May 2014

Network Theoretic Analyses and Enhancements of Evolutionary Algorithms

Karthik Kuber
Syracuse University

ABSTRACT

Information in evolutionary algorithms is available at multiple levels; however most analyses focus on the individual level. This dissertation extracts useful information from networks and communities formed by examining interrelationships between individuals in the populations as they change with time.

Network theoretic analyses are extremely useful in multiple fields and applications, e.g., biology (regulation of gene expression), organizational behavior (social networks), and intelligence data analysis (message traffic on the Internet). Evolving populations are represented as dynamic networks, and we show that changes in population characteristics can be recognized at the level of the networks representing successive generations, with implications for possible improvements in the evolutionary algorithm, e.g., in deciding when a population is prematurely converging, and when a reinitialization of the population may be beneficial to avoid computational effort, or to improve the probability of finding better points to examine.

In this dissertation, we show that network theoretic analyses can be applied to study, analyze and improve the performance of evolutionary algorithms. We propose various approaches to study the dynamic behavior of evolutionary algorithms, each highlighting the benefits of studying community-level behaviors, using graph properties and metrics to analyze evolutionary algorithms, identifying imminent convergence, and identifying time points at which it would help to reseed a fraction of the population.

Improvements to evolutionary algorithms result in alleviating the effects of premature convergence occurrences, and saving computational effort by reaching better solutions faster. We demonstrate that this new approach, using network science to analyze evolutionary algorithms, is advantageous for a variety of evolutionary algorithms, including Genetic Algorithms, Particle Swarm Optimization, and Learning Classifier Systems.

NETWORK THEORETIC ANALYSES AND ENHANCEMENTS OF EVOLUTIONARY ALGORITHMS

By

Karthik Kuber
B.E. Visveswaraiah Technological University, 2004
M.S. Syracuse University, 2009

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer and Information Science and Engineering

Syracuse University
May 2014

Advisors:
Prof. Chilukuri K. Mohan
Prof. Kishan G. Mehrotra

Acknowledgments

My student years at Syracuse University have been a great learning experience and an unforgettable phase of my life. I owe a debt of gratitude to a lot of people for making this happen.

I have been incredibly lucky to have had two very friendly advisors. Dr. Chilukuri Mohan has been advising me since my Masters thesis days early on in the program, and has patiently shaped the way I think about a research problem. Dr. Kishan Mehrotra has taught me to be precise about an idea and its expression, and has always reminded me to be thorough in exploring it. I thank them both for guiding me throughout with great knowledge and constant encouragement.

I would also like to thank Dr. Fred Schlereth, Dr. Raja Velu, and Dr. Stuart Card for the many discussions, helpful pointers, and pieces of advice over the years – be it for my dissertation research, or the many other projects that we have worked on.

I appreciate the time taken by Dr. Jae Oh, Dr. Qinru Qiu, and Dr. Chihwa Kao to examine my work, and I thank them for their thought-provoking questions and comments which helped improve this dissertation.

I consider myself very fortunate to have had more than my fair share of fantastic teachers right from my early school days in Mumbai, Chennai and Bangalore. I thank them not just for their dedication to teaching, but also for ingraining in me an interest for learning which has gone a long way in my decision to pursue higher education.

My years in Syracuse have been far more enriched with the many activity clubs that I was a part of. I'd like to thank the members of the Aikido, Badminton, Chess, Cricket,

Indian Classical Music, Scrabble, and Scuba diving clubs. And most of all, I thank the members of the Tae Kwon Do club, which I was a part of for over five years – my teacher Robert Britton and the others helped make the practice of the martial arts an integral part of my life.

I'd also like to acknowledge the support of my SENSE-Lab mates, peers, and all my friends in Syracuse, Bangalore, or elsewhere. They have been a part of some of my most memorable times.

Thanks and cheers to the city of Syracuse, especially to the university community. This city has been a very gracious host to my stay here as a graduate student.

Finally, I thank my parents, K. S. Krishnamurthy and R. Indira, and my sister, Tejashri Kuber, for everything that they have done for me. If not for their love, support, and encouragement, none of this could have happened.

To Wageesh Uncle, in place of the watch

TABLE OF CONTENTS

LIST OF TABLES

List of Figures

CHAPTER 1
INTRODUCTION

> *Science possesses, as yet, no word by which such a community of living beings may be designated; no word for a community where the sum of species and individuals, being mutually limited and selected under the average external conditions of life, have, by means of transmission, continued in possession of a certain definite territory. I propose the word 'Biocoenosis' for such a community.*
> *– Karl Möbius, 1877. The Oyster and Oyster Farming (English Translation of "Die Auster und die Austernwirtschaft")[49]*[1]

In the field of Artificial Intelligence or Computational Intelligence, Evolutionary Algorithms constitute a sub-field that focuses on heuristic optimization, search and learning. Metaheuristics are applied and instantiated to perform these tasks.

The field of *Evolutionary Algorithms (EAs)* acts as an umbrella covering many different classes of algorithms such as *Genetic Algorithms (GAs)*, *Particle Swarm Optimization (PSO)*, *Learning Classifier Systems (LCSs)*, *Genetic Programming (GP)*, *Evolutionary Programming (EP)*, and *Evolution Strategies (ESs)*, many of the concepts and algorithms are explained in [5]. In recent years, many difficult problems have been solved by these approaches. Although not identical, these

[1] This was the first known developed concept of community in ecological history, coined by Karl Möbius as "biocönose" (more often called by its German equivalents "Lebensgemeinde" and "Lebensgemeinschaf")[54] in the original 1877 version – although the more popular terms in English are "ecosystem" or "biotic community."

algorithms have many similarities – being population-based, driven by the principles of natural selection, and used for optimization problems.

The problems addressed by these algorithms are characterized by the existence of a large landscape of candidate solutions, and the need to effectively search this landscape for optimum solutions. Of course, such problems are computationally intractable, and brute force searching is not a feasible option. But we may pose a relevant question: Is there some information available in the landscape that we can use to find the next target points for the search?

Various algorithms address this question in different ways, determining different points of search for the population to explore or exploit. Evolutionary algorithms are characterized by the simultaneous exploration of multiple candidate solutions (or individuals) in a "population" that changes with time. Although different generations of individuals in such algorithms may lead to vastly different sets of individuals in the next generation (or iteration), one common feature is that all these algorithms learn from the landscape in every generation. This results in probabilistic moves towards a better solution, which may be slow or fast, in successive generations.

Evolutionary algorithms operate on sets or multi-sets of individuals (representations of candidate solutions), in which the relationships between individuals are usually ignored. Information contained in multiple individuals can be combined or modified in interesting ways, leading to the exploration of new candidate solutions, and selection strategies that favor better candidate solutions help guide the algorithm towards steady improvement in the best solutions[2] over time.

The techniques used in these algorithms are based on observations of nature. Biological processes have been a source of fascination for humans for a very long time. These processes are characterized by considerable complexity, be it evolution, or the search for food, or the process of mating to produce the next generation of offspring, or the struggle for limited resources. Evolution processes involve the struggle for survival from both an individual standpoint, as well as a species standpoint.

[2]In the case of an elitist algorithm, a steady improvement is noticed because the best individual is always retained. However, in a non-elitist algorithm, a solution could potentially get worse since the best individual also risks getting eliminated from the population, albeit with a very low probability.

The emergence of a population, and evaluation of individuals for their fitness, which in turn results in their respective probabilities of survival in the population, are ideas borrowed from Charles Darwin's theories of evolutionary biology [17]. Continuing with the theme that evolutionary computation borrows ideas from evolutionary biology, we proceed to another trait that living organisms exhibit in their ecosystems, viz., the formation of communities with some common characteristic(s) binding them together. Since in many biological ecosystems, the formation of communities leads to some very interesting phenomena, we explore the same in computational simulations of "intelligence". From a computer science/artificial intelligence perspective, viewing and understanding the behaviors of populations as networks can provide the means for depicting the phenomenon of community formation in ecology. This approach is the main focus of this dissertation.

1.1 Focus

In studies of evolutionary algorithms, relationships between individuals are usually ignored. Some efforts have focused on relationships between subsets of the population. Our focus is on examining relationships between individuals, asking what happens over time to the structural properties of networks representing evolving populations. Networks are important to study because of the following reasons: (i) Networks take into account relationships between individuals (via the existence of edges and their lengths), (ii) Networks are better suited to depict various other kinds of relationships as well, such as ancestral connections, and (iii) Networks provide useful models of relationships that biological species share.

We formalize and analyze network representations of well-known instances of evolutionary algorithms, viz., real-coded Genetic Algorithms [29], Particle Swarm Optimization [36], and Learning Classifier Systems [32] using nodes in the network to represent individuals in the population, applied to benchmark optimization problems studied by many researchers.

In most other network-theoretic applications where networks change over time, the nodes in a network are mostly the same from one instant to the next, although some edges (and a small

number of nodes) may appear or disappear. However, in applying this approach to evolutionary algorithms, we observe that individuals in an evolving population may not persist over time. Nodes in a network vanish in one generation, so that there is no obvious "continuity" between networks representing successive generations. Hence the key questions to be posed are at the level of the entire network (representing the entire population) or its subgraphs (representing sub-networks consisting of "related" individuals).

What kinds of relationships between individuals could be interesting in analyzing evolutionary algorithms? One possibility is based on ancestral relationships in the evolutionary process, based on the distance to the nearest common ancestor. For instance, an edge may be said to exist between nodes representing two individuals if they have at least one common parent. Another computationally simpler approach is based on proximity between individuals (in the problem-specific data space), measured using the Euclidean distance metric. Thus, an edge exists between two nodes if the distance between them is less than a threshold that may be chosen based on problem-specific characteristics or the properties of the current network.

This dissertation also addresses the problems of detecting convergence and initiating restarts in GAs, using an approach based on the thesis that observable phenomena in dynamical systems can be predicted based upon knowledge of an underlying structure which may be invisible by itself.

1.2 Outline

Chapter 2 presents some background material on EAs (GAs, PSOs, LCSs), and networks. We also present some related work that views the population as beyond just individuals.

Following this, we discuss the motivation for the use of networks in Chapter 3, define our network model, review some of the network parameters under study, and discuss parallels between a GA and a dynamic network. We also trace the evolution of these parameters over multiple generations. We discuss an inherent connection between the changing structure of the convex hull of the largest component, evolution of triangles amongst individuals, the identification of the space

where the optimum lies in a GA, and subsequently, its convergence.

Following this exploratory study, we present a few algorithms applied to real-valued Genetic Algorithms highlighting the potential uses of such analyses in Chapter 4. Specifically, we show a way to identify the region of interest by detecting convergence early, mainly by observing the structure of the convex hull of the largest component. We also demonstrate another utility of such metrics by "intelligently" identifying additional points in the search space to evaluate for better objective function value using the centroid of the largest component. This is analogous to multi-parent recombination in a niche, where the network structure is used to choose the parents. Early detection of convergence is used in another algorithm to enforce diversity and hence guarantee the exploration of other subspaces, which reduces the probability of premature convergence. We also extend our analysis to PSOs utilizing the common features between GAs and PSOs, and study early detection of convergence in a PSO.

We then begin our exploration of other forms of network construction. In Chapter 5, similar studies are carried out while obtaining the networks from another point of view, viz., ancestral connections in a GA, which provide information that is available within the evolving generations, but is often ignored in practice. In Chapter 6, we show that these network theoretic ideas extend readily to another form of EAs, viz., LCSs. Here, we observe many similarities in the evolution of network measures with the evolutions seen in a GA or a PSO, although there are many differences in the representation, structure and function of an LCS.

Finally, we summarize and discuss our observations in Chapter 7, and conclude and present future directions of research in Chapter 8.

CHAPTER 2
BACKGROUND: EVOLUTIONARY ALGORITHMS, NETWORKS, AND RELATED WORK

In this chapter, we describe some of the main characteristics of selected EAs. Following this, we also describe some of the essentials of network science, and further discuss how networks serve as a representative of interrelationships between individuals in the context of an EA.

2.1 Evolutionary Algorithms

Evolutionary Algorithms are inspired by biological evolution and are used to solve problems which are computationally hard. There are many algorithms with different characteristics, which still have broad similarities which classifies them as EAs. We discuss a select few (Genetic Algorithms, Particle Swarm Optimization, and Learning Classifier Systems) although there are many more. We have chosen to work with these three classes of EAs.

2.1.1 Genetic Algorithms

Genetic Algorithms [28][29][20][33][30] constitute a population-based class of heuristic algorithms that are used extensively for search problems. This class of algorithms includes genetic operators such as crossover, mutation, and parent selection based on fitness, all of which are ideas borrowed from evolutionary biology.

In order to use these ideas effectively in problem solving, it is necessary to translate a biological individual into a representation amenable to computer operations. This can be done in many forms, popular ones being using a bit string or a real-valued number. Another translation needed into the search space is the notion of fitness[1] which is usually chosen to be proportional to the value of the function being optimized.

The populations of individuals searching the landscape vary over time due to one generation giving birth to a new one using genetic operators. The entire set of individuals in a generation gets replaced by the next generation of offspring. However, the use of an elitist algorithm is popular, i.e., one or more of the best individuals are retained in the population by virtue of their high fitness. In the simulations carried out in this dissertation, we retain the best individual. This process is used to solve different kinds of search problems.

An overview of a GA is shown in Figure 2.1. We describe below some of the simulated biological processes (genetic operators) that are implemented in a GA.

Crossover

A crossover or recombination operator applied to two parents creates two new offspring individuals each having different portions of its parents' genes. In "one-point crossover", if two parents are represented as a chromosome each (i.e., a bit-string), and the point of crossover is determined to be at a particular point, the first child will have the genes (bits) of the first chromosome (parent)

[1]Historically, the term "fitness" has had many different definitions starting from Charles Darwin's work, *On the Origin of Species*[17], and Herbert Spencer's coining of the term "Survival of the fittest" [70] in his book *Principles of Biology*. A variety of different definitions have been proposed since then as comprehensively summarized and discussed in [6] and [55]. However, the broadly accepted meaning of the term is an organism's ability to survive and reproduce in its environment.

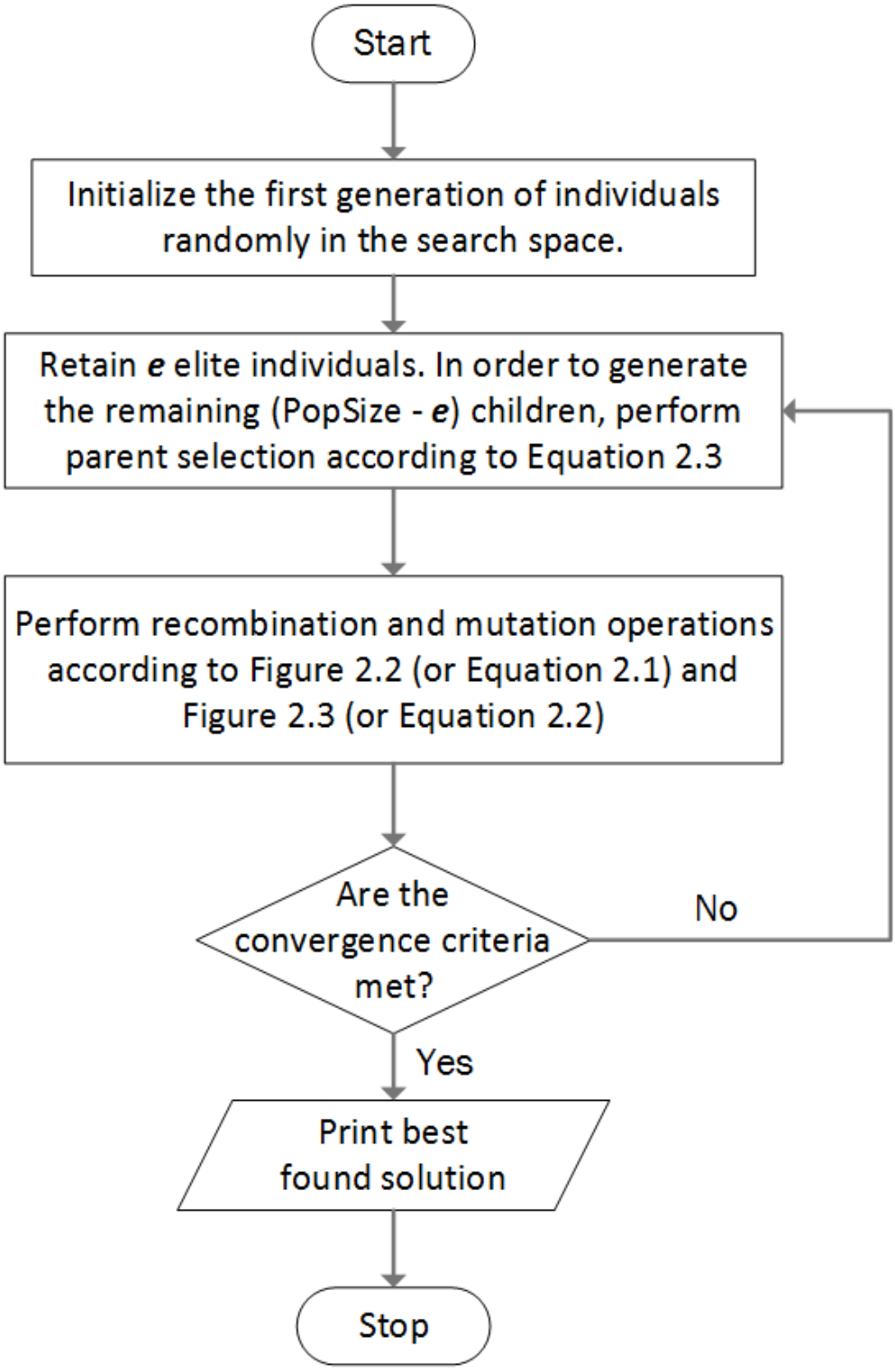


Fig. 2.1: Optimization performed by Genetic Algorithms

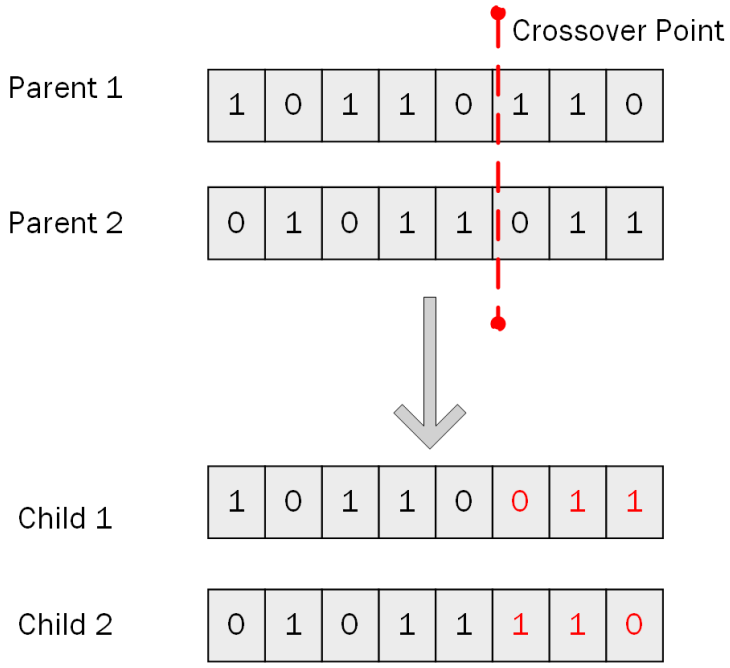


Fig. 2.2: Genetic Algorithm – One-point crossover operator

before the crossover point, and the genes of the second chromosome *after* the crossover point. The reverse applies to the second child, viz., it will have the genes (bits) of the second chromosome (parent) *before* the crossover point, and the genes of the first chromosome *after* the crossover point. This is illustrated with an example in Figure 2.2. In this figure, the two parents have 8-bit chromosomes, and they give birth to two children, both of whom have 8-bit chromosomes as well. Other crossover operators have also been proposed in the literature, e.g., two-point crossover and uniform crossover.

If the parents are represented as real-valued numbers, then an arithmetic recombination operation is often used, and the children are formed by linearly combining components of their parents. So, if the parents are X and Y, and the recombination (weighting) parameter is α, then the resulting children are shown in Equation 2.1.

$$\begin{aligned} &\text{Child 1: } \alpha X + (1-\alpha)Y; \\ &\text{Child 2: } (1-\alpha)X + \alpha Y; \end{aligned} \tag{2.1}$$

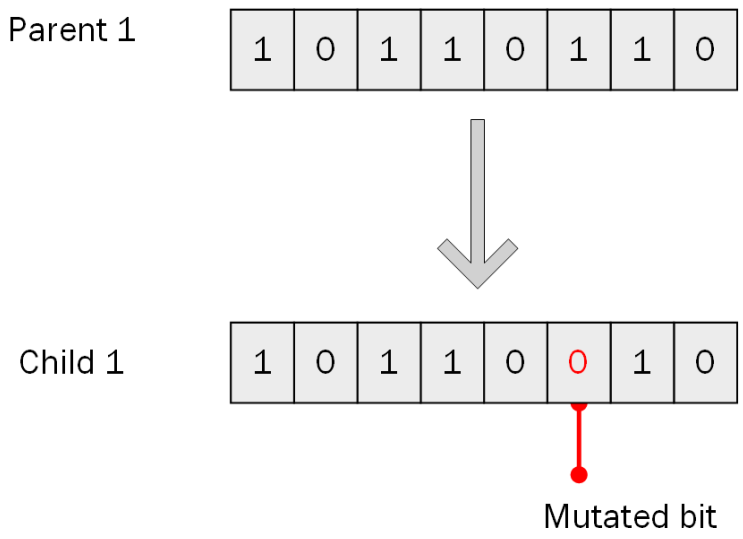


Fig. 2.3: Genetic Algorithm - Mutation operator

Mutation

A mutation operator modifies a chromosome by altering it slightly, typically as little as one bit. For a bit-string representation, if the k^{th} bit undergoes mutation, it means that the bit is toggled. This is also depicted pictorially in Figure 2.3. In this figure, the parent has an 8-bit chromosome, and gives birth to one child, which has an 8-bit chromosome as well.

If the parents are represented as real-valued numbers, then the magnitude of the mutation can be determined in many ways, a common one being proportional to the standard deviation, σ, of the Gaussian distribution. So, if the parent is denoted by X, and the mutation parameter is β, then the resulting child, C is shown in Equation 2.2.

$$C = X \pm \beta\sigma \tag{2.2}$$

If the parent is a vector of real values, this mutation operation occurs independently on each component of the vector in a GA.[2]

[2]Evolution Strategies use more sophisticated approaches that adapt the mutation rates, perhaps basing these on the learned relationships between different components of the vectors used for representation.

Parent Selection

"Fitness" is usually identified with a measurable estimate of solution quality.[3] Parent selection is based on fitness. A roulette wheel selection process may be used, where the probability of choosing a parent is proportional to its fitness. For any individual parent X_i,

$$P(X_i \text{ is chosen as a parent}) \propto \text{Fitness}(X_i), \quad \forall X_i \in \{\text{Mating Pool}\} \tag{2.3}$$

A parent can be chosen multiple times in a generation, in which case it contributes its genes to many children. The idea is that if a parent is of better quality than others in the population, it is desirable to try and improve upon it by using its genes as many times as possible, while not sacrificing diversity – the lesser fit individuals are also given a chance, albeit a smaller chance.

The practical application of GAs to new and difficult problems is an art requiring considerable empirical experimentation. In particular, an important obstacle is that populations may "converge" to suboptimal solutions, where convergence refers to the fact that most individuals in the population have become very similar, so that evolutionary operators are unable to explore new regions of the search space. Subsequent execution of the GA for additional generations is no longer helpful, and merely wastes computational effort. It would be best, at that point, to restart the GA with a new set of individuals, perhaps merged with some of the better individuals from the most recent generations.

Exploration vs. Exploitation

Two terms that are very common in EA terminology are exploration and exploitation. The term exploration refers to identifying (previously unexplored) regions of interest without attempting to find the regional best in any of the subspaces. Exploitation, however, is the fine-tuning of the search after exploration has identified a subspace of interest which is likely to contain an optimum,

[3] In minimization problems, a high fitness implies a low function value. In order to avoid this confusion, we will refer to this term as "function value" in our experiments as far as possible, since our experiments are mostly tested on minimization problems.

i.e., searching the neighborhoods.

The benefit of exploration is that new regions are visited, but it does not emphasize the improvement of solution quality. Exploitation is the exact opposite in that it focuses only on improving the result of a prior search by examining its neighborhoods, but not other spaces. Considerable work has been done in analyzing the balance between exploration and exploitation and their tradeoffs, summarized in a survey [16].

2.1.2 Particle Swarm Optimization

The idea of Particle Swarms was introduced by Kennedy, Eberhart and Shi [36][24][65][66]. It was initially designed to represent movements of individuals, such as a flock of geese, a colony of bees, a school of fish, and so on. When used to perform optimization tasks, several PSO models began to be developed, summarized in [58]. Figure 2.4 depicts a basic Particle Swarm Optimization algorithm structure. An implementation of the basic PSO is given in [13].

Unlike a GA, the next generation of individuals is determined by the position and velocity of the individuals and the corresponding position of the global best individual, which is the individual with the highest fitness, i.e., lowest function value. The term "generation" requires some clarification here – it is not a new set of individuals as in the case of a GA, but just an updated position and velocity vector of the same individuals from the previous time instant. In other words, it is an iteration, but not quite a generation in a GA sense.

As in a flock of birds, let us represent each individual as having position and velocity vectors $< X_i, V_i >$, each of which has D components (dimensions). So, the individual components can be represented as:

$$\begin{aligned} X_i &= (x_{i1}, x_{i2}, \ldots, x_{iD}) \\ V_i &= (v_{i1}, v_{i2}, \ldots, v_{iD}) \end{aligned} \tag{2.4}$$

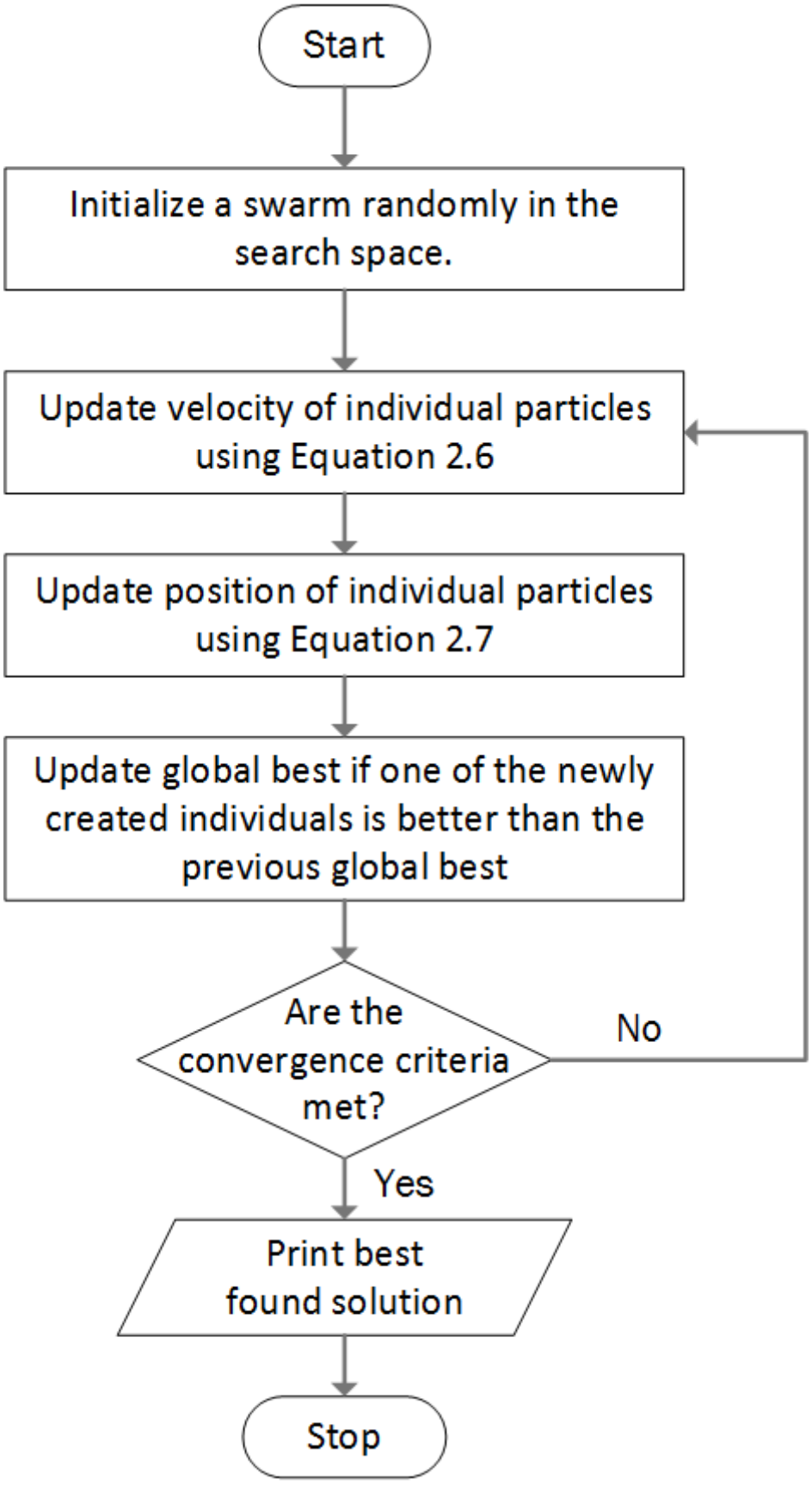


Fig. 2.4: Particle Swarm Optimization Flowchart

and, the best previous position of any particle is represented as:

$$P_i = (p_{i1}, p_{i2}, \ldots, p_{iD}) \tag{2.5}$$

Further, let us represent the global best individual index by g.

Using the above system representation, the velocity and position of a particle in each dimension $d \in D$ are manipulated as follows [65][66]:

$$v_{id} \leftarrow w v_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id}) \tag{2.6}$$

$$x_{id} \leftarrow x_{id} + v_{id} \tag{2.7}$$

where w, c_1, c_2 are values chosen by the practitioner and r_1, r_2 are values chosen from the uniform distribution over $[0, 1]$, that vary in each iteration.

In Equation 2.6, there are three main components:

- Inertial: In the first term (wv_{id}), v_{id} is proportional to the velocity at the previous iteration which is responsible for playing a significant role in determining the particle's position at the next time instant. The term w is termed the "inertia weight" and is typically chosen from the range $[0.9, 1.2]$.

- Cognitive: The second term $c_1 r_1 (p_{id} - x_{id})$ represents the particle's own "memory". Hence, it is dependent on particle i's best known position p_{id}.

- Social: The third term $c_2 r_2 (p_{gd} - x_{id})$ represents the particle's bias based on the global best found so far. This is explained by its dependency on the term p_{gd} which is the global best.

Frequently used values of the coefficients c_1 and c_2 are around 2, and the selection of these parameter values has been analyzed in detail to study the stability of swarms [14].

The random values r_1 and r_2 introduce the stochastic nature of dependencies on the cognitive and social components respectively. Since these vary in each iteration, there is a dynamic balance

of the weights assigned to the cognitive and social components, or a balance between exploration and exploitation, without ignoring the past observed values (due to the presence of the inertial component).

2.1.3 Learning Classifier Systems

Learning Classifier Systems (LCSs), originally proposed by Holland [32], use supervised and reinforcement learning concepts. In an LCS, a population of rules (also referred to as individuals or classifiers) evolves on the basis of intermittent stimuli and reinforcements from the environment, and learns which responses are appropriate when an input stimulus is presented to the system.

Simultaneously evolving multiple rules can be more efficient and effective than the alternative of sequentially searching for the best rule that handles data not satisfactorily handled by earlier rules. Furthermore, the evolutionary paradigm permits dynamic learning, in contexts where the environment itself changes, or wherein massive amounts of data arrive in real time and must be processed rapidly.

Rules in an LCS receive external inputs and generate actions or outputs applied to the environment. The results of these actions can then be evaluated, generating the feedback necessary to apply selection pressure to the rules. If firing some rules results in actions that enable the system to do well in the environment, then these rules are considered desirable, and persist in the system. On the other hand, if rule-firing leads to poor performance, then such rules are likely to disappear from the population, or be applied with considerably reduced frequency in the future. This approach is implemented by associating each rule with a "strength" that indicates its relative quality.

The earliest such system applied to a practical problem was Goldberg's LCS [28] to control the flow of natural gas through a pipeline, detecting leaks and optimizing profit in the face of seasonal and daily fluctuations in demand. LCSs have since been used in a wide variety of problem types such as in data mining, scheduling and optimization. Stochastic mutation and recombination operators based on trial and error (as opposed to a deterministic algorithm) have proven to be successful very often for complex problems.

At a high level, an LCS contains three components:

1. Performance System: This executes rules, possibly in parallel. Rule-firing may result in direct interaction with the environment, or the generation of "messages" (in the working memory of the system) that lead to the firing of other rules.

   A rule can be depicted as follows:

$$Rule \rightarrow Condition : Action$$

2. Credit Assignment: This mechanism evaluates rules and adjusts their strengths depending on system performance. Algorithms such as the "bucket-brigade" [77][29] are frequently used for this purpose.

3. Rule Discovery System: Evolutionary algorithms are used, using mutation and recombination operators along with selection mechanisms.

Performance can be improved in some problems by grouping together rules in some manner, just as modularization is helpful with traditional rule-based systems. Grouping can reflect interdependencies and close connections between rules. Subsets of rules may be collectively evaluated and rewarded or punished.

Two main varieties of LCSs have been proposed by researchers:

- In "Pittsburgh-style" classifier systems [68], a complete solution, i.e., a collection of many rules, is evaluated together. Rather than a single rule or small group of rules, the entire set is subjected to reward or punishment.

- "Michigan-style" systems (including Holland's LCS), in contrast, reward or penalize individual rules.

Most classifier systems use a simple language in which the right hand side is a bit string indicating which actions are to occur, and the left hand side is a string with elements drawn from the

alphabet 1, 0, #, where the "#" symbol represents don't-care cases taking either 0 or 1 as possible values, when the value of the corresponding attribute is irrelevant to the rule.

Each position in the left-hand-side of the rule represents a specific condition, and each position in the right-hand-side corresponds to a specific action or hypothesis. For example, the rule

$$\underbrace{0\#1\#110\#}_{\text{Condition}} : \underbrace{10}_{\text{Action}}$$

is interpreted to mean that if the first and seventh conditions are false, and the third, fifth and sixth conditions are true, then the first corresponding action (from the right-hand-side of the rule) is to occur, but not the second.

The right hand side of each rule is treated as identifying which "messages" are posted onto a "blackboard" data structure when a rule is fired. Posting messages is analogous to asserting facts in the working memory of a rule-based system. These messages may match with the left hand side of another rule, leading to its activation and possible firing. However, this does not occur in stimulus-response classifier systems, in which every rule interacts directly with the environment.

Holland's formulation allows for the left hand side of a rule to contain any number of ternary strings, and also allows negation [33]. For instance, the rule

$$\left.\begin{array}{l} 0\#1\# \\ \#0\#0 \\ \neg 11\#\# \end{array}\right\} : 10$$

can be fired if two messages matching $0\#1\#$ and $\#\#00$ are present, and if there is no message matching $11\#\#$.

These rules keep evolving through the generations and the performance of rule sets is expected to improve with time.

XCS (X-Classifier System) [79][8][9][10] is an implementation of a Learning Classifier System in which the fitness of a classifier is based on the payoff prediction accuracy, i.e., the accuracy

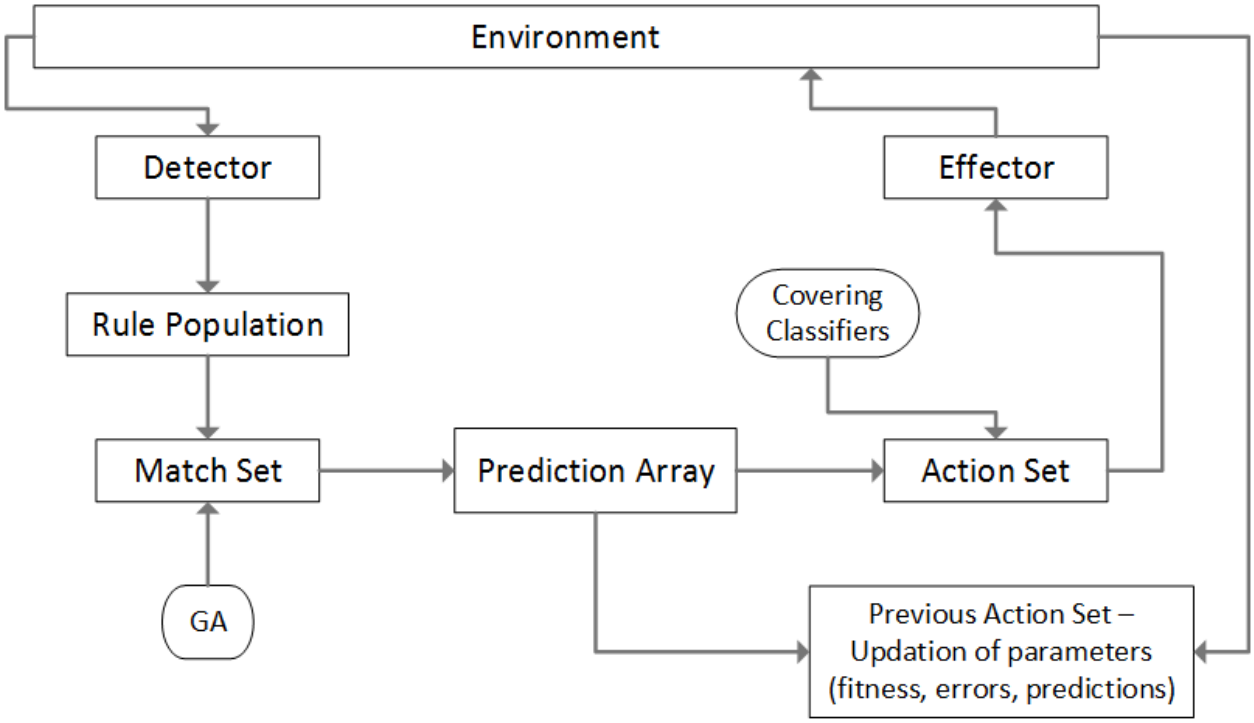


Fig. 2.5: The XCS model proposed by Stewart Wilson [79]

in making a decision, and a Genetic Algorithm is executed on the action sets. Reinforcement learning here is similar to Q-learning [75]. In this payoff-prediction model, the overall payoff is a combination of the expected reward and payoff prediction of the best possible action. As opposed to ZCS [78], it is an accuracy based algorithm, which means that the portion of the roulette-wheel assigned to each classifier in this algorithm is proportional to the fitness measure obtained from its accuracy (and not strength).

We use the XCS algorithm as the basis for our work – the algorithm and results shown in later sections are based on implementations that use this framework. Figure 2.5 shows the structure of Stewart Wilson's XCS [79]. Other variations of LCSs are summarized in [72].

The GA running at the heart of the XCS is intended to evolve "maximally general, maximally accurate" rules [11][44]. To achieve high generality of rules, we would like to have as many #s as possible, without sacrificing accuracy. For instance, a rule $\#\#\#\# : 1$ is very general, but since the four #s cover all possible inputs, they will have a high error, unless the problem being solved involves every data point belonging to class 1, no matter the input. On the other hand, a rule such

as $1011 : 0$ is likely to be very accurate, but is not general enough to cover any data point other than 1011.

Prior to exploring networks of rules in LCSs, we had begun the study of relationships between rules and classes in an XCS system [39][40][50]. We had also discussed interrelationships between multiple rules, rule-sets and classes [41]. Although these were information theoretic studies based on Shannon's entropy and mutual information [63] between rules and classes, they paved the way for a deeper study of relationships between rules in an LCS system.

2.2 Networks

The study of graphs and networks has been ongoing for a long time. However, the study of dynamic networks has been exciting considerable interest in recent years, particularly in the social networks context, mostly because of the success and popularity of many online social networks such as Facebook, Twitter, and LinkedIn. A comprehensive introduction to Network Science is provided by Newman [53].

Although the terms "graphs" and "networks" may be used interchangeably in many cases, a graph is an abstract representation of a network. For instance, a user's friends and social connectivities constitute a *network*, but their representation is a *graph*. Similarly, in our case, the population that is performing an optimization is a network of individuals, but is represented as a graph.

Any graph consists of a set of vertices (or nodes), $\mathbb{V}$, and a set of edges $\mathbb{E}$. An edge between two nodes represents some form of connection between them. The edge can be directed, implying that there is an asymmetric relation between the nodes, or undirected, implying a symmetric one. One example of a directed graph models the Twitter network where if a person A follows a person B, it does not imply that B follows A; for example, a celebrity on Twitter may have many more followers than he/she is following. In the case of an undirected graph, such as in Facebook, if a person A is a friend of person B, it implies that B is also a friend of A. This establishes an undirected edge between the nodes representing the two people.

We briefly describe below some of the fundamental network terminologies, metrics and measures, based on [53]. The example values in the descriptions are based on the Dolphin network [45][4]. We also describe some of these measures in the context of a GA in Section 3.3.

The Dolphin network is shown in Figure 2.6. Each node is a dolphin, and each edge represents an interaction between two dolphins. This graph was drawn using Wolfram's Mathematica [80].

- Weighted Networks: Edges can be assigned a weight, if there is a need to represent more than just the presence or absence of edges. This is typically a real number, and it can represent various measures in different kinds of networks. For example, in the dolphin network, if we wish to represent higher frequencies of interaction among one pair of dolphins when compared with another, we can assign that edge a certain weight indicative of this difference in interaction levels.

- Community: A community is a subgraph which is characterized by a tightly-knit structure. Each node in a community is connected to many others within that community, and significantly fewer edges between communities. A network is said to have a community structure if it can be sub-divided into such sub-networks (with or without overlaps).

- Degree centrality: The degree of a node is the number of edges connected to it.[5] Typically, the most important nodes of a network are considered to be the nodes with the highest degrees. So, in our dolphin network example in Figure 2.7, the dolphins at the centers of the communities have higher degrees (or degree centralities in social networks) than the peripheral ones.

- Adjacency Matrix: This is a two-dimensional matrix with vertices representing the rows and columns and the $(i, j)^{th}$ entry representing the presence or absence of an edge between vertices i and j. This matrix consists of 1s and 0s in case of an unweighted network, i.e., the presence or absence of an edge, and real values in the case of a weighted one, i.e., the

[4]This dataset is an undirected social network of statistically significant frequent associations between pairs of 62 bottlenose dolphins in a community living off Doubtful Sound, New Zealand [46].

[5]Directed networks have an in-degree and an out-degree.

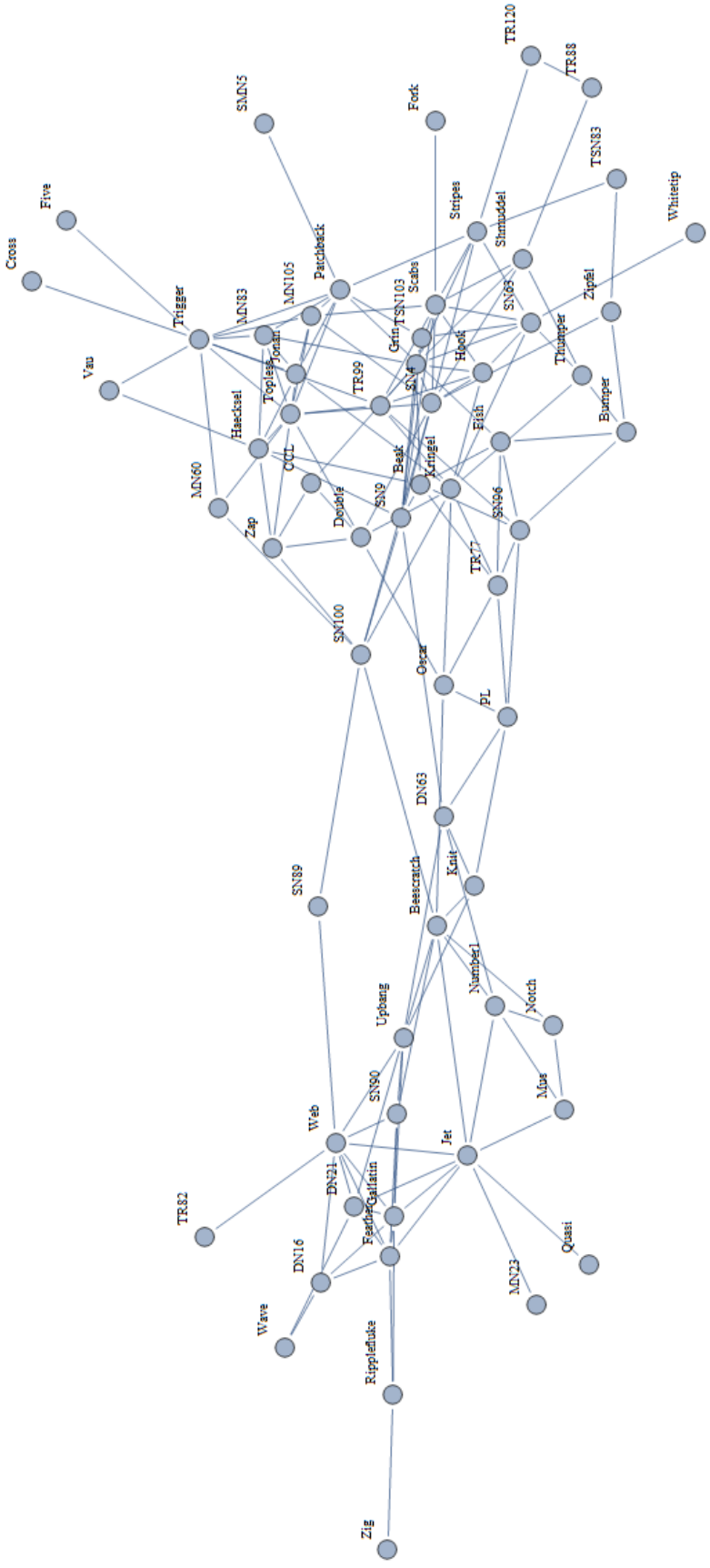


Fig. 2.6: Dolphin Network based on Lusseau et. al. [45], and Lusseau & Newman [46]

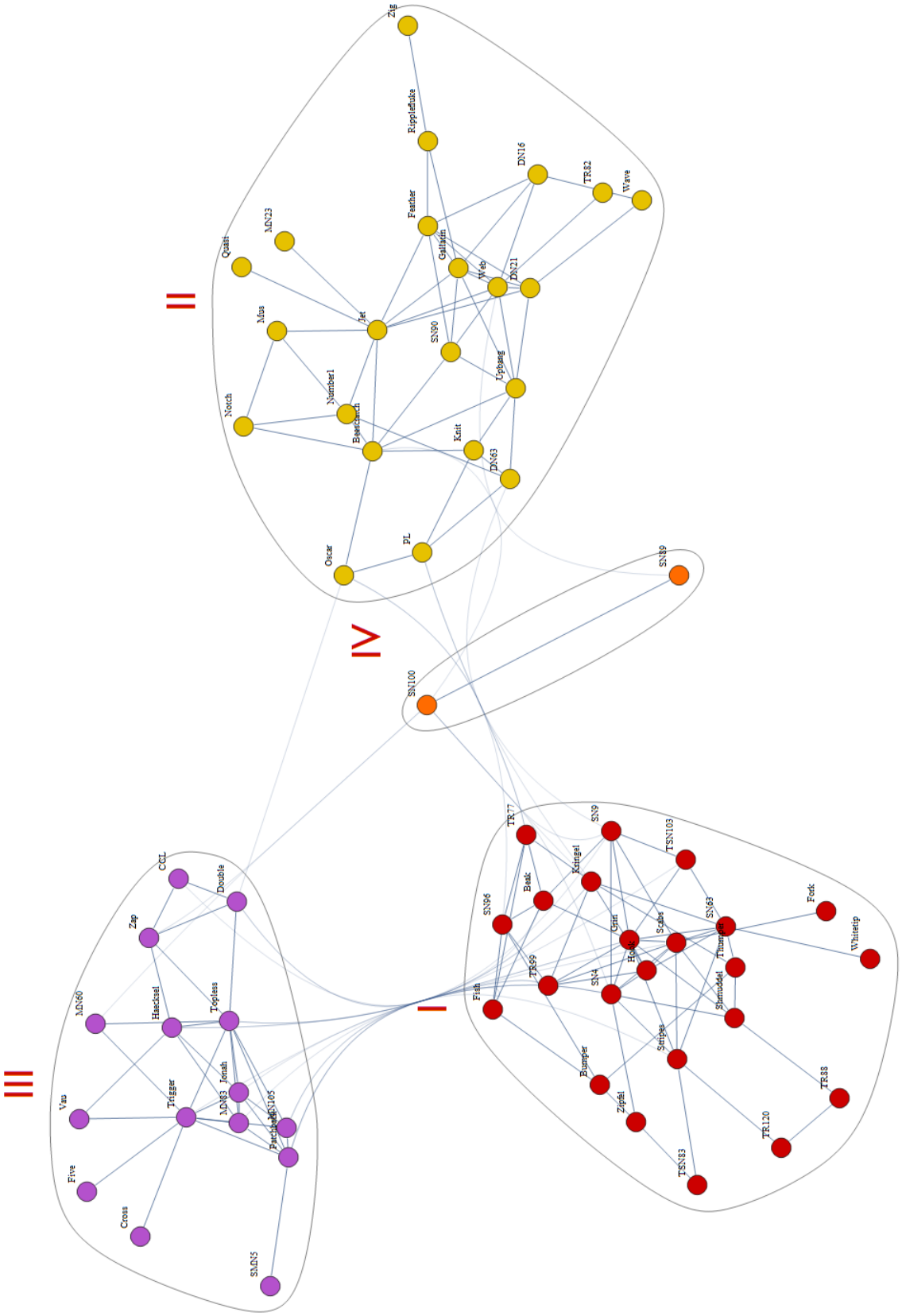


Fig. 2.7: Communities in the dolphin network

strength of the edge. Also, the matrix is symmetric across the principal diagonal if the network is undirected, and asymmetric if directed.

- Community Detection: We wish to separate the network into groups of vertices, often non-overlapping (although there could be instances of overlapping communities as well). However, no constraints exist on the number of groups or on the number of nodes that can form a community. Several algorithms have been suggested to perform this grouping based on the same concept of communities remains the same in them. One form of communitization is shown in Figure 2.7. This is the same network as in Figure 2.6, except that the communities have been separated. These are represented by the various grouped nodes with one color per community, and numbered I through IV. It is important to note that the network is not modified in any way, it is only rearranged to highlight distinct entities, or communities. We notice that there are two large communities (Communities I and II), one medium sized community (Community III), and a very small community consisting of just two nodes (Community IV).

- Components: If there exists a path from every node to every other node in a network, then the network is said to be fully connected. Figure 2.6 is an example of a fully connected network. However, if the network can be divided into two or more subgroups which have no connections to one another, then they are referred to as components of the network. For example, if in Figure 2.7, there existed only intra-community edges and no inter-community edges, then each community would have been a component.

- Large Component: If the size of a network component grows in proportion to the number of nodes in the network, it is called a "Giant Component" [53]. When the network is not associated with a specific growth process, we refer to a component that contains a large proportion (say $80 - 90\%$) of the nodes in the network as a "large" component.

- Hypergraphs: In certain cases, it is important to represent relations between three or more vertices at a time. Such relations are represented using "hyperedges", and graphs containing

such hyperedges are known as hypergraphs.

2.2.1 Networks in the context of EAs

Genetic Algorithms and other population-based evolutionary algorithms are rich sources of information contained in the ever-changing generations and populations. The effects of genetic operators and other ideas borrowed from observing various biological species serve as models to study the growth of a species from its infancy to maturity as a community or communities. Observing that the natural behavior of most species in the biosphere is to form cohesive communities (or networks) for survival and growth, we believe that a deeper understanding of such evolutionary systems can be greatly enhanced by the science of networks.

Considering also that each generation of an evolving population contributes towards one snapshot of a network, we have, in effect, created a dynamic network with a simple biological evolution system.

2.3 Related Work

We describe below some of the works that consider populations as more than just collections of disconnected individuals.

- One of the earliest works exploring different subspaces in a GA proposes the concept of crowding [19]. This is a specialization in steady state GAs where offspring are constrained to replace similar parents.

- In multi-deme algorithms [57], islands of individuals are formed and breeding is strictly local.

- Niching [47] involves the creation of sub-populations which explore different “niches” or regions of interest. These sub-populations may be stable through the runs especially when there are multiple global optima.

- Panmictic GAs [3] – This paper summarizes different panmictic and island models explored by researchers.

- Parallel GAs [12] – This approach involves the creation of isolated groups of individuals that explore different subspaces in parallel. Strategies are formulated to govern the occasional migration of individuals between islands.

All of these approaches (formulated in the works cited above) consider subsets amongst the population in different ways. Also, a few articles have been written focusing on related topics, such as the following:

- Clusters in an evolutionary system which focus on traditional clustering techniques [52].

- Adaptive crossover and mutation rates based on clusters [81], which suggests that crossover and mutation should not have a constant rate throughout the run.

- Formation of stable clusters, particularly in swarms [2], and communities in evolving networks [56].

- Creating rule clusters in an XCS system to specialize rule-sets and to merge rules which overlap, and the removal of redundant rules [64], and attribute co-occurrence networks in LCSs [73].

- Another alternate form of rule specialization involves the study of the development of specialized rule-sets from a mutual information viewpoint [69]. Here, interaction between different rules is defined by the mutual information between them.

Other related works include the study of graphs with edge weights denoting how often individuals place children in adjacent vertices, also discussing the balance of drift and selection being affected by graphs [43]. Networks based on shared parents at multiple levels are studied in [76], addressing genetic drift and the unification of parent selection processes.

By contrast, our focus is on examining relationships between individuals, interrelationships between individuals and different properties of the EA that we can gather from these networks, and possible ways of using this information to enhance them. We ask what happens over time to the structural properties of networks representing evolving populations, and then treat these networks as a useful characterization of the EA, in order to study potential improvements.

In other words, we wish to first run an EA without altering it, and then observe the dynamics from the point of view of the evolving networks. We attempt to study the behavior of the EA without looking at the EA parameters. We then draw parallels between EA measures and network measures and show evidence of the underlying inter-relationships between individuals. We also use information obtained from such evolving networks – again, without using information from the EA – to take decisions about altering the runs of the EA.

Typically, reseeding criteria are based on:

- an *a priori* bound on the number of fitness evaluations,
- values of the best or average function value, or
- increments in fitness values below a set threshold.

Various other stopping criteria have been proposed in [4], [31], [61] and [27]. Also, various parent selection mechanisms, cataclysmic mutation, and other approaches are described in [26]. Work in [59] focuses on increasing search around points of good fitness values by sampling the search space around the most fit points.

However, the focus of our work is not to modify the GA parameters or set different thresholds for them, but to:

- use the evolving networks as a means to identify potential convergence,
- decide on time instants to reseed,
- identify regions of interest, and

- identify points likely to be better candidates (higher fitness), based on changes in populations.

The focus is mainly on functions which have many potential basins of attraction, i.e., functions prone to premature convergence. We demonstrate the role that networks (or interactions between individuals) play here.

This work explores in detail the inherent relationships between individuals, their behavior when communities or components form, and the support that networks provide in improving the search features of an evolutionary algorithm. Emphasis is not just on the positions of the optimum points, but the paths that the individuals take to reach them.

We have now introduced the essential background material on different types of EAs and networks, and described some of the related work. In the next chapter, we describe one way of defining a network that models an EA, and study the evolution of network and EA parameters.

CHAPTER 3
DISTANCE-BASED NETWORK MEASURES

In this chapter we begin by defining the *Euclidean Network*, describe various network metrics and measures under study in the context of an EA, study the evolution of a large component, and list a selection of benchmark optimization problems with different characteristics.

3.1 The Network

We define a *Euclidean Network* $G =< \mathbb{V}, \mathbb{E} >$ as follows:

- $\mathbb{V}$: All n individuals in the population are vertices or nodes.
- $\mathbb{E}$: Two individuals have an edge between them if the Euclidean distance between them is less than the "Edge Threshold" value (δ). In our simulation, this value is chosen to be proportional to the length of the body-diagonal of the search space, since the "closeness" of two points is relative to the space being searched.

We discuss GAs and PSOs in the context of the above definition of a Euclidean network. We also define and analyze another form of network for GAs based on ancestral information. Finally, we define a different network for LCSs as the individuals are ternary strings, and we define similarities between individuals based on comparing individual bit positions.

In Euclidean networks, we look for the formation of a large component. Through this, we find that a majority of high fitness individuals often lead the component down a somewhat deterministic path of high exploitation, leading eventually to convergence.[1]

The identification of the largest component is done efficiently using Tarjan's algorithm [71]. The complexity of this algorithm is $O(V + E)$ where V is the number of nodes, and E is the number of edges.

3.2 Evolution of a large component

Snapshots of the network and component formation along with the emergence of a large component are shown in Figures 3.1a-3.1f, for a real-coded GA, with population size = 30, two-point crossover rate = 0.8, adaptive feasible mutation rate = 0.2, and roulette selection, generated using the GA Toolbox in MathWorks's MATLAB [48]. Edge lengths shown are longer than typical values for clarity of demonstration. The background is the contour plot[2] of Rastrigin's function [60], having global minimum at (0,0), and the networks are superimposed on it. Solid (red) lines indicate connections within the largest component, dotted (black) lines are used within smaller components. Shaded (green) circles are the individuals with highest centrality. Squares denote other individuals.

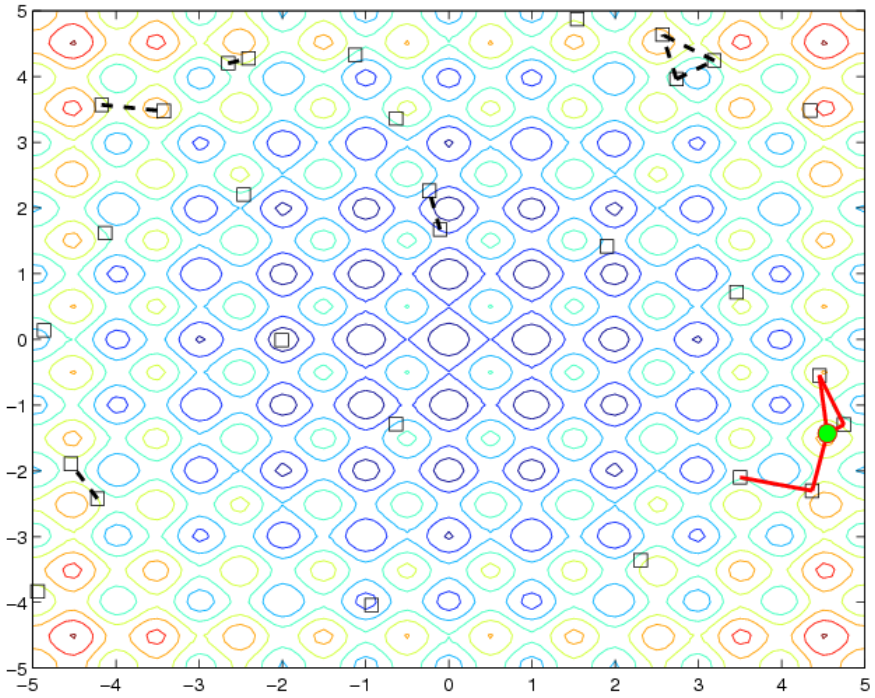
The individuals in these components, especially those in the large component, demonstrate a form of "democracy" in making GA decisions. A large number of individuals "voting" that an optimum solution lies in their vicinity will help in many ways, as shown in later chapters.

3.3 Network measures under study

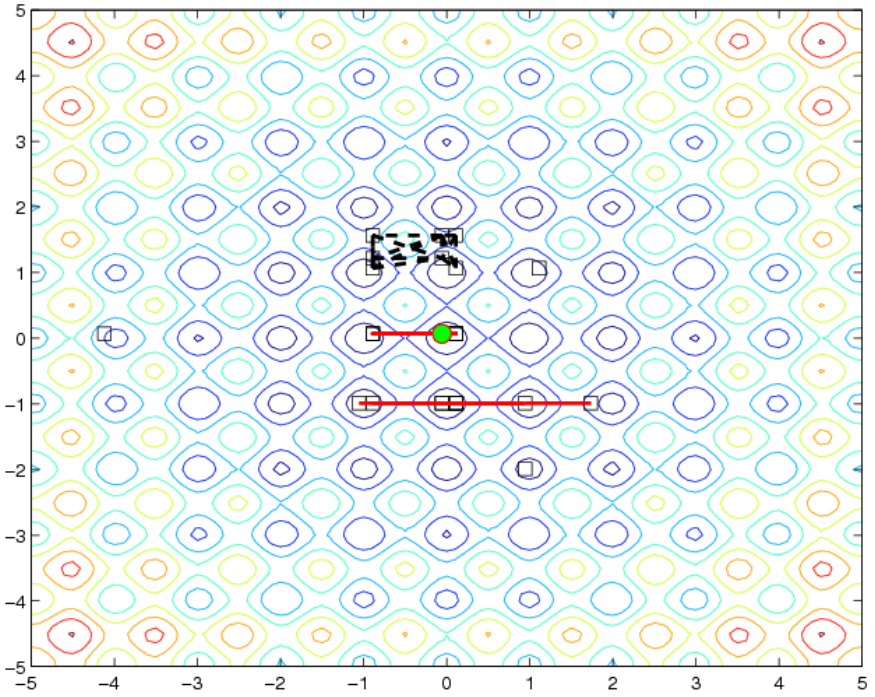
We list below some network measures under study. In Figures 3.2a-3.2h, it is important to note the strong correlations between the evolution of network measures and GA properties. These correla-

[1]The emergence of this component and its collapse leads to a comparison with "Exploration vs. Exploitation" as discussed in Section 2.1.1. This is because formation of the component signifies the exploration phase, and its collapse corresponds to exploitation.

[2]The dark (blue) contours are lines joining points having lower function values and the function values increase as the contour colors change towards a lighter shade (red).

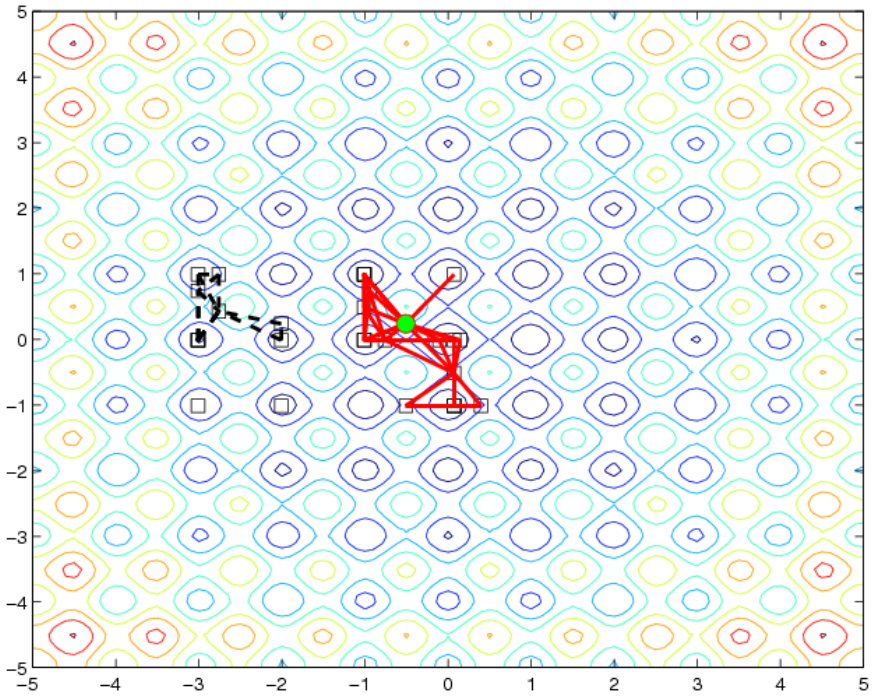


(a) Exploratory phase – We see a few components, one larger than the rest, and a few unconnected individuals.

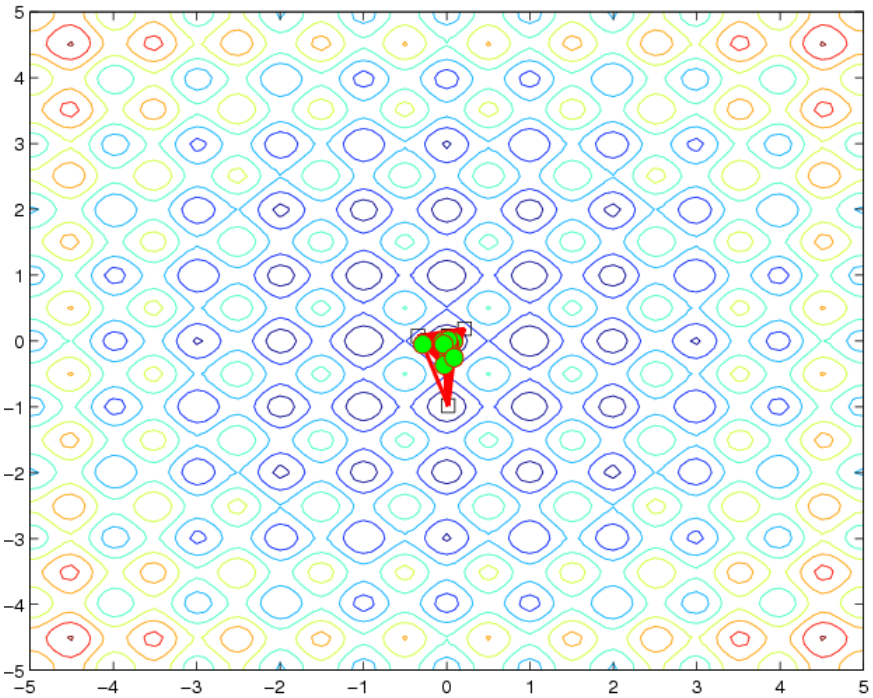


(b) A few generations later – This illustrates the kinds of components that can be formed. They are not clusters in the typical sense.

Fig. 3.1: Different phases in a GA optimization run

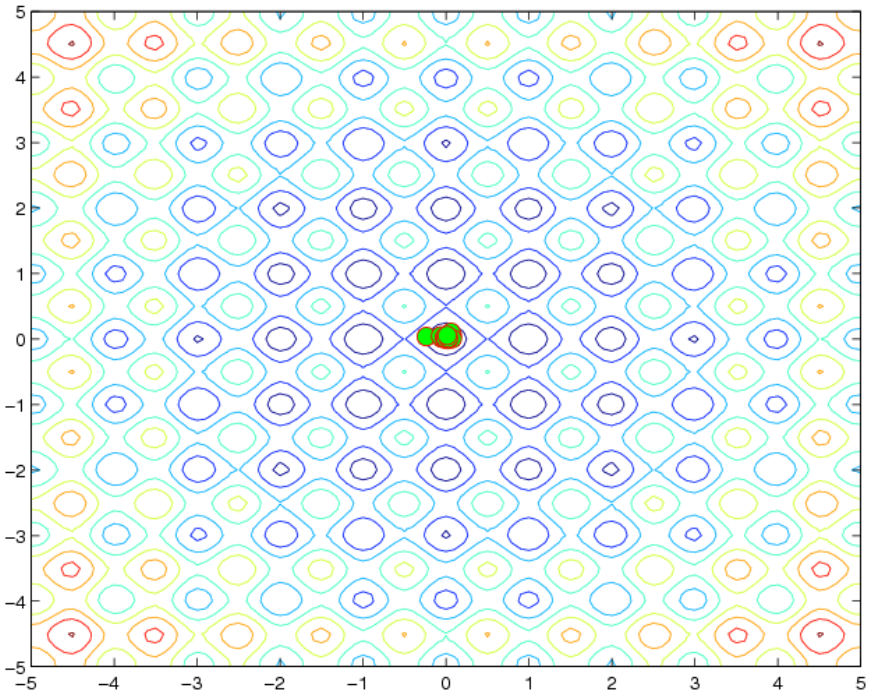


(c) Midway – We see that a large component has formed. Smaller components and unconnected individuals can also be seen.

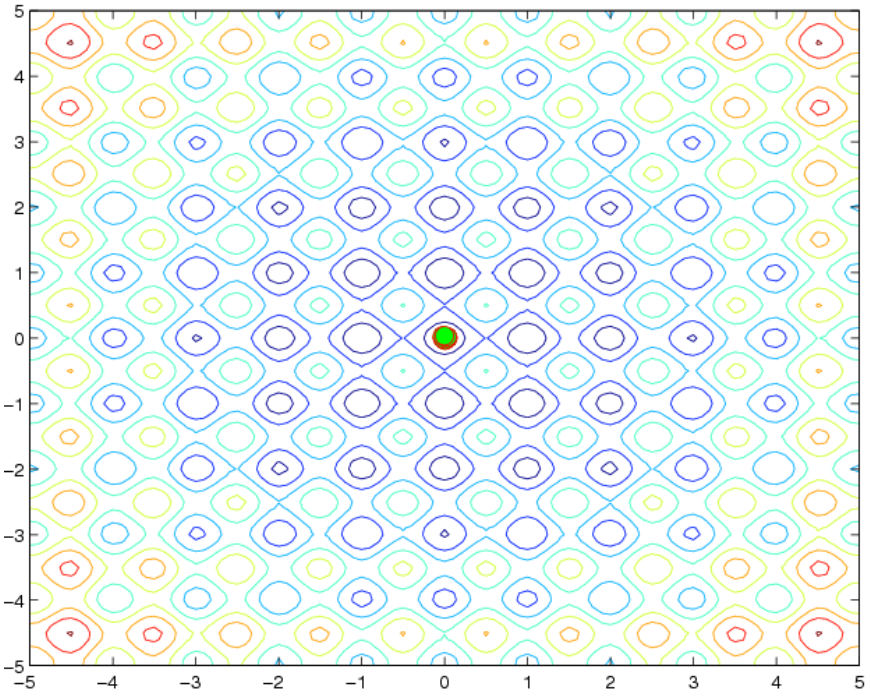


(d) Exploitation Phase, late in evolution – Only one large component is seen (and the convex hull has become much smaller), but the algorithm has not converged.

Fig. 3.1: (...contd.) Different phases in a GA optimization run



(e) Almost converged



(f) Converged

Fig. 3.1: (...contd.) Different phases in a GA optimization run

tions highlight the main point of this study, viz., since there are many network metrics that correlate well with GA evolution, we can harness these to understand the dynamics of relationships between populations of a GA better and hence detect convergence early, and improve GA performance (as discussed in later chapters).

(a) Number of nodes in the largest component: Since multiple components are formed, our main focus is on the component which has the largest number of nodes. It steadily increases and reaches the population size while the GA approaches convergence.

(b) Average Degree and Component Average Degree: We now consider the average degree of all nodes in the population and the average degree of all nodes in the largest component. Both these measures grow steadily, and convergence is implied when the largest component degree ceases to grow.

(c) Volume of component: The convex hull (hyper) volume of the component. The definition of a convex hull is as per Cormen [15]:

The convex hull of a set Q of points is the smallest convex polyhedron P such that each point in Q is either on the boundary of P or in its interior..

Many algorithms have been proposed to compute the hypervolume of a hull. Figure 3.2b illustrates (for a network representing a GA) how a hull area rises initially and then shrinks. Algorithm 1 shows how this measure can be used.

(d) Number of edges: The total number of edges in the whole network. As seen in Figure 3.2c, this measure has small values in the early stages of EA evolution, incremental jumps in the middle, and finally a large jump near EA convergence, to remain essentially constant from there onwards.

(e) Number of triangles: This measure counts the number of instances of nodes A, B, and C such that all three nodes are within the edge threshold distance δ, i.e., all three have edges

connecting each other. Its evolution is illustrated in Figure 3.2d, and has characteristics similar to edge count.

(f) Objective function values of nodes with highest degree: There could be multiple nodes with maximal degree and their objective function values are averaged and plotted in Figure 3.2e. This has a plot similar to the mean or best function value.

(g) Centroidal function values: Centroidal function value is the value of the objective function at the centroid of the largest component. The centroid, G, for a set of k points in $\mathbb{R}^n$ is defined as follows:

$$G = \frac{\mathbf{x_1} + \mathbf{x_2} + \cdots + \mathbf{x_k}}{k} \tag{3.1}$$

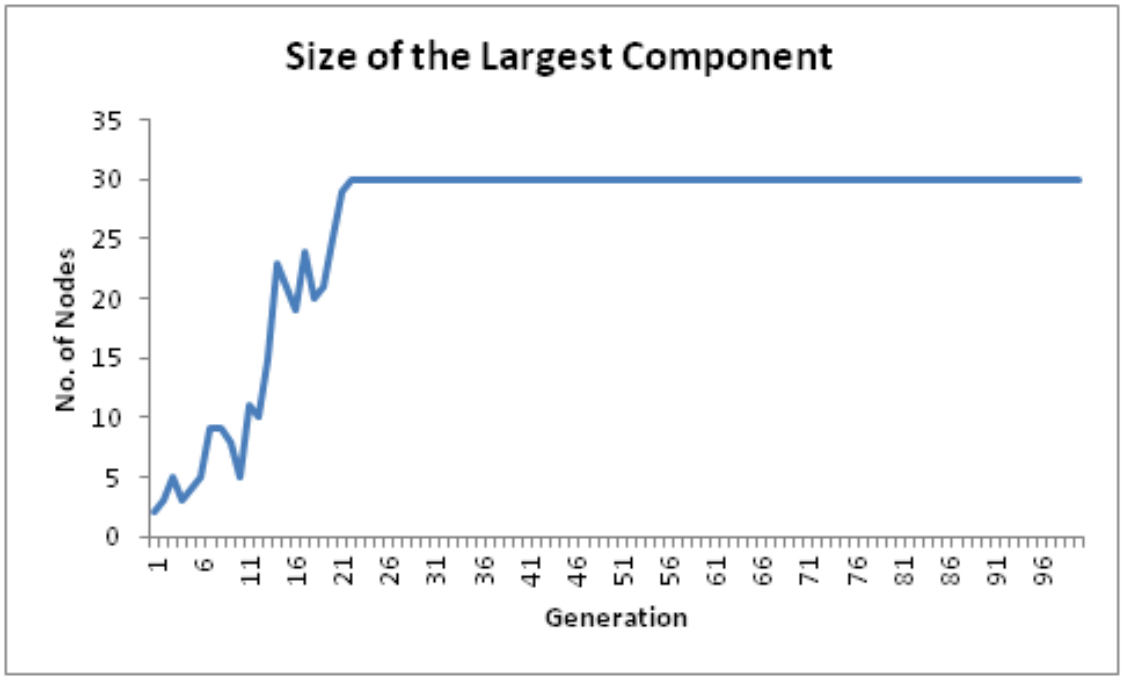
where each $\mathbf{x_i}$ is an n-dimensional vector.

We observe from Figure 3.2g that there are some generations where this value is lower than the best function value, i.e.

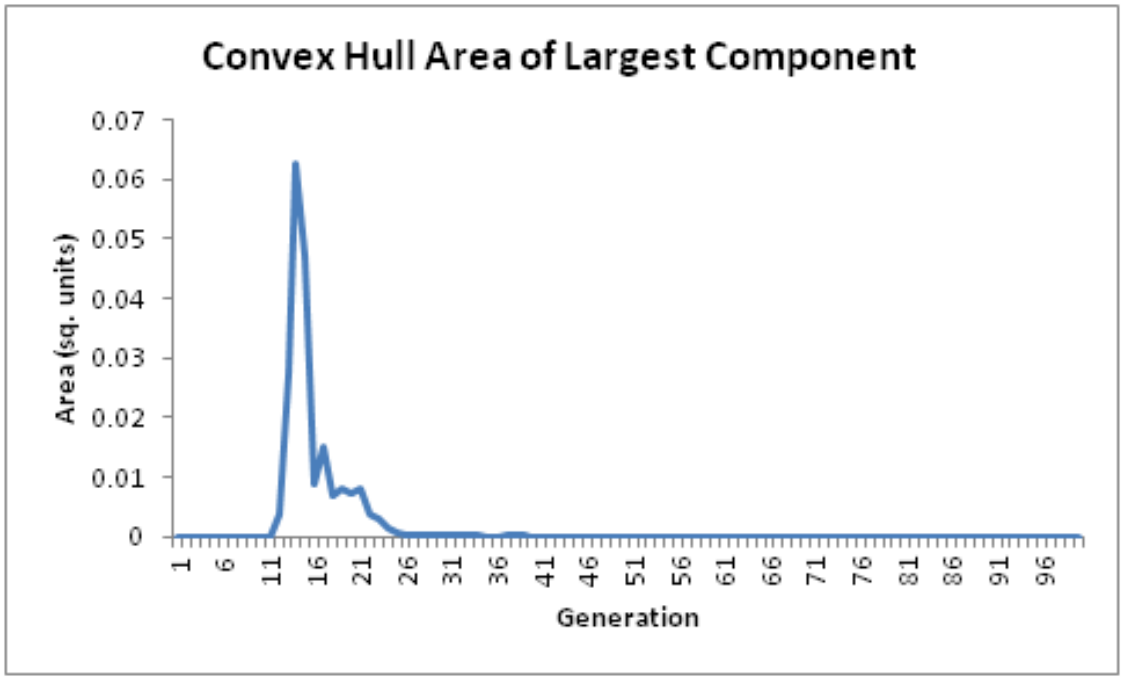
$$f(G) \geq f(x_i), \quad i = 1, 2, \ldots, n. \tag{3.2}$$

The plots in Figure 3.2a-3.2h are for Rastrigin's function [60], and similar results were obtained for many other test functions as well: Ackley's path function [1], DeJong's first function [19], Easom's function [23] and Schwefel's function [62]. The plots for these functions also have similar characteristics as observed in Rastrigin's function, but we have shown only the plots of one function (Rastrigin's) for demonstration.

From the plots of network metrics over time, we make a few interesting observations. In Figure 3.2b, the area of the largest component (convex hull) typically is very low in the beginning and rises slowly during the exploratory phase. We then notice a steep rise when a large component forms, and then a fall as if the convex hull has "collapsed under its own gravity". In Figure 3.2g, we see that there are a few instances where the centroid is a better point than all the other individuals in

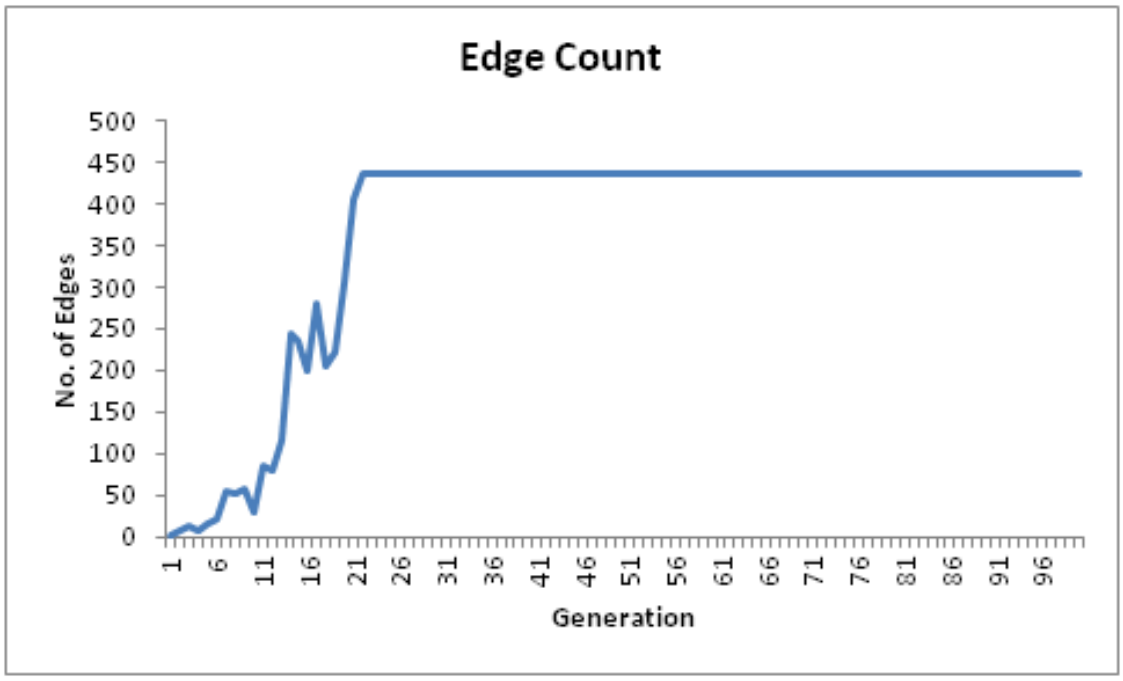


(a) The number of individuals in the largest component

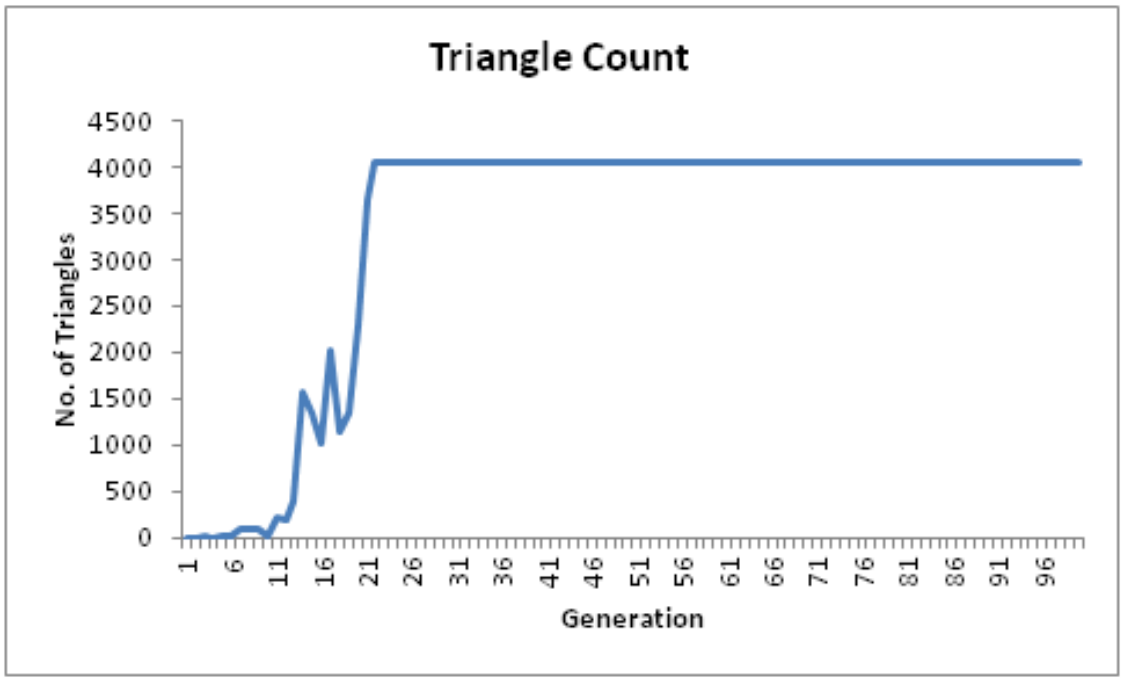


(b) Convex Hull area of the largest component

Fig. 3.2: Network Metric/Measure Plots for a GA run

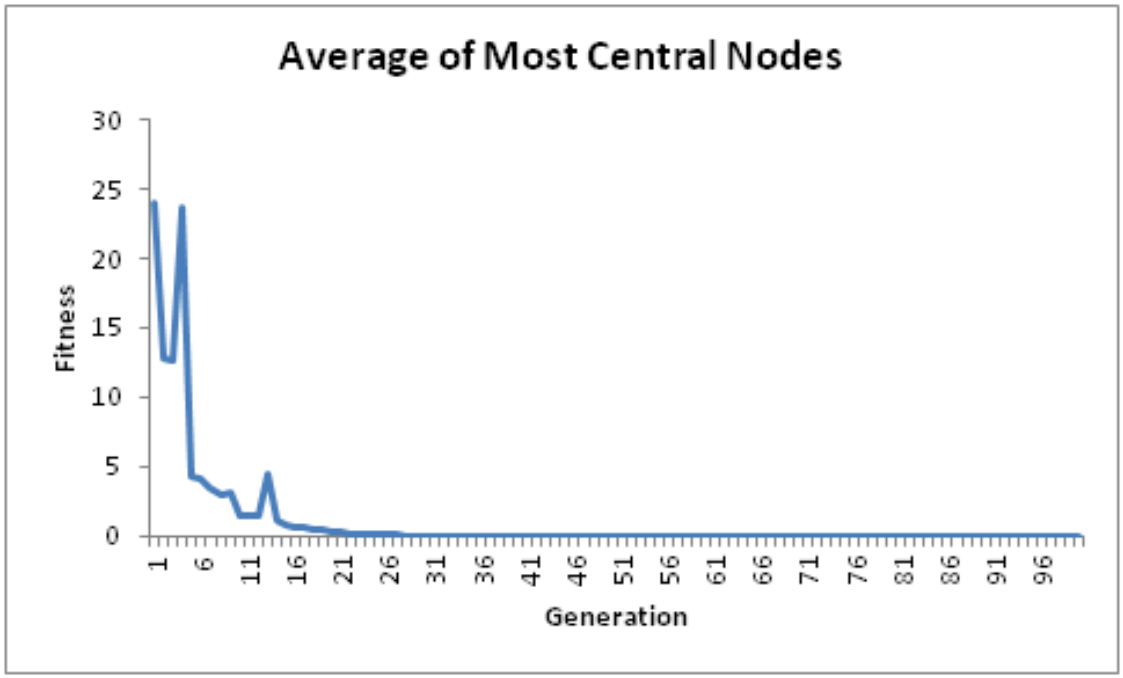


(c) Edge count – As the number approaches the maximum, we see cliques forming.

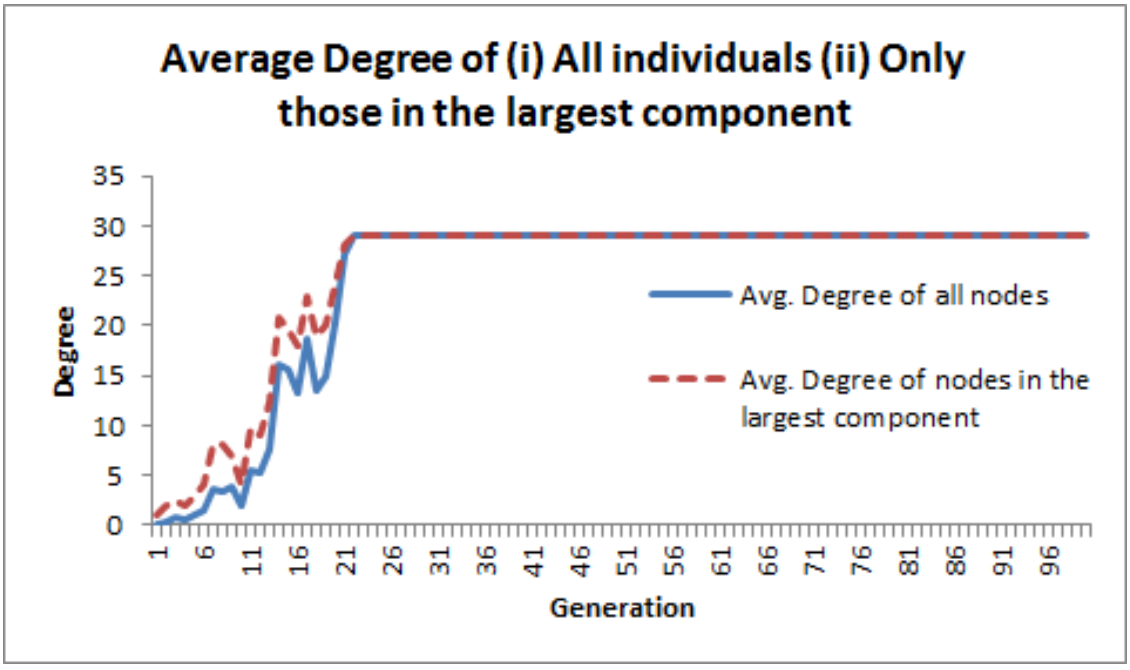


(d) Number of triangles formed – As in the previous figure, when this number approaches the maximum, again we see cliques forming.

Fig. 3.2: (...contd.) Network Metric/Measure Plots for a GA run

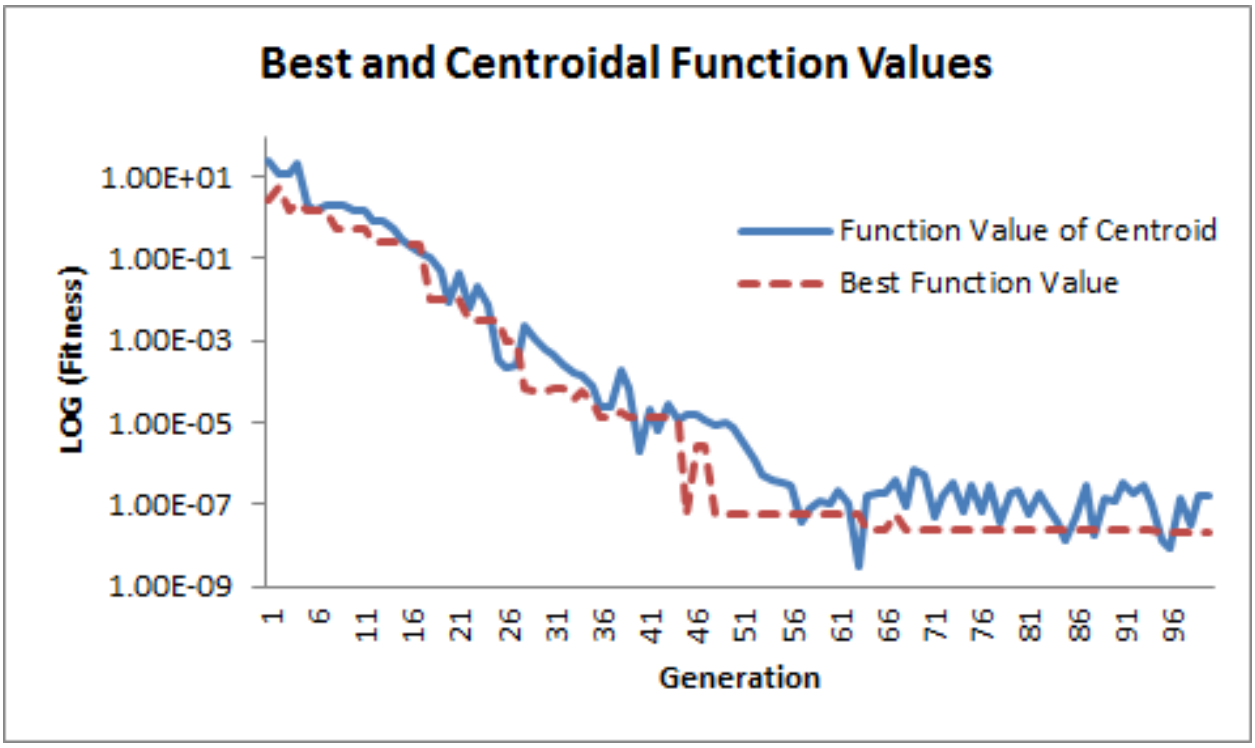


(e) Average of nodes with highest degree

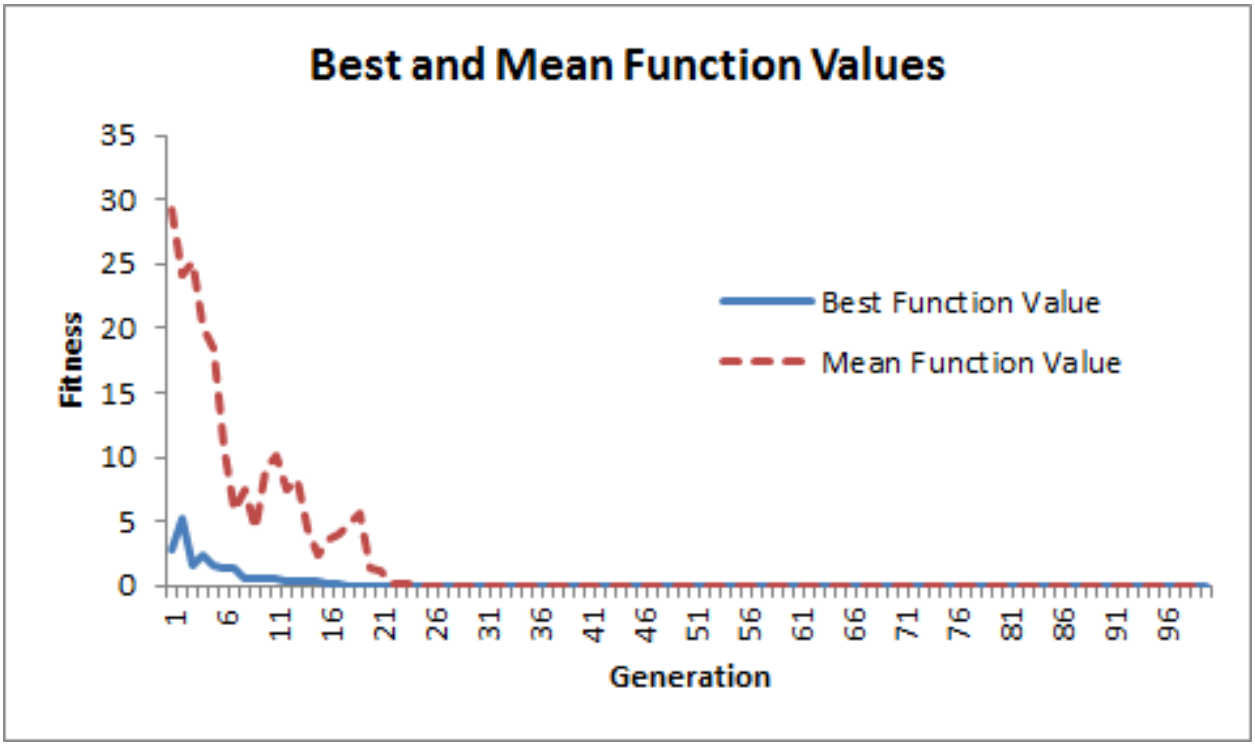


(f) Comparison: Average Degree of (i) All individuals (ii) Only those in the largest component – Typically the average degree in the largest component will be higher.

Fig. 3.2: (...contd.) Network Metric/Measure Plots for a GA run



(g) Comparison: Best Function Value and Centroidal Function Value



(h) Best and Mean function values – Included to compare other network measures with the GA.

Fig. 3.2: (...contd.) Network Metric/Measure Plots for a GA run

the population. This is likely because points are converging from different parts of the landscape, which indicates that there is a region of interest at the center of these points. Assuming that the surface is not very randomly structured (in which case, a random search could work equally well), it is likely that there are some points in the center of these converging individuals which have better fitness than the individuals explored so far. We use these observations to develop algorithms to understand better and enhance the EA, in the next two chapters.

3.4 Test problems - Optimization

We address network models of EAs, applied to minimization problems. Mathematically, a minimization problem in the real space is defined as:

For any function,

$$f : \mathbb{R}^n \to \mathbb{R} \tag{3.3}$$

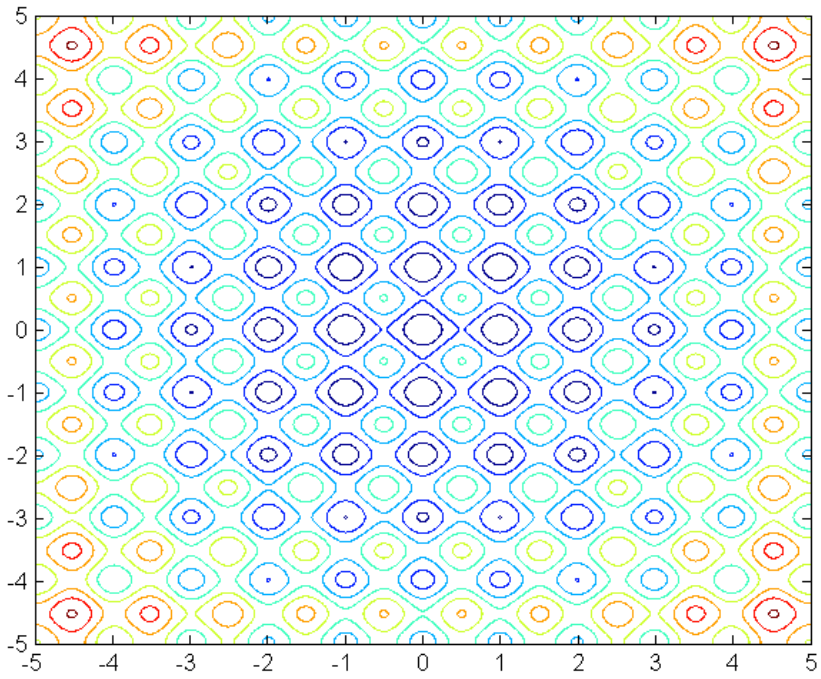
we wish to find the vector:

$$\mathbf{x}^* \in \mathbb{R}^n \text{ such that } f(\mathbf{x}^*) \leq \mathbf{x}, \quad \forall \mathbf{x} \in \mathbb{R}^n \tag{3.4}$$

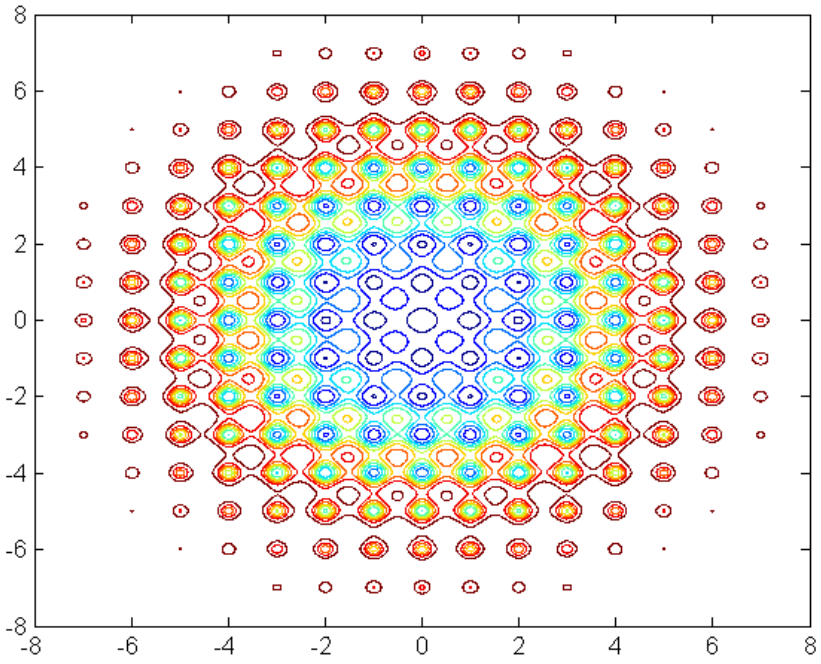
To this end, we use the following widely used benchmark test problems. Some special features of each problem are also listed and contour plots (for two-variable versions) are also shown in Figures 3.3a-3.3e, in order to visualize the main problems optimized throughout this dissertation. (All are minimization problems with a global minimum of 0.)

1. **Rastrigin's Function** [60]: This is a widely used multimodal function, having multiple local minima uniformly distributed.

$$f(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\,cos(2\pi x_i)),\ x_i \in [-5.12, 5.12] \tag{3.5}$$

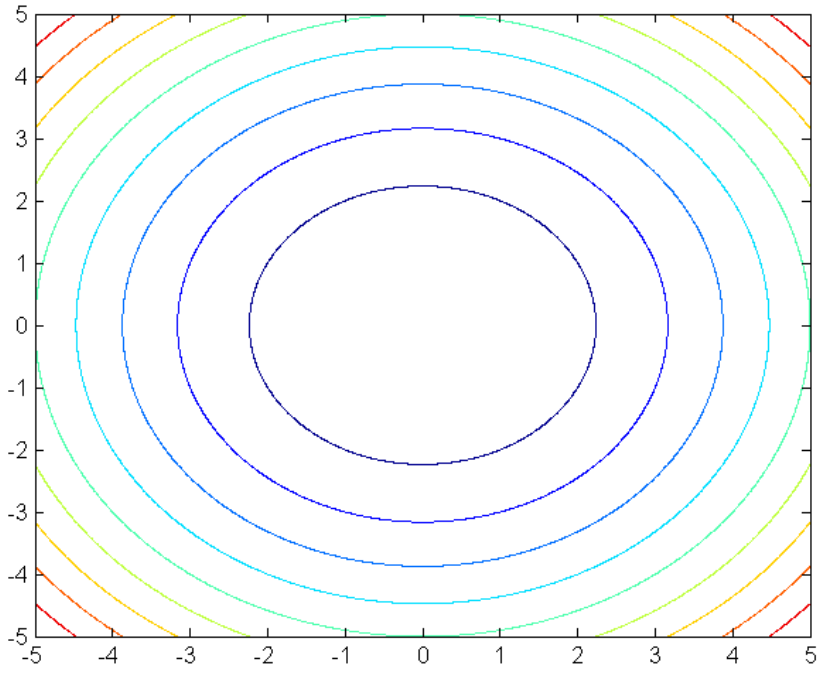


(a) Rastrigin's function

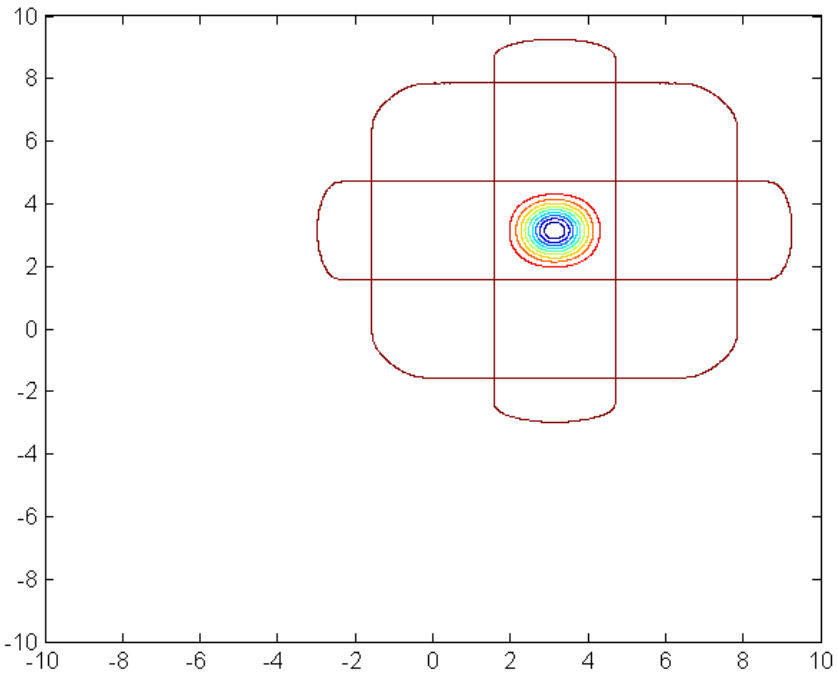


(b) Ackley's function

Fig. 3.3: Contour plots of test functions

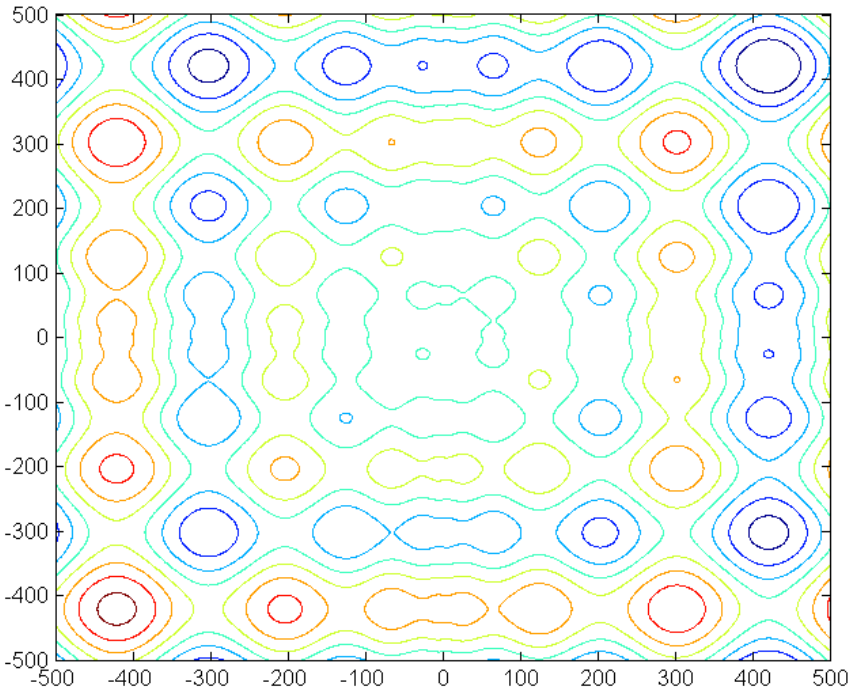


(c) DeJong's First function



(d) Easom's function

Fig. 3.3: (...contd.) Contour plots of test functions



(e) Schwefel's function

Fig. 3.3: (...contd.) Contour plots of test functions

2. **Ackley's Path Function** [1]: This is another widely used multimodal test function.

$$f(x) = 20 + e - 20\,exp\left(-0.2\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}\right) - exp\left(\frac{\sum_{i=1}^{n} cos(2\pi x_i)}{n}\right) \qquad x_i \in [-32.768, 32.768] \tag{3.6}$$

3. **DeJong's first function** [19]: This test problem is a simple unimodal sphere like function.

$$f(x) = \sum_{i=1}^{n} x_i^2 \qquad x_i \in [-5.12, 5.12] \tag{3.7}$$

4. **Easom's Function** [23]: This function is characterized by a single narrow mode.

$$f(x) = 1 - \prod_{i=1}^{n}(cos(x_i))exp\left(-\sum_{i=1}^{n}(x_i - \pi)^2\right),\ x_i \in [-100, 100] \tag{3.8}$$

5. **Schwefel's Function** [62]: This is well-known to be a tough problem, and is the toughest test problem of the five. It is a multimodal function with geographically distant local minima.

$$f(x) = \sum_{i=1}^{n}\left(-x_i sin(\sqrt{|x_i|}\right) \qquad x_i \in [-500, 500] \tag{3.9}$$

3.5 Summary

In summary, we have addressed five topics in this chapter:

1. We have shown a way to represent the EA population using a network representation of the population. We have demonstrated one way to create this network, i.e., based on Euclidean distances. In later chapters, we discuss two other kinds of networks, one based on ancestry and another on bitwise differences in an LCS. However, we emphasize that these are only a few examples of network formation, and there could be many more.

2. We have related the development of a large component to convergence. While the example shown in Figures 3.1a-3.1f serves as an observation of an interesting pattern, we will show similar results on several test problems (over many runs), in the next two chapters.

3. We then observed a pattern in how the size of the large component changes with time, and used it to predict imminent convergence. We will formalize an algorithm in the next chapter using this observation.

4. We have also identified several network measures are strongly correlated with convergence in the EA, and examining changes in these with time. We use some of these measures in the later chapters to develop different algorithms.

5. Finally, we described the benchmark test problems on which the EAs are tested. These are widely used by researchers to test EA algorithms because of the special characteristics of these problems. The set of problems was chosen such that they represent a wide variety of characteristics, such as a simple unimodal problem, a narrow unimodal problem, multimodal problems, and a multimodal problem with traps.

In the next three chapters, we describe experiments with various EAs, by defining and studying the evolution of different kinds of networks of populations. The first set of experiments involves constructing Euclidean networks in a GA, and we demonstrate that similar networks can be extended to a PSO as well.

Chapter 4
Euclidean Networks in Genetic Algorithms and Particle Swarm Optimization

In this chapter, we use the emergence of large components in a network to perform early detection of convergence in an EA. Whereas traditionally used convergence criteria in GAs remain somewhat strict since the criteria are based on properties of the best point or the mean of the entire population as a whole, the corresponding criteria in the networks domain are relaxed, i.e., the formation of a large component is observed more easily than measuring minute changes in best/mean function values. This is because an additional piece of information is available, viz., *movements of groups of individuals* towards a basin of attraction, when compared with the other typically used factors such as best and mean function values.

PSOs have different dynamics than GAs, with particles that possess position and velocity components. However, with respect to our network formation, they are similar to GAs, i.e., they have a global best individual acting as an attractor, i.e., influencing the movement of other individuals in the population. In this chapter, we explore the utility of Euclidean networks in both GAs as well as PSOs.

4.1 GAs - Largest Component and Clique Formation

In past work, the convergence based termination criteria of a GA depend on values of the best objective function value and average function value. In some instances, the criteria depends on some combination of the two, e.g., the function value has changed less than a predetermined threshold value. It could also be an upper limit on the number of generations. We propose below two algorithms, based on network metrics to detect convergence and hence terminate the GA.

4.1.1 Convergence and its relation to the convex hull volume

The pattern in the rise and fall of the convex hull area, as seen in Figure 3.2b, is used to formalize the following observation:

Observation 1 *Initially there is no significant large component. As a large component emerges, the convex hull of this component grows. After the hull swells, it shrinks in size to a much smaller volume (barring minor size fluctuations) and reaches a very small fraction of its previous highest peak. We hypothesize that as the hull shrinks while still retaining the majority of the individuals, the GA converges, and cannot be expected to do much better unless the population is significantly perturbed.*

The above observation leads us to the following algorithm.

Algorithm 1 Convergence detection based on the convex hull of the largest component

1: Initialize a population of n individuals;
2: α takes values in $[0.6, 0.9]$, ϵ takes values in $[0.005, 0.03]$;
3: **repeat** Perform recombinations and mutations to obtain next generation of GA
4: **until** number of individuals in the largest component, $\geq \alpha.n$ (This indicates that a large component has emerged.)
5: $\gamma \leftarrow$ Greatest Hull Area (observed so far) of the largest component
6: **repeat** Perform recombinations and mutations to obtain next generation of GA
7: **until** hull area $\leq \epsilon.\gamma$, and number of individuals in the largest component $\geq \alpha.n$ (This indicates that the convex hull has collapsed and further iterations are not needed.)

In Algorithm 1, α indicates the fraction of population which must constitute the largest component in order to distinguish it from other "big" components that occur by chance. ϵ is the fraction

of the largest hull area obtained to which the largest component should collapse in order to detect convergence. Hence, once the hull area goes from γ to $\epsilon.\gamma$, we recommend that the algorithm be stopped, provided the number of individuals in the component,

$$n^* \geq \alpha.n \tag{4.1}$$

Most standard GA crossovers are convex operators [51]. This observation of convexity is consistent with our view that a component forms and then shrinks, and this notion has been exploited in the above algorithm. The results are in Table 4.1. The main observation in this table is that most Δ (the difference between the function value at the generation recommended by the algorithm for stopping and the value obtained if the GA had continued until convergence) values are very close to 0 indicating very little loss in solution quality, even if the run is stopped very early.

Another alternative approach to convergence detection is triangle counting. Triangle counting also tends to be a good indicator of convergence especially since convergence implies that there is a large number of individuals connected to each other, which in turn leads to formation of cliques based on our definition of edges. We now make an observation regarding triangle formation.

Observation 2 *The count of the number of triangles has a positive trend. Initially, it remains well below the value of the total possible number of triangles, viz.,* $\binom{n}{3}$*,where* n *is the total population size, and then the count increases suddenly and is comparable to* $\binom{n}{3}$.

This phenomenon is seen in Figure 3.2d, and Table 4.1 shows an example of the results indicating that a large number of triangles are formed. An algorithm utilizing this observation and the formation of cliques is described in Section 4.2.1.

Time complexity: For the classical GA, on average, a run of the GA on Rastrigin's function (population size of 30, 200 generations) took 0.61 seconds to run. Whereas, using the proposed algorithm, stopping at Generation 48 , including computation of network metrics, took only 0.19 seconds to run. Although there is a slight overhead time per generation, stopping early clearly saves a lot of computation time without sacrificing quality (note that Δ, the difference between the

Table 4.1: Results of running algorithm 1 (Average of 30 runs). (i) δ – Edge threshold (ii) Pop – Population Size (iii) Gen – Generation at which the algorithm is stopped according to the algorithm, (iv) Δ – The difference between the function value at this stopped generation and the value obtained if we had continued the GA until convergence occurred, (v) r – Ratio of number of triangles at the stopped generation and total number of triangles possible.

		δ=**0.001**			δ=**0.005**			δ=**0.01**		
Function	**Pop**	**Gen**	Δ	r	**Gen**	Δ	r	**Gen**	Δ	r
Rastrigin	30	48	0	1	36	8.00E-5	1	35	0.00016	1
	100	44	7.43E-7	0.99994	33	2.10E-5	0.99746	33	9.10E-5	0.99913
Ackley	30	39	0	1	29	0.0003	1	27	0.0008	1
	100	39	0	0.99796	31	0.0002	0.99988	24	0.001	0.99802
Easom	30	48	1.00E-5	0.97116	41	0.00017	1	40	0.0001	0.97919
	100	35	6.00E-6	0.99054	29	0.00015	0.94589	32	6.00E-5	0.98565
DeJong	30	49	4.90E-8	1	36	1.30E-7	1	34	1.83E-6	1
	100	44	3.44E-9	0.99845	32	1.43E-7	0.99951	33	4.19E-7	0.99647
Schwefel	30	129	0	1	162	0	1	131	0	1
	100	59	0	0.99685	57	3.42E-5	1	55	0.048228	1

function value at this stopped generation and the value obtained if we had continued the GA until convergence occurred, is close to 0). This implies that the earlier termination of the GA is indeed acceptable.

Experimentally, we also found that the values of "Gen" do not differ much for $\alpha \in [0.6, 0.9]$ or $\epsilon \in [0.005, 0.03]$, they all lie within 1 or 2 generations of each other. Smaller values of α typically yield components that are too small to indicate convergence, and large values of ϵ may stop the algorithm too early. Further, we observed that it is best to keep the edge threshold (δ) low, e.g.

$$\delta \leq PopSize^{-1} \tag{4.2}$$

As δ increases, we again run the risk of stopping the algorithm too early, which may yield unacceptable values of Δ. However, we observed that as long as δ satisfies Equation 4.2, the algorithm is not very sensitive to fluctuations in the value of δ. As δ gets lower, the accuracy gets higher, i.e., Δ reaches even smaller values, although it would mean that the recommended stopping generation also increases correspondingly.

Although the approach described in this algorithm does not always yield as good a result as when the GA is allowed to converge fully, our focus is on identifying a small space in which the optimum solution lies and an "acceptable solution" is attained. If further refinement is required based on the application, local search techniques (such as hill climbing) may suffice since the global optimum zone has been identified with reasonable confidence.

In certain problems where premature convergence is likely (e.g., due to local and global optima being distant), it can be useful to restart GAs. However, waiting until the algorithm has converged can prove computationally expensive. An implementation of this idea, using early identification of premature convergence to perform multiple GA restarts, is shown in Section 4.5.

Table 4.2: Detection of Convergence in a GA – Various dimensions – Population Size 50

Dim	Function	Gen	Δ
3	Rastrigin	38.733	0.001748
3	Ackley	30.667	0.001433
3	DeJong1	41.6	3.37E-06
3	Easom	31.433	1.80E-05
3	Schwefel	103.03	0.009694
5	Rastrigin	51.2	0.027936
5	Ackley	38.067	0.009637
5	DeJong1	61.1	0.000312
5	Easom	55.567	0.000121
5	Schwefel	145.23	5.1762
7	Rastrigin	67.4	0.31953
7	Ackley	44.533	0.062387
7	DeJong1	74.7	0.000871
7	Easom	59.867	0
7	Schwefel	152.93	24.723

4.2 GAs - Identification of Imminent Convergence – Multiple Dimensions

In Tables 4.2-4.4, we list the findings of imminent convergence as per Algorithm 1 for different values of selected algorithm parameters. Table 4.5 and Figure 4.1 summarize the same for all dimensions and parameters. We work with the same set of five benchmark problems – Rastrigin's function [60], Ackley's path function [1], DeJong's first function [19], Easom's function [23] and Schwefel's function [62]. Here, we examine problem instances of higher dimensionality. The gist of the algorithm is that we identify when a component forms with a majority of the individuals, and then proceed to observe the convex hull collapse due to convex crossover behavior [51].

Tables 4.2-4.4 present results for population sizes in the range $(50 - 150)$, different dimensions $(3 - 7)$ (number of variables in the optimization problem), and for Distance Threshold value of $\delta = 0.5$ (percent of the body diagonal of the search space). In each case, the generation at which the collapse of the convex hull described before and the cost of stopping early, denoted by Δ, are listed. By cost, we mean the improvement that we end up sacrificing if we stopped at the prescribed

Table 4.3: Detection of Convergence in a GA – Various dimensions – Population Size 100

Dim	Function	Gen	Δ
3	Rastrigin	37.533	0.000392
3	Ackley	31.033	0.0013
3	DeJong1	40.367	1.94E-06
3	Easom	32.4	1.17E-05
3	Schwefel	80.967	6.52E-05
5	Rastrigin	53.033	0.008336
5	Ackley	40.233	0.004607
5	DeJong1	51.933	9.59E-05
5	Easom	60.5	6.53E-05
5	Schwefel	103.4	0.002297
7	Rastrigin	64.467	0.10989
7	Ackley	43.167	0.02399
7	DeJong1	66.7	0.000252
7	Easom	98.267	0
7	Schwefel	142.83	0.16308

Table 4.4: Detection of Convergence in a GA – Various dimensions – Population Size 150

Dim	Function	Gen	Δ
3	Rastrigin	37.5	0.000165
3	Ackley	29.467	0.00141
3	DeJong1	36.467	1.13E-05
3	Easom	39.167	9.00E-06
3	Schwefel	68.3	3.57E-05
5	Rastrigin	48.433	0.04186
5	Ackley	40.333	0.005203
5	DeJong1	55.5	3.61E-05
5	Easom	62.167	3.43E-05
5	Schwefel	97	0.000964
7	Rastrigin	62.733	0.021634
7	Ackley	42.1	0.01363
7	DeJong1	64.6	0.000187
7	Easom	101.93	0
7	Schwefel	121.33	0.023488

Table 4.5: Euclidean Networks in a GA – Frequency Distribution of Δ – Over all parameters: Dimensions $3-7$, Population sizes $50-150$, Distance Threshold, $\delta = 0.01, 0.05, 0.1$

Bins	Frequency	Cumulative %
1E-07	14	3.73%
0.00001	44	15.47%
0.001	120	47.47%
0.1	160	90.13%
1	21	95.73%
10	12	98.93%
More	4	100.00%

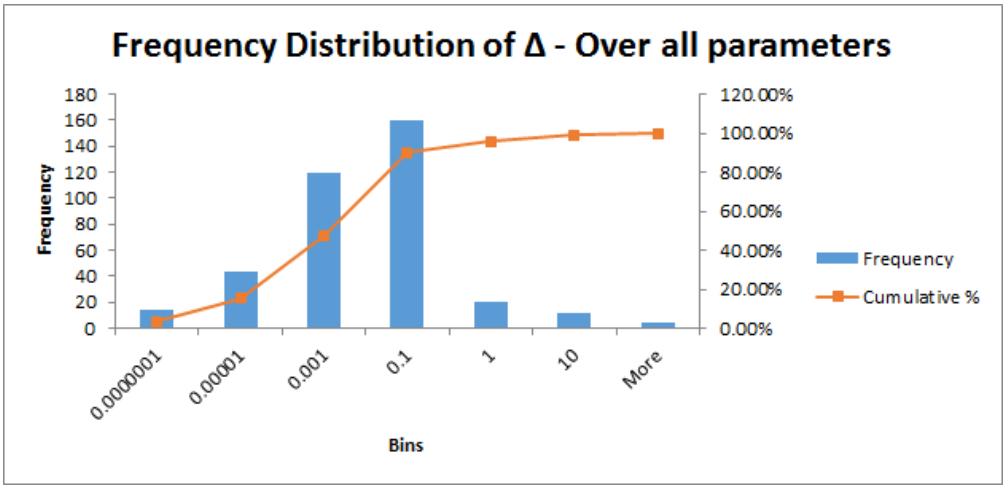


Fig. 4.1: Euclidean Networks in a GA – Frequency Distribution of Δ – Over all parameters

generation as opposed to continuing until all convergence criteria are met. This was also tested for a few other population sizes, dimensions and distance threshold values as well, but only a summary is listed here.

So for instance, row 1 of Table 4.2 should be interpreted as follows: "For population size of 50, in the 3-variable version of Rastrigin's function, stopping at Generation 39 as opposed to continuing on to Generation 200 (or the minimum function value improvement threshold), yields a function value that is 0.001748 more than the converged value, when averaged over 30 runs." This, and other rows in the tables provide evidence that exploitation is the dominant process in the later generations of the algorithm. We hypothesize that we do not need this degree of exploitation and the algorithm in Section 4.5 describes a process that continues to explore with some individuals (i.e., some fraction of the population), while leaving the remainder of the population to exploit.

4.2.1 GA convergence and its relation to clique formation

The growth of the number of edges and the number of triangles as seen in Figures 3.2c and 3.2d lead us to the following observation:

Observation 3 *The number of connected edges or triangles grows from zero (or close to zero) to the maximum possible, i.e., it grows from a completely disconnected graph structure and develops into a clique towards convergence. Once a clique is formed, the region of interest has been identified and the algorithm cannot be expected to do much better.*

This observation leads us to Algorithm 2.

Algorithm 2 Convergence based on clique formation

1: Initialize the population;
2: Edge threshold, δ takes values $< PopSize^{-1}$;
3: **repeat** Perform recombinations and mutations to obtain next generation of GA
4: **until** A complete clique containing $PopSize$ nodes is formed.

The results of Algorithm 2 are tabulated below (in Table 4.6). Here $\delta = 0.001$ which is much less than all the population sizes used for experimentation. However, we do not dwell on this

Table 4.6: Results of running algorithm 2 (Average of 30 runs). (i) Pop – Population Size (ii) Gen – Generation at which the algorithm is stopped according to the algorithm, i.e., the first occurrence of a clique (iii) Δ – The difference between the function value at this stopped generation and the value obtained if we had continued the GA until convergence occurred.

		Rastrigin	Ackley	Dejong	Easom	Schwefel
Pop=100	Gen	48	42	57	45	62
	Δ	2.59E-7	0	2.73E-10	0	0
Pop=30	Gen	48	37	47	70	129
	Δ	0	1.00E-4	6.50E-8	0	0
Pop=10	Gen	69	63	114	70	Clique not formed
	Δ	4.00E-4	1.32E-2	5.25E-5	6.10E-4	-

algorithm for too long since we note that there could be instances of the clique not forming at all depending on the problem being solved. Another reason that we do not examine this further is the high complexity of this approach, i.e., the identification of triangles, which involves examining every triplet of points. This would lead to a computational complexity of $O(n^3)$ which would be an issue at higher population sizes.

4.3 GA and Centroid of the Largest Component

In Figure 3.2g, we observe that the function value at the centroid of the largest component is occasionally better than the best individual in the population. This observation is exploited to improve the GA, and is summarized below:

Observation 4 *The individuals of a population often converge from different directions into the optimum and hence the centroid of the largest component, which lies in the middle of this component, often has the best objective function value amongst all individuals. In other words, the largest component identifies the region of interest and its centroid may have a better value than all of the individuals in the population.*

This leads us to Algorithm 3, which treats this additional point as an individual and evaluates its fitness.

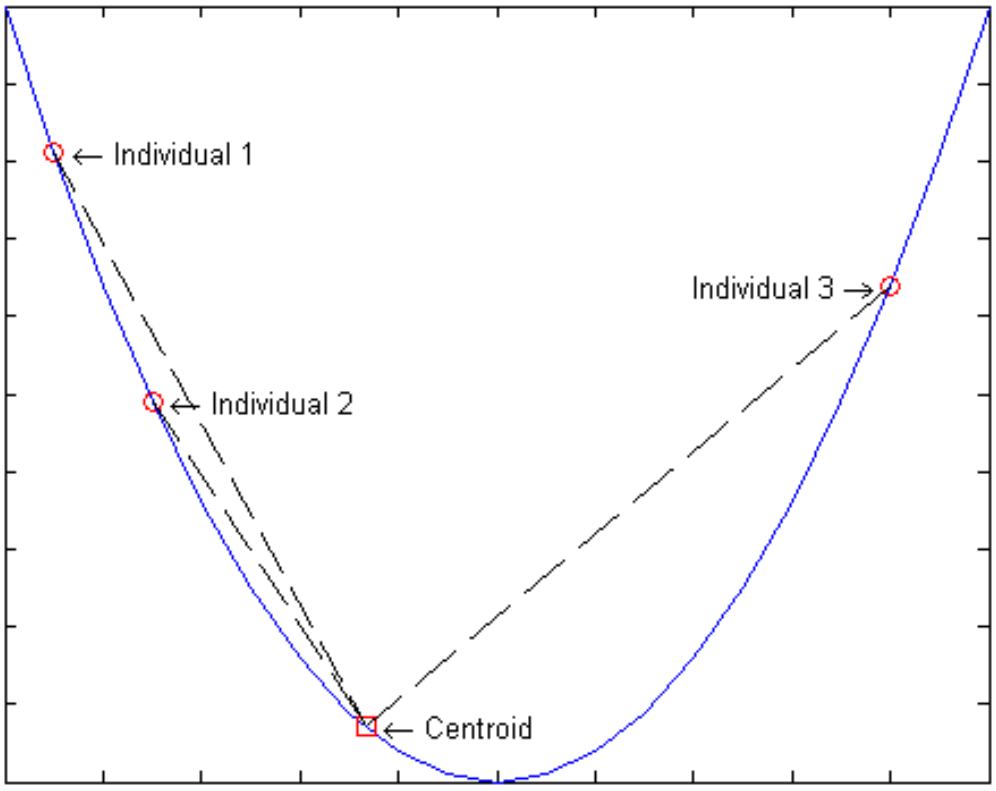


Fig. 4.2: Population around a global minimum – Additional point to evaluate

The additional point that we evaluate is shown in Figure 4.2. The points 1,2, and 3 lie in different directions from the global minimum. Hence, it can be expected that the point at the center (center of gravity) of these individuals is often better than all three of them.

This algorithm is not the same as multi-parent recombination [25]. Selection of parents in multi-parent recombination systems is non-trivial, since it is necessary to decide the number of parents, select them, and decide how to recombine them. Our approach can, however, be viewed as a network-based deterministic way to identify parents for *one* multi-parent recombination per generation.

The emphasis of this algorithm is on the *approach* of the population towards the optimum, rather than just the position of individuals. Essentially, we are betting in favor of eliminating the least fit point so that we can keep the centroid, provided the centroid is fitter than the best individual, i.e., we include one additional point in the exploitative set and one less in the exploratory set. Table 4.7 shows that in 22 out of 25 cases, the modification yields a better function value. The

Algorithm 3 Centroid of the Largest Component

```
Initialize the population;
repeat Compute function value of centroid of the largest component
    if There is more than one largest component then
        Pick one of these components at random;
    end if
    if Function value of centroid is better than the function value at all other points
     in the population then
        Delete worst individual in population;
        Introduce new individual at centroid;
    end if
    Perform recombinations and mutations to obtain next generation of GA
until Number of generations ≥ Maximum number of generations, or convergence occurrence,
 whichever is earlier
```

t-test compares best values obtained at the $100th$ generation for 30 runs of both the regular and modified algorithms.

Table 4.7 also includes cases in which premature convergence[1] may occur. This is included to show that the added exploitation does not cause any more cases of premature convergence than before. In Table 4.8, the results of running the above algorithm for 30 cases of regular convergence are shown.

We note from Table 4.8 that the additional exploitation performs better in general, or at least no worse when the GA approaches convergence. Hence, the above two tables illustrate that (i) this algorithm does not increase the possibility of premature convergence, and (ii) it achieves a better solution, in general.

4.4 A Restart Policy for GAs

In problems where premature convergence is likely (due to local and global optima being distant), it can help to restart GAs, i.e., reinitialize a population, and begin from scratch. However, waiting until the algorithm has converged can prove computationally expensive.

We propose the following approach: using algorithm 1, identify convergence. Once the al-

[1]The convergence of an algorithm at a local optimum that is not the global optimum.

Table 4.7: Best function values at various generations using regular GA and Algo 3 (Avg. of 30 runs). Pop=30 (Reg: Regular GA, Mod: Modified using Algo 3)

	Rastrigin		**Ackley**		**Easom**		**DeJong**		**Schwefel**	
Gen	**Reg**	**Mod**	**Reg**	**Mod**	**Reg**	**Mod**	**Reg**	**Mod**	**Reg**	**Mod**
10	1.1230	0.979	0.268	0.224	0.854	0.933	0.007	0.009	116.94	76.869
30	0.763	0.531	0.115	0.085	0.437	0.241	3.99E-6	2.42E-5	97.03	58.508
50	0.763	0.531	0.114	0.085	0.433	0.2	9.05E-9	4.01E-9	88.06	46.281
70	0.763	0.531	0.114	0.085	0.433	0.2	2.36E-10	9.98E-11	85.289	40.273
90	0.763	0.531	0.114	0.085	0.433	0.2	1.68E-10	1.98E-11	84.373	37.009
t-test (p-values)	0.0004		0.2815		0.0252		0.0001		0.0218	

Table 4.8: Best function values at various generations using regular GA and Algo 3 (Avg. of 30 runs). Pop=100 (Reg: Regular GA, Mod: Modified using Algo 3)

	Rastrigin		**Ackley**		**Easom**		**DeJong**		**Schwefel**	
Gen	**Reg**	**Mod**	**Reg**	**Mod**	**Reg**	**Mod**	**Reg**	**Mod**	**Reg**	**Mod**
10	0.166	0.244	0.061	0.052	0.002	0.0009	0.4	0.366	9.658	3.108
30	4.28E-5	0.0002	0.0002	0.0002	4.83E-7	1.99E-7	6.87E-5	0.001	0.886	0.036
50	1.01E-7	3.15E-7	6.45E-6	3.33E-6	5.51E-10	6.8E-10	4.36E-8	4.88E-7	0.0003	2.83E-5
70	1.30E-8	3.91E-9	0	0	1.07E-10	3.28E-11	3.48E-10	1.39E-10	2.55E-5	2.55E-5
90	4.72E-9	1.55E-9	0	0	3.4E-11	1.38E-11	8.33E-11	2.95E-11	2.55E-5	2.55E-5

gorithm has converged, change the function values of the landscape in and around the identified optimum to a very large value (or infinity), and restart. The algorithm will now avoid such a "blanked out" zone and continue to explore other areas. Continuing thus, the algorithm is more likely to identify the global optimum in future generations.

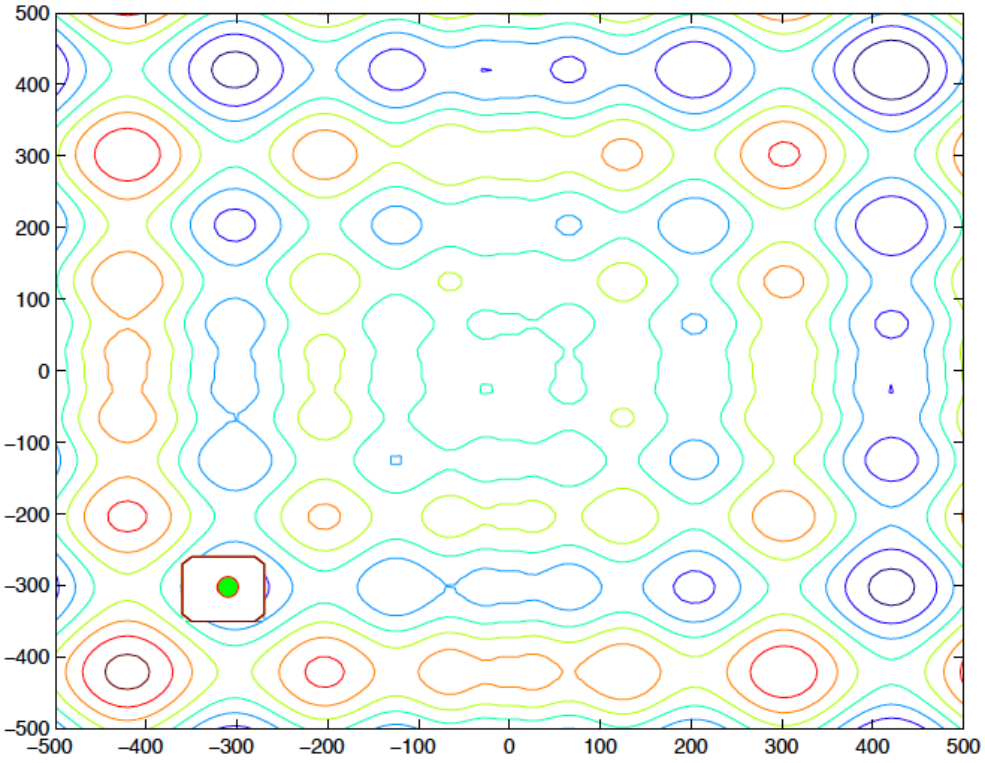
The above approach leads to the algorithm below. This is a consequence of Algorithm 1 which is used to identify if the algorithm is headed towards a convergence. If we were to wait until convergence is reached, we will not be in a position to maximize the benefits of using a restart procedure.

Algorithm 4 GA Restarts based upon early identification of convergence

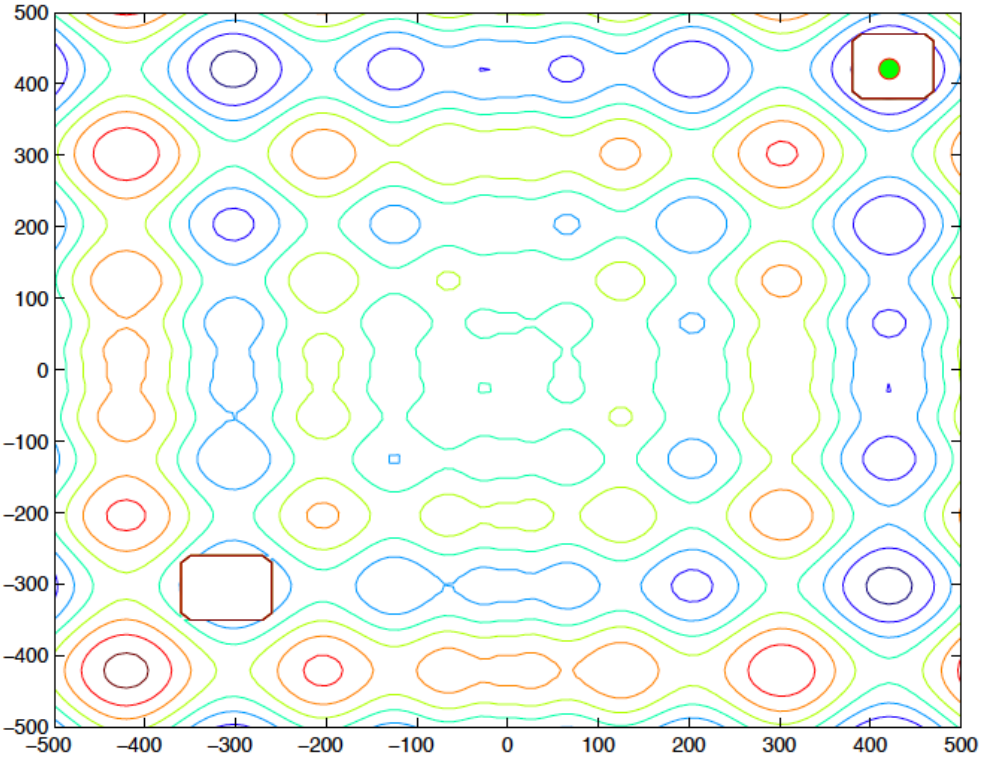
1: $\omega \leftarrow \infty$
2: **repeat**
3: Initialize the population;
4: Use Algorithm 1 to identify the vector x^* where the algorithm is likely to converge;
5: Significantly diminish the fitness of points surroundingx^*;
6: $\omega \leftarrow min(\omega, f(x^*))$
7: **until** Restart limit is reached
8: Return ω

In Figures 4.3a-4.3j, we present the results of applying this algorithm to the Schwefel function [62]. This is a multimodal function with distant local optima, one of which is the global optimum. Hence, many algorithms can be trapped in the local optima. For the two-dimensional case, this function has 4 geometrically distant local optima (and several smaller local optima), at regions close to the four corners of the search square, with one of them being the global optimum with a value of 0, at $(420.9867, 420.9867)$. We see that this is reached after a restart. The algorithm is restarted a few more times in case there is a better optimum found, we note that in the process it identifies other local optima.

The shaded (green) circle shows the algorithm identifying the optimum. The square areas are the "blanked out" spaces. These areas are very poor by way of function value, and hence the algorithm is forced to explore other areas. This continued exploration increases the probability of finding the global optimum zone. For further refinement, results may be improved by either a GA restart seeding the population around the blanked out zone with best function value, or a local

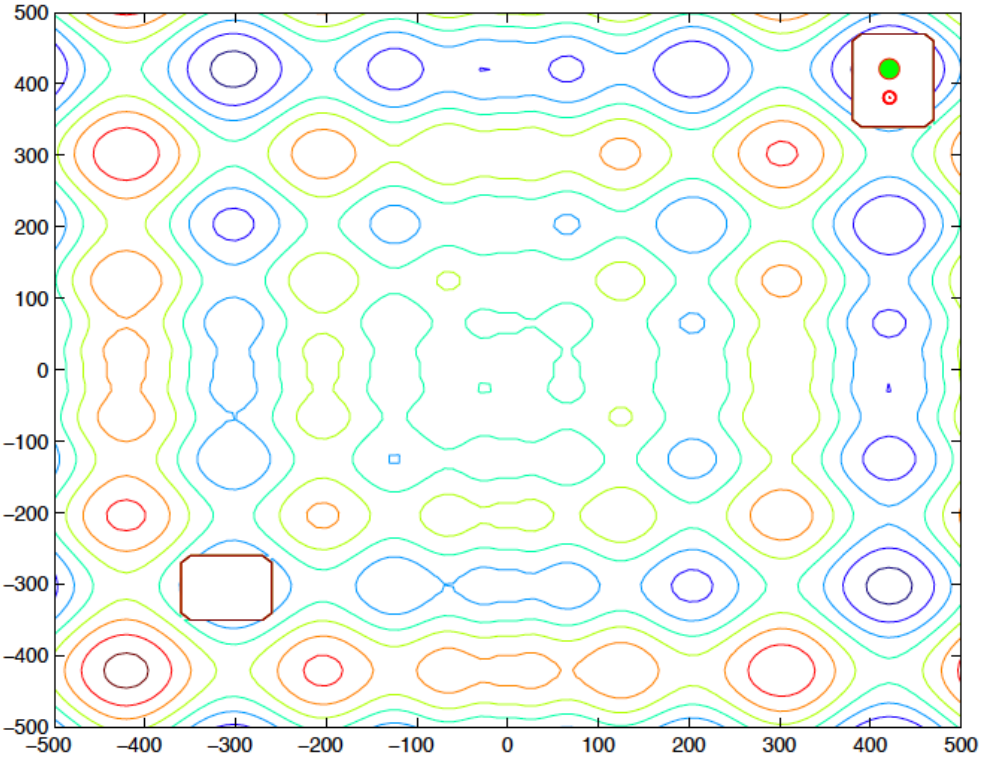


(a) Premature Convergence

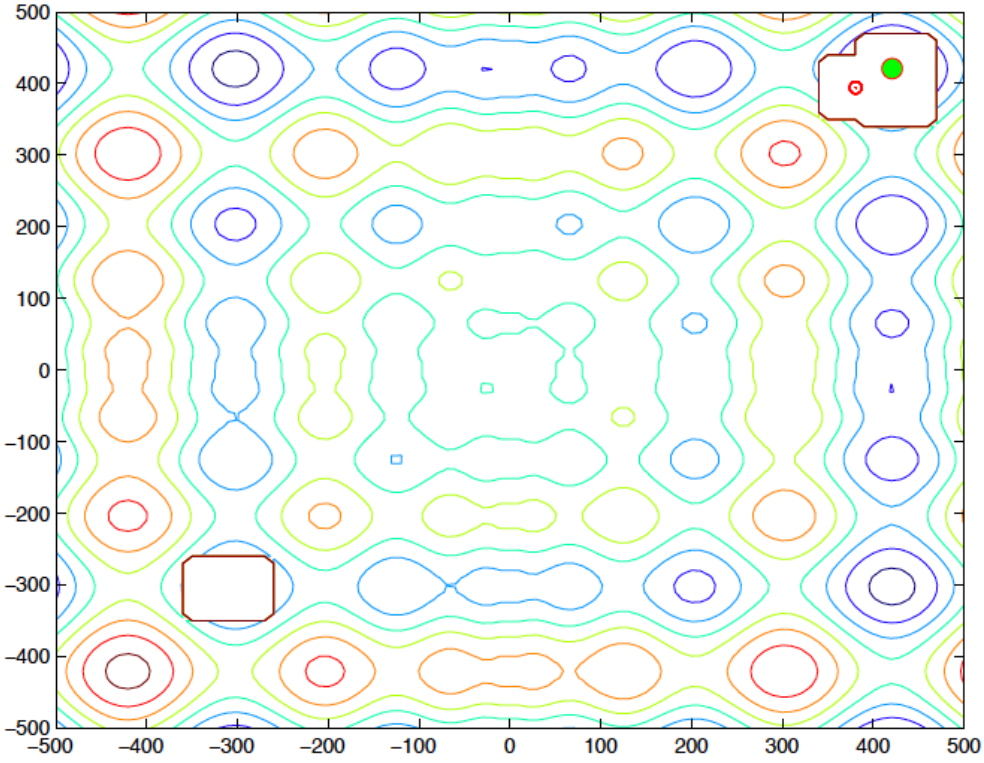


(b) After a restart – Global optimum identified

Fig. 4.3: Restart Policy

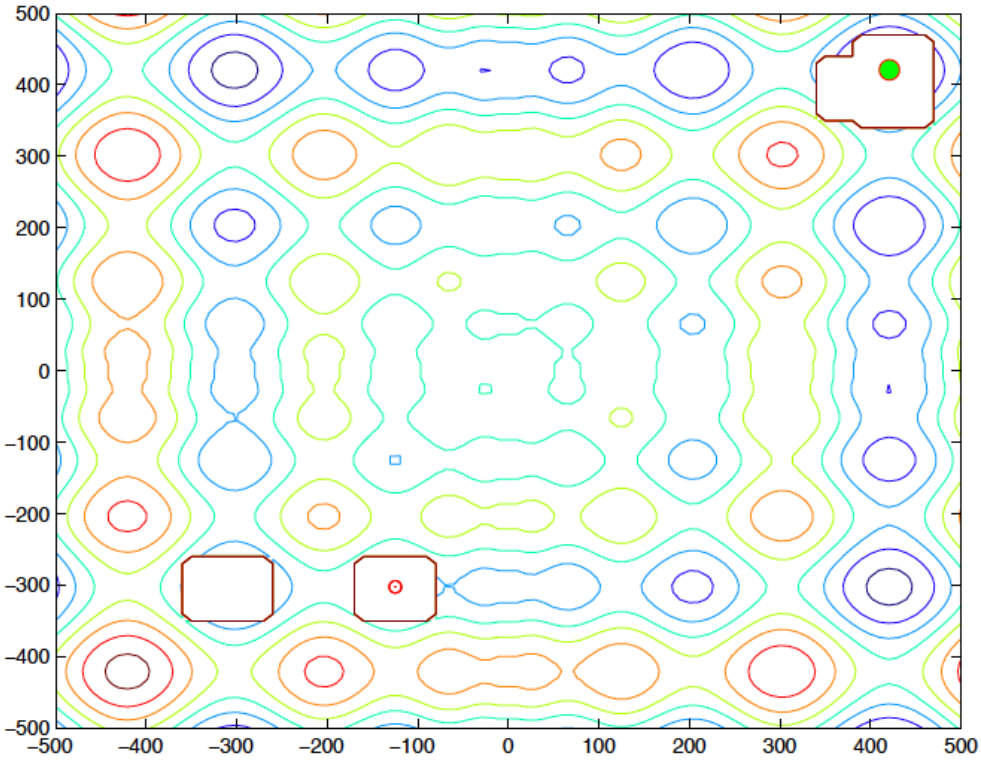


(c) After another restart – Convergence at local optimum

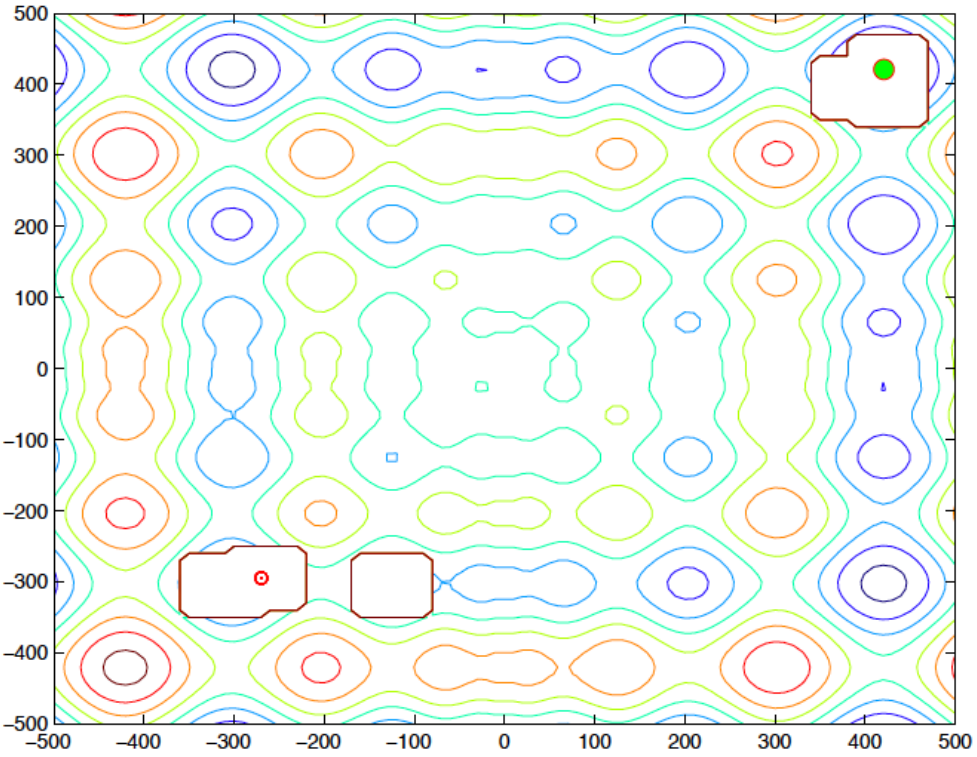


(d) Restart 3

Fig. 4.3: (...contd.) Restart Policy

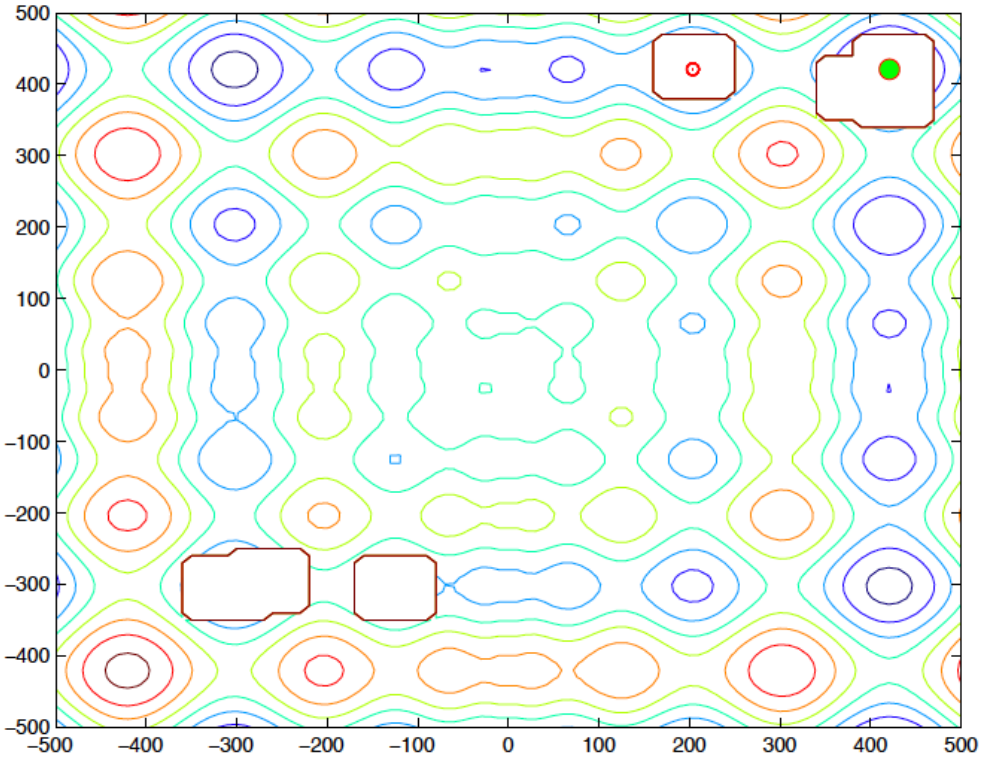


(e) Restart 4

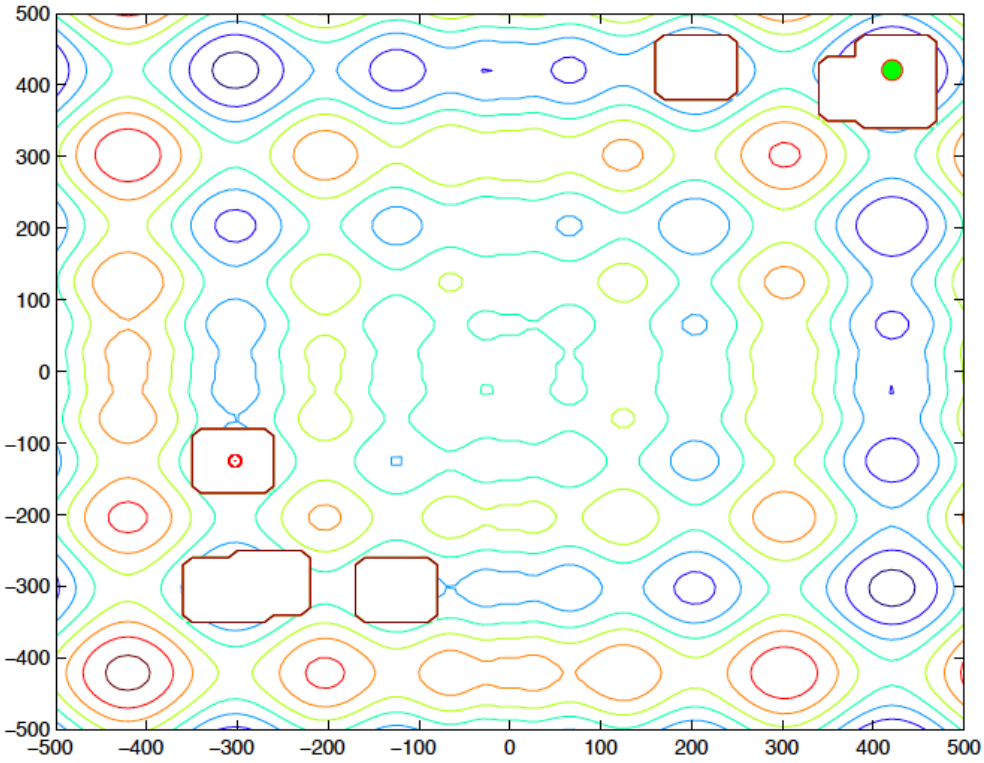


(f) Restart 5

Fig. 4.3: (...contd.) Restart Policy

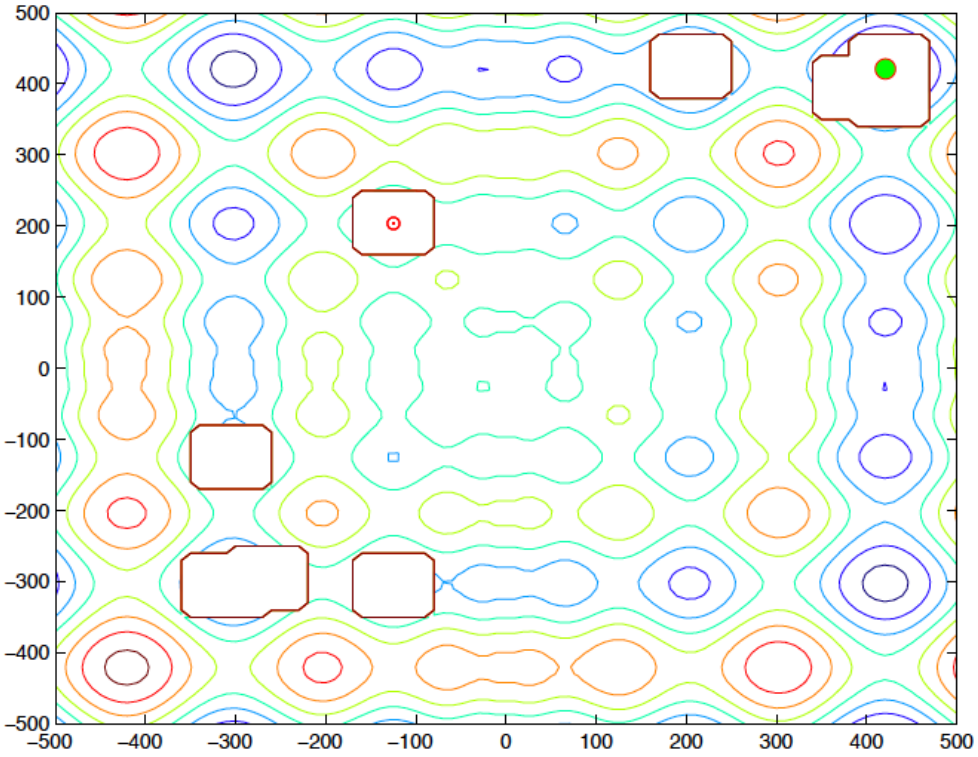


(g) Restart 6

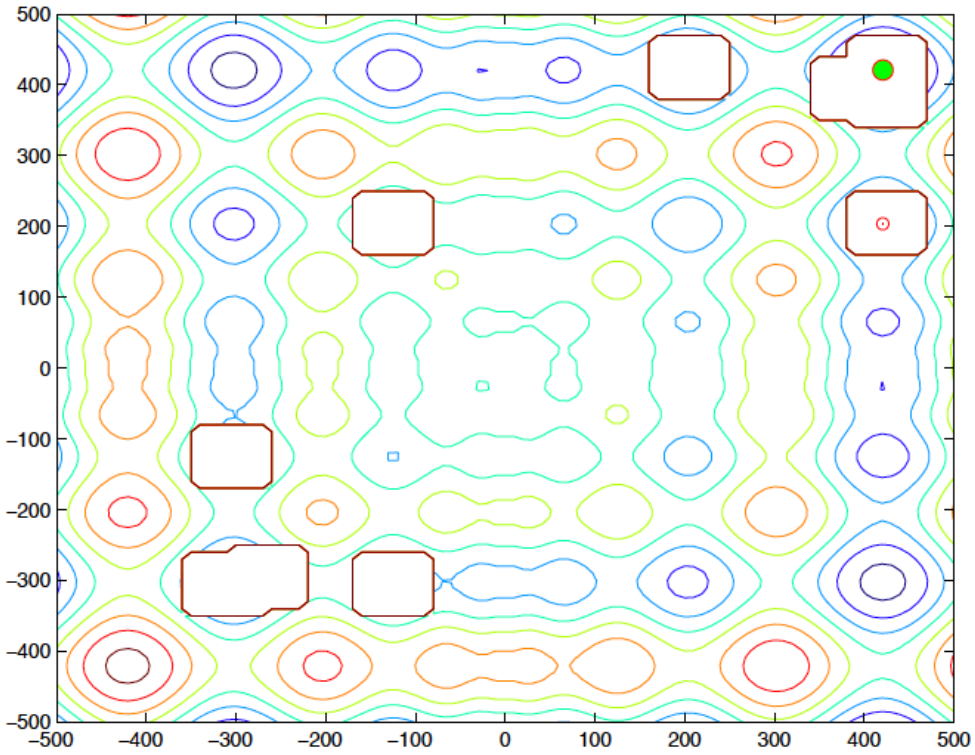


(h) Restart 7

Fig. 4.3: (...contd.) Restart Policy



(i) Restart 8



(j) Restart 9

Fig. 4.3: (..contd.) Restart Policy

search of the area.

If the region around the identified optimum is not "blanked out", then it is possible that the algorithm could get caught in the same local minimum again. Due to the creation of a very poor fitness zone, the same region will not be explored again, leading to exploration of other spaces. The number of restarts also depends on the number of potential basins of attraction. However, in these experiments, we have set the number of restarts to 10, so that most local minima can be identified and not re-explored.

However, there is one major problem with this restart model. We do not know how to define the ideal dimensions of the poor fitness regions. If we have *a priori* information regarding the surface, which makes it problem dependent, we can perform this method of restarting, but in many cases, we may not have this information. Hence, we approach it from a different perspective as seen in the next section. In this policy, we do not require any additional information about the search space.

4.5 Another Reseeding Policy for GAs

Waiting for the "collapse" of the convex hull is essentially over-exploitation. Hence, as soon as the largest component (with a majority of the population forming it) is formed, we reseed a portion of the component to force exploration of other spaces. This leads to interesting phenomena as shown in Figures 4.4a-4.4k, and the events are summarized in Table 4.9. In other words, in our greedy reseeding policy, we do not wait for the collapse to happen, we wait for the large components to form and then reseed immediately.

The details of the procedure that we recommend to solve complex problems such as the Schwefel function [62], i.e., to determine *when* to reseed, is described in the next algorithm.

Algorithm 5 sketches the reseeding procedure. This algorithm determines when to reseed, but the process of reseeding itself is done randomly over the entire search space. Note that when the reseeding happens, only some individuals from the largest component are chosen to be reseeded.

Individuals that are not part of the component are left alone, since they are still exploring and have not yet been drawn to the basin of attraction.

Algorithm 5 A Reseeding Policy Based on Early Detection of Convergence

Initialize the population;
$n \leftarrow$ Population Size;
$\delta \leftarrow$ Edge Threshold, i.e., the fraction of the body-diagonal of the search hyperspace that determines whether an edge exists or not;
$\alpha \leftarrow$ Fraction of components in order to identify a large component;
repeat Perform recombinations and mutations to obtain next generation of GA
if number of individuals in the largest component $\geq \alpha.n$ (i.e., a likely convergence zone) **then**
Retain a representative fraction (ω) of the component
Allow the remaining population to search the space, i.e., reseed them randomly in the entire search space and include them in the mating pool.
end if
until Number of generations $\geq$ Maximum number of generations, or convergence occurrence based on changes below the set thresholds for mean/best fitness or genotypic similarity, whichever is earlier

Our typical experimental settings were: $\alpha \in [0.6, 0.9]$, $\delta \in [0.01, 0.1]$, $\omega \in [0.5, 0.8]$ and results have been found to be insensitive to parameter values as long as they are in these ranges. These parameters can take values from a wide range without significantly affecting the algorithm's performance.

Figures 4.4a-4.4k show the gradual progression of the effect of reseeding while solving the Schwefel's function in two variables for easy visualization. The distances between the local optima of this function result in the traditional GA converging prematurely without reaching the global optimum. We use this function mainly to test our reseeding mechanisms in various experimental settings. In these figures, we demonstrate how a large component forms around one of the local optima, then moves to a better fit region, albeit still not the global one, and finally settles on the global optimum.

The background in Figures 4.4a-4.4k is the contour plot of Schwefel's function. The foreground consists of red circles which are the individuals, red dashed lines show connections in the largest component, and black dotted lines connect vertices within smaller components. The

Table 4.9: Summary of results observed in Figures 4.4a-4.4k

Generation	Observation
11	Large component formed, convergence headed towards local optimum near $(-500, 500)$
12	Reseeding
14	Identified new region of interest (near $(500, -500)$), although most of the population is still around the previous optimum
19	Another large component formed, identified a better fit region near $(500, -500)$, genetic operators drive population near this optimum
20	Reseeding
33	Large component forms again; population still gravitates back towards $(-500, 500)$. The reseeding has not yet found the global optimum region.
34	Reseeding
51	Now some individuals are around the $(500, 500)$ region, containing the global optimum (which is at $(420.97, 420.97)$)
58	Large component forms once more; the high-fitness around that region causes the genetic operators to drive the population towards $(500, 500)$
59	Reseeding
69	Yet another large component forms – since the fitness around the top-right corner is the best there is, the population will continue gravitating toward this over and over again with a very high probability

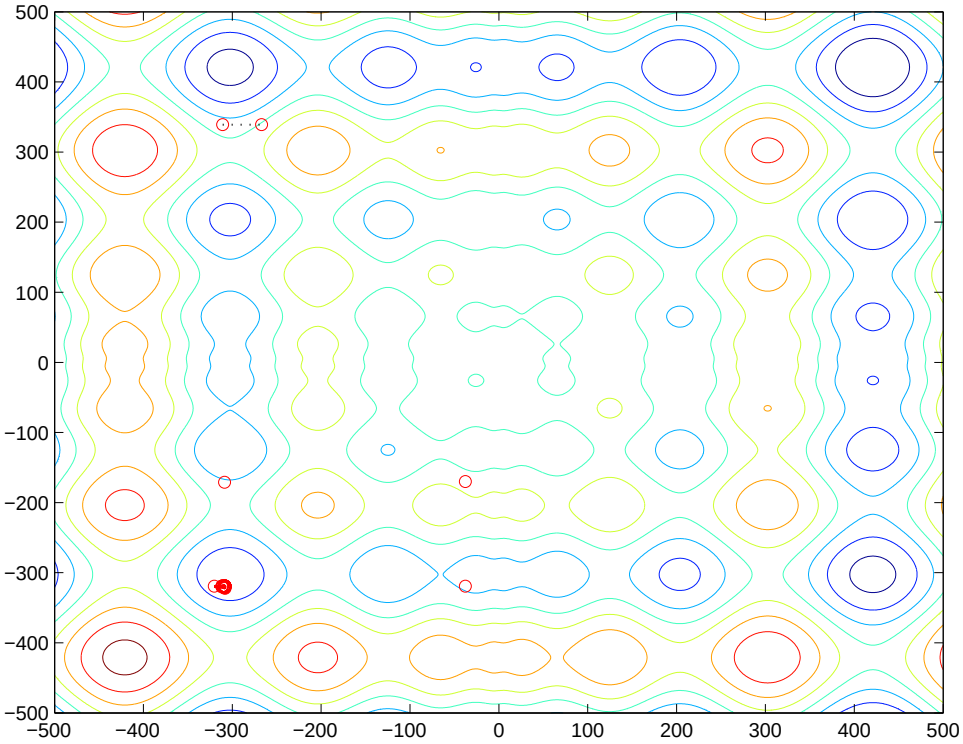
observations from these figures are summarized in Table 4.9.

4.5.1 Observations

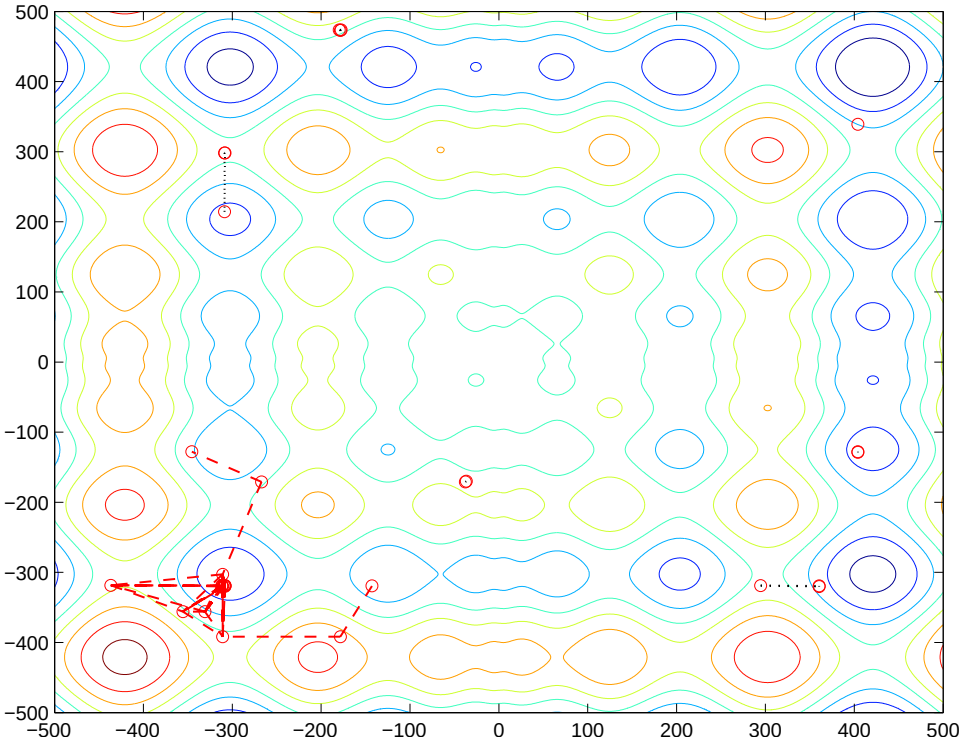
Tables 4.10, 4.11 and 4.12 contain a summary of results obtained while running Algorithm 5 for different settings. We observe the function value at Generations $10, 50, 90$, i.e., early, middle and late in evolution, for an unseeded version, $20\%, 30\%$ and 40% of the population in the largest component replaced. The tables show the following results:

1. Table 4.10 – Different Population Sizes, all values are averaged over all Dimensions and Edge Thresholds (δ).

2. Table 4.11 – Different Dimensions, all values are averaged over all Population Sizes and Edge Thresholds.

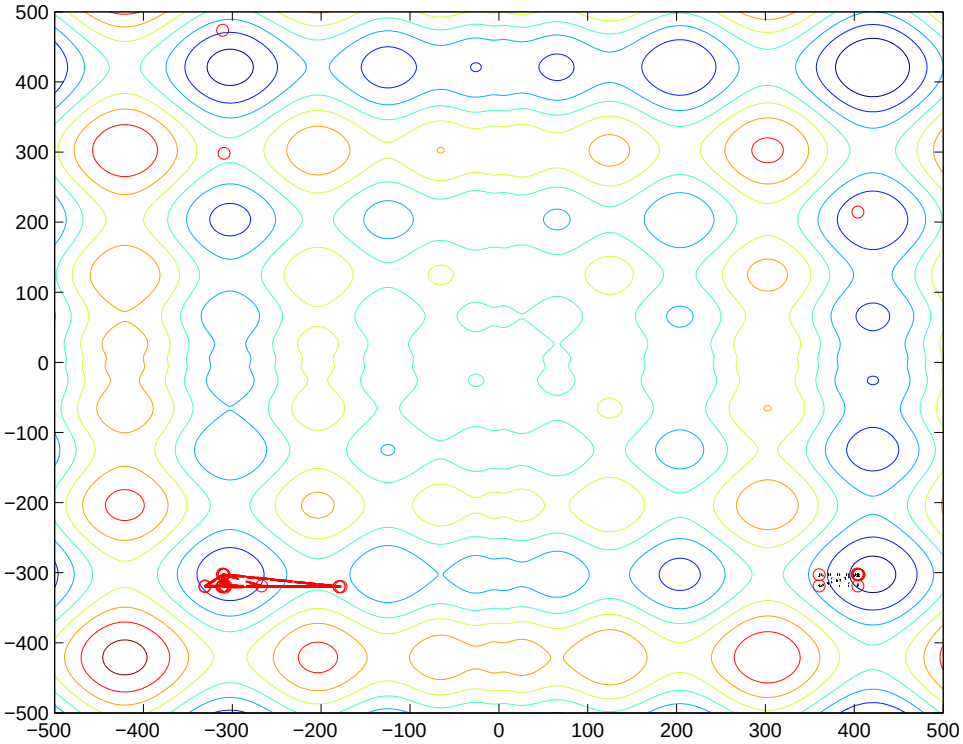


(a) Generation 11: Convergence appears to be heading to the bottom left local optimum.

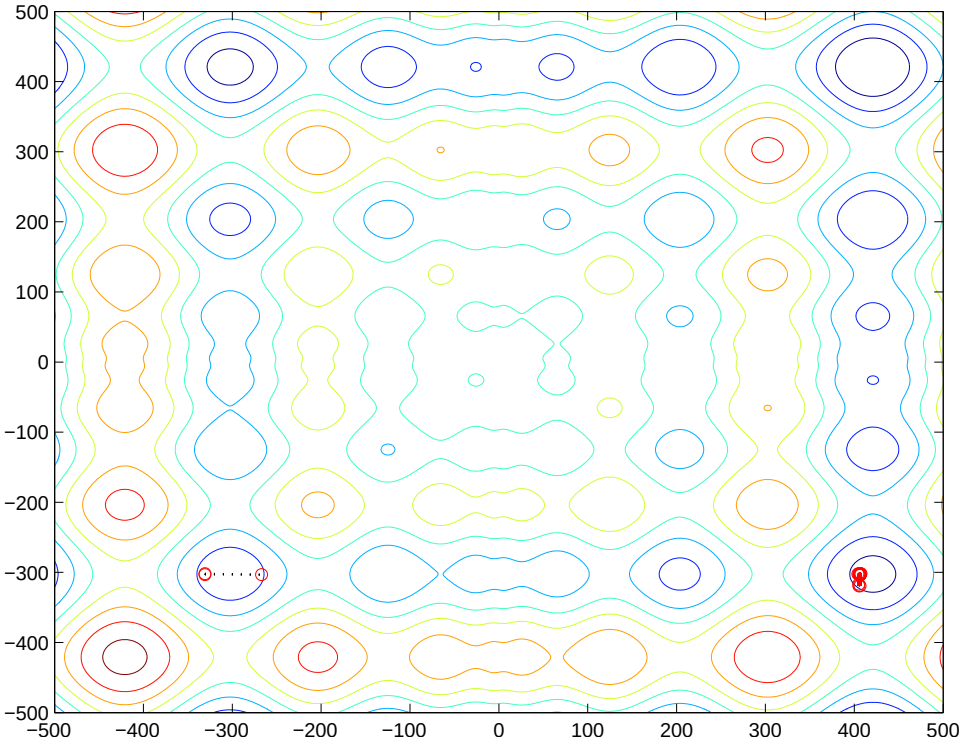


(b) Generation 12: Reseeding

Fig. 4.4: Reseeding Policy

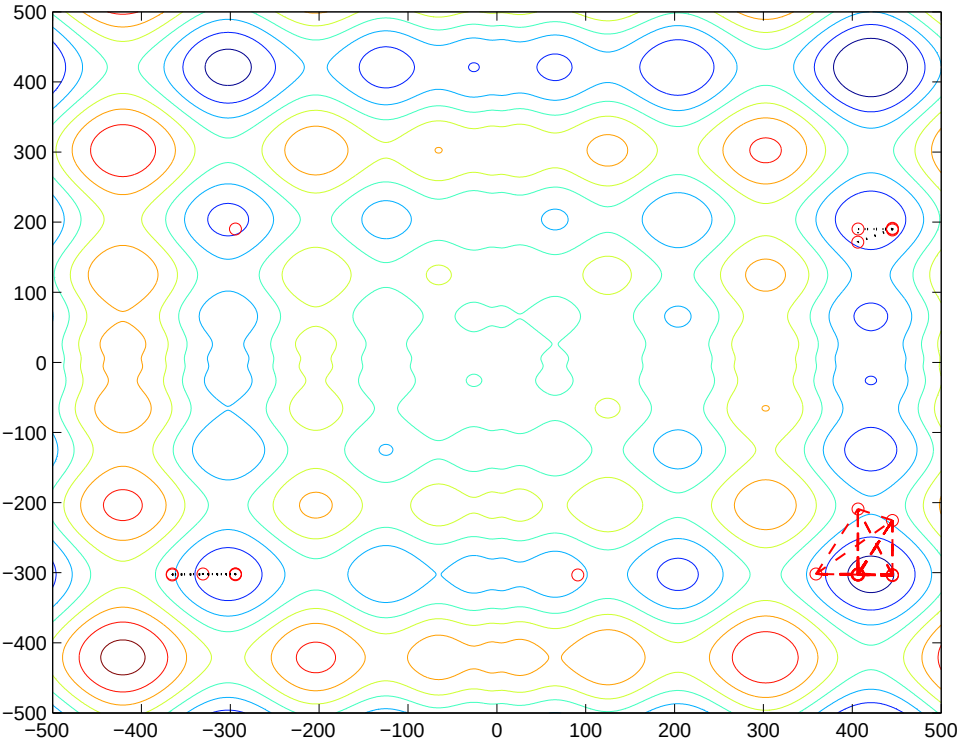


(c) Generation 14: Identified a new local region of interest (bottom-right), although most of the population is still around the previous optimum.

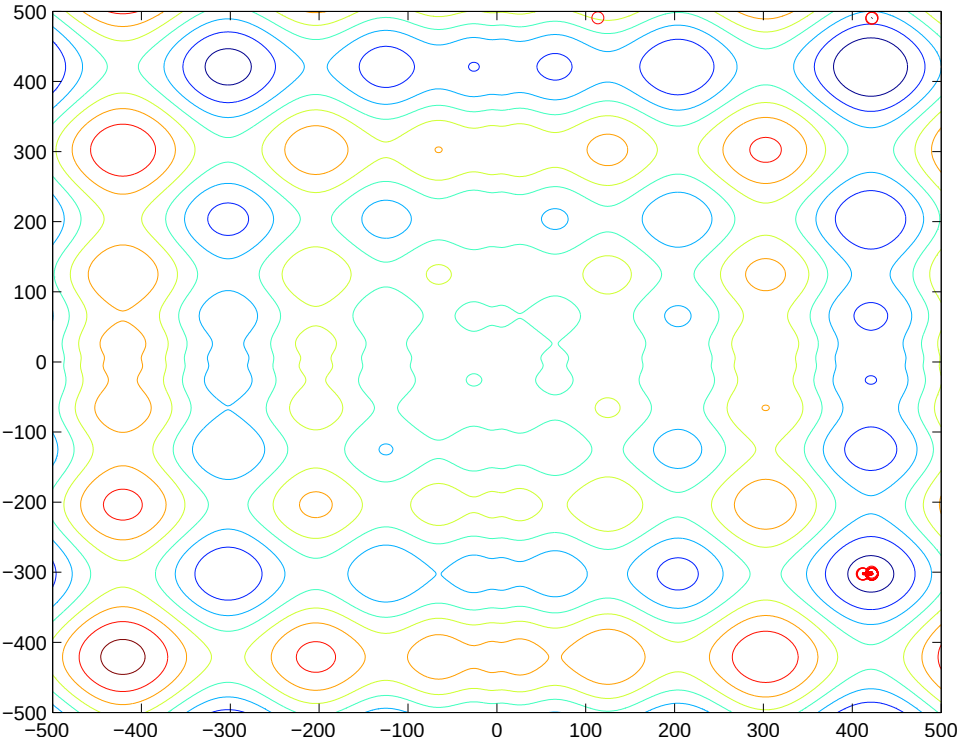


(d) Generation 19: Being a better fit, individuals crowd around the new optimum.

Fig. 4.4: (...contd.) Reseeding Policy

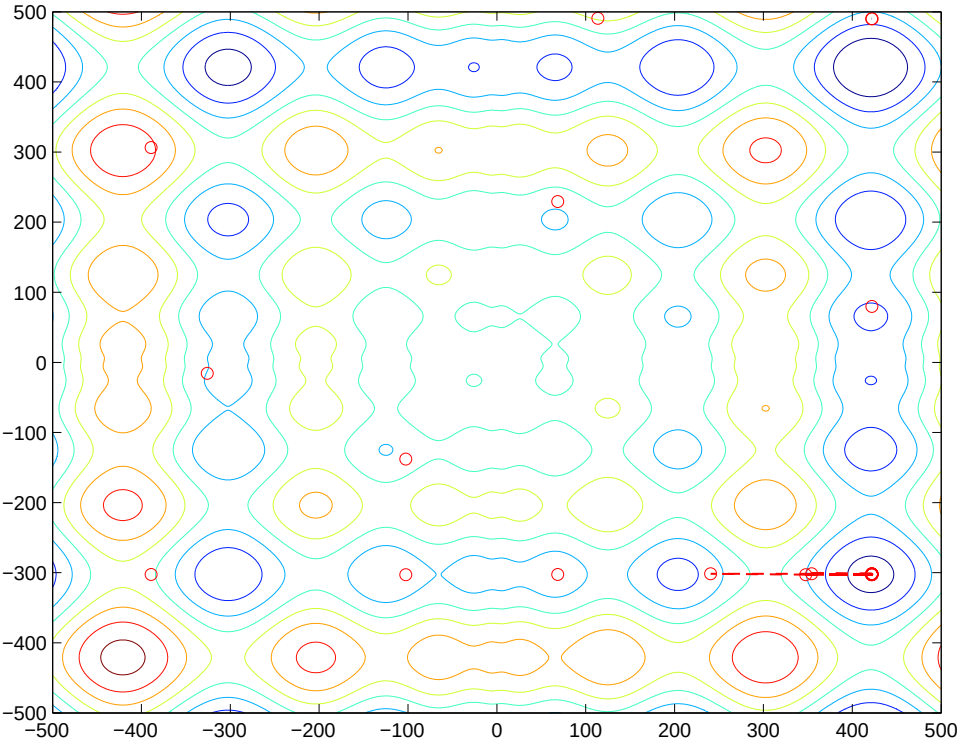


(e) Generation 20: Reseeding.

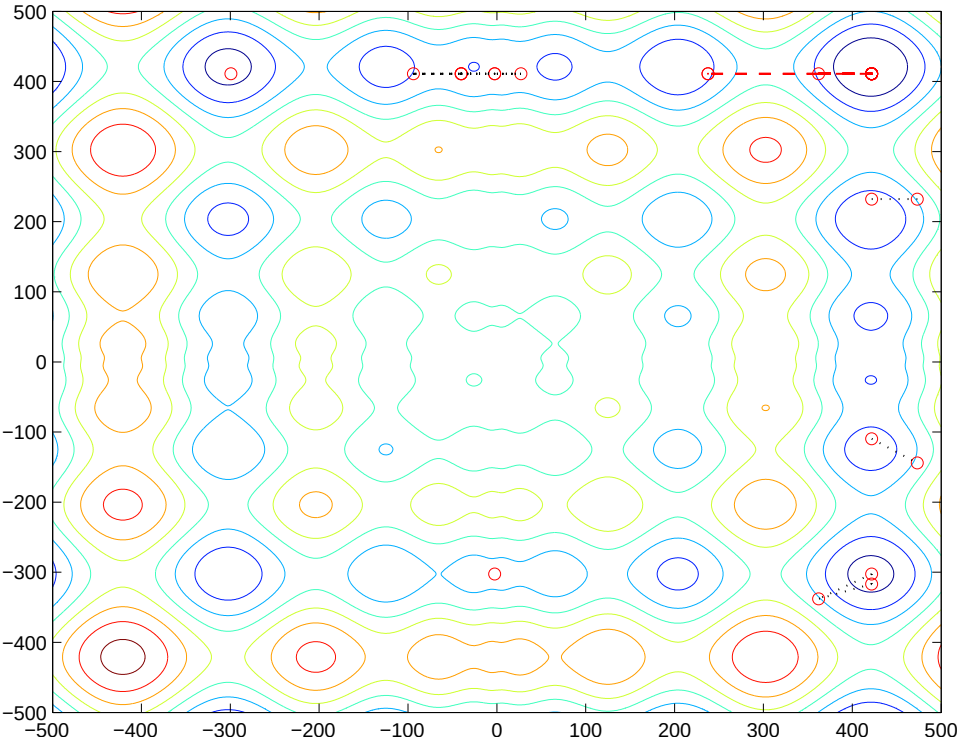


(f) Generation 33: Population still appears to gravitate back towards the bottom right optimum.

Fig. 4.4: (...contd.) Reseeding Policy

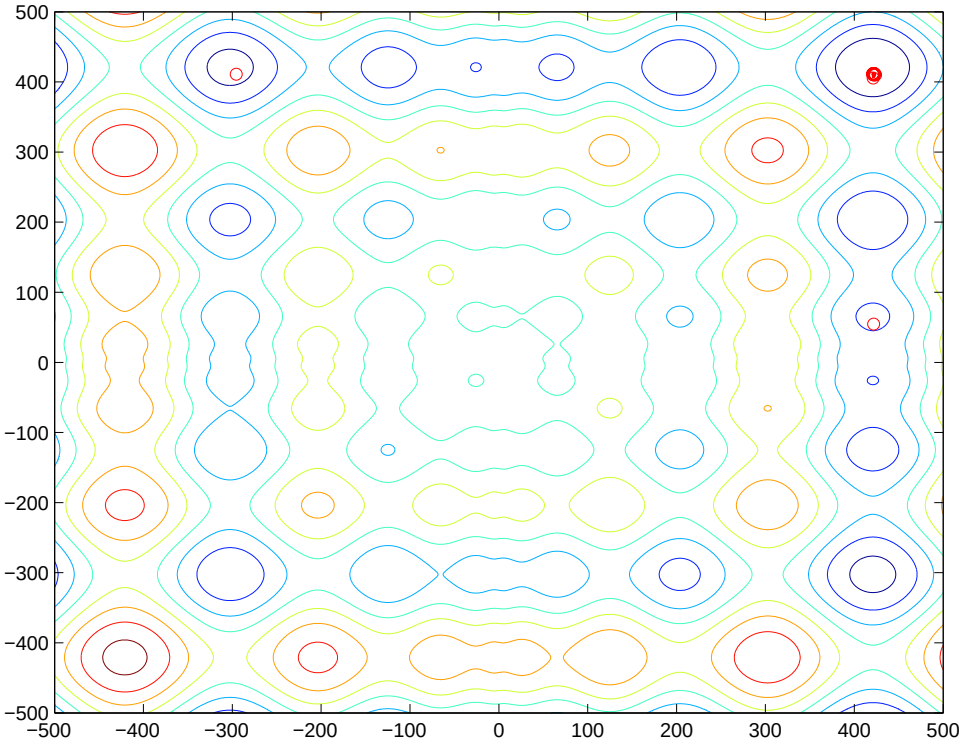


(g) Generation 34: Reseeding.

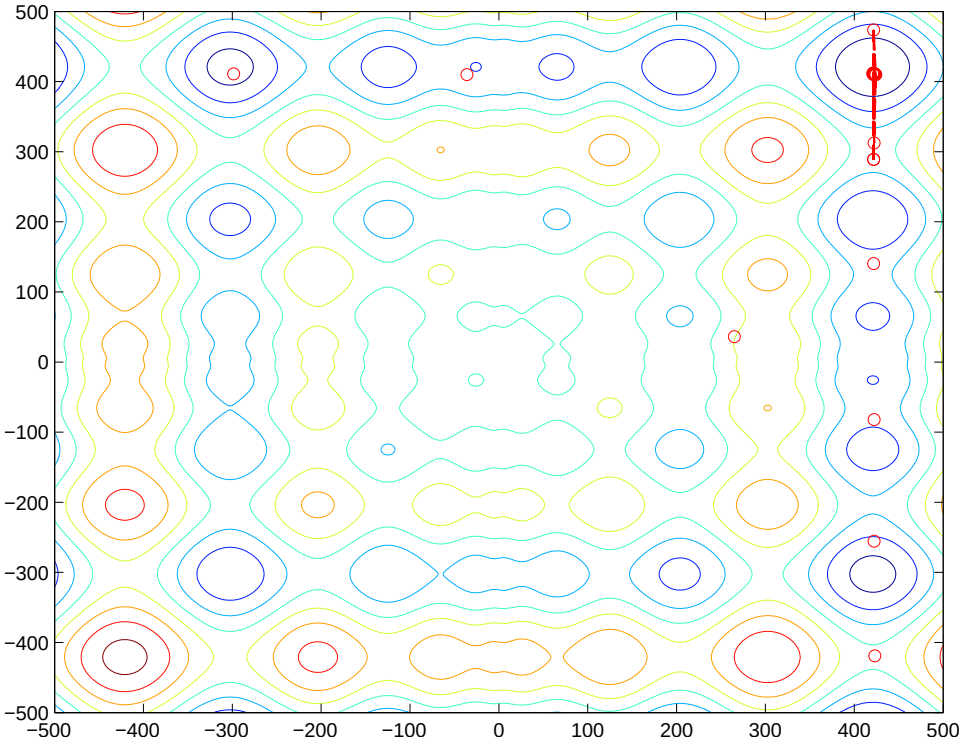


(h) Generation 51: Now some individuals are around the top-right optimum, the global optimum.

Fig. 4.4: (...contd.) Reseeding Policy

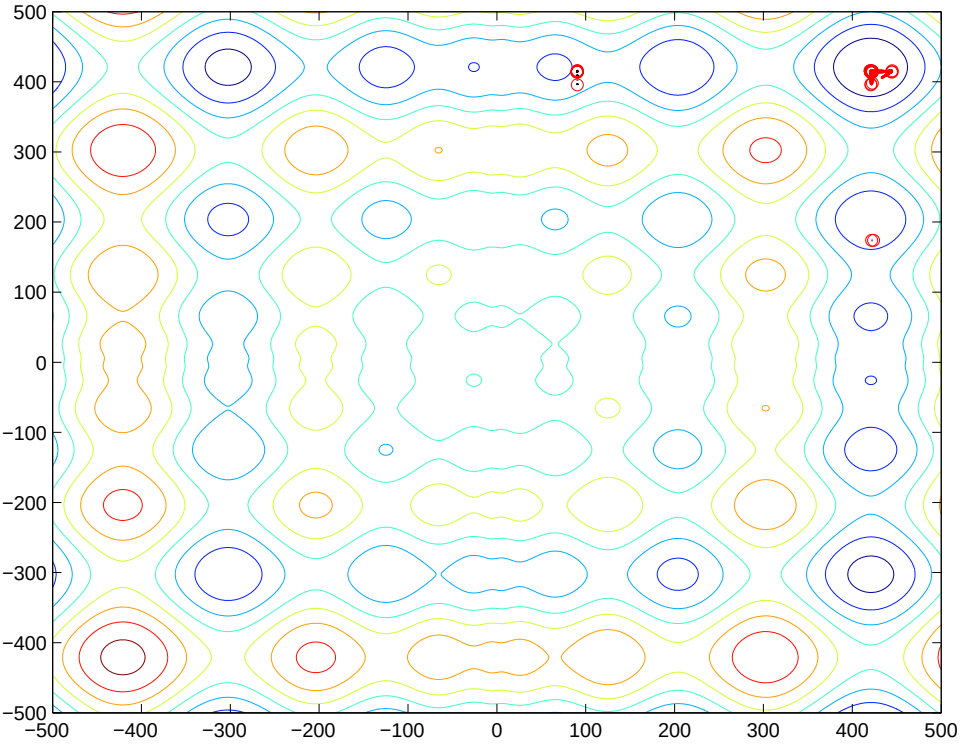


(i) Generation 58: The high-fitness around that region makes the individuals abandon the previous optima and crowd around the new optimum.



(j) Generation 59: Reseeding.

Fig. 4.4: (...contd.) Reseeding Policy



(k) Generation 69: Since the fitness around the top-right corner is the best there is, the population gravitates toward this again. This process continues until convergence criteria are met.

Fig. 4.4: (...contd.) Reseeding Policy

3. Table 4.12 – Different Edge Thresholds, all values are averaged over Population Sizes and Dimensions.

The main observations are that the initial values (Generation 10) are somewhat equal for the different replacement settings, and we observe improvements during the middle of the run (Generation 50). Finally, when we reach the late stages of the run (Generation 90), all the experiments with different values for the parameters (population sizes, dimensions, and edge thresholds) yield much better function values than the original unmodified GA. This arises as a result of the phenomena observed in Figures 4.4a-4.4k, i.e., the forced exploration reduces the occurrences of premature convergence, thus leading to better function values.

We observe that in all the three tables, viz., Tables 4.10, 4.11 and 4.12, although performance starts off similar in the early stages (Generation 10 in the tables), it improves greatly compared to the original GA late in evolution (shown at Generation 90), since there would have been few instances when reseeding would have occurred by this time.

Figure 4.5 shows the above observations pictorially and includes runs from populations of sizes $50 - 150$ in generating the figure. These are plotted late in evolution (here, Generation 90). Each point in the graph is a result of 30 runs. Function values are, in general, much better with any fraction of replacement in a wide range of values. We also note from Figure 4.5 that as dimensions increase, the problem complexity increases leading to worse function values; and with increase in population sizes, the function values improve. One section of this figure, having a population size of 100, is also plotted separately in Figure 4.6 to examine the details.

In Figure 4.7, we show a boxplot comparing different versions of the algorithm; unmodified, 20%, 30% and 40% modifications. This plot shows that there is significant benefit to using this reseeding policy, and that reseeding yields results of similar quality for a wide range of fractions of population reseeded.

One set of values, i.e. Population size=100, Number of variables=3, Distance threshold=0.5 is examined in more detail. In this, we focus on the best function values late in evolution. A histogram of these values is given in Figure 4.8, and shows the number of occurrences of function

Table 4.10: Replacement results by Population Size

	Original			20% replaced			30% replaced			40% replaced		
Pop \ Gen	10	50	90	10	50	90	10	50	90	10	50	90
50	428.2	328.5	278.5	413.4	209.2	118.3	418.0	194.3	104.1	411.0	188.7	103.3
75	328.2	227.1	185.1	328.3	144.4	73.9	328.6	136.5	65.4	325.7	131.7	63.1
100	262.5	167.3	133.8	261.8	103.3	48.0	266.4	96.9	42.0	266.8	93.9	36.5
125	216.9	129.2	104.6	223.4	78.0	33.2	228.8	71.6	28.4	217.2	69.6	28.2
150	199.3	116.8	95.0	190.7	62.3	24.4	202.2	59.3	21.6	192.5	52.4	17.2

Table 4.11: Replacement results by Problem Dimensionality

	Original			20% replaced			30% replaced			40% replaced		
Dim \ Gen	10	50	90	10	50	90	10	50	90	10	50	90
2	15.7	4.6	4.2	15.1	0.2	1.6E-4	16.3	0.3	2.6E-5	12.5	0.1	3.0E-5
3	187.7	120.4	106.2	175.5	54.7	21.9	158.5	44.0	14.7	175.2	49.6	18.2
4	238.3	153.1	127.1	240.6	84.4	39.2	241.9	83.1	36.1	233.4	72.9	29.2
5	324.1	215.3	174.5	317.7	123.0	55.5	341.2	123.4	54.8	320.9	107.9	43.9
6	414.4	283.0	228.6	409.5	181.4	91.4	428.0	167.3	75.1	414.8	167.5	78.3
7	541.7	386.2	316.0	542.8	273.0	149.5	546.8	252.3	133.1	538.9	245.6	128.5

Table 4.12: Replacement results by Edge Threshold

	Original			20% replaced			30% replaced			40% replaced		
δ \ Gen	10	50	90	10	50	90	10	50	90	10	50	90
0.1	282.6	187.7	152.8	286.9	143.3	79.2	287.8	132.0	69.1	280.9	121.4	63.8
0.5	294.2	200.3	165.1	283.0	106.6	49.9	287.3	100.0	43.3	286.1	99.9	42.9
1	284.3	193.3	160.3	280.6	108.5	49.7	291.2	103.1	44.5	280.9	100.5	42.4

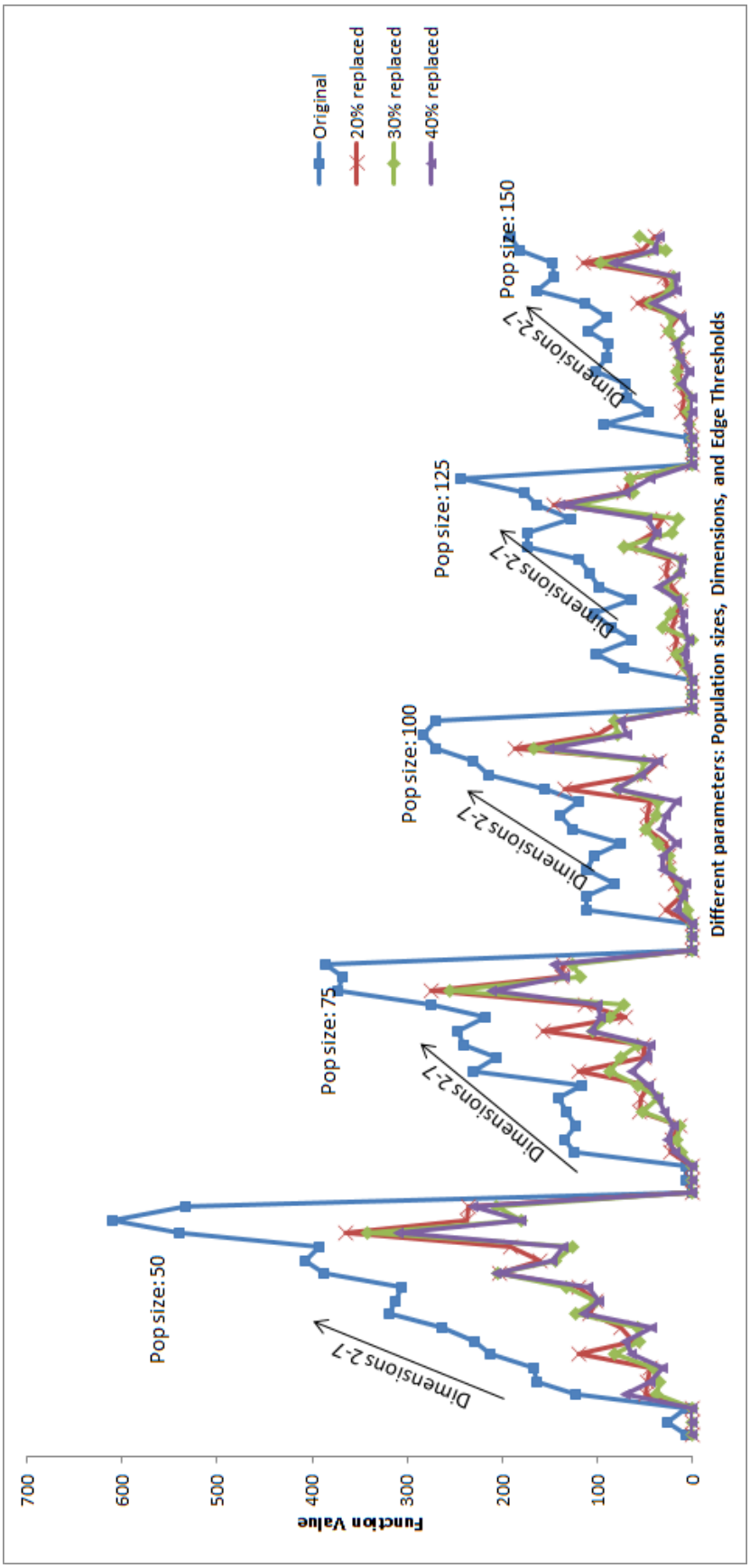


Fig. 4.5: Results of reseeding (Schwefel function, Various parameters)

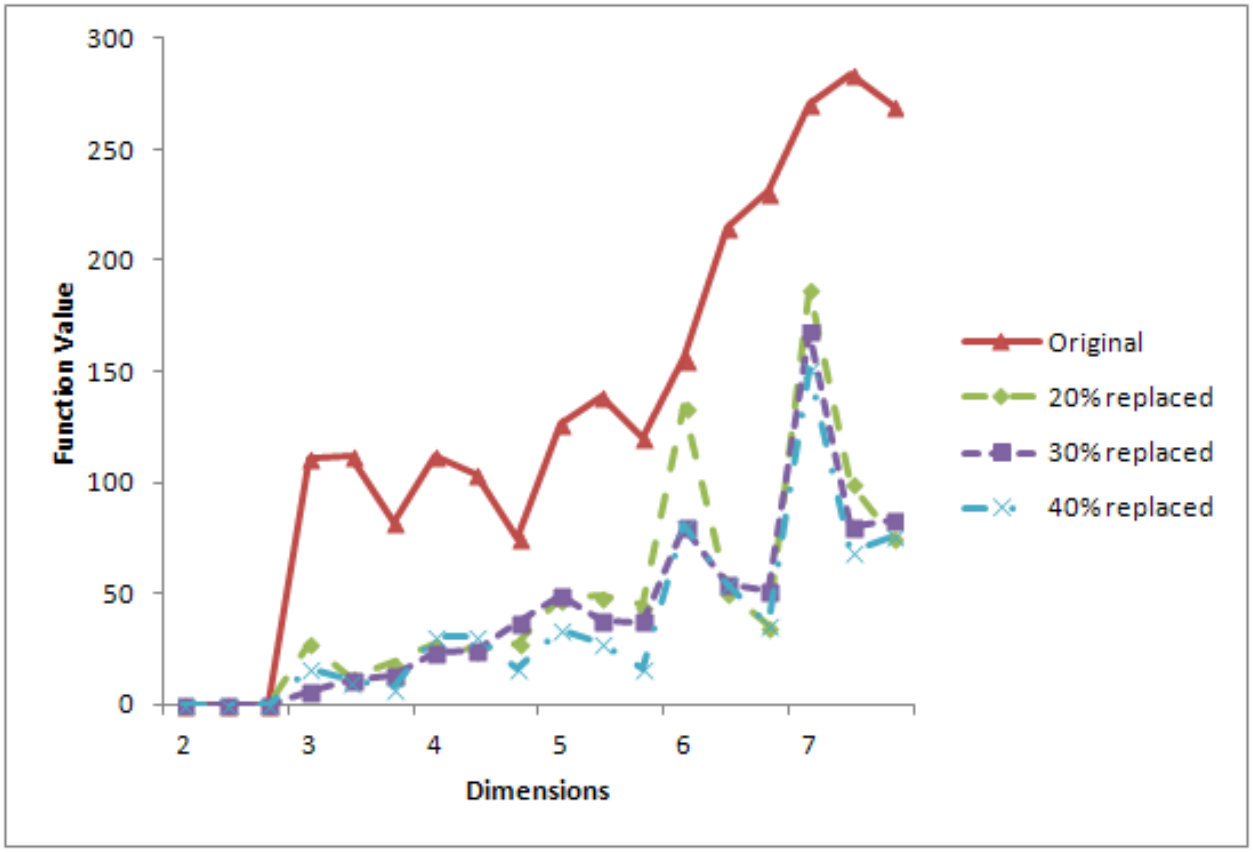


Fig. 4.6: Results of reseeding (Schwefel function, three different edge thresholds per dimension, different fractions replaced, Population size = 100)

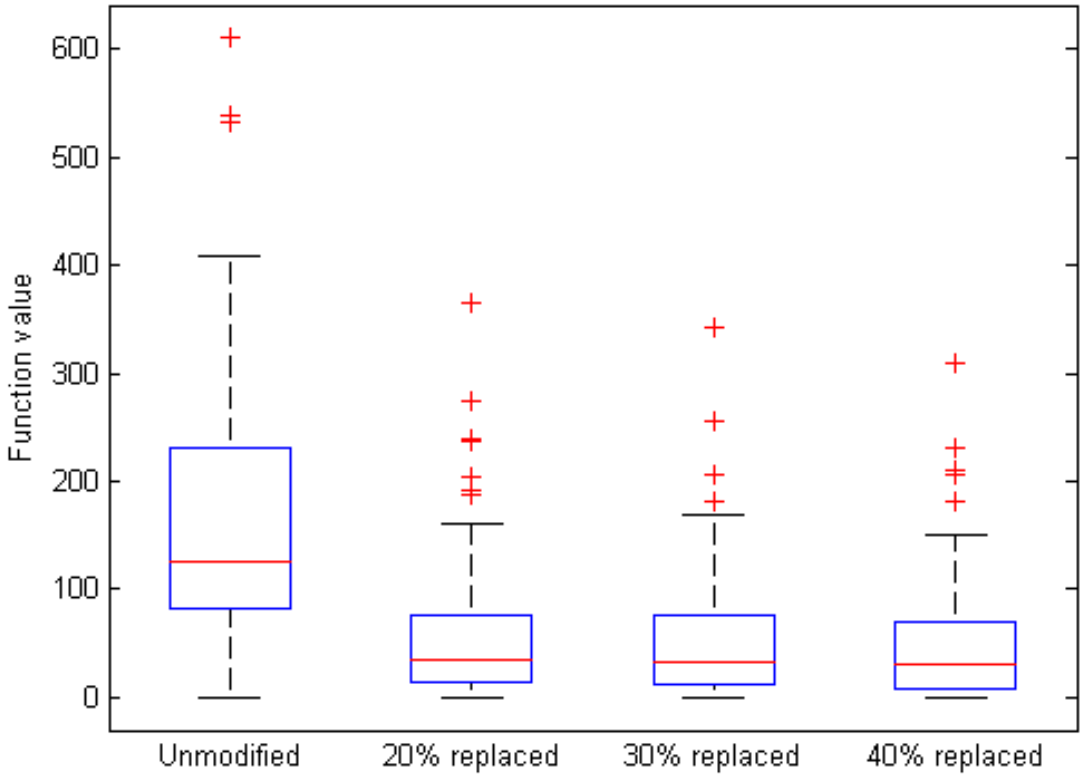


Fig. 4.7: Boxplot of function values

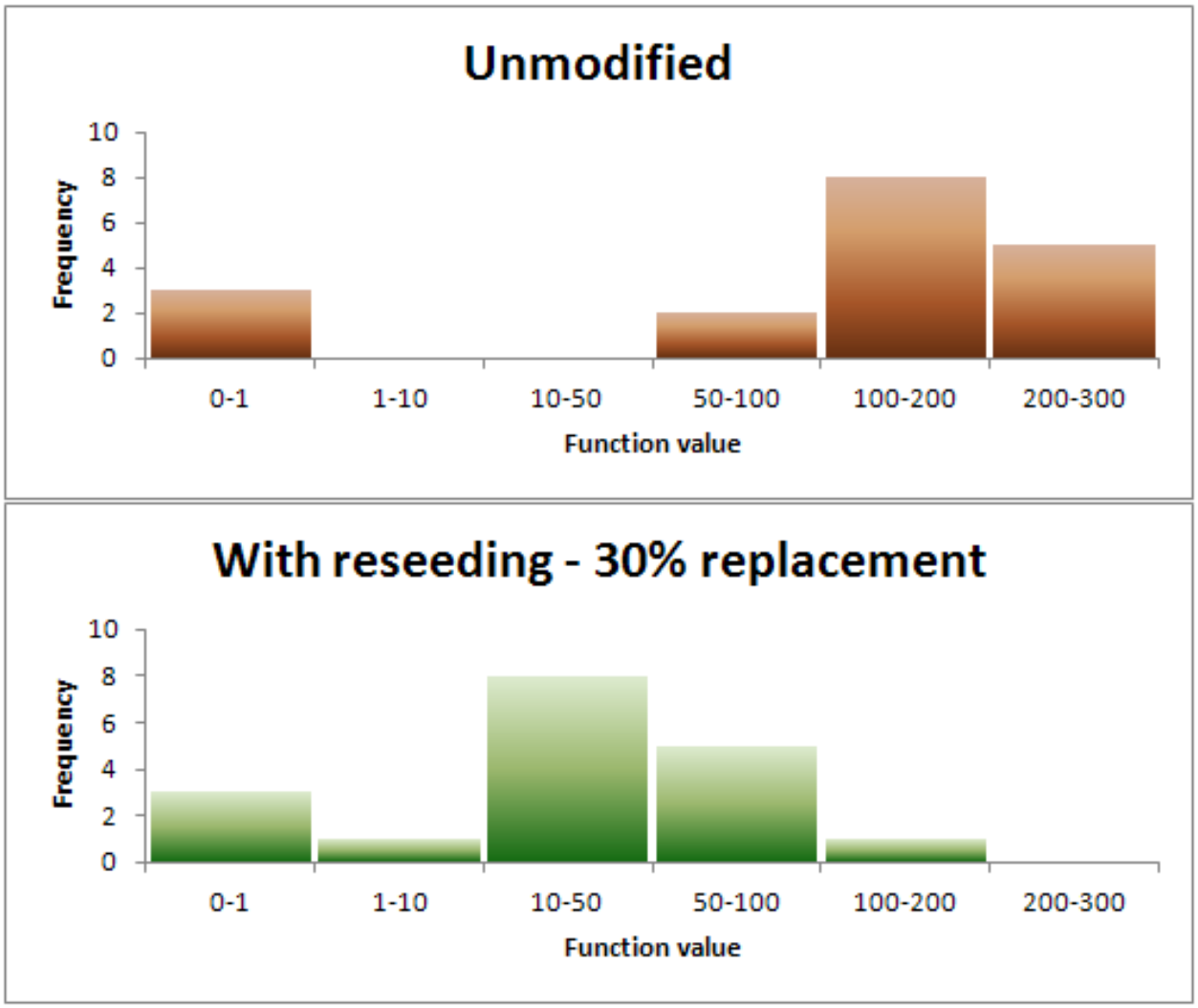


Fig. 4.8: Histogram of function values (Schwefel function, Average of 30 runs)

values in various ranges, with and without reseeding. Clearly, there is an improvement in function values as a result of the reseeding. Other settings also yield similar kinds of plots.

We also performed all these experiments for other test functions listed previously. On average, in approximately 60% of the cases, improvements were observed. Hence, even if there are no traps, this method of reseeding results in improvements, though not substantial, since the problems themselves are not too difficult.

Reseeding often creates a set of individuals that are of poor quality (compared to individuals in an evolved population), because they are randomly chosen. It follows that the mean value of the function is likely to get worse each time there is a reseeding because the newly replaced points

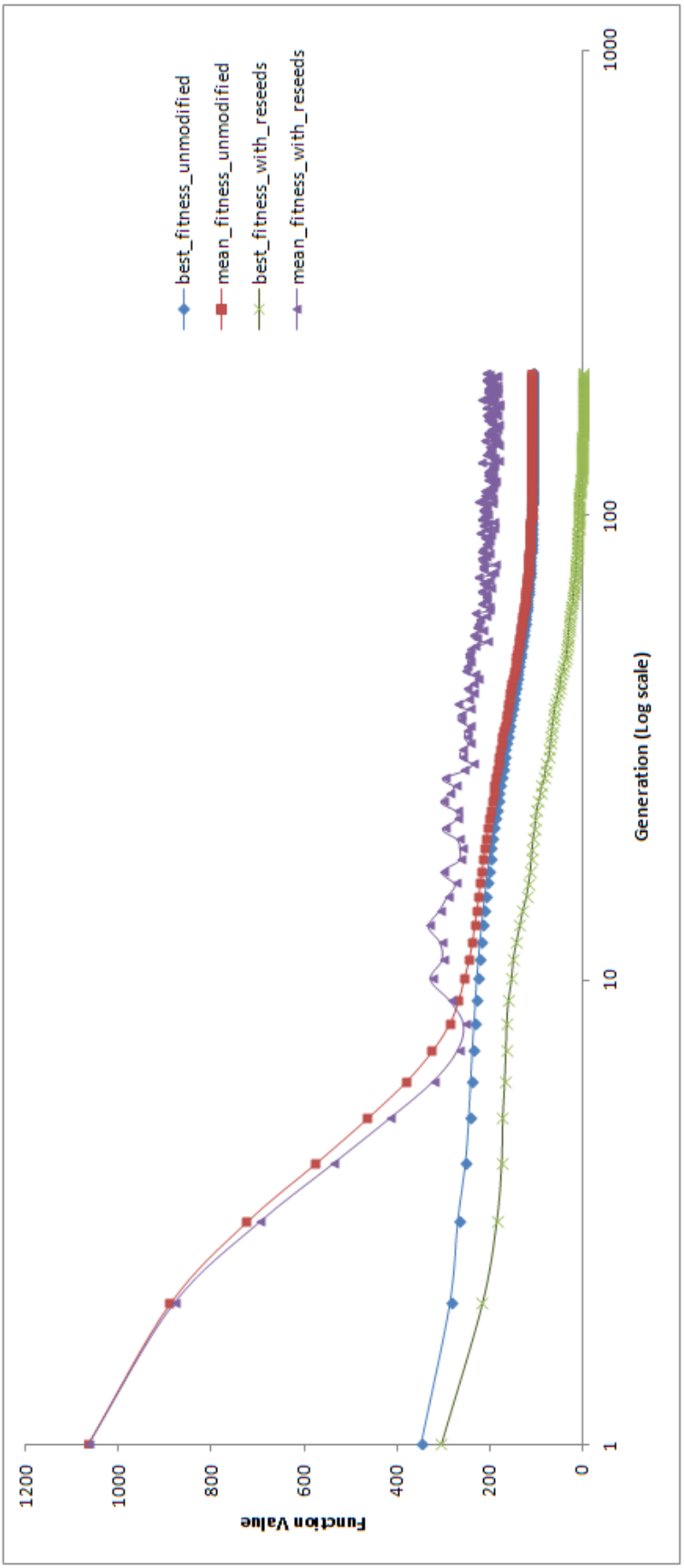


Fig. 4.9: Comparison of best and mean values, with and without reseeding (Schewefel function, Average of 30 runs)

will, in most cases, be placed at a point of poorer fitness (or higher function value). However, the best value appears to be usually better as is seen in plots and tables. Typically, near convergence, a GA has somewhat equal mean fitness and best fitness values. Although reseeding improves the best fitness values, it sacrifices the mean fitness values, which is expected and acceptable, since the final desired result is just the best value. Figure 4.9 shows this pictorially.

So far, we have explored in detail the roles that Euclidean network analysis can play in a GA. In the next section, we examine similar (Euclidean) networks in modeling a PSO.

4.6 Networks in Particle Swarms

In this section, we define and study Euclidean networks in a PSO. A brief introduction and a description of the processes in a PSO was given in Chapter 2. As mentioned earlier in this chapter, although a PSO is different from a GA, both share some similarities in that they are both population-based, and the best individuals act as attractors. We exploit this similarity later in this section by developing an algorithm very similar to the one used for GA convergence detection.

4.6.1 What is different about the PSO?

Particle Swarms were initially developed to model groups of individuals. But unlike a swarm of bees or a flock of geese, the individuals in an instance of the PSO algorithm do not group together into well-defined structures. There is a probabilistic attraction for each particle towards the global best (or the leader of the flock), as well as attraction towards its own best known position. In addition, we must also consider the stochastic nature of the search process which is inherent in almost all forms of EAs.

In a GA, we had the advantage of a convex operator, *crossover*. Since PSO does not utilize crossover, the global best acting as an attractor has a similar effect, being one of three components that determines a particle's next position and velocity as seen in Equation 2.6. Hence, we can draw a rough parallel between the convexity of the crossover operator of a GA and the likelihood

of global best values exerting continuous pressure drawing individuals towards it. We attempt to exploit this information through experimentation.

The global component of Particle Swarms acts as the main attractor; this is roughly analogous to the best individual of a GA and the effect it has on the rest of the population. Both of these behave as attractors, but in very different ways as explained below.

A GA exerts pressure by choosing the choosing the individuals of higher fitness more frequently as parents, and hence the individuals in the next generation are hence more likely to resemble the fitter individuals of the current generation. However, the global best in the PSO exerts pressure by directly attracting all individuals. In a GA, it is likely but not certain that the best individual will contribute much to future offspring since the best being chosen as a parent is probabilistic. Similarly, the global best in a PSO may exert only a weak attraction if the randomly chosen weighting parameter r_2 is small. Consequently, we expect that the network analysis of PSO and GA should be similar. From the point of view of the evolving networks, the mechanism by which the global best acts as an attractor does not matter, since we exploit the commonalities between a GA and a PSO, i.e., large subsets of individuals moving in the same direction indicate a region of interest. In the remainder of this chapter, we evaluate this hypothesis.

4.6.2 Experiments

We explore the issue of convergence detection based on the communities and structures formed by the swarms. This convergence detection is somewhat greedy in nature, and identifies instants in time when the efficacy of a PSO is no longer significant. This leads to answering the important question of how far is far enough. Although there will likely continue to be very minor improvements, the minor changes will be far too small to justify application of a computationally expensive process such as a PSO (or in the previous cases, a GA).

We present below an algorithm very similar to the one seen in Algorithm 1. The Euclidean networks are defined in exactly the same way as in Genetic Algorithms in Section 3.1, i.e., by defining a threshold (to determine the existence of an edge) based on the body-diagonal of the

search space. Algorithm 6 exploits the convergence of a PSO, exactly as in the GA context, i.e., using the collapse of the convex hull.

Algorithm 6 Early convergence detection in a PSO based on the convex hull of the largest component

1: Initialize the population;
2: α takes values in $[0.6, 0.9]$, ϵ takes values in $[0.005, 0.03]$;
3: **repeat** Perform velocity and position based computations to obtain next generation, as in Equations 2.6 and 2.7
4: **until** number of individuals in the largest component $\geq \alpha.n$ (This indicates that a large component has emerged.)
5: $\gamma \leftarrow$ Greatest Hull Area (observed so far) of the largest component
6: **repeat** Perform velocity and position based computations to obtain the next swarm
7: **until** hull area $\leq \epsilon.\gamma$, and number of individuals in the largest component $\geq \alpha.n$ (This indicates that the convex hull has collapsed and further iterations are not needed.)

Test problems and Results

We test Algorithm 6 on the same benchmark problems as before – Rastrigin's function [60], Ackley's path function [1], DeJong's first function [19], Easom's function [23] and Schwefel's function [62].

Tables 4.13-4.15 show the results for a selected set of parameters and dimensions. Table 4.16 and Figure 4.10 summarize our observations for all dimensions and parameters. We note that most Δ values are very close to 0 which indicates that we sacrifice very little accuracy by predicting early convergence and stopping our algorithm.

4.6.3 Observations: Convergence Detection in PSOs

In this section, we have explored the applicability of convergence detection in a way very similar to the ones used for a GA by making use of the similarities between a GA and a PSO, and not focusing on the differences. As we observe from Tables 4.13-4.15, the cost of stopping our algorithm early by detecting impending convergence is indeed low. A more detailed discussion on the different kinds of networks that are formed in GAs and PSOs (as well as LCSs) is given in Chapter 7.

Table 4.13: Detection of Convergence in a PSO – Various dimensions – Population Size 50

Dim	Function	Gen	Conv	Δ
3	Rastrigin	100.33	141.4	0.066354
3	Ackley	92.767	160.1	0.00238
3	DeJong1	92.433	107.57	7.11E-08
3	Easom	70	70	0
3	Schwefel	85.067	132.27	0.000278
5	Rastrigin	103.07	164.17	0.39994
5	Ackley	88.9	174.6	0.024197
5	DeJong1	88.433	120.5	3.37E-06
5	Easom	49	49	0
5	Schwefel	81.567	145.97	0.005433
7	Rastrigin	98.567	167.93	0.74628
7	Ackley	88.633	184.1	0.08751
7	DeJong1	87.8	131.03	2.73E-05
7	Easom	49	49	0
7	Schwefel	82.733	161.6	4.1643

Table 4.14: Detection of Convergence in a PSO – Various dimensions – Population Size 100

Dim	Function	Gen	Conv	Δ
3	Rastrigin	102.27	133.73	8.2E-06
3	Ackley	94.133	157.8	0.001306
3	DeJong1	92.933	102.93	1.55E-08
3	Easom	63.5	63.5	0
3	Schwefel	86.8	129.23	2.48E-05
5	Rastrigin	103.5	155.93	0.001601
5	Ackley	88.867	169.93	0.013319
5	DeJong1	89.233	117.6	9.54E-07
5	Easom	49	49	0
5	Schwefel	80.933	145	3.9501
7	Rastrigin	99.633	162.27	0.1389
7	Ackley	88.467	178.3	0.049274
7	DeJong1	88.033	125.7	1.13E-05
7	Easom	49	49	0
7	Schwefel	81.233	153.83	0.036333

Table 4.15: Detection of Convergence in a PSO – Various dimensions – Population Size 150

Dim	Function	Gen	Conv	Δ
3	Rastrigin	100.93	127.37	4.17E-07
3	Ackley	94	156.03	0.00092
3	DeJong1	93.667	98.8	2.21E-09
3	Easom	69.267	69.267	0
3	Schwefel	83.633	125.97	2.46E-05
5	Rastrigin	104.4	154.47	0.034096
5	Ackley	89.3	166.8	0.007991
5	DeJong1	89.5	114.33	5.28E-07
5	Easom	49	49	0
5	Schwefel	80.2	141.47	3.9495
7	Rastrigin	103.93	168.43	0.26655
7	Ackley	88.267	174.8	0.0298
7	DeJong1	88.333	124.3	4.24E-06
7	Easom	49	49	0
7	Schwefel	80.9	152.2	0.042434

Table 4.16: Euclidean Networks in a PSO – Frequency Distribution of Δ – Over all parameters: Dimensions $3-7$, Population sizes $50-150$, Distance Threshold, $\delta = 0.01, 0.05, 0.1$

Bins	Frequency	Cumulative %
1.0E-07	148	32.89%
1.0E-05	63	46.89%
1.0E-03	70	62.44%
0.1	117	88.44%
1	40	97.33%
10	12	100.00%
More	0	100.00%

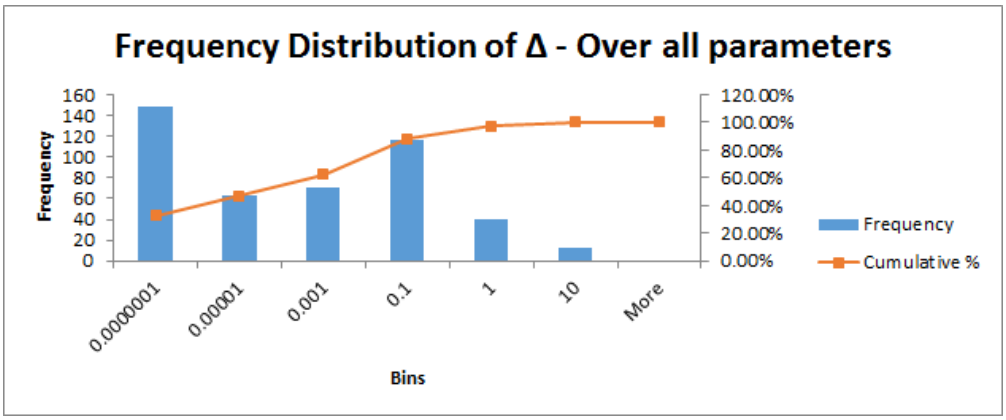


Fig. 4.10: Euclidean Networks in a PSO – Frequency Distribution of Δ – Over all parameters

The observations of low Δ values are further strengthened by looking at the summarized results when performed over all the different experimental values of population size, edge threshold and problem dimensionality. Table 4.16 shows the distribution of Δ values, i.e., the cost of stopping the PSO early over all these parameters and they show most values as being negligibly small.

4.7 Summary

The preceding sections demonstrate the applicability and uses of Euclidean networks in GAs and PSOs. Networks provide additional information regarding the performance of the evolutionary algorithm, and this information has been used to develop convergence detection and reseeding algorithms.

We have observed that in the case of convergence detection, very little is sacrificed by stopping early. In other words, after a certain point in the EA, there is very little to gain from the evolutionary processes, since the later generations of the EA are mainly over-exploiting. Expensive genetic operators such as crossover or mutation in a GA, or position and velocity updates in a PSO, are not essential.

However, there could be many situations where these methods of convergence detection would not necessarily be an ideal choice. For example, if there are deceptive problems which mainly require a very long search process, early detection based purely on component formation might be too aggressive, i.e., the recommended stop time may be too soon. Also, in the case of mostly flat landscapes, it might be more prudent to use the movement of the component as well as the component formation and convergence.

Also, some issues with convex hull volume computations are their increasing complexity in higher dimensions. We have described a greedy and computationally less intensive way here by observing large component formation and not performing such hull volume computations for re-seeding, since we are satisfied with a very early indicator of impending convergence in order to reseed. However, a reasonable approximation with much lesser computation cost is to use ellip-

soids instead of the convex hull.

Thus far, we have studied two forms of EAs, viz., GAs and PSOs, in the context of a Euclidean network in this chapter. In the next chapter, we construct another form of network, based on shared genetic material and ancestry. Since this is based on ancestry and PSOs do not have this feature, ancestral networks are applicable only to GAs and not to PSOs.

CHAPTER 5
NETWORKS FORMED FROM ANCESTRAL CONNECTIONS

In this chapter, networks are defined not based on distance between individuals in the Euclidean space, but based on the genetic material that individuals share between them, i.e., based on ancestry.

Here, we define an ancestral network, $G =< \mathbb{V}, \mathbb{E} >$ as follows:

- $\mathbb{V}$: All n individuals in the population are vertices or nodes.
- $\mathbb{E}$: Two individuals have an edge between them if they share at least one parent, i.e. if they are at least half-siblings.

These are similar to the networks based on shared parents which are studied at multiple levels [76], although the main focus of that study is genetic drift. An example of edges based on parents in the previous generation is shown in Figure 5.1.

The advantage of using this kind of network definition is that the edges are "hard", i.e., we do not require any additional information or need to set thresholds (as in the previous chapter), since the ancestral connections are very clear as the nodes and edges are a direct consequence of the genetic processes that led to the formation of the current generation.

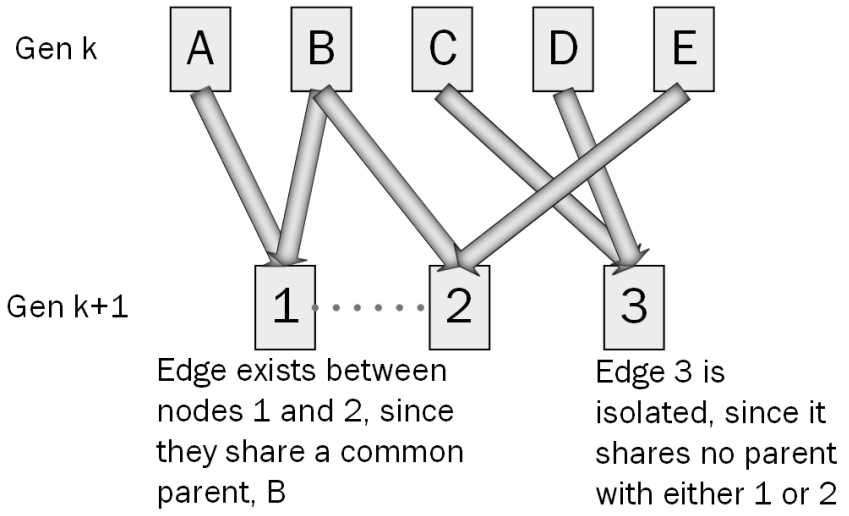


Fig. 5.1: Edges in the network, based on parents in the previous generation

The motivation to use an ancestral network as described, is that two nodes will share a link because they are likely to contain some common genetic material. Common genetic material would imply a form of "similarity" amongst individuals which we exploit by creating an edge and observing formations of components. The existence of many "similar" individuals would imply the identification of a region of interest, which in the problem at hand would be the possible location of the global optimum. Translating the above into the context of the optimization problem and strings/real-valued representations, the nodes that share an edge will contain a common bit-string or the corresponding real-valued number.

There are many implications of such a network. For instance, the formation of a large component means that a few individuals may have become "super-individuals" with very high fitness (viz., low function values). This would lead to a much higher probability of them getting selected for reproduction. This eventually leads to numerous individuals sharing an edge because of the above-mentioned common parent(s). In certain situations such as a very flat surface (where there are equally fit individuals, i.e., individuals having similar function values), where all are equally likely to be selected, we could see many different kinds of edge sets. This is because there are no super-individuals as described. But since most surfaces in real-life problems are uneven, this kind of occurrence is unlikely. An example of a largely flat surface that can lead to such a phenomenon

is the Easom's function [23].

We use the above observations to develop an algorithm in the next section. This algorithm determines a time instant to reseed.

5.1 Reseeding Algorithm

We begin by formalizing our observations below, and then develop it into an algorithm.

Observation 5 *The EA approaches convergence when its population is "largely" connected. Once a component emerges, certain fit individuals begin contributing a lot of their genetic material to newer individuals, leading to similar individuals as before, which implies very little change. We hypothesize that this would be an instant in time to retain a majority of the population for the search results it has already produced, and also to reseed a small fraction, thus giving the algorithm some more exploration opportunities.*

We use the ideas in the proposition above to generate Algorithm 7.

Algorithm 7 A Reseeding Policy Based on Early Detection of Convergence

Initialize the population;
$n \leftarrow$ Population Size;
$\alpha \leftarrow$ Threshold fraction of population in order to identify a large component;
repeat Perform recombinations and mutations to obtain next generation of GA
  if number of individuals in the largest component $\geq \alpha.n$ (i.e., a likely convergence zone) **then**
    Reseed a fraction, β, of the population in the search space and include them in the mating pool.
    Retain the remaining representative fraction, $(1 - \beta)$ of the component, which will be a representation of the work done by the GA so far.
  end if
until Convergence criteria are met, i.e., difference between best/mean function values is less than a prespecified threshold, or the maximum number of generations is reached.

Algorithm 7 takes into account multiple ideas: component formation, identification of the time instant when convergence is likely to occur, reseeding a fraction to continue exploring, and retaining the rest. We now discuss the results of running this algorithm.

We pick a difficult test-bed function that is used widely by researchers, the Schwefel function [62], which has distant local optima. We test our approach by experimenting with various population sizes, different dimensions (or number of variables), and different values of representative fractions to retain and reseed.[1]

We record and examine the results of running these experiments at Generations 10, 30, 50, 70, and 90, of a total of 100 generations each. The results are shown in Table 5.1, depicting function values late in evolution, i.e., Generation 90, at around the point of convergence. A more detailed version of Table 5.1 is shown in Table 5.2, which contains other experimental parameters of α and β. Results reported in these tables are all combinations of the above parameters, and are averaged over 30 runs.

A pictorial depiction of the above for a population size of 50 is shown in Figure 5.2. Comparing the original GA performances against the modified versions as seen in Table 5.1 as well, the algorithm shows improvements in terms of better (lower) function values (since we are minimizing), with occasional exceptions.

It is of interest to note the high values of standard deviation, occurring because there are significant numbers of optimally converged values as well as a significant number of premature convergences. In this scenario, a better mean implies having better function values as well as the fact that the algorithm got trapped fewer times in local optima (due to reseeding). The Schwefel function (global optimum = 0) has many distant optima, which lead to the observation above.

We also study the variation in function values with changing population sizes (Figure 5.3) for select values of α and β, as well as percentages of improvement over regular GA with changing problem dimension (Figure 5.4). From these, we observe that reseeding at prespecified instances does help, but it is clearly better to depend on component formation in order to determine when to reseed.

[1] We also ran the experiments with the other benchmark functions which we have been testing on thus far, viz., Rastrigin's function [60], Ackley's path function [1], DeJong's first function [19], and Easom's function [23] as well. Since these are simpler problems, we found that they did not require reseeding as these were not much prone to premature convergence, although reseeding did not hamper the results.

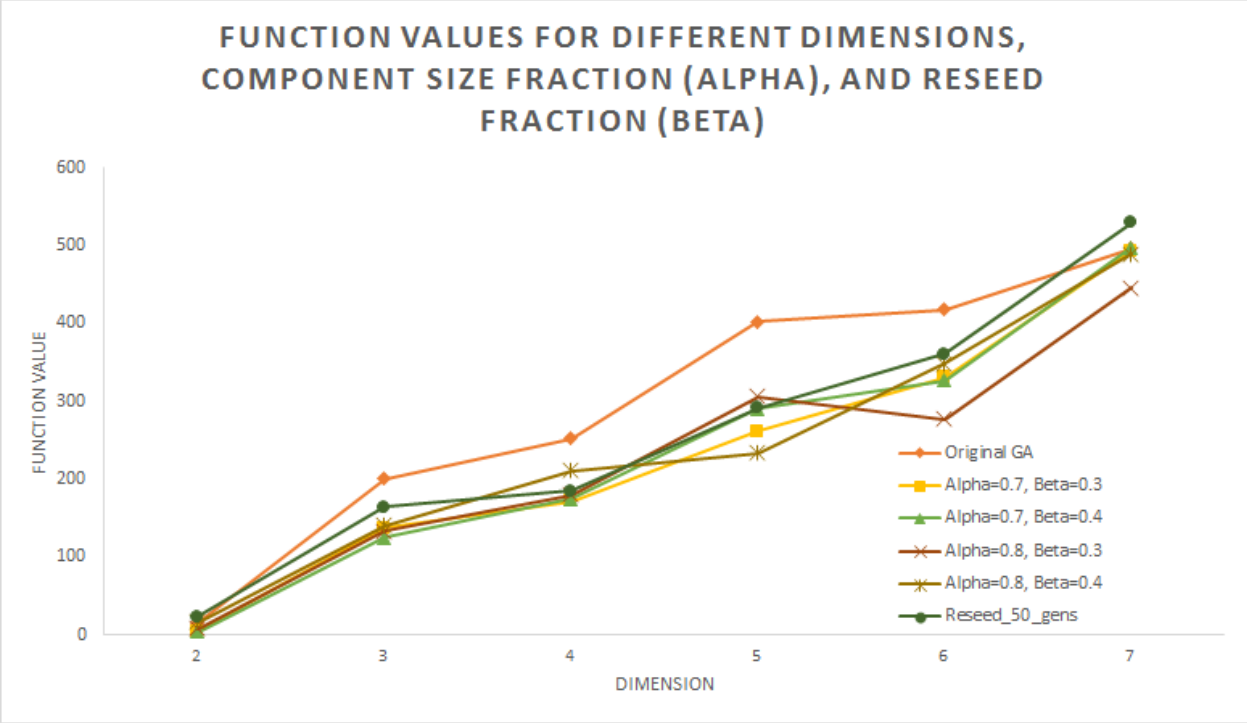


Fig. 5.2: Function values for various dimensions, and parameters α and β

Table 5.3 shows the range of values of α and β for which the function values are relatively good, for a high-dimensional Schwefel problem (7 dimensions), and large population (Pop. size = 150). The values in bold indicate the recommended parameters, viz., the best is to replace just a small fraction since we do not want to lose the results of the GA run thus far. If larger fractions need to be replaced, it is advisable to consider the size of the large component – waiting for too high a component size implies minimal reseeding, and too low implies very frequent reseeding. Results obtained are relative unaffected by the precise choice of parameter values, e.g., as shown in Figure 5.2, what fraction determines a large component, or how much to reseed, as long as the parameter values are within the described bounds. This shows that the algorithm is not too sensitive to changes in the parameters, which is helpful from a practitioner's point of view. We would, however, recommend the values of α and β to be among the ones for which the function values are in bold.

In order to highlight the effects of reseeding based on component generation, we also try re-seeding at predetermined instances (labeled 'Reseed_50_gens') at every 50th generation. This can

Table 5.1: Results: Function values for different dimensions and population sizes, at Generation 90. Best values in bold. Average of 30 runs. Standard Deviations in brackets.

Dim	Pop size	Original GA	Reseed_50_gens	Alpha=0.7, Beta=0.3
2	50	15.432 (47.659)	23.339 (21.263)	**4.4467 (53.879)**
3	50	200.06 (139.12)	163.75 (97.146)	**137.46 (115.5)**
4	50	251.94 (140.91)	184.94 (103.71)	**171.86 (132.29)**
5	50	402.59 (180.44)	291.1 (154.09)	**260.98 (154.93)**
6	50	417.84 (171.5)	360.65 (144.93)	**331.23 (183.76)**
7	50	496.65 (232.46)	529.72 (210.43)	**493.43 (198.35)**
2	75	7.896 (29.544)	4.1002 (1.8E-10)	**2.55E-05 (3.39E-21)**
3	75	126.73 (102.97)	128.43 (117.43)	**120.96 (85.483)**
4	75	133.06 (127.56)	149.84 (85.882)	**110.23 (112.8)**
5	75	**167.63 (128.48)**	186.46 (117.55)	174.05 (135.14)
6	75	235.98 (120.24)	262.53 (161.7)	**225.09 (120.46)**
7	75	407.96 (142.95)	**293.84 (150.56)**	304.51 (134.87)
2	100	**2.55E-05 (3.39E-21)**	4.1863 (4.34E-08)	**2.55E-05 (3.39E-21)**
3	100	113.62 (95.535)	77.18 (82.826)	**71.98 (80.42)**
4	100	117.02 (107.43)	106.19 (91.643)	**76.249 (99.749)**
5	100	142.46 (125.4)	**121.59 (102.08)**	131.03 (122.28)
6	100	225.24 (132.08)	194.39 (134.6)	**171.64 (128.18)**
7	100	291.71 (147.98)	**237.16 (145.19)**	278.23 (121.27)
2	125	3.948 (21.261)	3.948 (3.39E-21)	**2.55E-05 (3.39E-21)**
3	125	63.249 (83.255)	**44.414 (65.466)**	51.524 (64.307)
4	125	78.323 (99.981)	91.262 (73.982)	**73.775 (69.153)**
5	125	111.05 (96.667)	103.44 (91.177)	**94.238 (114.47)**
6	125	**126.15 (94.034)**	146.34 (157.51)	149.78 (112.98)
7	125	230.39 (145.58)	197.35 (129.19)	**170.09 (136.15)**
2	150	**2.55E-05 (3.39E-21)**	**2.55E-05 (3.39E-21)**	**2.55E-05 (3.39E-21)**
3	150	65.965 (86.725)	55.293 (57.774)	**45.14 (66.515)**
4	150	71.357 (78.779)	87.762 (59.728)	**46.316 (66.744)**
5	150	113.93 (99.133)	104.55 (69.005)	**82.249 (105.29)**
6	150	**94.242 (112.89)**	128.23 (119.24)	119.18 (108.88)
7	150	166.95 (131.59)	159.59 (105.73)	**133.77 (107.91)**

Table 5.2: Expanded version of Table 5.1 with different values of α and β

				Alpha=0.7		Alpha=0.8	
Dim	Pop size	Original GA	Reseed_50_gens	Beta=0.3	Beta=0.4	Beta=0.3	Beta=0.4
2	50	15.432	23.339	4.4467	**4.0057**	7.898	15.818
3	50	200.06	163.75	137.46	**124.35**	133.25	140.19
4	50	251.94	184.94	**171.86**	174.52	179.38	210.99
5	50	402.59	291.1	260.98	290.74	306.45	**233.9**
6	50	417.84	360.65	331.23	326.89	**277.09**	348.67
7	50	496.65	529.72	493.43	497.14	**446.2**	489.03
2	75	7.896	4.1002	**2.55E-05**	3.53E-05	**2.55E-05**	3.948
3	75	126.73	128.43	120.96	80.093	73.59	**71.153**
4	75	133.06	149.84	110.23	**107.91**	145.98	115.7
5	75	167.63	186.46	174.05	171.28	185.94	**145.46**
6	75	235.98	262.53	**225.09**	232.99	229.57	234.02
7	75	407.96	293.84	304.51	363.85	**290.99**	269.84
2	100	**2.55E-05**	4.1863	**2.55E-05**	**2.55E-05**	3.948	**2.55E-05**
3	100	113.62	77.18	71.98	**52.277**	61.403	79.634
4	100	117.02	106.19	**76.249**	77.656	110.91	113.79
5	100	142.46	121.59	131.03	**105.32**	119.62	153.55
6	100	225.24	194.39	171.64	**138.73**	151.07	156.89
7	100	291.71	237.16	278.23	**202.4**	213.61	212.17
2	125	3.948	3.948	**2.55E-05**	**2.55E-05**	**2.55E-05**	0.0024077
3	125	63.249	**44.414**	51.524	58.346	77.264	47.73
4	125	78.323	91.262	**73.775**	82.639	78.734	75.672
5	125	111.05	103.44	94.238	104.11	102.24	**89.059**
6	125	**126.15**	146.34	149.78	135.29	135.99	148.9
7	125	230.39	197.35	170.09	234.58	186.13	**167.08**
2	150	**2.55E-05**	**2.55E-05**	**2.55E-05**	**2.55E-05**	**2.55E-05**	**2.55E-05**
3	150	65.965	55.293	45.14	42.77	44.17	**27.866**
4	150	71.357	87.762	46.316	48.461	52.24	**34.286**
5	150	113.93	104.55	82.249	**78.868**	99.907	85.323
6	150	**94.242**	128.23	119.18	115.33	145.14	111.52
7	150	166.95	159.59	**133.77**	147.74	147.26	166.3

Table 5.3: Function values obtained by varying α and β for Population size 150, and 7 Dimensions. Bold values represent recommended parameters of α and β

β \ α	0.6	0.7	0.8	0.9
0.1	**48.879**	**56.05**	**37.277**	**50.617**
0.2	668.14	795.24	1121.3	1228.1
0.3	1153.3	**133.77**	**147.26**	1291.6
0.4	1416.4	**147.74**	**166.3**	1273.3

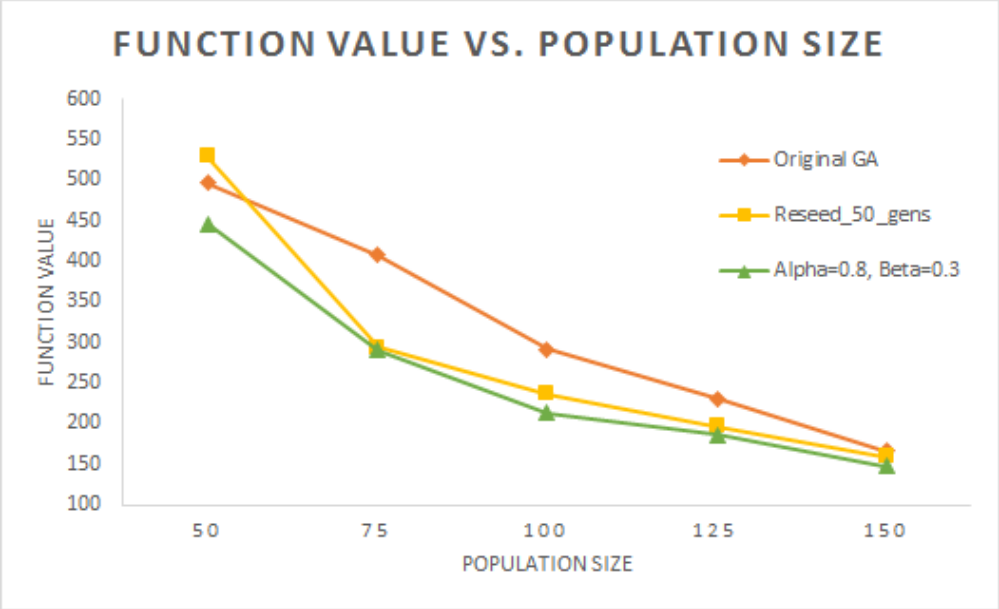


Fig. 5.3: Variation in function values with Population Size

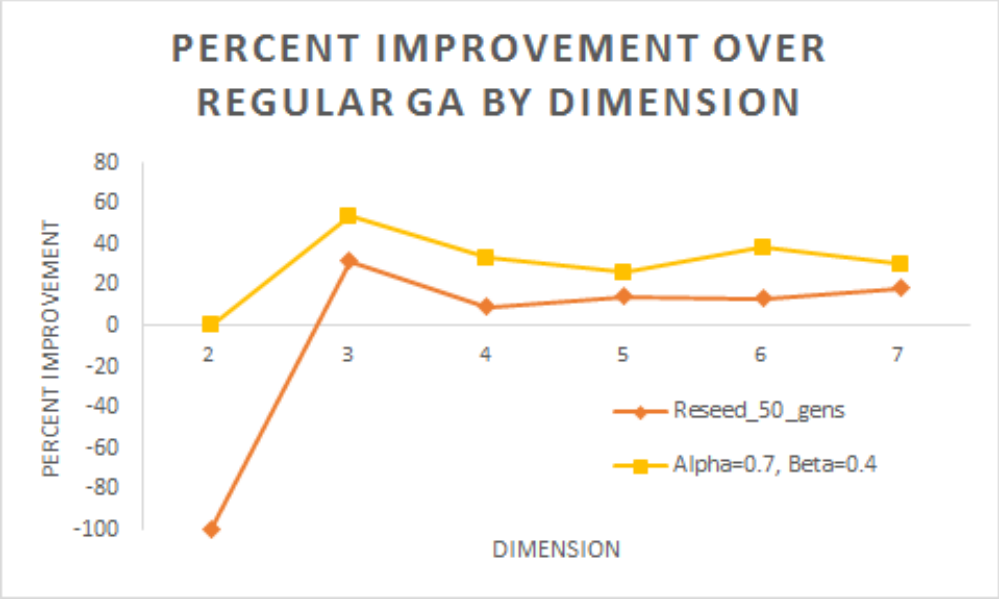


Fig. 5.4: Percent improvement over Regular GA with Problem Dimension

also help, however it is better to estimate convergence and then reseed because choosing an instant based on impending convergence leads to better values. Also, this method of choosing *when to reseed* is data driven, whereas it is difficult to decide generation numbers beforehand for different problems where we do not know in advance how many generations are needed to converge.

In a few instances, the unmodified GA is better, perhaps because the reseeding was too aggressive when a very good solution was already observed.

From the perspective of time complexity, there is a little overhead due to the reseeds, when compared with the predetermined reseeding option. A single predetermined run for $n = 2$, population size$= 50$ took 1.63 seconds whereas with the component occurrence based reseeding took 1.80 seconds, for an overhead of 10.4% which is the time taken to perform the extra reseeds.

Another observation we make is that reseeds typically begin occurring at about Generations 15 to 25 or so. The frequency of reseeds depends on the fraction of population being reseeded, size of the component that is required to be formed, etc.

In the next section, we describe another way to use the ancestral network to modify the algorithm.

5.2 Modifying Fitness Values Based on Centrality

We performed experiments to modify the fitness value using a linear combination of node centrality and the function value. In the process of linearly combining the two, we wish to achieve both of the following:

- increase diversity by penalizing highly central nodes[2], and
- retain the results of the function values obtained so far, and reward the fit nodes.

We normalize both the centrality as well as function value to equal ranges of [0,1]. The hypothesis is that the change should be most noticeable if the algorithm were to get trapped in local

[2]Similar to nodes with highest centrality as shown earlier: shaded green in Figures 3.1a-3.1f

Table 5.4: Linear combination - Function values close to convergence for Population Size 50

Dim	Test Function	Original GA	Modified
2	Rastrigin	0.29849	0.21889
2	Ackley	3.042	3.0864
2	Dejong	9.72E-11	1.17E-10
2	Easom	0.37998	0.31998
2	Schwefel	27.895	14.283
3	Rastrigin	1.3532	1.8506
3	Ackley	3.1535	3.1375
3	Dejong	1.30E-09	2.19E-09
3	Easom	0.99387	0.99405
3	Schwefel	189.84	174.4
4	Rastrigin	1.7711	2.1491
4	Ackley	3.0989	3.1191
4	Dejong	2.43E-07	1.44E-06
4	Easom	0.66	0.60004
4	Schwefel	244.45	245.48
5	Rastrigin	2.5675	1.5526
5	Ackley	3.1313	3.0964
5	Dejong	2.42E-05	8.13E-06
5	Easom	0.99838	0.99874
5	Schwefel	370.68	335.62
6	Rastrigin	2.1997	1.4685
6	Ackley	3.1221	3.1026
6	Dejong	0.000116	0.000147
6	Easom	0.98	0.94001
6	Schwefel	447.48	447.61
7	Rastrigin	2.5613	1.8356
7	Ackley	3.1805	3.1566
7	Dejong	0.001127	0.000909
7	Easom	1	0.99982
7	Schwefel	658.06	559.25

Table 5.5: Linear combination - Function values close to convergence for Population Size 75

Dim	Test Function	Original GA	Modified
2	Rastrigin	0.079597	0.079597
2	Ackley	3.0262	3.0016
2	Dejong	9.10E-11	6.46E-11
2	Easom	0.29999	0.21999
2	Schwefel	0.010906	2.4282
3	Rastrigin	0.59697	0.85566
3	Ackley	3.0865	3.0546
3	Dejong	1.02E-09	9.08E-10
3	Easom	0.99423	0.99351
3	Schwefel	146.81	112.8
4	Rastrigin	0.71637	0.61688
4	Ackley	3.0621	3.0967
4	Dejong	6.43E-08	8.63E-08
4	Easom	0.44	0.5
4	Schwefel	183.91	163.36
5	Rastrigin	0.75644	1.3534
5	Ackley	3.0311	3.0698
5	Dejong	4.54E-06	2.73E-06
5	Easom	0.99784	0.99802
5	Schwefel	227.03	192.87
6	Rastrigin	0.74045	1.1983
6	Ackley	3.0651	3.0535
6	Dejong	5.05E-05	3.41E-05
6	Easom	0.98	0.96005
6	Schwefel	291.76	274.17
7	Rastrigin	1.3098	1.735
7	Ackley	3.0653	3.0614
7	Dejong	0.00033	0.000185
7	Easom	1	0.99982
7	Schwefel	410.24	422.2

Table 5.6: Linear combination - Function values close to convergence for Population Size 100

Dim	Test Function	Original GA	Modified
2	Rastrigin	0.019899	0.019899
2	Ackley	3.0316	2.9858
2	Dejong	5.22E-11	5.17E-11
2	Easom	0.23999	0.21999
2	Schwefel	4.7376	2.3689
3	Rastrigin	0.39798	0.49748
3	Ackley	3.0472	3.0223
3	Dejong	7.47E-10	7.36E-10
3	Easom	0.99351	0.99243
3	Schwefel	112.15	83.871
4	Rastrigin	0.47759	0.45768
4	Ackley	3.0515	3.025
4	Dejong	1.50E-08	2.73E-08
4	Easom	0.6	0.48
4	Schwefel	126.1	113.49
5	Rastrigin	0.67676	0.71654
5	Ackley	3.0332	3.0144
5	Dejong	1.27E-06	7.46E-07
5	Easom	0.99838	0.9982
5	Schwefel	155.48	195.39
6	Rastrigin	0.66167	1.0173
6	Ackley	3.0345	3.0393
6	Dejong	1.78E-05	2.22E-05
6	Easom	0.90013	0.96
6	Schwefel	274.29	265.56
7	Rastrigin	0.74575	0.77871
7	Ackley	3.0438	3.0196
7	Dejong	0.000144	0.000131
7	Easom	1	1
7	Schwefel	321.49	336.55

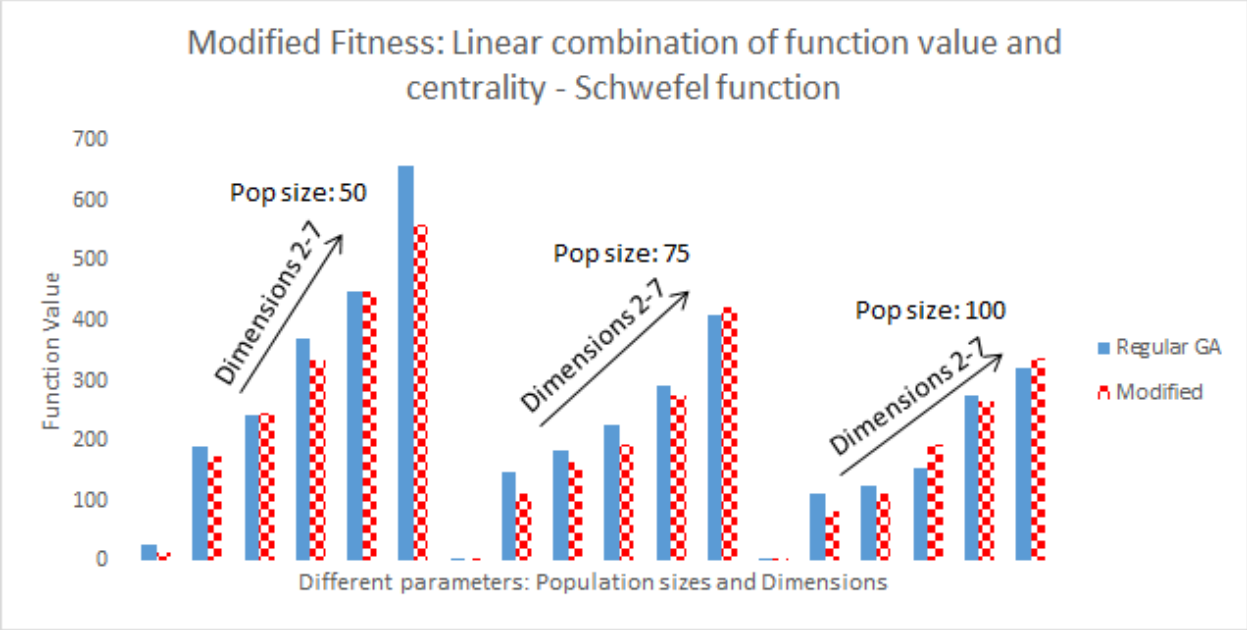


Fig. 5.5: Linear combination based fitness - Function values for various populations and dimensions - Schwefel function

optima. This is because the most central node (likely to have contributed genes to many children) is penalized for its high centrality, and this improves diversity.

$$Fitness(x) = \alpha Centrality(x)_{normalized} + \beta Function\ Value_{normalized} \tag{5.1}$$

Here, we use $\alpha = \beta = 1$.

We examine the performance (averaged over 30 runs) of GAs with population sizes $50, 75, 100$ for multiple functions and dimensions in Tables 5.4-5.6. We notice a better value in 60% of the cases, and worse in 40% of them. However, if we add up the differences of function values between the old and the new, the sum of differences is -250.69 indicating that the biggest improvements are in cases where the algorithm was likely headed towards a premature convergence. We compare the largest differences, viz., in the Schwefel function in Figure 5.5. From the figure, we observe that the linear combination in Equation 5.1 results in many better function values.

5.3 Summary

In this chapter, we have studied two different ways to use an ancestral network in a GA. One involved reseeding based on large component formation. The second improvement was to increase diversity by including a penalty component to an individual's fitness which is done by increasing the function value if the degree centrality of the vertex is high. These two different ways to use the information provided by the population networks continue to support our initial thesis that networks provide a good way to analyze, enhance and improve our understanding of EAs.

In the next chapter, we move on to discuss the applicability of networks to another form of EAs, viz., LCSs, by creating networks of rules.

CHAPTER 6
NETWORKS IN LEARNING CLASSIFIER SYSTEMS

Thus far, the individuals we have been considering have been real-valued vectors. We now consider another class of algorithms which uses a GA, but has other operations involved in creating the next generation. This chapter addresses networks in *Learning Classifier Systems* with the following features:

- We work with XCS, which uses ternary representations (consisting of 0, 1, and $\#$).
- There is a GA running at the center of the LCS.
- This is an online learning system, i.e., new points come in from the environment every generation.

6.1 Test problems

One of the most common benchmark problems for an LCS is a multiplexer problem [79]. This is a bitwise representation of the typical functioning of a multiplexer with the following details:

- A condition string of length

$$C_L = k + 2^k, \text{ where } k \in I^+ \tag{6.1}$$

  In the above equation, k represents the number of select lines of the multiplexer, and 2^k refers to the possible binary strings input to the multiplexer.

- An action bit – This is the output of the multiplexer given the inputs and the select lines.

This function is popular because of many reasons:

- The complexity of the problem can be increased or decreased easily by varying k. For example, if $k = 2$, we have a 6-bit multiplexer problem, $k = 3$ yields a 11-bit problem, and so on, based on Equation 6.1. Typical test functions involve $k = 2, 3, 4$ for problem sizes $6, 11, 20$, respectively. We focus mostly on the 20-bit problem.

- While the problem can be as complex as possible, we do not depend on large datasets since the functioning of a multiplexer is very well understood, i.e., the bit-value in the input as indicated by the select lines is the output (action). So, for example, an input string of $01\#10\#$ is broken into $01, \#10\#$ which implies that the address of the output bit is 01 or the first bit (in a zero-indexed system), which is the bit 1 of $\#10\#$, and hence the output is 1.

- It is very well suited for online learning, where each point can be presented with minimal effort, i.e., a string can be generated at random, and the correct output (for the purposes of supervised or reinforcement learning) can be computed easily and independently.

An example of a perfect solution (rule-set) is described in [79] and is shown below. Here, the set of rules required to have a perfectly evolved 6-mux system is listed.

$$
\begin{aligned}
&000\#\#\# : 0\\
&001\#\#\# : 1\\
&01\#0\#\# : 0\\
&01\#1\#\# : 1\\
&10\#\#0\# : 0\\
&10\#\#1\# : 1\\
&11\#\#\#0 : 0\\
&11\#\#\#1 : 1
\end{aligned}
\tag{6.2}
$$

The first rule indicates that if the select input is 00, then the first of the remaining four bits is the only one that matters, and the action is the same as that bit. Hence, there are 8 cases: One rule for an input value of 0 and another for an input of 1, for each of the four select line combinations $00, 01, 10, 11$.

6.2 XCS Evaluation

Thus far, in the GA and the PSO, we could evaluate the function value of the best individual and compare its evolution with the dynamics of the network properties. However, we cannot do the same here since we are solving a prediction problem using rule-sets. Hence, we describe the measures used to evaluate the performance of an XCS.

Two measures are typically used for evaluating an XCS [79]:

- Accuracy (or Performance): The fraction of the last 50 trials that were correctly predicted.
- System Error: Absolute difference between system prediction and external payoff normalized over the total payoff range (here, 1000), averaged over the last 50 trials.

Hence, the goal is to improve accuracy while reducing system error. Performance analysis of an LCS is also discussed in detail in [11] and [22].

We will compare these XCS evaluation parameters with the network parameters that we will describe next. These network parameters are used to detect rule-set convergence from a network perspective. An alternate view of rule-set convergence based on parent-trees of rules has been studied in [35].

6.3 Similarity Measure

Since the genotypes of the rules have bit-string representations, a similarity measure based on the Hamming distance can be applied, as in [34]. We define the similarity between two rules in a way similar to the Hamming distance, i.e., we consider bitwise comparisons. But these rules contain "don't cares" $(\#)$, therefore we not only consider the number of differences, but also by how much. In other words, we define similarity between pairs of individuals in a ternary system involving a don't care as:

- Similarity between 1 and 1 or 0 and 0 is 1.
- Similarity between 1 and $\#$ as well as that between 0 and $\#$ is 0.5. This is because in each of these cases, there can be two values for each don't care (0 or 1) where one of the values is a perfect match having a similarity of 1, and the other is a mismatch, an exact opposite with a similarity of 0. Hence, we use the average value which is 0.5.
- Similarity between $\#$ and $\#$ is 0.5. This is because there are four combinations here: If both are the same (both 0 or 1), then there is a perfect match with a similarity of 1. Else, they are negations of each other, implying complete dissimilarity.

The more similar the rules are, the stronger the edge. We compare only the condition part of the rules. The action bit is not considered in this computation, since each incoming point is matched only with the condition string to trigger a rule.

6.4 Network Construction

We create networks in two different ways:

1. A weighted network: An edge is assigned a weight proportional to the similarity between the two vertices. Hence, the edge strengths can vary from 0 (i.e., no edge), to C_L, i.e., an exact match.

2. An unweighted network: This network has a similarity threshold for two rules. In other words, two nodes are connected by an edge if the edge strength is greater than a threshold. Hence, all nodes are not connected to each other, unlike the previous case. In this context, we perform a degree-based analysis.

We examine convergence indicators from both of these methods.

6.5 Experiments and Results

The Weighted Network

In the first network, we have a completely connected network, but with varying edge weights. Network measures as well as XCS measures are plotted in a dual Y-axis against Generation $(1 - 50,000)$ on the X-axis in Figure 6.1. We observe that there is a rise and fall of the average edge weight per rule, akin to the component structures seen in a GA and a PSO, and their collapse. If a function is monotonically increasing or decreasing, without knowledge of the distribution, it is hard to determine when it will stabilize. But by having a rise and fall structure, we have a baseline to work with.

Further, we note the strong correlation between the average edge weight per rule (and also the same measure per microclassifier[1]), and the XCS evaluation measures of Accuracy and System

[1]A microclassifier is a term given to each individual classifier that a rule (or a macroclassifier) represents. The number of microclassifiers that each macroclassifier represents is recorded in the "Numerosity" of the rule.

Error. These graphs are plotted for every 50^{th} generation, and the values are averages over the most recent 50 generations.

Table 6.1: XCS Evaluation measures: Accuracy, System Error compared with Edge weight per microclassifier and Edge weight per rule

Generation	Accuracy	System Error	Edge weight per microclassifier	Edge weight per rule
5000	0.6	467.658	13.564	541.320
10000	0.6	425.089	8.570	260.379
15000	0.78	303.681	5.934	244.428
20000	0.96	174.084	3.696	164.941
25000	1	61.981	2.422	118.396
30000	1	28.587	2.648	144.453
35000	1	12.024	2.345	98.749
40000	1	55.630	2.884	84.737
45000	1	6.185	2.128	72.584
50000	1	5.182	2.323	241.563
60000	1	10.316	2.311	52.768
70000	1	18.228	1.858	55.337
80000	1	5.059	1.974	54.883
90000	1	6.127	2.435	222.440

In Table 6.1, we list the observations at various generations. We also compute two measures of correlations over all data from the experiments:

1. Correlation between the System Error and the Average edge weight per microclassifier is 0.848, and

2. Correlation between the System Error and the Average edge weight per rule is 0.857.

Both correlations have p-values < 0.001, which are indicative of strong correlations between the XCS evaluation measures and the corresponding network measures.

The Unweighted Network

In the second network described above, we set a threshold of half the maximum similarity possible as the criterion for the existence of an edge. Creating the network thus, this time we plot the distributions of degrees (after normalization to the range $[0, 1]$). We align boxplots of these distributions

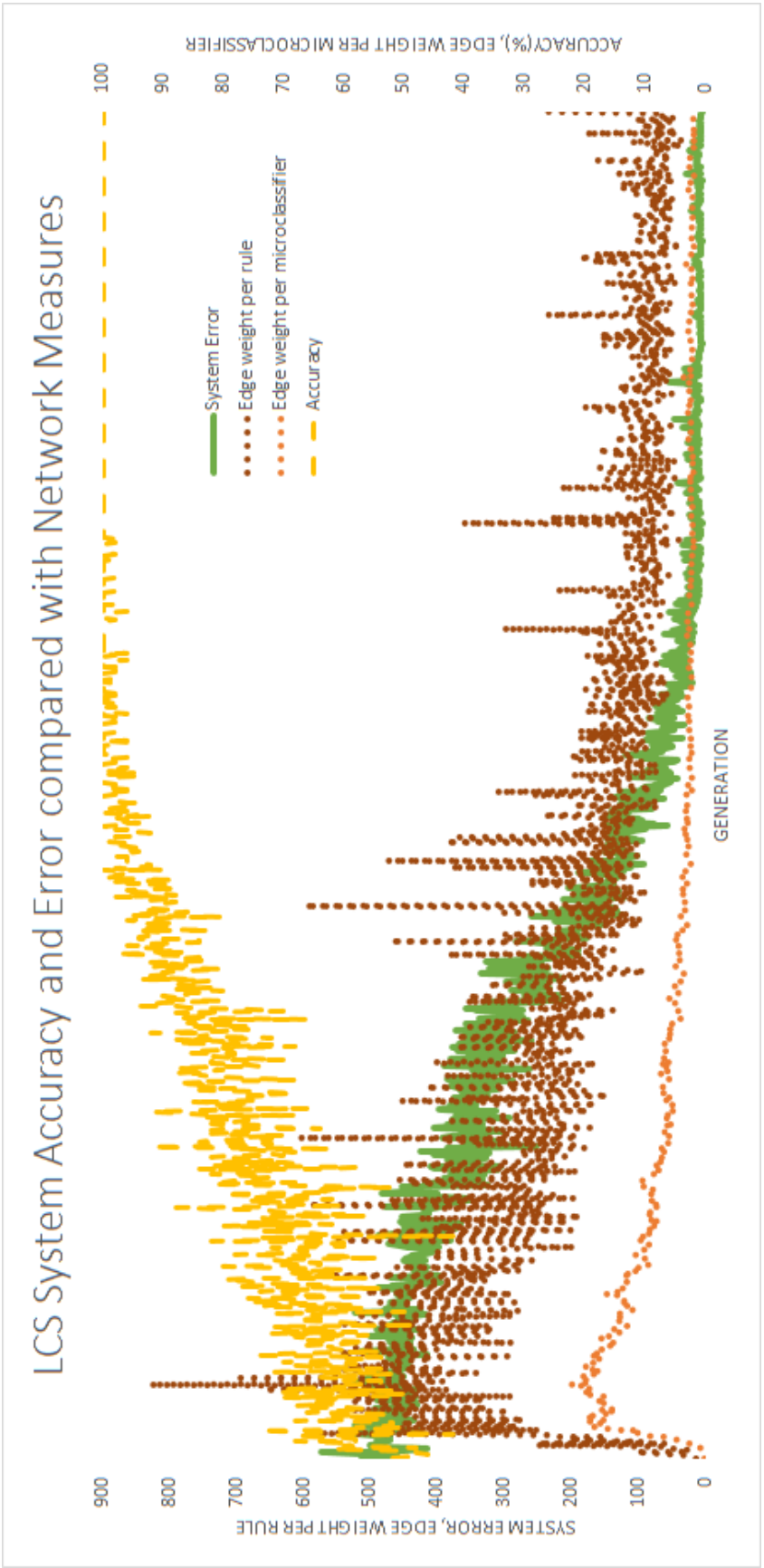


Fig. 6.1: Accuracy, System Error compared with the Edge weights in XCS: 20-mux

and the XCS measures, and plot them against the Generation $(1 - 50,000)$ on the X-axis as seen in Figure 6.2. Here, too, except the degree distributions (which are plotted at those instants), all values are averaged over the past 50 runs.

Table 6.2: XCS Evaluation measures: Accuracy, System Error compared with Average Rule Degree and its Standard Deviation

Generation	Accuracy	System Error	Avg. Rule Degree	Rule Std. Dev.
5000	0.66	445.507	0.399	0.166
10000	0.7	414.770	0.380	0.183
15000	0.86	312.837	0.582	0.211
20000	0.92	292.491	0.825	0.139
25000	0.94	168.776	0.905	0.119
30000	0.98	59.863	1	0
35000	1	45.090	1	0
40000	1	23.420	0.974	0.051
45000	1	6.714	1	0
50000	1	9.198	1	0
60000	1	7.546	1	0
70000	1	20.975	1	0
80000	1	13.618	1	0
90000	1	8.356	1	0

From Table 6.2 and Figure 6.2, we observe that once the boxplots stabilize, there is negligible change in the System Error. We let the experiment run for about $100,000$ generations and note that in the second half, there is very little improvement as is noticed from the various properties. In other words, if we had observed the run purely from the network theoretic standpoint, we could have stopped the run reasonably with very little accuracy sacrificed by Generation $30,000$.

We also observe from Figure 6.2 that the degree distribution is well aligned with the accuracy/system error plots. This is yet another indication of network measures being well synchronized with EA measures.

6.6 Summary

In a bit-string system as seen in an LCS, the similarities between rules are high (which is dependent on the problem being solved and the input points at each generation) when we compare

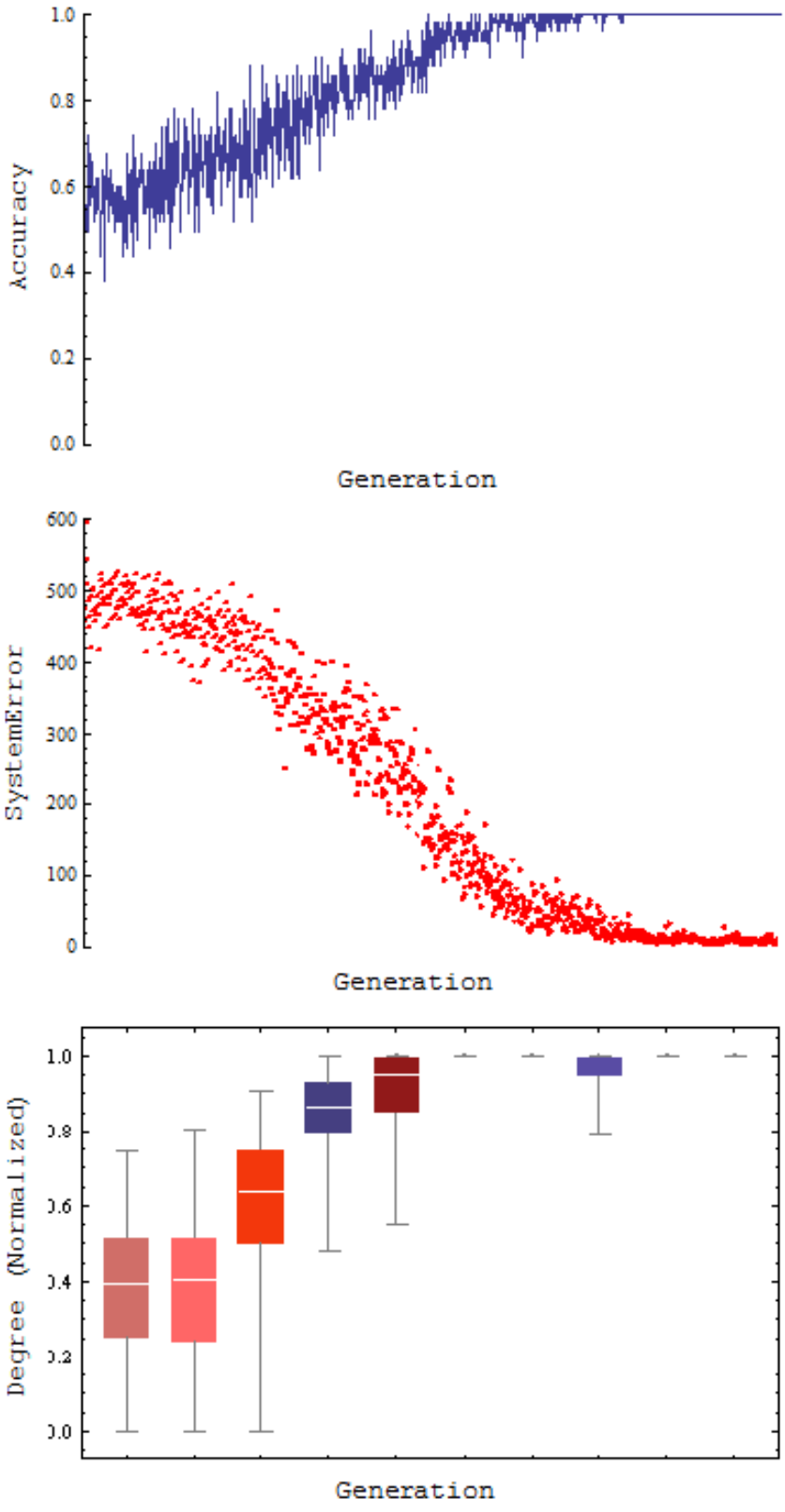


Fig. 6.2: Accuracy, System Error compared with Degree Distributions in XCS for the 20-mux problem

bit-positions. The weight threshold (to determine when an edge exists in the network) can be modified for the problem at hand. More importantly, we notice a strong correlation between the edge weights and node degree distributions and the corresponding convergence criteria in the XCS algorithm (high accuracy as well as low system error).

Although it is hard to determine when the system error will reach 0, if at all, it is easier to observe the network measures which clearly converge, which is noticed far more easily in the case of degree distributions. For the same parameters, we continued running the XCS for another $50,000$ generations, but the graph was a continuation of the values seen from about $30,000 - 50,000$ and are hence not plotted.

Through these experiments, we observe that although we examine the LCS, which is a completely different EA system (that involves ternary strings, has many other processes alongside a GA, and involves multi-layered individuals – macroclassifiers and microclassifiers), we are still able to identify and draw parallels with the interrelationships that they share in space. Strong correlations between the network measures and the LCS evaluation measures suggest that the networks and the EA are in synchrony, i.e., the dynamical changes in network and EA measures are very well synchronized.

We also wish to note that these observations can lead to enhancements in the LCS as we have done for GAs and PSOs. However, the translation from those EAs into LCSs is unclear because of the following reasons:

- The convex hull property as noticed before does not apply here because each rule could represent varying numbers of points in space, depending on the number of don't-cares ($\#$).
- This is an online learning system which is another difference when compared with GAs and PSOs. That is, new points are being input to the system at each generation. This leads to rule-sets evolving continuously until the rule-set yielding the exact match is evolved as shown in the example rule-set for 6-mux in (6.2). As the problem complexity increases, the rule-set may or may not evolve to the exact solution for the chosen parameters. This leads to the rule-set constantly evolving, and can represent varying positions in space. Hence, since

the position of the rules may not be nearby, the convex hull collapse equivalent is not yet clear.

Nevertheless, the similarity between rules in an XCS and the distance threshold between individuals in a GA or a PSO have similar characteristics and would hence promises to be fruitful. This is a subject for future work.

Having examined different varieties of EAs and different kinds of networks, we now begin a summary of the dissertation by discussing our observations in the next chapter.

CHAPTER 7
DISCUSSION

In this dissertation, we have shown that the application of EAs for difficult optimization problems in multiple dimensions can be assisted significantly by the use of network theoretic abstractions. Two key questions faced by a practitioner involve identifying whether (premature) convergence has occurred, and timing the initiation of random restarts, i.e., reseeding the population to trigger exploitation of new regions of the search space. Results in this work have shown that rather than *detect* convergence after it has occurred, we can *predict* convergence very early, using network-theoretic analysis, thereby saving large amounts of computational effort. Indeed, most evolutionary algorithms spend much greater computational effort in (exploiting) a very small region of the search space, whereas significantly better results could be obtained by shifting the balance towards exploration of other regions. Whereas other researchers have relied on inferences drawn from the slowing down of improvements obtained by a GA, we rely on rapid analysis of the underlying network, well before such slowing down has occurred.

In the context of reseeding, the well-known *stability-plasticity dilemma* applies in this context: too much replacement leads to loss of much useful information, and too little replacement may be ineffectual, especially since the selection process may immediately eliminate randomly generated individuals that are likely to be poorer in quality than individuals in well-explored (though sub-optimal) regions of search space. This is the reason we attempted a few different combinations

of fractions of populations to retain and reseed. But we assume that a majority must be retained, because if we reseed the majority, that would mean that the work done by the EA would have been mostly undone.

At a high level, this research illustrates that many dynamical systems can be understood by first representing the system as a network, then identifying the most relevant network-theoretic properties that influence, mirror, or predict observable EA system behavior, and finally executing systemic modifications based on the network model in order to yield desirable behavior such as more optimal function values or better learning.

In the next three sections, we point out three topics of interest that require some discussion – the kinds of networks that form in different types of EAs, highlight the role of reseeding, and finally compare networks and clusters in an EA.

7.1 Comparison of Network Formations in Different EAs

Apart from the many benefits of networks, we wish to look at the differences in network formations in each of these EAs.

In GAs, looking at Tables 4.2, 4.3, and 4.4, we find that the number of generations required to determine convergence varies considerably based on the characteristics of the function being optimized. That is, when the values are compared with that required for a PSO as seen in Tables 4.13, 4.14 and 4.15, where the number of generations required to detect imminent convergence are somewhat in between the large varying corresponding values seen in a GA. The tables we compare are for three different population sizes – $50, 100,$ and 150.

Arguably, this is because a GA is more "greedy" than a PSO in the sense that there could be wild jumps in the position of new individuals in a GA, since the new position depends on the crossover and mutation alone, and not on other features as in a PSO, such as inertia and local best which render its movement more "measured" and "steady". In other words, the new position in a GA is dependent only on the fitness of points selected for reproductive operations which is

comparable to the effect of the global best in a PSO.

This difference becomes clearer when we divide the test problems into its characteristic features determining its complexity, i.e., the modality of the problem and the regularity of the surface. For instance, De Jong's first function is the easiest of the lot where a greedy approach is likely to be very successful, where the networks in a GA identify the sufficient number of generations to be around $40 - 60$ whereas the components in a PSO recommend around $80 - 90$. For Schwefel's function, however, the GA required in many cases $120 - 150$ generations, when compared with the PSO requiring about $80 - 90$ generations. One reason for this could be that the inertial component as well as the personal best components of the PSO (which are both lacking in the GA) contribute to a steadier movement of each point across the search space since they emphasize the current position, best position, and velocity of the point, as opposed to those in a GA which could involve larger jumps due to crossover between points far away from each other.

From the perspective of edge weights, the rise and fall of edge weight per rule in an LCS are similar to the corresponding rise and fall of the components seen in a GA or a PSO. In other words, there is a rise in the average edge weight early on, and then a drop, which is similar to the properties observed in the convex hull volume in a GA. Since we retain these rules as ternary strings and do not convert them to real values, the convex hull and component/community definitions are much less useful. For this reason, edge weighted networks are more appropriate in LCSs. Monotonicity makes it hard to predict when the measure stabilizes, as described in the system error plot, however, an appropriate algorithm may use the top of a rise as a baseline and consequently stop the run similar to the termination criteria used for GAs and PSOs.

7.2 Reseeding

Reseeding plays a very important role, especially in a GA, where the population may ignore many potential basins of attraction and converge. Observing the rise and fall of the component size establishes clear time instants where reseeding may help in setting a fraction of the population

loose to explore other regions, and yet retain another fraction to preserve the work done by the GA.

We have explored various ways of reseeding as described previously, and observe the roles that networks and communities play in determining time instants to reseed. Essentially, it is a tradeoff between exploration and exploitation. In our experiments, the main indicator that the networks have provided towards this process is the determination of *when* to reseed, as opposed to *how* to reseed. However, many other pieces of information inherent in networks may provide better reseeding solutions.

7.3 Comparison: Network formation vs. Clusters

Distance based networks have some similarities with clusters, although ancestral networks do not. We now address one potential question that might arise, viz., why not use clusters in place of distance-based networks? We explain below with examples of cluster formation as well as a complexity analysis argument that there are differences between the two.

Network creation is preferred to clustering for analyzing the population structure in EAs because of the following reasons:

1. Clustering methods such as k-means clustering require the knowledge of k, the number of clusters, which can be difficult to determine in such evolving populations. Such methods also assume that clusters are symmetric (spherical, ellipsoidal, etc.), preventing the discovery of asymmetric structures in data.

2. In the case of clustering techniques such as hierarchical clustering where this is not an issue (since a hierarchical tree or dendrogram is created), the following concerns arise:

   (a) The tree is suitable to be pruned at any level which requires an input such as the number of clusters, or a threshold to determine natural divisions within the data (e.g., tree depth).

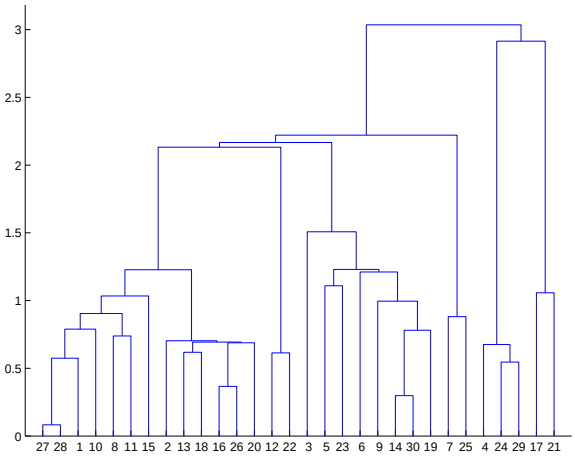
(b) The complexity exceeds that of creating a network. This form of clustering has $O(n^3)$ complexity for an agglomerative clustering [18], whereas network creation needs only $O(n^2)$ steps, since only pairwise distances need to be computed. In fact, distance computations between all pairs of nodes is the first step in the hierarchical clustering process, before the creation of the tree structure and the pruning process.

3. Finally, there is also the requirement in clustering that the intra-cluster distances have to be low, and inter-cluster distances must be high. This is a strict requirement that we believe is not necessary in order to solve optimization problems. Long chains, spirals, or other form of irregular structures are acceptable, as long as they are connected. We also presented an example (in Figure 3.1b) of such structures.[1]

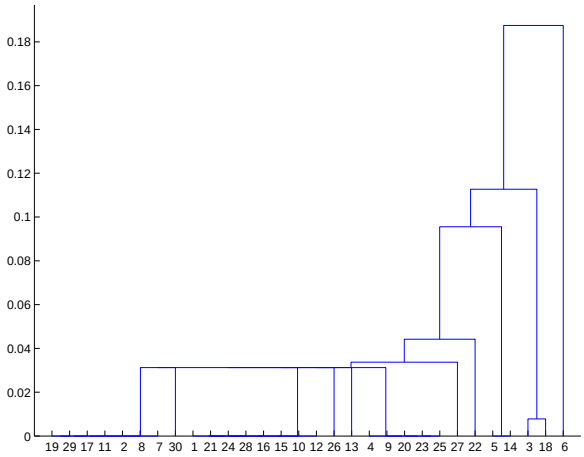
The clustering approach that most resembles the network approach is single linkage clustering [67] (similar to Kruskal's minimum spanning tree algorithm [38]). This form of network creation stops at the first step of this algorithm, i.e., determining pairwise distances. Although this form of clustering requires only $O(n^2)$ effort, since only the relative distances are of interest for our problem, we cannot determine the difference between tree structures that are generated early in evolution and those generated late in evolution as seen in Figures 7.1a-7.1k.

These figures display results for every 20^{th} generation of Rastrigin's function. The X-axis denotes the cluster ID, and the Y-axis denotes the function value. Joined objects imply individuals linked together, numbers on the X-axis are population identifiers. We observe that no single cluster is formed toward convergence although the function values get smaller, and it does not appear easy to draw any conclusions from the structure of the clusters. This is a result of relative distances between individuals maintaining an almost fractal-like behavior. The only change is that the absolute distances between individuals and clusters themselves reduce, but this is not a factor in cluster formation. In other words, the kinds of clusters that form do not vary in structure by much even

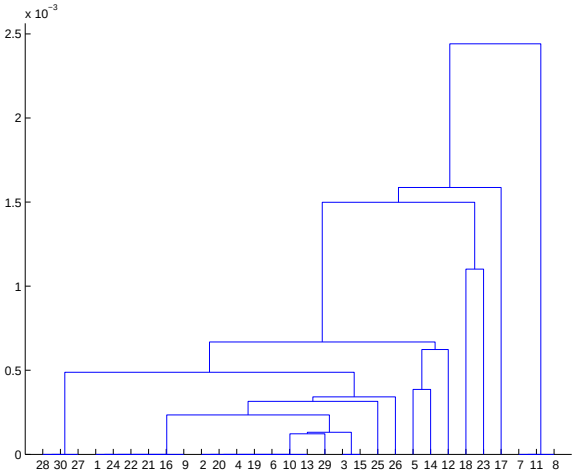
[1]The differences between clusters and networks from the structural point of view is also apparent when observing examples of community activity in nature. A flock of geese form a V-shaped network, but not a regular cluster. A group of ants headed towards food, form a connected component by going very close to each other in a line, but not a regular cluster.



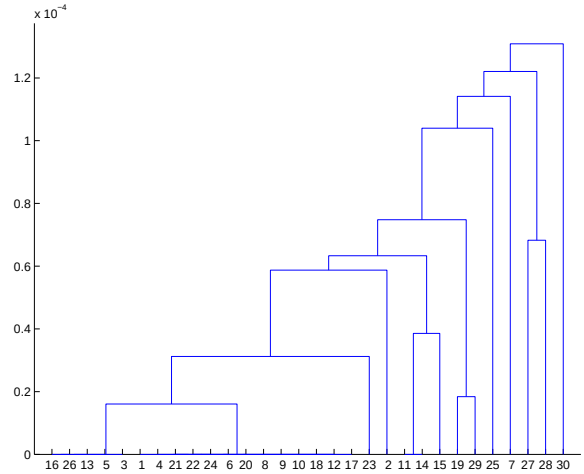
(a) Initial Population



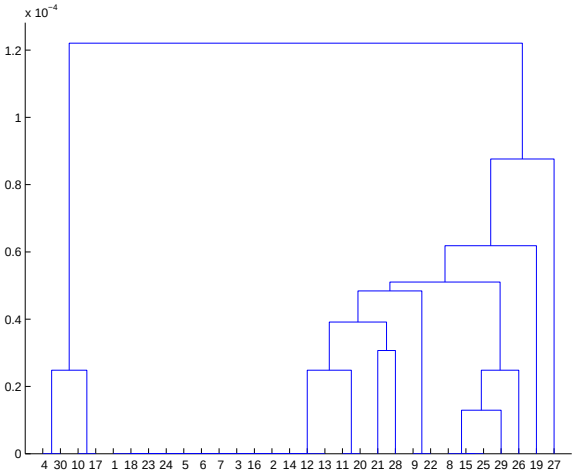
(b) Generation 20



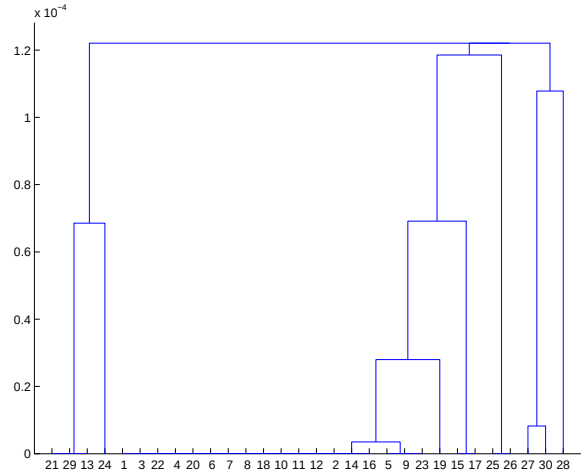
(c) Generation 40



(d) Generation 60

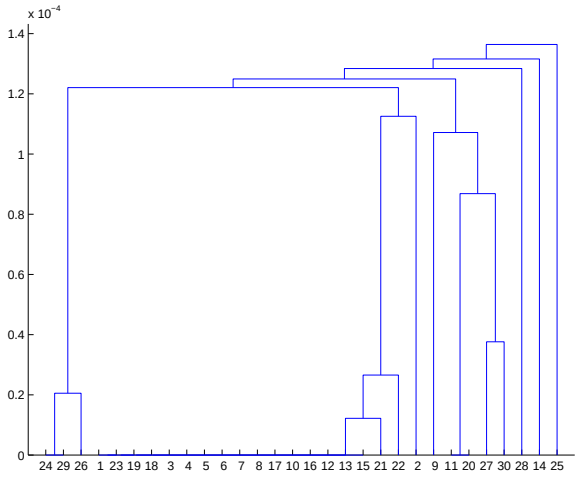


(e) Generation 80

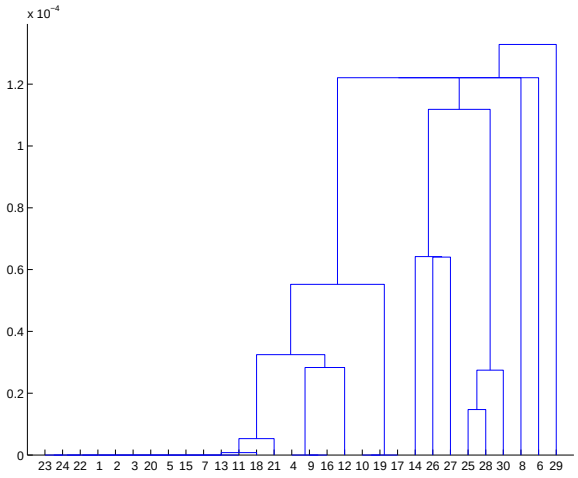


(f) Generation 100

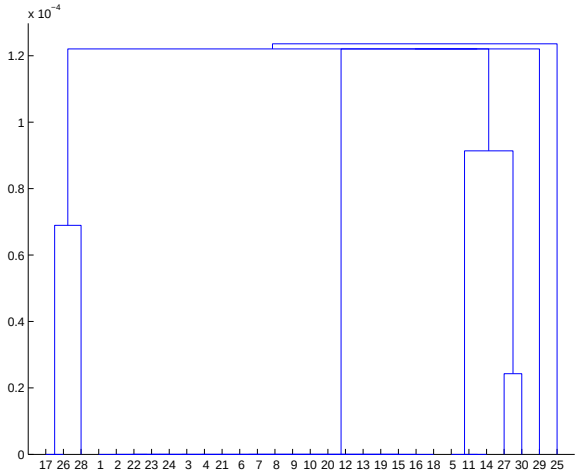
Fig. 7.1: Clusters formed at Generations 0-200. Function Values vs. Cluster IDs



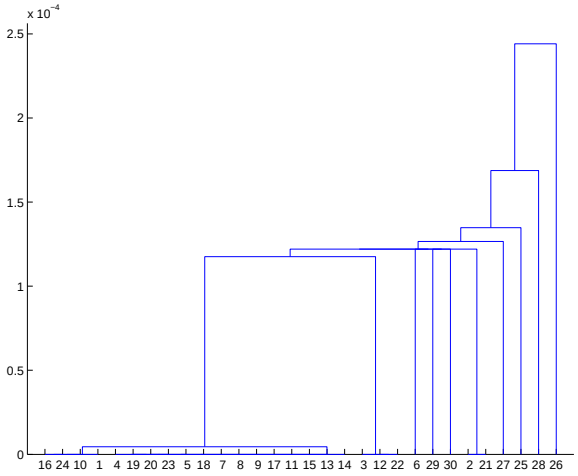
(g) Generation 120



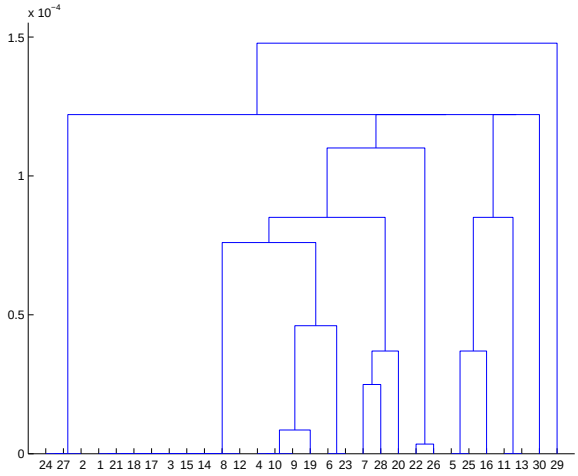
(h) Generation 140



(i) Generation 160



(j) Generation 180



(k) Generation 200

Fig. 7.1: (..contd.) Clusters formed at Generations 0-200. Function Values vs. Cluster IDs

though they are much closer to each other, or near convergence, as opposed to early phases of evolution.

CHAPTER 8
CONCLUSIONS AND FUTURE WORK

There are many benefits to analyzing EAs as evolving networks or communities or components as opposed to evolving individuals of a population. We have described a few ways of using networks through a series of observations, correlations and algorithms, but emphasize that there are potentially many more uses of networks, and a detailed study of networks can prove to be very useful.

In other words, while observing and imitating ideas from nature, we have barely scratched the surface of biology or ecology, and the next step we have begun to take through these experiments is that we imitate the interrelationships and communities formation characteristics that are observed in the biosphere.

8.1 Contributions

The central thesis that we have pursued in this dissertation is that network models can be formulated to represent inherent relationships amongst individuals in the populations of an EA, and that there is potential benefit in studying relationships between them.

We divide the contributions in this dissertation into five main sections. We have formulated and attempted to answer the following problems:

1. **Introduction and Exploration of Network Views of a Population-based EA**

   We addressed a mostly-ignored area of population-based EAs, viz., the presence of underlying interrelationships among individuals of a population. Can we use such additional information that may lie hidden in the evolving generations of individuals?

2. **Convergence Detection**

   We then applied the network view towards identifying communities or connected components, identifying impending convergence, and extracting information about the EA (using the network view) in various ways. The crucial questions are: When do we believe that we have gotten all that we can from the EA? When can we stop the EA run without sacrificing quality?

3. **Restarts or Reseeds**

   Given the identification of imminent convergence (above), how can we put this information to use and identify time instants to restart, or reseed the algorithm?

4. **Applicability to Other EAs**

   We explored swarm networks in PSOs and rule networks in LCSs. Are the networks in these other EAs comparable to those seen in GAs? Can we generalize?

5. **Mimicking Nature**

   Finally, continuing along the philosophy that we are mimicking natural (biological) evolution, where do we stand? How much have we managed to imitate, and what can we strive for in the future? We discuss this now.

8.2 Open Questions

Having opened up the issue of unexplored or under-explored ideas in interrelationships between individuals working towards an optimization goal, we continue on the same theme to identify future directions of research.

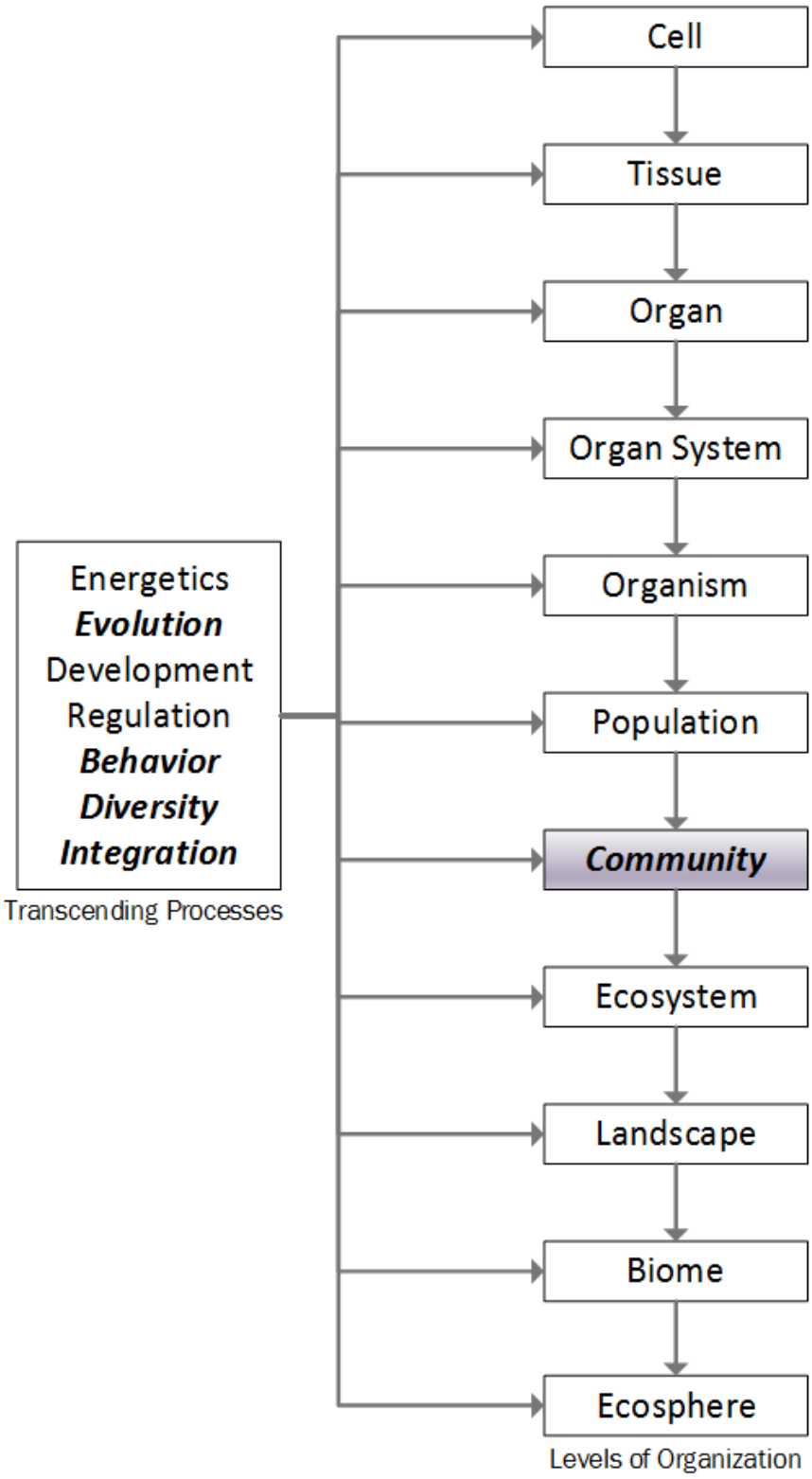


Fig. 8.1: Levels of Organization based on Barrett et. al. [7] - Seven transcending processes in each of eleven levels of organization - Our focus in bold/italics

1. We, as individuals, experience many parallel networks in our lives, such as a social network of friends, a family network, a professional network of colleagues that we work with, a school or company or city network where we live and is geography-based, and so on. Translating this into our optimization problem, can we identify and build multiple parallel networks and identify how they co-evolve in order to better understand the search and optimization problem structure?

   These could be as simple as the Euclidean and Ancestral networks generated in parallel. It could also involve another form of network, for instance, the similarity measure as in the LCS, when the real values are converted to bit-strings. For example, the real-numbers 32 and 48 might appear very far on the real-scale, but if we convert it into bit-strings, we notice that they are 100000 and 110000, which is a difference in only one bit. This can be considered roughly analogous to very minor DNA differences between humans and apes, but very large physical differences in appearance, behavior, and intelligence.

2. The notion of partitioning and cell structures are evident in Self-Organizing or Kohonen Maps [37], Voronoi Tessellation [74], or its dual form, Delaunay Triangulation [21], among others. The spatial information that is contained in these techniques is evident with the well-defined cells, neighborhoods, adjacencies, and cell partitions. The availability of such distance and neighborhood information immediately brings to mind the likelihood of underlying Euclidean networks. How would the view of individuals forming networks help in improving the problem solving ability of these techniques?

3. Consider a hierarchical structure that is usually seen in our everyday life: workplace hierarchies, family hierarchies, political hierarchies, legal hierarchies, etc. This is not just a human created concept, it is known to exist in other species as well, obviously at a more rudimentary level, e.g., a bee colony with a queen bee, the lead goose in a flock of geese, and so on.

   Observing this, we ask how best can we create such hierarchies in an EA? Could they correspond to highest fit individuals having veto-like powers over the choice of parents? Or could

it be the same at the component or community level, i.e., at the sub-population level?

These questions are interesting from many points of view: First, from the standpoint of the *breadth* of ideas that we can borrow from nature, and second, the potential *depth* of unexplored ideas in underlying networks, parallel communities, hierarchies, and more complex interactions between members of a species or different species.

In the big picture, we can evolve communities with a focus on evolution, behavior, diversity, and integration based on Barrett's levels of organization [7], shown in Figure 8.1. The final question is – How best can we mimic the plethora of unexplored biological ideas at the community level and above (until the ecosphere) into evolutionary algorithms to achieve better search, optimization, and understanding of EAs? Can we create an ecosphere in an EA?

REFERENCES

[1] D. Ackley. 1987. An Empirical Study of Bit Vector Function Optimization. Genetic Algorithms and Simulated Annealing, pp. 170-215.

[2] M. M. al-Rifaie and J. B. Mark. 2013. Stochastic Diffusion Search Review. Paladyn, Journal of Behavioral Robotics 4 (3): 155-73.

[3] E. Alba and J. M. Troya. 1999. A Survey of Parallel Distributed Genetic Algorithms. Complexity 4 (4): 31-52.

[4] H. Aytug and G. J. Koehler. 2000. New Stopping Criterion for Genetic algorithms. European Journal of Operational Research 126, no. 3, pp. 662-674.

[5] T. Bäck, D. B. Fogel, and Z. Michalewicz. 2000. Evolutionary Computation 1: Basic Algorithms and Operators. CRC Press.

[6] J. S. F. Barker. 2009. Defining Fitness in Natural and Domesticated Populations. In Adaptation and Fitness in Animal Populations, pp. 3-14. Springer.

[7] G. W. Barrett, J. D. Peles, and E. P. Odum. 1997. Transcending Processes and the Levels-of-organization Concept. Bioscience 47 (8): 531-5.

[8] M. V. Butz. 2000. XCSJava 1.0: An Implementation of the XCS Classifier System in Java. Technical Report 2000027, Illinois Genetic Algorithms Laboratory.

[9] M. V. Butz. 2000. XCS Implementation in Java, version 1.0 (Source Code: http://www.illigal.uiuc.edu/pub/src/XCSJava/XCSJava1.0.tar.Z). Retrieved Apr 4, 2009.

[10] M. V. Butz and S. W. Wilson. 2002. An Algorithmic Description of XCS. Journal of Soft Computing, 6 (2002) 144–153.

[11] M. V. Butz. 2005. Rule-based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design. Vol. 191. Springer.

[12] E. Cantú-Paz. 2000. Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, Norwell, MA.

[13] S. Chen. 2009. Another Particle Swarm Toolbox (Source Code: http://www.mathworks.com/matlabcentral/fileexchange/25986), MATLAB Central File Exchange. Retrieved Sep 17, 2013.

[14] M. Clerc and J. Kennedy. 2002. The Particle Swarm - Explosion, Stability, and Convergence in a Multidimensional Complex Space. Evolutionary Computation, IEEE Transactions on 6 (1): 58-73.

[15] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. Introduction to Algorithms. 2001. 2nd ed. McGraw-Hill Higher Education.

[16] M. Črepinšek, S. H. Liu, and M. Mernik. 2013. Exploration and Exploitation in Evolutionary Algorithms: A survey. ACM Computing Surveys (CSUR) 45 (3): 35.

[17] C. Darwin. 1859. On the Origin of Species. John Murray, London.

[18] W. H. E. Day and H. Edelsbrunner. 1984. Efficient Algorithms for Agglomerative Hierarchical Clustering Methods. Journal of Classification 1 (1): 7-24.

[19] K. A. De Jong. 1975. An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Ph.D. Thesis, University of Michigan.

[20] K. A. De Jong. 1990. Genetic-algorithm-based Learning. San Francisco, CA. Morgan Kaufmann Publishers Inc.

[21] B. Delaunay. 1934. Sur la sphère vide. À la mémoire de Georges Voronoï. Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk 7: 793-800.

[22] J. Drugowitsch. 2008. Design and Analysis of Learning Classifier Systems: A Probabilistic Approach. Vol. 139. Springer.

[23] E. E. Easom. 1990. A Survey of Global Optimization Techniques. M. Eng. Thesis, University of Louisville.

[24] R. C. Eberhart and Y. Shi. 2001. Particle Swarm Optimization: Developments, Applications and Resources. Evolutionary Computation, 2001. Proceedings of the 2001 Congress on, Vol.1, pp.81-86, 2001.

[25] A. E. Eiben, P. Raue, and Z. Ruttkay. 1994. Genetic Algorithms with Multi-parent Recombination. In Parallel Problem Solving from Nature – PPSN III, edited by Yuval Davidor, Hans-Paul Schwefel, and Reinhard Manner, 866:78-87. Berlin, Heidelberg: Springer Berlin Heidelberg.

[26] L. J. Eshelman. 1990. The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. Foundations of Genetic Algorithms: 265-83.

[27] M. S. Gibbs, H. R. Maier, G. C. Dandy, and J. B. Nixon. 2006. Minimum Number of Generations Required for Convergence of Genetic Algorithms. In IEEE Congress on Evolutionary Computation, 2006. CEC 2006, 565-572. IEEE.

[28] D. E. Goldberg. 1983. Computer-aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning. Ph.D. Dissertation, University of Michigan.

[29] D. E. Goldberg. 1989. Genetic Algorithms in Search, Optimization and Machine Learning. 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

[30] D. E. Goldberg. 2002. The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, Norwell, MA.

[31] D. Greenhalgh and S. Marshall. 2000. Convergence Criteria for Genetic Algorithms. SIAM J. Comput. 30, no. 1, pp. 269-282.

[32] J. H. Holland and J. S. Reitman. 1978. Cognitive Systems Based on Adaptive Algorithms. In D. Waterman & F. Hayes-Roth (eds) Pattern-directed Inference Systems. Academic Press.

[33] J. H. Holland. 1992. Adaptation in Natural and Artificial Systems. Cambridge, MA. MIT Press.

[34] J. Horn, D. E. Goldberg, and K. Deb. 1994. Implicit Niching in a Learning Classifier System: Nature's Way. Evolutionary Computation 2 (1): 37-66.

[35] M. Iqbal. 2014. Improving the Scalability of XCS-based Learning Classifier Systems. Ph.D. Thesis, Victoria University of Wellington.

[36] J. Kennedy and R. C. Eberhart. 1995. Particle Swarm Optimization. In Proceedings of IEEE International Conference on Neural Networks, vol. IV, pp.1942–1948.

[37] T. Kohonen. 1982. Self-organized Formation of Topologically Correct Feature Maps. Biological Cybernetics 43 (1): 59-69.

[38] J. B. Kruskal Jr. 1956. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society, Vol. 7, No. 1, pp. 48-50.

[39] K. Kuber and C. K. Mohan. 2009. Biasing Evolving Generations in Learning Classifier Systems using Information Theoretic Measures. In Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. GECCO '09. ACM, New York, NY, 2077-80.

[40] K. Kuber. 2009. Improving Learning Classifier System Performance Using Information Theoretic Fitness Measures. M.S. Thesis, Syracuse University.

[41] K. Kuber and C. K. Mohan. 2010. Information Theoretic Fitness Measures for Learning Classifier Systems. In Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation. GECCO '10. ACM, New York, NY, 1885-1892.

[42] K. Kuber, S. W. Card, K. G. Mehrotra, and C. K. Mohan. 2012. A Network Theoretic Analysis of Evolutionary Algorithms. In Swarm, Evolutionary, and Memetic Computing. Lecture Notes in Computer Science 7677, pp. 585-593. Springer Berlin Heidelberg, 2012.

[43] E. Lieberman, C. Hauert, and M. A. Nowak. 2005. Evolutionary Dynamics on Graphs. Nature 433 (7023): 312-6.

[44] X. Llorà, K. Sastry, D. E. Goldberg, and L. de la Ossa. 2006. The χ-ary Extended Compact Classifier System: Linkage Learning in Pittsburgh LCS. Advances at the Frontier of Learning Classifier Systems, 2.

[45] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson. 2003. Dolphin Social Network. Behavioral Ecology and Sociobiology 54, 396-405.

[46] D. Lusseau and M. E. Newman. 2004. Identifying the Role that Animals Play in their Social Networks. Proceedings. Biological Sciences / the Royal Society 271 Suppl 6 (Dec 7): S477-81.

[47] S. W. Mahfoud. 1995. Niching Methods for Genetic Algorithms. Ph.D. Dissertation, University of Illinois at Urbana-Champaign.

[48] The Mathworks, Inc. 2011. MATLAB. Version 2011b, Natick, MA.

[49] K. Möbius. 1877. Die Auster und die Austernwirtschaft. (tr. The Oyster and Oyster Farming) Berlin. (English translation) U.S. Commission Fish and Fisheries Report, 1880: 683-751.

[50] C. K. Mohan and K. Kuber. 2009. Improving Performance of Learning Classifier Systems. In Proceedings of Frontiers of Interface between Statistics and Sciences, Hyderabad, India, December 2009.

[51] A. Moraglio. 2007. Towards a Geometric Unification of Evolutionary Algorithms. Ph.D. Thesis, Department of Computer Science, University of Essex.

[52] P. B. Nair and A. J. Keane. 2001. Passive Vibration Suppression of Flexible Space Structures via Optimal Geometric Redesign. AIAA Journal 39, pp. 1338-1346.

[53] M. E. J. Newman. 2010. Networks: An Introduction. Oxford University Press.

[54] L. K. Nyhart. 1998. Civic and Economic Zoology in Nineteenth-century Germany: The Living Communities of Karl Möbius. Isis 89 (4): 605-30.

[55] H. A. Orr. 2009. Fitness and its Role in Evolutionary Genetics. Nature Reviews Genetics 10 (8): 531-9.

[56] S. Pandit, Y. Yang, V. Kawadia, S. Sreenivasan, and N. V. Chawla. 2011. Detecting Communities in Time-Evolving Proximity Networks. In 2011 IEEE Network Science Workshop (NSW), 173-179. IEEE.

[57] C. B. Pettey, M. R. Leuze, and J. J. Grefenstette. 1987. A Parallel Genetic Algorithm. In Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, 155-161. Hillsdale, NJ, USA: L. Erlbaum Associates Inc.

[58] R. Poli. 2008. Analysis of the Publications on the Applications of Particle Swarm Optimisation. Journal of Artificial Evolution and Applications 2008 : 3.

[59] B. Raphael and I. F. C. Smith. 2003. A Direct Stochastic Algorithm for Global Search. Applied Mathematics and Computation 146 (2-3) (12/31): 729-58.

[60] L. A. Rastrigin. 1974. Extremal Control Systems. Theoretical Foundations of Engineering Cybernetics Series, Moscow, Nauka, Russian.

[61] M. Safe, J. Carballido, I. Ponzoni, and N. Brignole. 2004. On Stopping Criteria for Genetic Algorithms. In Advances in Artificial Intelligence – SBIA 2004, edited by Ana L. C. Bazzan and Sofiane Labidi, 3171:405-413. Berlin, Heidelberg: Springer Berlin Heidelberg.

[62] H. P. Schwefel. 1981. Numerical Optimization of Computer Models. John Wiley & Sons, Chichester. English translation of Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie (1977).

[63] C. E. Shannon. 1948. A Mathematical Theory of Communication. Bell System Technical Journal, vol. 27, pp. 379-423, 623-656, July, October, 1948.

[64] L. D. Shi, Y. H. Shi, Y. Gao, L. Shang, and Y. B. Yang. 2011. XCSc: A Novel Approach to Clustering with Extended Classifier system. International Journal of Neural Systems 21 (01): 79-93.

[65] Y. Shi and R. Eberhart. 1998. A Modified Particle Swarm Optimizer. Paper presented at Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence.

[66] Y. Shi and R. Eberhart. 1998. Parameter Selection in Particle Swarm Optimization. Paper presented at Evolutionary Programming VII.

[67] R. Sibson. 1973. SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method. The Computer Journal 16 (1): 30-4.

[68] R. E. Smith. 1980. A Learning System based on Genetic Algorithms. Ph.D. Dissertation, Computer Science Department, University of Pittsburgh.

[69] R. E. Smith and M . K. Jiang. 2007. MILCS: A Mutual Information Learning Classifier System. In Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation. London, United Kingdom: ACM, pp. 2945-2952.

[70] H. Spencer. 1864. The Principles of Biology.

[71] R. Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. SIAM Journal on Computing 1 (2): 146-60.

[72] R. J. Urbanowicz and J. H. Moore. 2009. Learning Classifier Systems: A Complete Introduction, Review, and Roadmap. Journal of Artificial Evolution and Applications.

[73] R. J. Urbanowicz, A. Granizo-Mackenzie, and J. H. Moore. 2012. An Analysis Pipeline with Statistical and Visualization-guided Knowledge Discovery for Michigan-style Learning Classifier Systems. Computational Intelligence Magazine, IEEE 7 (4): 35-45.

[74] G. Voronoï. 1908. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites. Journal Für Die Reine Und Angewandte Mathematik 133 : 97-178.

[75] C. J. C. H. Watkins. 1989. Learning from Delayed Rewards. Ph.D. Thesis, Cambridge University.

[76] P. A. Whigham and G. Dick. 2008. Exploring the use of Ancestry as a Unified Network Model of Finite Population Evolution. Paper presented at Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence).

[77] S. W. Wilson. 1985. Knowledge Growth in an Artificial Animal. In Proceedings of the 1st International Conference on Genetic Algorithms, John J. Grefenstette (Ed.). L. Erlbaum Associates Inc., Hillsdale, NJ, 16-23.

[78] S. W. Wilson. 1994. ZCS: A Zeroth Level Classifier System. Evolutionary Computation 2 (1): 1-18.

[79] S. W. Wilson. 1995. Classifier Fitness based on Accuracy. Evolutionary Computation 3 (2): 149-75.

[80] Wolfram Research, Inc. 2012. Mathematica. Version 9.0, Champaign, IL.

[81] J. Zhang, H. S. H. Chung, and W. L. Lo. 2007. Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms. IEEE Transactions on Evolutionary Computation 11, no. 3, pp. 326-335.

VITA

NAME OF AUTHOR: Karthik Kuber

PLACE OF BIRTH: Hyderabad, Andhra Pradesh, India

DATE OF BIRTH: 31 December 1982

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

- B. M. S. College of Engineering, Visveswaraiah Technological University, Bangalore, India, 2000-2004
- L. C. Smith College of Engineering and Computer Science, Syracuse University, Syracuse, NY, USA, 2007-2014

DEGREES AWARDED:

- Bachelor of Engineering in Electronics and Communication Engineering, 2004, Visveswaraiah Technological University
- Master of Science in Computer Science, 2009, Syracuse University

ASSOCIATION MEMBERSHIPS:

- Student Member: ACM, IEEE

PROFESSIONAL EXPERIENCE:

- Assistant Systems Engineer, Tata Consultancy Services, Ltd., Bangalore, India, 2004-2007

PUBLICATIONS:

1. K. Kuber, S. W. Card, K. G. Mehrotra, and C. K. Mohan. 2012. A Network Theoretic Analysis of Evolutionary Algorithms. In Swarm, Evolutionary, and Memetic Computing (Bhubaneshwar, India). Lecture Notes in Computer Science 7677, pp. 585-593. Springer Berlin Heidelberg, 2012.
2. K. Kuber and C. K. Mohan. 2010. Information Theoretic Fitness Measures for Learning Classifier Systems. In Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation (Portland, OR, USA). GECCO '10. ACM, New York, NY, 1885-1892.
3. C. K. Mohan and K. Kuber. 2009. Improving Performance of Learning Classifier Systems. In proceedings of Frontiers of Interface between Statistics and Sciences (Hyderabad, India).
4. K. Kuber and C. K. Mohan. 2009. Biasing Evolving Generations in Learning Classifier Systems using Information Theoretic Measures. In Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers (Montréal, QC, Canada). GECCO '09. ACM, New York, NY, 2077-80.

MASTERS THESIS:

- K. Kuber. 2009. Improving Learning Classifier System Performance Using Information Theoretic Fitness Measures. M.S. Thesis, Syracuse University.

PUBLICATIONS IN PROGRESS:

1. K. Kuber, S. W. Card, K. G. Mehrotra, and C. K. Mohan. 2014. Rule Networks in Learning Classifier Systems. Accepted for presentation at the Seventeenth International Workshop on Learning Classifier Systems (IWLCS). GECCO '14 (Vancouver, BC, Canada).
2. K. Kuber, S. W. Card, K. G. Mehrotra, and C. K. Mohan. 2014. Ancestral Networks in Evolutionary Algorithms. Accepted for presentation as a Poster/Short Paper. GECCO '14 (Vancouver, BC, Canada).