0.05	7768.05	126.95	162.84	31	64.1602
0.10	7760.88	119.46	176.25	35	68.1619
0.15	7743.82	130.35	183.94	21	71.0634
0.20	7712.07	126.87	191.04	21	73.4111
0.25	7716.97	135.92	198.18	18	74.8906
0.30	7714.22	136.11	191.89	13	73.1739
0.35	7723.17	136.64	189.81	9	74.6916
0.40	7725.72	134.76	198.23	18	76.9391
0.45	7707.67	129.26	184.54	7	71.5075
0.50	7729.46	140.34	189.83	11	73.5706

Table 11 : Heritage Factor with GESA (10 Chains, 20 Children) for caa200

H Factor	*Avg. Final Cost*	*Avg. Best Cost Loop Index*	*Avg.Num Iterations*	*Num of times All Died But One*	*Avg. Duration (sec.)*
0.00	7702.54	123.45	162.2	0	131.4663
0.05	7684.81	118.61	140.64	0	111.3153
0.10	7692.17	117.40	145.89	0	116.0855
0.15	7698.75	116.44	156.78	0	122.4769
0.20	7686.90	117.06	158.46	0	125.6899
0.25	7687.63	122.03	150.01	0	116.4753
0.30	7687.17	123.97	150.65	0	115.2789
0.35	7691.11	122.99	155.68	0	125.7589
0.40	7688.88	125.55	161.75	0	124.1577
0.45	7689.87	115.21	151.76	0	117.3011
0.50	7686.52	119.66	143.84	0	113.9658

Table 12: Heritage Factor with GESA (20 Chains, 20 Children) for caa200

H Factor	*Avg. Final Cost*	*Avg. Best Cost Loop Index*	*Avg.Num Iterations*	*Num of times All Died But One*	*Avg. Duration (sec.)*
0.00	7680.80	114.26	139.78	0	225.3505
0.05	7679.78	107.55	127.34	0	204.1992
0.10	7679.71	118.19	139.71	0	225.8760
0.15	7680.04	113.50	133.91	0	215.1480
0.20	7680.30	109.74	131.61	0	211.5904
0.25	7678.77	108.44	122.28	0	195.0951
0.30	7678.77	111.38	125.56	0	196.4883
0.35	7679.12	110.95	117.02	0	181.6149
0.40	7679.33	110.03	131.59	0	209.1756
0.45	7679.59	109.31	128.23	0	201.8456
0.50	7678.76	110.54	119.55	0	187.5176

Table 13: Heritage Factor with GESA (5 Chains, 20 Children) for oaa110

H Factor	*Avg. Final Cost*	*Avg. Best Cost Loop Index*	*Avg.Num Iterations*	*Num of times All Died But One*	*Avg. Duration (sec.)*
0.00	7030.46	101.4	160.62	18	66.0241
0.05	7013.94	99.47	166.00	11	68.6489
0.10	7031.45	98.04	175.56	13	70.3182
0.15	7048.72	113.26	180.82	9	71.2176
0.20	7040.67	106.42	180.46	7	71.9677
0.25	7029.70	113.7	190.64	10	75.6301
0.30	7027.88	108.15	170.41	8	69.2060
0.35	6972.83	113.26	180.86	3	71.6650
0.40	6995.47	114.19	182.54	2	74.2060
0.45	7001.88	111.14	175.64	6	67.5310
0.50	6964.86	117.01	184.01	4	73.3453

Table 14: Heritage Factor with GESA (10 Chains, 20 Children) for oaa110

H Factor	*Avg. Final Cost*	*Avg. Best Cost Loop Index*	*Avg.Num Iterations*	*Num of times All Died But One*	*Avg. Duration (sec.)*
0.00	6902.59	90.45	132.95	0	109.3919
0.05	6915.03	102.69	143.44	0	116.7513
0.10	6909.48	90.96	128.26	0	104.8354
0.15	6905.88	101.6	134.35	0	107.9916
0.20	6902.81	105.29	142.29	0	116.5781
0.25	6898.70	100.66	137.63	0	113.2255
0.30	6903.56	101.85	140.35	0	114.1443
0.35	6885.49	112.7	135.12	0	112.3661
0.40	6894.35	111.41	152.86	0	125.8467
0.45	6889.39	97.65	129.79	0	108.7101
0.50	6895.92	108.71	150.27	0	123.7344

Table 15: Heritage Factor with GESA (20 Chains, 20 Children) for oaa110

H Factor	*Avg. Final Cost*	*Avg. Best Cost Loop Index*	*Avg.Num Iterations*	*Num of times All Died But One*	*Avg. Duration (sec.)*
0.00	6878.54	93.01	126.25	0	206.6352
0.05	6857.05	84.24	91.87	0	152.1805
0.10	6857.75	94.82	109.50	0	185.5580
0.15	6863.62	91.89	114.42	0	192.1911
0.20	6859.93	90.47	104.52	0	178.9680
0.25	6859.43	115.38	121.55	0	210.7190
0.30	6857.75	105.09	123.48	0	207.2115
0.35	6863.49	100.48	109.79	0	179.8755
0.40	6862.37	101.98	113.77	0	190.1550
0.45	6857.08	97.82	108.12	0	181.5467
0.50	6856.00	95.22	95.22	0	159.1565

To test the effect of varying the GESA parameters, the RLD analysis is done to the three groups of (5 chains 20 children), (10 chains 20 children) and (20 chains 20 children). The number of runs for each group was 100 runs, without applying the heritage factor, and with saturation 50. To make a fair comparison, the number of runs for the group (10 chains, 20 children) was multiplied by 2 and the number of runs of (20 chains, 20 children) was multiplied by 4, to make them equivalent to the run length of (5 chains, 20 children). Table 16 shows the maximum number of iterations to reach the global optimum using each group for the six CBA instances; along with how many times the global optimum was reached.

Table 16: RLD Analysis for GESA with different number of chains

	GESA (5, 20)		*GESA (10, 20)*		*GESA (20, 20)*	
Instance	*Max # Iterations*	*Reached Global*	*Max # Iterations * 2*	*Reached Global*	*Max # Iterations * 4*	*Reached Global*
raa180.cba	7109	4	7033 * 2	7	5887 * 4	16
caa200.cba	1623	46	1237 * 2	68	542 * 4	88
oaa110.cba	1336	45	1217 * 2	70	966 * 4	89
lab070.cba	512	77	373 * 2	86	240 * 4	95
rab050.cba	52	100	37 * 2	100	29 * 4	100
qab030.cba	34	100	19 * 2	100	14 * 4	100

Figure 40 to Figure 45 show the RLD graphs for the six instances. For the smallest two instances qab030 and rab050, 5 chains 20 children is the best combinations especially that the global optimum is always found, which indicates that there is no need for the extra complexity of adding more chains. For the other problems, it seems also that 5 chains 20 children dominate most of the time, with some exceptions where 10 chains 20 children perform better at parts of raa180 and oaa110. In general, the added complexity of more chains is not needed unless the problem is harder for less number of chains.

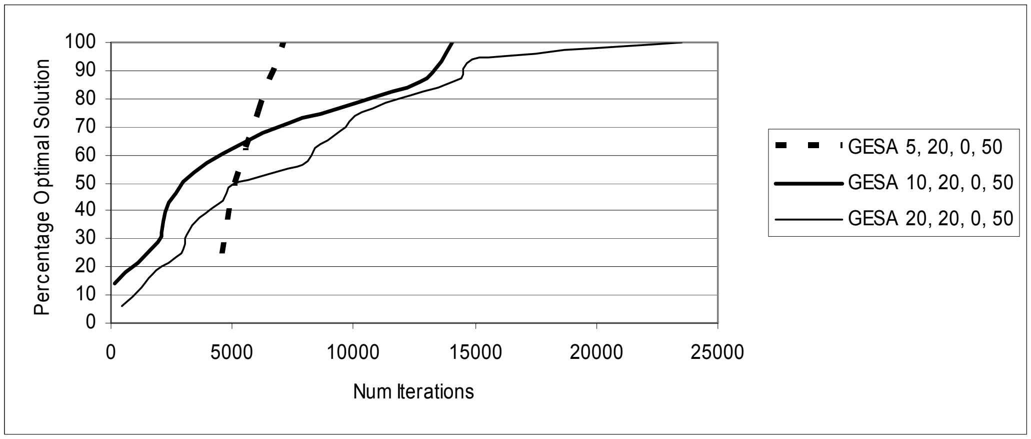

Figure 40: RLD Analysis for GESA (5,20), (10,20), (20,20) for raa180

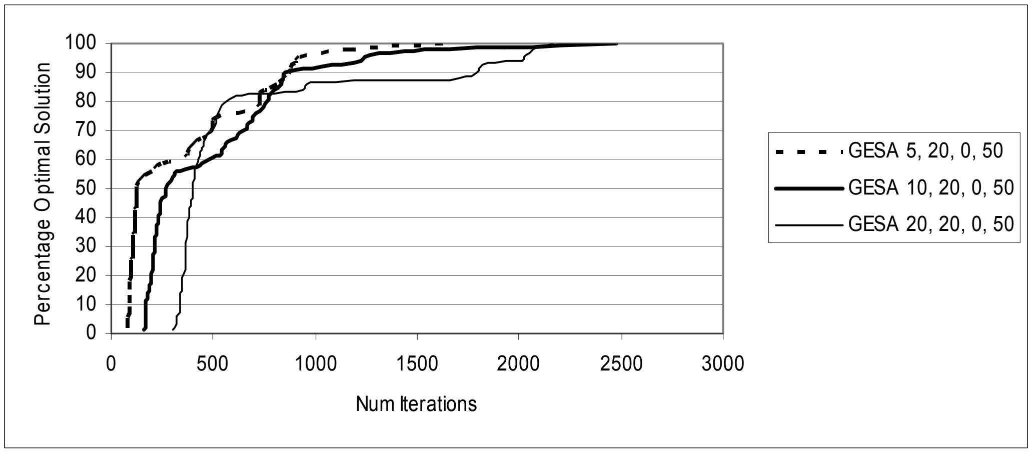

Figure 41: RLD Analysis for GESA (5,20), (10,20), (20,20) for caa200

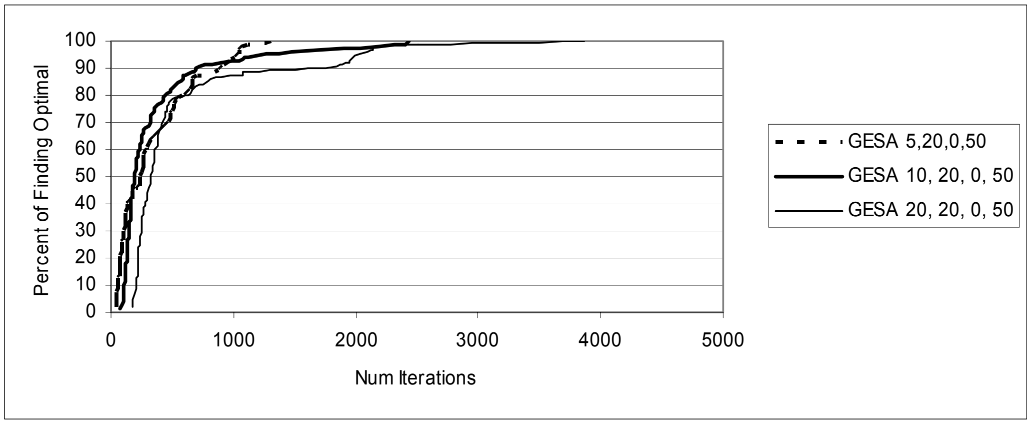

Figure 42: RLD Analysis for GESA (5,20), (10,20), (20,20) for oaa110

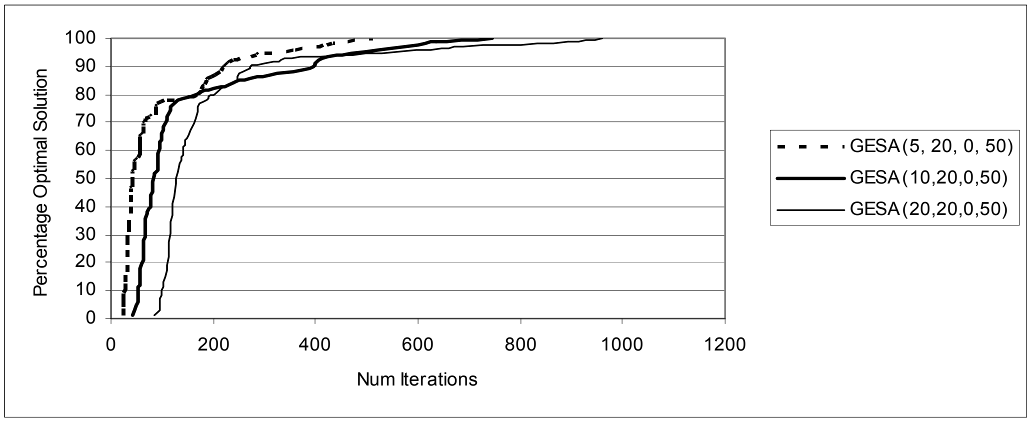

Figure 43: RLD Analysis for GESA (5,20), (10,20), (20,20) for lab070

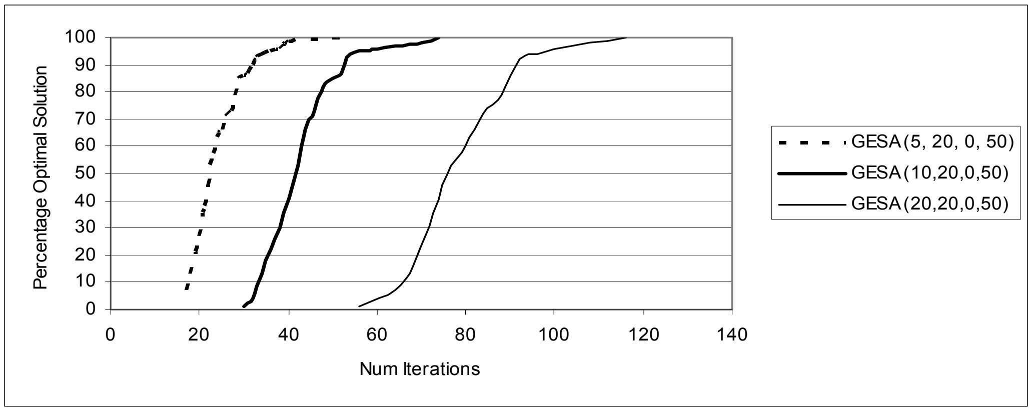

Figure 44: RLD Analysis for GESA (5,20), (10,20), (20,20) for rab050

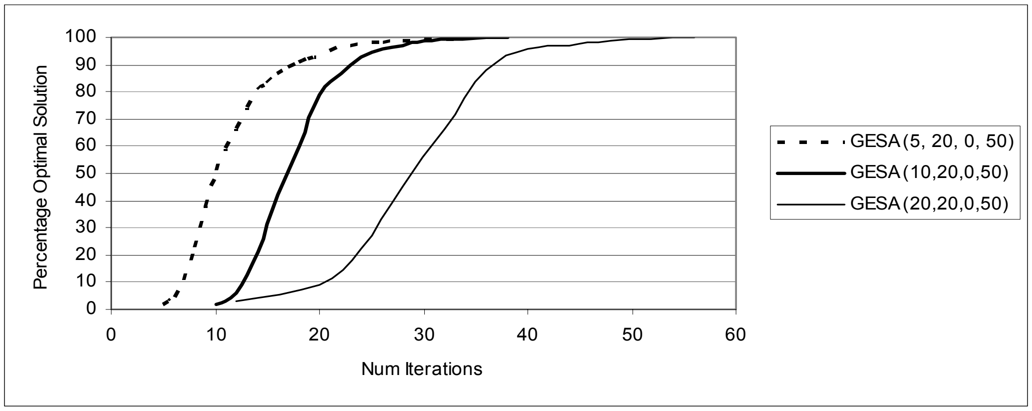

Figure 45: RLD Analysis for GESA (5,20), (10,20), (20,20) for qab030

Chapter 5 CONCLUSION AND FUTURE WORKS

5.1 Conclusion

In this research, we have applied population oriented and sequential chained simulated annealing in the form of Guided Evolutionary Simulated Annealing and Repeated Simulated Annealing to Cost-Based Abduction problems. A comparison was made between the two approaches using run-length analysis. The experiments' results showed GESA to outperform pure RSA in most of the instances used in testing. Using the Heritage Factor even improved finding the best-cost-loop-index but made the search saturate after more number of iterations and took longer time, as it kept the generations alive for a longer duration. Whether or not to use the heritage factor and the best value for it to be used is problem dependent and can only be determined empirically.

The Fitness-Distance analysis of CBA problems is done here showing that the instances used are considered straightforward for genetic algorithms. It was also shown that the Fitness-Distance Correlation coefficient strongly depends on the local minima taken into account, and that a large number with a large variation in distance and fitness of local minima has to be accounted for to some extent, as the very far local minima with very high fitness value could affect the FDC coefficient although they might be considered irrelevant.

5.2 Future Work

As a start, a benchmark of CBA problem instances has to be established and classified, to make it easier for search algorithms to be evaluated using CBA problems.

All the test cases used in this work have only one global optimum. Doing the analysis to other problems that have more than one global optimum is an interesting point that was not accounted for in this thesis. It could be more investigated as one of the keys to CBA problem difficulty, as it might affect the FDC coefficient value and the fitness-distance plots of CBA problems. It might also make it harder or easier for GESA to solve a multi-global optima problem.

The GESA was compared to RSA in this thesis. It could be also compared to evolutionary programming and genetic algorithms to see its performance and effectiveness with CBA problems.

For the FDC analysis, more work can be done to determine how many local minima should be taken into account to calculate the FDC and to what percentage of deviation from the global optimum. Setting these standards might give more precise values for the FDC coefficient which give better understanding for problem difficulty.

More emphasis can be given to the Heritage Factor to determine the best ways of assigning a value to it and using it in other search problems.

REFERENCES

[Abd01] A. M. Abdelbar. "Heritage Factors: Extending Guided Evolutionary Simulated Annealing" *Proceedings IEEE International Joint Conference on Neural Networks,* vol.4, pp. 2568-2573, 2001

[Abd04] A. M. Abdelbar. "Approximating Cost-Based Abduction is NP-Hard" *Artificial Intelligence, to appear,* 2004

[AA03] A. M. Abdelbar and H. Amer. "Applying Guided Evolutionary Simulated Annealing to Cost-Based Abduction" *Proceedings IEEE International Joint Conference on Neural Networks,* vol 3, pp. 2428-2431, 2003

[AGA04] A. M Abdelbar, S. H. Gheita and H. Amer. "Exploring the Fitness Landscape and the Run-Time Behavior of an Iterated Local Search Algorithm for Cost-Based Abduction" *under review*, 2004

[Alt97] L. Altenberg. "Fitness Distance Correlation Analysis: An Instructive Counterexample" *Proceedings of the 7th Int. Conference on Genetic Algorithms (ICGA97),* pp. 57-64, San Francisco, CA. Morgan Kaufmann. 1997

[BBM93] D. Beasley, D. R. Bull and R. R. Martin. "An Overview of Genetic Algorithms: Part 1, Fundamentals" *University Computing,* vol. 15, no. 2, pp. 58-69, 1993

[BSW02] H.-G. Beyer, A.-P. Schwefel, and I. Wegener. "How to Analyse Evolutionary Algorithms" *Theoretical Computer Science*, vol. 287, no. 1, pp. 101-130, Aug 2002

[CM86] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence,* pp. 21, 453-457. Addison-Wesley 1986

[CS94] E. Charniak and S. E. Shimony. "Cost-Based Abduction and MAP Explanation" *Artificial Intelligence,* vol. 66, pp. 345-374, 1994

[COC98] H. J. Cho, S. Y. Oh, and D. H. Choi. "Population-Oriented Simulated Annealing Technique Based on Local Temperature Concept" *Electronics Letters,* vol. 34, no. 3, pp. 312-313, 1998

[CC02] P.J. Chuang and C.W.Cheng. "On File and Task Placements and Dynamic Load Balancing in Distributed Systems" *Tamkang Journal of Science and Engineering*, vol. 5, no. 4, pp. 241-252, 2002

[CLR90] T. H. Cormen, C. E. Leiserson and R. L. Rivest. *Introduction to Algorithms,* MIT Press, McGraw-Hill, NY, 1990

[DZ+01] D. Dean, P. Zhang, A. Metzger, C. Sibata, and R. J. Maciunas "Medial Axis Seeding of a Guided Evolutionary Simulated Annealing (GESA) Algorithm for Automated Gamma Knife Radiosurgery Treatment Planning" *Proceedings of the 4th Int. Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, Springer-Verlag. pp. 441-448, 2001

[EM94] H. Esbensen, and P. Mazumder. " SAGA: A Unification of the Genetic Algorithm with Simulated Annealing and Its Application to Macro-Cell Placement" *Proceedings Seventh International Conference on VLSI Design,* pp. 211-214, 1994

[FSL02] M. Finger, T. Stützle and H. Lourenço. "Exploiting Fitness Distance Correlation of Set Covering Problems" *Applications of Evolutionary Computing,* S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. R. Raidl (eds.), Springer- LNCS, pp. 61-71, 2002

[Gol90] D. E. Goldberg. "A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-Oriented Simulated Annealing" *Complex Systems,* pp. 445-460, 1990

[GZ01] G. W. Greenwood and Q. J. Zhu. "Convergence in Evolutionary Programs with Self-Adaptation" *Evolutionary Computation*, vol. 9, no. 2, pp.147-157, 2001

[Hef01] M. S. Hefny. "A New Linear-Programming Based Admissible Heuristic for Cost-Based Abduction" MSC Thesis, Department of Computer Science, The American University in Cairo, May 2001

[HS+88] J. R. Hobbs, M. E. Stickel, P. Martin and D. Edwards. "Interpretation as Abduction" *Proceedings 26th Annual Meeting of the Association for Computational Linguistics*, Buffalo, NY, pp. 95-103, 1988

[IM98] M. Ishizuka and Y. Matsuo. "SL Method for Computing a Near-Optimal Solution Using Linear and Non-Linear Programming in Cost-Based Hypothetical Reasoning" *Proceedings of Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, Singapore, 1998

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. "Data Clustering: A Review" *ACM Computing Surveys (CSUR),* vol. 31, no. 3, pp. 264-323, Sept. 1999

[JF95] T. Jones and S. Forrest. "Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms" L. Eshelman, editor, *Proceedings of the 6th Int. Conference on Genetic Algorithms,* pp. 184-192, San Francisco, CA. Morgan Kaufman, 1995

[KO+99] S. Kato, S. Oono, H. Seki and H. Itoh. "Cost-Based Abduction Using Binary Decision Diagrams" *Proceedings of Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE),* Cairo, 1999

[KSI96] S. Kato, H. Seki, and H. Itoh. "Parallel Cost-Based Abductive Reasoning for Distributed Memory Systems" *Proceedings of Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, Cairns 1996

[KGV83] S. Kitpatrick, C. D. Gellat, and M. P. Vecchi. "Optimization by Simulated Annealing" *Science* 220(4598), pp.671- 680, 1983

[LKH91] F. T. Lin, C. Y. Kao, and C. C. Hsu. "Incorporating Genetic Algorithms into Simulated Annealing" *Proceedings of the Fourth International Symposium on Artificial Intelligence,* pp 290-297, 1991

[LKH93] F. T. Lin, C. Y. Kao and C. C. Hsu. "Applying the Genetic Approach to Simulated Annealing in Solving Some NP-Hard Problems" *IEEE Transactions on Systems, Man and Cybernetics,* vol. 23, no. 6, pp. 1752-1767, 1993

[MG95] S. W. Mahfoud and D. E. Goldberg. "Parallel Recombinative Simulated Annealing: A Genetic Algorithm" *Parallel Computing,* vol. 21, pp. 1-28, 1995

[OI97] Y. Ohsawa and M. Ishizuka. "Networked Bubble Propagation: A Polynomial-Time Hypothetical Reasoning Method for Computing Near-Optimal Solutions" *Artificial Intelligence,* vol. 91, pp. 131-154, 1997

[PR90] Y. Peng and J. A. Reggia. *Abductive Inference Models for Diagnostic Problem-Solving,* pp. 1-9. Springer-Verlag, NY, 1990

[PR+03] A. P. Plumb, R. C. Rowe, P. York, and C. Doherty. "Effect of Varying Optimization Parameters on Optimization by Guided Evolutionary Simulated Annealing (GESA) Using a Tablet Film Coat as an Example Formulation" *European Journal of Pharmaceutical Science,* vol. 18, issue 3-4, pp. 259-266, March 2003

[RK91] E. Rich and K. Knight. *Artificial Intelligence*, pp.70-72. McGraw-Hill, Inc.2nd ed. 1991

[San94] E. Santos Jr. "A Linear Constraint Satisfaction Approach to Cost-Based Abduction" *Artificial Intelligence,* vol.65, no.1, pp. 1-28, 1994

[SS96] E. Santos Jr., and E. S. Santos. "Polynomial Solvability of Cost-Based Abduction" *Artificial Intelligence,* vol. 86, pp. 157-170, 1996

[SS03] T. Schiavinotto and T. Stützle. "Search Space Analysis of the Linear Ordering Problem" In S. Cagnoni, J. J. Romero Cardalda, D. W. Corne, J. Gottlieb, A. Guillot, E. Hart, C. G. Johnson, E. Marchiori, J.-A. Meyera, M. Middendorf and G. R. Raidl, editors. *Applications of Evolutionary Computing,* vol 2611 in Lecture Notes in Computer Science, Springer Verlag, pp. 322-333, 2003

[SH00] T. Stützle and H. H. Hoos. "MAX-MIN Ant System" *Journal of Future Generation Computer Systems,* 16: pp. 889-914, 2000

[VT03] L. Vanneschi and M. Tomassini. "Pros and Cons of Fitness Distance Correlation in Genetic Programming" *Proceedings of the Bird of a Feather Workshop, Genetic and Evolutionary Computation Conference (GECCO 2003),* A. Barry, Editor. pp 284 – 287, 2003

[YP95] P. Yip, and Y.-H. Pao. “Combinatorial Optimization with Use of Guided Evolutionary Simulated Annealing” *IEEE Transactions on Neural Networks,* vol. 6, no. 2, pp. 290-295, 1995

APPENDIX A: CBA FILE FORMAT

http://www.cs.aucegypt.edu/~abdelbar/CBAlib/format.txt

1. The first line of the file always contains the number 1.
2. Throughout the file, any blank lines are ignored.
3. This is followed by a line with a single integer *H*, where *H* is the number of hypotheses.
4. This is followed by *H* lines, where line i contains a single real number equal to the assumability cost of hypothesis number i. If a hypothesis's assumability cost is infinity, then the line contains the number zero.
5. This is followed by a line containing the integer *R*, where *R* is the number of rules.
6. This is followed by *R* lines, where each line represents a rule. Each of these lines contains a number of integers with the following meanings: the first integer is the number of antecedents *u* for this rule, this is followed by *u* integers equal to the hypothesis numbers of the *u* antecedents for this rule (hypothesis numbers start at 1 not at 0), this is followed by the hypothesis number of the consequent of this rule.
7. This is followed by a line with a single integer equal to the hypothesis number of the goal hypothesis. (This line is actually redundant because the goal hypothesis is always equal to the maximum-numbered hypothesis, which is equal to the number of hypotheses).

APPENDIX B: QAB030.CBA

```
1

100

 190.00
0
0
 836.00
0
0
 662.00
0
0
0
0
0
 848.00
0
 747.00
0
0
0
0
0
0
 331.00
0
0
0
0
0
0
0
```

488.00
917.00
0
0
782.00
0
0
0
725.00
0
0
857.00
468.00
653.00
492.00
0
0
0
272.00
0
0
0
0
0
0
0
0
890.00
0
0
0
739.00
477.00
0
0

454.00

0

237.00

190.00

0

0

0

483.00

0

0

0

538.00

0

0

386.00

263.00

537.00

0

0

0

0

0

0

983.00

466.00

0

0

396.00

0

0

0

0

400.00

0

0

```
0


300
3   11   7   17   21
5   8   9   1   41   4   45
4   54   24   53   44   83
2   20   43   46
1   2   3
7   23   17   30   48   6   32   22   54
6   11   5   6   3   8   15   17
4   77   63   61   60   94
5   70   52   39   73   44   87
8   15   29   24   26   31   7   16   19   32
3   38   71   7   84
7   12   32   6   7   23   27   4   33
5   40   3   23   54   73   75
3   2   1   16   18
7   9   4   10   13   15   7   11   17
4   46   15   29   32   78
5   36   30   14   7   18   51
8   1   14   11   7   4   3   13   8   19
8   97   44   58   90   67   68   76   80   99
7   26   6   41   19   1   52   2   60
1   1   6
8   37   38   79   57   33   20   93   54   94
1   12   25
4   5   11   12   3   14
2   63   43   82
2   35   18   54
1   24   58
4   21   7   22   26   27
5   4   63   43   50   19   93
8   82   26   22   69   45   59   81   13   85
6   84   15   82   45   53   67   95
```

```
3   4   5   6   8
5   50   11   62   56   1   66
3   29   39   31   77
2   7   5   19
4   50   5   43   31   51
5   43   15   31   42   34   59
1   1   2
5   33   17   23   51   45   64
1   29   52
2   4   9   12
2   11   33   64
2   45   35   51
8   12   13   25   20   19   2   16   22   27
5   16   6   14   11   10   19
5   51   75   38   91   23   95
7   28   78   38   22   6   41   8   96
5   38   18   57   47   8   59
7   46   21   16   30   42   6   14   53
4   60   85   2   67   91
6   41   30   7   17   19   6   49
3   45   30   56   75
2   1   22   23
8   8   19   25   20   15   2   27   4   37
3   31   52   53   54
2   4   3   16
6   14   6   7   10   8   5   19
5   1   5   4   8   7   10
1   49   52
3   23   16   21   35
2   53   54   63
5   50   30   69   83   24   84
2   8   14   16
8   76   15   1   53   24   41   62   8   83
7   6   1   7   5   4   2   3   8
6   12   11   5   16   8   6   18
```

```
5   9   17   23   5   7   27
2   8   2   21
8   12   50   17   25   37   44   66   63   69
8   36   24   54   10   11   28   5   52   78
4   14   45   22   47   77
7   20   17   8   2   14   5   21   25
1   12   46
8   80   28   30   8   48   89   71   21   94
4   34   2   72   29   75
5   58   37   42   47   78   95
4   45   19   4   63   70
1   1   2
1   2   3
7   30   58   43   32   12   44   4   75
4   35   26   4   23   49
4   44   79   19   54   86
6   13   9   7   17   1   15   19
5   7   11   9   13   10   16
6   23   22   9   11   4   26   27
5   56   68   12   38   63   74
4   11   8   7   14   26
1   28   64
6   46   15   67   35   66   56   75
6   3   15   6   14   9   10   16
6   11   13   5   6   3   8   14
3   28   62   10   69
8   11   45   48   51   54   19   42   53   60
8   10   18   14   23   6   1   12   9   24
6   10   5   8   11   9   2   12
5   72   28   57   44   65   84
8   2   12   1   15   16   11   24   14   26
1   1   45
3   2   4   3   11
3   1   3   2   6
1   7   9
```

```
8   21   18   41   44   30   3    19   48   54
4   3    18   10   23   45
5   57   47   23   49   22   77
6   14   71   35   52   22   74   77
7   4    6    7    9    2    3    1    11
5   1    24   6    35   38   73
4   39   14   40   31   50
5   7    37   36   28   23   40
6   7    5    1    2    3    4    9
6   46   40   29   42   49   4    63
6   27   50   41   13   51   46   54
2   32   4    40
1   1    2
4   22   41   51   63   71
4   6    1    5    4    8
1   14   33
3   5    2    4    6
6   46   1    75   26   48   50   85
5   7    16   2    12   4    19
7   32   47   9    40   13   20   11   50
2   2    1    3
4   15   43   18   13   75
8   16   2    8    9    17   5    22   25   28
1   1    6
6   50   5    10   39   16   34   60
2   9    6    29
3   36   30   44   64
8   34   8    43   32   27   44   17   47   51
4   76   26   87   20   96
8   33   28   61   37   39   85   48   77   95
2   48   12   56
7   49   10   92   46   79   82   60   93
3   33   20   28   49
8   1    24   20   2    11   36   33   14   53
4   10   20   7    14   21
```

```
6   50   11   34   5    64   19   82
3   11   13   83   87
1   38   52
2   54   16   98
5   67   59   4    32   69   77
8   69   48   47   9    25   55   35   66   73
8   67   20   80   30   1    19   10   73   87
1   49   59
4   19   8    15   23   32
1   15   90
6   26   59   79   32   57   3    96
1   1    9
2   28   2    56
4   16   9    12   18   27
1   47   83
6   9    26   7    43   39   4    64
1   66   78
6   5    8    3    13   12   11   14
8   57   48   72   70   67   64   46   42   77
1   13   63
6   16   66   55   23   33   28   77
7   44   28   30   27   16   7    15   49
4   82   97   53   37   99
5   60   63   44   12   50   77
7   8    12   11   2    13   14   10   18
4   8    7    9    5    18
6   32   16   37   21   20   22   49
4   15   1    41   46   47
5   8    3    2    7    13   21
1   6    55
5   64   7    58   24   14   69
3   15   2    5    27
5   33   90   76   29   35   94
5   6    3    7    1    8    9
8   22   56   13   28   53   30   35   41   64
```

```
8   33   24   34   9    23   44   4    46   47
1   1    2
5   21   7    6    39   27   45
4   19   11   6    22   49
3   15   9    12   26
3   3    20   9    28
3   14   2    9    35
8   41   80   62   54   51   72   35   79   83
7   23   28   48   39   58   61   24   63
4   15   22   26   9    35
5   5    1    2    3    4    6
7   34   31   46   7    15   12   37   52
3   40   46   26   56
8   25   28   14   23   20   15   5    2    32
8   33   36   83   31   12   81   56   14   85
7   22   35   68   39   45   5    43   69
3   17   3    34   45
1   54   87
8   53   40   28   50   13   11   61   44   66
5   8    9    4    22   5    23
8   55   10   25   39   34   7    13   43   66
1   7    8
1   1    3
3   5    1    4    9
7   25   40   43   1    31   34   4    51
7   3    12   11   10   4    2    5    14
6   9    13   6    1    3    7    14
2   78   26   91
2   21   17   23
6   9    2    10   23   19   18   33
3   23   17   1    29
1   26   53
2   9    5    16
1   1    2
3   77   70   42   86
```

```
7  6  3  7  13  9  12  18  21
1  4  32
3  19  7  15  39
5  25  20  26  38  23  46
7  19  4  51  42  37  20  33  54
2  30  16  58
5  13  5  7  9  11  18
2  8  2  11
8  14  13  20  25  22  19  36  8  51
7  59  23  26  15  13  48  7  60
8  5  22  8  17  57  48  30  58  64
6  22  2  19  5  13  16  26
5  17  81  26  8  3  86
1  2  5
1  6  40
2  9  2  14
7  13  24  37  25  8  7  11  59
7  46  20  22  61  73  36  34  75
1  8  32
8  5  17  12  6  1  10  15  14  19
5  23  5  39  42  40  71
4  63  61  60  29  86
3  18  16  3  49
1  79  95
4  1  34  23  2  35
5  9  19  13  24  18  49
8  60  45  26  63  47  5  23  65  66
7  14  34  29  30  27  11  19  36
8  15  7  2  41  27  16  46  33  56
6  20  29  54  61  3  66  70
5  49  29  3  40  4  54
4  10  1  7  4  11
2  73  23  86
3  15  31  18  55
5  23  29  67  69  37  77
```

```
5   21   8    17   16   3    46
3   85   16   73   93
2   5    7    12
3   27   11   2    95
4   12   2    25   30   50
3   21   9    14   36
4   41   42   4    23   94
7   1    13   31   46   2    9    24   52
4   88   61   94   11   95
2   5    4    6
6   28   73   53   36   34   40   84
6   4    10   26   36   41   52   53
3   13   14   8    18
4   31   54   8    23   83
2   29   17   70
3   5    3    4    6
1   4    9
5   19   14   6    12   2    20
3   50   47   74   82
7   3    14   44   28   19   33   20   45
1   1    3
3   31   20   27   45
2   37   5    83
8   9    16   4    18   12   2    14   13   19
1   10   71
6   13   17   7    18   5    10   19
7   12   16   2    10   3    1    4    17
7   11   14   10   31   29   36   2    40
1   2    3
5   4    34   32   2    37   47
1   6    9
5   58   64   51   13   70   73
4   55   36   53   34   73
1   36   70
8   44   11   21   7    34   20   45   2    51
```

```
6    27    36    18    15    22    34    56
4    6    65    54    46    66
4    16    24    4    5    25
2    26    15    55
2    5    17    28
8    54    19    20    79    74    58    60    38    84
6    25    13    6    14    17    3    28
8    10    3    5    6    4    2    12    13    17
3    42    27    31    47
5    18    13    85    73    50    91
2    2    1    3
3    6    7    5    12
7    11    7    9    8    5    10    2    12
6    51    59    36    54    7    30    87
4    35    75    3    31    78
4    29    4    6    55    63
8    10    23    59    18    20    56    19    12    66
2    31    63    95
8    7    37    33    9    31    44    85    20    87
4    27    30    47    56    60
2    15    30    55
3    45    49    20    71
2    1    2    3
11    19    20    36    49    50    52    54    56    77    93    94    100
100
```

APPENDIX C: PUBLISHED PAPER