2010

Heuristics for Cultural Algorithm Knowledge Driven Search in Dynamic Social Systems

Viranthi Peiris
University of Windsor

HEURISTICS FOR CULTURAL ALGORITHM KNOWLEDGE DRIVEN SEARCH IN DYNAMIC SOCIAL SYSTEMS

By

Viranthi R. Peiris

A Thesis

Submitted to the Faculty of Graduate Studies
through Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada
2009

HEURISTICS FOR CULTURAL ALGORITHM KNOWLEDGE DRIVEN SEARCH IN DYNAMIC SOCIAL SYSTEMS

by

Viranthi R. Peiris

APPROVED BY

M. Khalid, External Reader
Department of Electrical and Computer Engineering

R. Kent, Internal Reader
School of Computer Science

Z. Kobti, Advisor
School of Computer Science

S. Bandyopadhyay, Chair of Defense
School of Computer Science

Date: October 21, 2009

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Population evolution algorithms such as Cultural Algorithms (CA) enable a global repository known as the belief space consisting of common cultural traits or generalized schemas to influence the population space. Two important aspects of CA are the knowledge and its propagation. Individuals in the population use social networks for communication. Knowledge representation is generally dependent on the application at hand. In this thesis the role of CA belief space knowledge in application neutral simulation is explored. A standard benchmark function is used to study the performance of various heuristics on the quality of the belief space knowledge. The function captures the characteristics of a neutral world in static and dynamic settings. A multi-agent simulation was designed where autonomous agents are able to communicate, acquire and exploit various knowledge types including topographic, domain, historical and situational. While all these strategies showed improvements when searching for the global maximum in both static and dynamic environments, we found that domain based topographic exploitation strategies of the landscape were the more efficient.

DEDICATION

To my loving parents, husband and son.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Kobti for his encouragement, guidance and immense help that enabled me to complete my research while juggling a new born and the thesis. Without his constant persistence, his innovative ideas, his own research and support throughout my entire M.Sc program I would not have been able to make this research a success. My sincere gratitude goes to Dr. Kent for his unconditional support and valuable advice. Also I would like to thank Dr. Khalid for his advice and support.

Finally I would like to thank my husband for his encouragement, immense patience, tolerance and support taking care of our son when ever he can to give me time to work on the thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Problem Definition

To explore population based strategies in which individuals within a specified proximity are grouped into social networks where they communicate with one another to share required information to reach an optimum global in static and dynamic environments. Furthermore, evaluate the role of different knowledge types in Cultural Algorithms on population based strategies.

1.2 Motivation

Finding the global optimum faster is a very important problem in Artificial Intelligence. Evolutionary algorithms such as, Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Cultural Algorithms (CA) and Genetic Algorithms(GA) are some of the algorithms that have been used to find the global optimum.

In ACO [Dorigo 2006]once the ants know the source and the destination, they are able to find the shortest path between the source and the destination. ACO has been applied to NP-hard problems, routing problems in telecommunication networks, industrial problems and others. This can also be used in many dynamic applications. Convergence is guaranteed but it takes a long time than most of the other heuristics. Even though the destination is unknown ants will be able to find the global optimum. It will take a long time to find it because the ants would follow the ant who has the highest fitness. This process would be repeated till the entire problem area has been explored. Then once the entire area is explored all the ants would go to the best fit location. This process would take a long time even though the global optimum is found. More information is given in

chapter 2.5.1.

With PSO [Eberhart, 1995] there is no guarantee that it will be able to find the global optimum since it is problem dependent. Similarly with GA there is no guarantee that it will be able to find the global optimum since this is also problem dependent and when it comes to our problem, GA will most likely find only suboptimal solutions. More information is given in Chapter 2.5.

CA enables population evolution over longer generation time, allowing faster convergence of the population. Knowledge plays an important role in Evolutionary Algorithms, particularly Cultural Algorithm. However, CA knowledge development is highly dependent on the application. Therefore an application neutral environment is required to evaluate the role of knowledge.

1.3 Thesis Statement

In this thesis we evaluate the role of different knowledge types in Cultural Algorithms. Using an application independent social simulation, one can isolate the role of each knowledge type in terms of its influence on the population's belief space observed by the population performance and convergence. The quest is to identify the extent to which each type of knowledge plays in goal optimization with the hypothesis that some types have more influence over others.

1.4 Thesis Contribution

In this research a multi-agent simulation was designed where autonomous agents within a defined proximity are grouped into social networks. They are able to communicate, acquire and exploit various knowledge types including topographic, domain, historical and situational. The Cones World Problem Generator [Morrison, 1999]a standard

benchmark function to capture the characteristics of a neutral world in static and dynamic settings was used. The following nine strategies were developed to find which strategy(s) has the best performance. The strategies implemented are briefly listed here:

- Strategy 1 (S1) - Hill Climbing Algorithm.
- Strategy 2 (S2) - Equal probability of Reputation for selecting a neighbour.
- Strategy 3 (S3) - Biased probability of Reputation for selecting a neighbour.
- Strategy 4 (S4) - S3 + Situational Knowledge of Cultural Algorithm.
- Strategy 5 (S5) - S4 + Mutation.
- Strategy 6 (S6) - Neighbours and Domain Knowledge of Cultural Algorithm.
- Strategy 7 (S7) - Situational, Domain, Historical, and Topographical knowledge of Cultural Algorithm along with surrounding heights and the neighbours.
- Strategy 8 (S8) - Topographical Knowledge of Cultural Algorithm.
- Strategy 9 (S9) - Modified S8.

The question in this research is how knowledge types play a role in Cultural Algorithms efficiency.

1.5 Thesis Organization

In chapter 2 literature review is presented on Multi Agent Systems, Social Complex Systems, Social Networks, Hill Climbing Algorithm and Evolutionary algorithms which are Ant Colony Optimization, Genetic Algorithms, Particle Swarm Optimization and Cultural Algorithm. In chapter 3 detail descriptions are given in regards to the developed nine strategies followed by chapter 4 where the test results of the nine strategies are presented. Chapter 5 describes the results and attempts to validate the proposed hypothesis against the test results. In the last chapter, chapter 6, the conclusion and future work are explained.

Chapter 2: Literature Review/ Survey

This chapter includes a short survey on Multi Agent Systems (MAS), complex social systems, social networks, Evolutionary Algorithms such as Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) , Cultural Algorithms (CA) and Genetic Algorithms (GA).

2.1 Multi Agent Systems (MAS)

Recently there has been a growing interest in agent based systems technology in artificial intelligence research because it has been hailed as a new paradigm for conceptualizing, designing and implementing software systems [Sycara, 1998]. Agents are active, persistent (software) components that perceive, act, reason and communicate in environments that are distributed and open to solve many complex problems. There are many agent based systems which consists of only a single agent. Due to the increased technological complexity, the need for complex applications have risen that require systems consisting of multiple agents who can communicate in a peer to peer fashion.

A Multi Agent System consists of many autonomous agents such as software programs or robots who interact with each other to perform a set of tasks or to achieve a set of goals.

In theory, MASs are usually characterized in terms of internal behaviours and external interactions between agents [Poslad, 2007]. Some of the characteristics of an agent's internal behaviour are the type of cognition used and the performance measure they utilize when choosing how to behave in model based, reactive, goal based and utility based environments. Some of the characteristics of an agents external behaviour are how the agents interact with each other to share information and to perform tasks. MASs main internal and external behaviours characterize them as a unique type of distributed computation system which has to be supported by a generic distributed computer infrastructure or a set of middleware services. Software agents should be able to

communicate with each other to find out the tasks other agents are able to perform. The middleware services are provided by an agent's Information Communication Technology (ICT) environment, which is often called the MAS platform, where the agents are embedded. Some examples of middleware services are, data storage, message transport, and service discovery. Replacing these services with agent equivalents may cause some disadvantages such as overhead in message invocation and less robust service designs. Also agents not only should be able to communicate with other agents but also should be able to invoke non agent software services. Sometimes agents not only interact locally within their own platform but also interact with remote agents in other platforms. In the future there would be many heterogeneous MASs.

MASs are used to develop complex, large or unpredictable systems as they are mainly developed in a modularized manner. The modular components (agents) will use the most appropriate paradigm to solve a particular problem. When interdependent problems arise, the agents would communicate and coordinate with one another to ensure that interdependencies are handled appropriately.

Most of the distributed computation problems involve open and dynamic environments. In an open system the structure of the system itself is capable of dynamically changing [Katia Sycara, 1998]. In such a system an agent alone is not able to solve the entire problem due to certain constraints, incompetency in performing certain tasks, inaccessibility to critical data, etc. Therefore open systems require MAS.

2.1.1 Applications of Multi-Agent Research

MAS applications cover a variety of domains including: Aircraft maintenance, Electronic book buying coalitions, Military de- mining, Wireless collaboration and communications, Military logistics planning, Supply-chain management, Joint mission planning and Financial portfolio management.

2.2 Social Complex Systems

2.2.1 Complex Systems

A "complex system is a system for which it is difficult, if not impossible to restrict its description to a limited number of parameters or characterising variables without losing its essential global functional properties" [Pavard].

A complex system consists of parts that interact in a non-linear fashion within the environment and their behaviours are non-predictable, emergence also is an essential property. It is important to differentiate between a complicated system such as a computer or a plane and a complex system such as economic systems or ecological systems. Complicated systems contain many functionally distinct parts which are predictable . But in complex systems the parts interact non-linearly within their environment, and their components have properties of self organization which make them non-predictable. In addition, complex systems are completely irreducible. This means that it is impossible to derive a model from this system without losing all its relevant properties.

Recently there has been a growing interest in applying complex system approaches to social systems [Eve et al 1997, McKelvey 1997, McKelvey 1999, Goldspink 1999 and Marion 1999]. Many social systems are modelled using simple agents [Epstein and Axtell 1996]. Therefore Social Complex Systems consists of many individuals (autonomous agents) who interact non-linearly within the environment and their behaviour is unpredictable .

Some of the characteristics of Social Complex Systems are:

- Non-determinism and non-tractability [Pavard] –

  It is impossible to precisely anticipate the behaviour of these systems even if we completely know the function of its constituents. It is fundamentally non-deterministic.

- Agent based -

  Systems comprise of many agents.

- Dynamic -

  Some parts of the system may change, as the agents adapt to their environment. The behaviour of the agents change since they are unpredictable and they learn from their experiences. The dynamics of these systems are usually non-linear sometimes even chaotic. Also the systems are rarely stabilized.

- Organization -

  Most of the time the agents are categorized into groups. These groups are mostly structured and will influence how the system evolves over time.

- Distributed nature of information and representation [Pavard]-

  A complex system possesses properties comparable to distributed systems (in the connectionist sense), example: some of its functions cannot be precisely localised. In addition, the relationships that exist within the elements of a complex system are short-range, non-linear and contain positive and negative feedback loops.

- Emergence and self-organization [Pavard] -

  This is the process of deriving some new and coherent structures, patterns and properties of the system due to non-linear and distributed interactions between agents of the system. They are observed at a macro level even though they are generated in the micro level.

2.3 Social Networks

A social network is a group of people who are connected by a set of relationships, such as

friendship or co-working. By analysing the social networks, one can find out different categories of relationships that occur between people, organizations and groups etc. People are represented as nodes in the network and relationships among the people are represented as edges. Also the structure of the network can contain many sub structures such as, groups and cliques. By analysing these substructures we can find out the likely behaviour of the network as a whole.

The term social network was first coined by J.A Barnes in 1954. Since then a lot of work has been carried out in social science fields such as, anthropology, sociology and social psychology.

With the rapid growth of Internet many cyber social networks have emerged recur to social network software and social network websites [Zhao, 2006]. Cyber social networks have virtual relationships, but reflect some common law of real world social relationships. The most popular Internet social network is Friendster which is a network of virtual friends.

[Zhao, 2006] have proposed the EigenForwarding(k,t) search approach because searching in social networks was a problem. They applied this approach on a P2P social network by leveraging Maze file sharing system. Maze is a Napster-like P2P system, and enables a function to construct social networks. In order to make the system run in self-consistency, a point-based incentive system was imported to motivate contribution. This consists of two kinds of networks which are the friend network and the download network. These two networks are merged to prove the “small world” property of the combination network. Then they computed the EigenTrust value of peers to find the critical node in the network. By leveraging the “small world” social network and EigenTrust, they proposed their search approach EigenForwarding(k,t). A search query is forwarded to k friends who have the highest EigenTrust values. This has better search performance than flooding and random forwarding, and this also indicates the impacts of k and t parameters to consult real deployment.

Then there is a fairly new phenomenon called Social Networking Services (SNSs) [Ahn, 2007]which enable individuals and tools to communicate with people by providing private space for them. Also they help people to find others who have common interests, to create forums for discussions, to exchange photographs and personal news. The social networks of SNSs most of the time accurately reflect the real life social relationships of people than any other online networks. Also because of their size it gives a good opportunity to study human social networks.

Online SNSs –

Social Networking Services enable users to share personal information such as, hobbies, photographs, birthdays, etc. Most SNSs offer features that will help users to form and maintain online networks with other users. One such feature is called a "friend". A user can invite another user to be one's friend. Then if the invited user accepts the invitation then a friend relationship is established between them.

The following gives a brief description about the three SNSs:

- **Cyworld** –

  This is the oldest and largest online social networking service in South Korea. Cyworld was started in September 2001 and has been growing ever since. This consisted about 12 million registered users in November 2005. Cyworld users can establish, maintain and can dissolve a friend relationship online.

- **MySpace** –

  MySpace was the largest social networking service in the world, with more than 30 million users in November 2006. It was started in July 2003. A new user by default gets a friend relationship with Tom Anderson, the cofounder of MySpace. This offers services such as, writing testimonials, checking upcoming birthdays, shortcuts to friends's front pages. By crawling the Myspace online web site from September to October 2006, the authors obtained 100,000 user information. The crawler will randomly select a starting user site, and will crawl to the user's

friends' pages, their friends' pages so on.

- **Orkut** –

  This was initially started in September 2002, by some Google employees and became an official Google service in January 2004. Till recently one could create an account only if one was invited by an existing user, which is different from Cyworld and MySpace. Ever since it allowed a user to create an account without an invitation it expanded greatly. Today it consists of more than 33 million users. When a user joins Orkut, the user can publish one's profile, join other communities and upload photographs. This also offers friend relationships.

What influence individuals to join particular communities, which communities grow rapidly and how the communities evolve over time? Are some of the questions that are addressed in [Backstrom, 2006]. The probability of an individual joining a community depends on the following:

1. The number of friends one has in the community.
2. Activity level of a community.
3. Friendship level of one's friends in the community (Connectedness).

The probability of an individual joining a community is higher when that individual's friends in the community are connected.

The members of a social network fall into three categories [Kumar, 2006]. They are,

- **Singletons** –

  Singletons are users who are loners. Those who have never made any connections with others and are very inactive in the network.

- **Giant component** –

  Giant component consists of a large group of members who have lots of connections with others in the network. These members are highly active and they

are either indirectly or directly connected to the large potion of the entire network.

- **Middle region** –
  Middle region consists of isolated communities who have created small groups and communicate with each other.

Then there are sub groups and sub structures in social networks [Jamali, 2006]. Members who have similar interests, or who are in the same age, race, ethnicity, gender, religion may form cliques or sub groups to share information among each other.

Even the web can be considered as a social network [Jamali, 2006]. These Social networks are formed by hyper- linking web pages to other web pages. Creating communities is one of the important activities in a web. Bibliographic metrics and Bipartite Cores are two approaches which are used to identify communities in a link topology.

There are some social networks, such as LinkedIn, which has incorporated trust in the network connections. Then Orkut allows users to rate trustworthiness of one another. Friend Of A Friend project also has developed a trust module so that the users can rate the trustworthiness [Golbeck, 2006]. When trust is explicitly rated on a numerical scale, then that data can be used to produce information about how much two individuals who do not have a direct connection trust each other. The specific problem that the authors have looked at is how to use the data in the network to accurately tell to one user (source) how much to trust another person (sink). The authors have taken advantage of the explicit trust ratings to find out the trust that may exist between two people who are not directly connected. Trust has three main properties. They are asymmetry, transitivity and personalization. To generate a trust network, the edges have to be augmented with values representing the trust relationship between individuals. The process of adding trust is explained in detail in the paper [Golbeck, 2006].Detail description about the algorithms for inferring trust relationships between individuals who are not directly connected to

each other are given below. There are two algorithms. They are:

- **Rounding Algorithm –**
  In this algorithm the users will assign {0,1} ratings to all the neighbours, and in every step of the algorithm a node will return {0,1} values. Then the source polls the neighbours who have got positive ratings (1) and the others (0) are ignored. Each neighbour will return their ratings for the sink. The source will then find the average of these ratings and will round the final value. This rounded value is the reputation rating from source to sink.

- **Nonrounding algorithm –**
  In this algorithm the nodes do not round the values before they return the values. At the end of the algorithm the original source will round the average of the values returned by the neighbours. Therefore the final inferred value will be 0 or 1.

After analysing the above mentioned algorithms the authors have come up with the following results.

- In the rounding algorithm the accuracy at each step increased.
- Rounding algorithm will out perform the nonrounding algorithm.
- Algorithms will be more effective on larger networks with higher average degrees.

Nowadays it is almost impossible to do one's work without email. But when a lot of spam and junk mails are received it is a very tedious task to find the important emails. Therefore the authors have introduced the TrustMail prototype to filter out the unwanted messages.

TrustMail prototype –

This is an email client that adds trust ratings to each message in the folder. Therefore a user can read the trust rating and can sort the messages accordingly. The users highly benefit from this because users do not have to open unwanted junk mail to find whether they are from trusted parties or not. This program extracts the sender and uses trust algorithms to find out whether the person can be trusted or not.

According to [Goecks, 2004] current user applications and end user applications in social networks provide very limited support such as, current applications mostly do not allow the users to manage and to be aware how their information is transferred to the other users. Therefore lots of users have privacy concerns because they do not know who has access to their information and information dissemination was a hassle. This brings to the fact that there is a need for an infrastructure that enables users to manage and control information sharing. Therefore the authors built Saori to solve this problem. Saori is a computational infrastructure that enables users and applications to manage and control information dissemination within social networks. All information dissemination occurs along ties between individuals who are connected to the social network. Users can leverage their personal networks and the extended networks to share information. Saori maintains an information database that can be shared. It presumes that the information is in the form of a pointer or a url and each instance of information is owned. Saori has two policies. They are Level of Detail (LoD) and Word of Mouth (WoM). To enforce these policies Saori must be able to access users' social networks and attributes. Also maintains a database of this information, by mining users' email messages. Saori obtains user attributes and personal social networks and stores these in the database.

Friendster, [Boyd, 2004] is an online dating site that utilizes social networks to encourage friend of friend connections. Recently many social networking sites have emerged for various reasons, such as dating, job hunting, to find friends, to get recommendations and listings etc. Compared to many other sites, Friendster is a very good site to study the value and implications of popularity, diverse usage and press coverage on the HCI community.

The author has been an active participant with Friendster users and also an active observer with the social networking software creators.

The following are some social networks

- **Yahoo! – Flickr – [Kumar, 2006]**

  Flickr is a popular and an active network. It was launched in Feb 2004. This allows the users to Upload photographs and share them with their friends. A Flickr user can invite a friend to join the network or can add an existing user as a friend. According to Jan 2006 data the Flickr time graph consisted about one million nodes and eight million directed edges.

- **Yahoo! 360 – [Kumar, 2006]**

  Yahoo! 360 is a part of the Yahoo! User network. This is a social networking website. This is mainly used to create and to share albums, photographs among friends. Users can add contacts and invite others to join the network. According to the Jan 2006 time graph Yahoo! 360 had about five million nodes and about seven million directed edges.

2.4 Hill Climbing Algorithm

Hill Climbing is a mathematical optimization technique which falls into the family of local search [Wikipedia, 2006]. Hill Climbing algorithm begins with an initial solution which is usually chosen at random. The string is then mutated, and if the mutation gives a higher fitness for the new solution than the previous solution, then that solution will be used at the next iteration. This process is repeated till a higher fitness is not found [Marczyk, 2004]. Usually the current solution is close to the optimal solution, but cannot guarantee that it will always be close to the optimal solution. This algorithm is widely used in Artificial Intelligence for reaching a goal state from a starting node.

Example [Marczyk, 2004]:

There is a three-dimensional contour landscape. A given set of coordinates on that

landscape represents one particular solution. Those solutions that are better are higher in altitude, forming hills and peaks; those that are worse are lower in altitude, forming valleys. A "hill-climber" is then an algorithm that starts out at a given point on the landscape and moves inexorably uphill.) Hill-climbing is what is known as a *greedy algorithm*, meaning it always makes the best choice available at each step in the hope that the overall best result can be achieved this way.

Pseudo code of the hill climbing algorithm is given in figure 1.

```
1 Each agent will check the surrounding height
2 If surrounding height > current height then
3 Move agent to the surrounding height.
4 Else
5  If surrounding height = current height then
6        Move agent randomly by one step within the allowed  x, y cones space range.
7  Else
8        Agent remains at the current position
9  Repeat till surrounding height <= current height.
```

Figure 1: Pseudo code of Hill Climbing Algorithm

2.5 Evolutionary Algorithms

Problem solving approaches that use computational models which are based on the principles of natural evolution are called evolutionary algorithms (EA) [Saleem, 2001]. In evolutionary algorithms an individual would have to go through the process of selection, recombination and mutation. Then the individuals who have the high fitness values would most likely be selected to form the next generation. Some of the popular evolutionary algorithms are Ant Colony Optimization (ACO), Genetic Algorithms (GA), Particle Swarm Optimization (PSO) and Cultural Algorithms (CA).

2.5.1 Ant Colony

Ant Colony Optimization (ACO) takes inspiration from the foraging behaviour of ants [Dorigo 2006]. Ants deposit a chemical substance called pheromone on the path when walking to and from a food source. Other ants sense the presence of pheromone and tend to follow a trail that is rich in pheromone. Thus they are able to find the shortest path from a food source to the nest. Also ants are capable of adapting to changes in the environment. For example being able to find a new shortest path when the old path is no longer feasible due to obstacles.

In the Ant Colony Optimization Algorithms proposed by Dorigo [2006] the ants are defined as computational agents, which iteratively build solutions to an optimization problem. ACO has been applied to optimization problems in areas such as asymmetric and symmetric travelling salesman problem, scheduling, routing and partitioning problems.

In [Peng, 2005] Each ant will move from a state ι to another one ψ corresponding to a more complete solution. Each agent at each step will compute a set of feasible expansions to its current state by using the following probability distribution,

$$\rho^k\ \iota\ \psi = \begin{cases} \dfrac{\alpha.\ \tau\ \iota\ \psi + (1-\alpha).\ \eta\ \iota\ \psi}{\sum (\tau\ \iota\ v) + (1-\alpha).\ \eta\ \iota\ v} & \text{if } \iota\ \psi \in \text{tabuk} \notin \text{tabuk} \\ 0, \text{otherwise} \end{cases}$$

Will move to the new position only if it is better than the previous step. $tabu_k$ represents a set of feasible moves for an agent k. Parameter α defines the relative importance of the trail. Then after each iteration t of the algorithm, the trails are updated by the following formula,

$$\tau\ \iota\ \psi(t) = \rho\ \tau\ \iota\ \psi(t-1) + \Delta\ \tau\ \iota\ \psi.$$

ρ is a user defined co-efficient and $\Delta \tau \iota \psi$ represents the sum of the contributions of all agents that used the move $\iota \psi$ to find the next position.

The pseudo code from [Maniezzo, 2000] describes how the basic Ant Colony optimization works.

1. (initialization)
   initialize $\tau \iota \psi, \forall \iota, \psi$
2. (construction)
   for each ant k do
   repeat
   compute $\eta \iota \psi, \forall \iota, \psi$
   choose in probability the state to move into
   append the chosen move to the kth ant 's set
   $tabu_k$
   until ant k has completed its solution
   [apply a local optimization procedure]
   end do
3. (Trail update)
   For each ant move (ι, ψ) do
   compute $\Delta \tau \iota \psi$ and update the trail values
4. (Terminating condition)
   If not (end condition) and go to step 2.

Figure 2: Pseudo code of the basic Ant Colony Optimization [Maniezzo, 2000]

2.5.2 Genetic Algorithms (GA)

Genetic Algorithm (GA) was developed by J. H. Holland in 1975 [Holland 1975]. In GA initially a set of solutions or a population would be randomly generated. Then at each iteration by using a fitness function which is problem domain specific would select the

fittest solutions. Most of these functions are stochastic so that only a small proportion of less fit solutions would be selected. This is done to keep the diversity of the population large and to avoid premature convergence on poor solutions. Then would test the selected solutions to find whether the optimal solution is found. If not found then would apply recombination and mutation to breed the next generation (new solutions). Generally the average fitness would have increased in this generation since only the best solutions (parents) are selected for recombination and mutation. This process will be repeated till the optimal solution is found or till a fixed number of generations have been reached or successively better solutions have not been found.

Figure 3 gives the basic algorithmic steps presented by Goldberg [Goldberg 1989].

```
1 t = 0;
2 initialize P(t)
3 evaluate structures in P(t)
4 repeat
5       t = t + 1
6       select_reproduction C(t) from P(t-1)
7       recombine and mutate structures in C(t) forming C'(t);
8       evaluate_structures in C'(t)
9       select_replace P(t) from C'(t) and P(t+1);
10Until (termination condition satisfied)
```

Figure 3: Algorithmic steps for GA [Goldberg 1989]

Initially a population of individuals would be generated (usually randomly). Then the individuals' structures would be evaluated. A probability that is proportional to its fitness would be assigned to each individual. A selection buffer C(t) is created which will contain the better individuals from P(t – 1) "select_reproduction" because they have a higher probability of creating better offspring. Next mutation and crossover would be applied to individuals in buffer C(t) and the new generation would be stored in the buffer C'(t). Then the structural fitness of C'(t) would be evaluated and P(t) would be replaced with the

individuals from C'(t) and P(t-1). This process would be repeated till the termination condition is satisfied.

Cannot guarantee that GA s are able to produce optimal solutions because of its evolutionary nature. For example:

Requirement is to find the highest peak of a given landscape. The algorithm will prefer solutions that lie on a peak. The individuals will generate solutions and at each iteration the best solutions will be kept and mutated expecting better solutions and the rest will be eliminated. Once the individuals have reached either the same peak or different peaks the solution with the highest value will be selected assuming that is the highest peak. But there can be another higher peak which has not been found by any of the individuals since when they reach a peak, the surrounding area is not higher so they will stop exploring the landscape. Therefore the solution that was thought as the optimal is actually only a sub optimal solution.

2.5.3 Particle Swarm

When compared with genetic search, Particle Swarm Optimization (PSO) is a relatively recent optimization technique of the swarm intelligence paradigm [Windisch, 2007]. PSO is a population based optimization technique [Eberhart, 1995], inspired by the behaviour of schools of fish, herds of animals or flocks of birds. Particle Swarm Optimization is somewhat similar to genetic algorithms because both of them are population based. In PSO the system is initialized with a population of random solutions called particles. These particles move through the problem space in search of the global minima or maxima. Each particle keeps track of its past best performance/ fitness and its neighbours (specified proximity radius) best performance to decide on its next move. Also the swarm is aware of the global best achieved by all the particles. At each iteration the particles will update its velocity and position by using the following (a) and (b) equations.

v[] = v[] + c1 * rand() * (pbest[] - present[]) + c2 * rand() * (gbest[] - present[]) (a)

present[] = present[] + v[] (b)

v[] is the agent's velocity, present[] is the current position of the agent and rand() is a random number between 0 and 1. C1 and C2 are the learning factors and usually c1 = c2 = 2. The previous best position of particle i is denoted by pBesti and the previous global best is denoted by gBest.

The pseudo code of the initial version of PSO for real valued variables is given in [Kennedy 2001] as follows,

```
1 For each particle
2       initialize particle
3 End For
4 Do
5       For each particle
6               calculate fitness value
7               if the fitness value is better than the best fitness value (pBest) in history
8               set the current value as the new pBest
9       End
10      Choose the particle with the best fitness value of all the particles as the gBest
11      For each particle
12              calculate particle velocity according to equation (a)
13              update particle position according to equation (b)
14      End
15While maximum iterations or minimum error criteria is not attained.
```

Figure 4: PSO pseudo code [Kennedy 2001]

By simulating individual learning and social cultural transmission PSO attains both simplicity and efficiency (speed of convergence). Some of the advantages of PSO are, it has performed well on a variety of benchmark problems, such as, Schaffer f6 function [Peng, 2005] and global minimum. Also in a wide range of applications such as, minimizing the weight of a tension spring (engineering optimization problem) and neural network optimization. But it does not accurately reflect the accurate human belief system

and performance is problem dependent.

2.5.4 Cultural Algorithm Framework

The Cultural Algorithm (CA) is a class of computational models derived by observing the cultural evolution process in nature [Reynolds 1978, 1994]. The Cultural Algorithm consists of three major components which are population space, belief space and a communication protocol that describes how knowledge is exchanged between population space and belief space [Reynolds, 2005]. The knowledge that is generated in the population space is selectively passed to the belief space which is a global knowledge repository which will influence the changes made by the next generation in the population space. Each individual in the population space has access to the knowledge that is in the belief space, therefore as these individuals evolve to achieve a goal the knowledge is modified and passed to the belief space. Cultural algorithms are mainly designed to model the society including humans. But it is equivalently applicable to non social search problems such as, Constraint Satisfaction or optimization problems. Following diagram shows the basic CA framework.

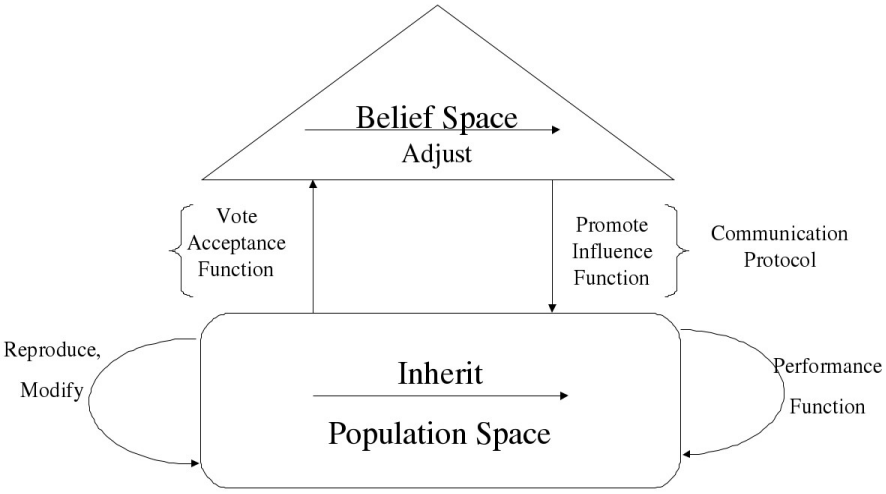


Figure 5: Cultural Algorithm Framework [Reynolds 2002]

The population space consists of autonomous entities or agents of the specific problem. These agents have knowledge or a set of strategies or beliefs. The belief space contains

some of the knowledge that is generated in the population space. In Cultural Algorithms evolution happens both in the population space and belief space through communication. As the population in the population space evolves the belief space will be updated accordingly by the acceptance function and the knowledge in the belief space will influence the agents in the population space. The Reproduction function will generate the new generation and the modify function causes mutation or changes to the new generation. The performance function will identify the optimal state of the system.

The CA pseudo code presented by Reynolds [Reynolds 2002] is given in the figure 6.

```
1Begin
2       t = 0;
3       Initialize Population POP(t);
4       Initialize Belief Space BLF(t);
5       repeat
6               Evaluate Population POP(t);
7               Adjust (BLF(t), Accept(POP(t)));
8               Adjust (BLF (t));
9               Variation(POP (t) from POP (t-1));
10      until termination condition achieved
11End
```

Figure 6: Cultural Algorithm pseudo code from [Reynolds 2002]

The Cultural Algorithm is a population based algorithm just like Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO), but unlike ACO and PSO, Cultural Algorithm uses five basic knowledge models in the problem solving process. They are,

- Situational Knowledge: Situational knowledge was proposed by [Chung, 1997] for real valued function optimization problem solving in static environments. This contains knowledge of exemplars.
- Normative Knowledge: Defines a set of standards within which maximum fitness is expected.
- Topographic Knowledge: Provides a spatial or array framework in which

environmental patterns can be identified.

- Domain Knowledge: This was introduced into CA by Reynolds and Saleem in [Reynolds, 2005] in order to solve dynamic optimization problems. This is generated from the problem domain to predict trends in the resource landscape.
- Historic Knowledge: This was also introduced into CA by Reynolds and Saleem in [Reynolds, 2005] in order to find about global dynamics and to be able to backtrack when necessary. While domain knowledge provides information about local changes in terms of geometrical or gradient considerations, historic knowledge provides a more global perspective of the changes.

A hybrid algorithm with GA and CA is presented in [Xue, 2007] . In [Coelho 2006] PSO was used within the CA framework. [Peng 2003] and [Peng 2004] has carried out a thorough investigation on these types of knowledge and conclude that by amalgamating these knowledge sets in the right combination the CA might be able to converge faster. Also these papers mention how each knowledge category can affect the others to evolve.

Chapter 3: Proposed Algorithms

We have incrementally developed a number of strategies of increasing complexity that can be applied on an application neutral system which contains static/dynamic environments which are inspired from a standard benchmark function the Cones World Problem Generator.

3.1 Cones World Problem Generator [Morrison 1999]

This was originally developed by De Jong and Morrison [Morrison 1999]. They called it DF1. It is a standard and a rigorous test function generator that is used to study the performance of evolutionary algorithms in changing environments [Peng 2005]. It is capable of providing instances of a wide variety of dynamic landscapes.

The cones world generates a multi dimensional problem landscape in which resource cones of different heights and slopes would be scattered. This would be carried out in two steps.

Step1:

Specify a baseline static landscape of the desired complexity. The base landscape will be generated according to the following formula,

$$f(< x_1, x_2, \ldots, x_n >) = \max_{j=1,k}(H_j - R_j * \sqrt{\sum_{i=1}^{n}(x_i - C_{j,i})^2})$$

Figure 7: Formula for the base landscape [Peng, 2005]

Where,

K : The number of cones

n: dimensionality

H_j: Height of cone j

R_j: Slope of cone j

$C_{j,i}$: coordinate of cone j in dimension i

The values for each cone (H_j, R_j and C_{ji}) are randomly assigned based on user specified ranges.

$H_j \in$ (Hbase, Hbase + Hrange)

$R_j \in$ (Rbase, Rbase + Rrange) and

$C_{ji} \in (-1, 1)$

Each of these independently specified cones are blended together using the max function. If two cones overlap then the height at a point is the height of the cone that has the highest value at that point. [Peng, 2005]

For this thesis a three dimensional landscape was used in which K = 5, Hbase = 3, Hrange = 3, Rbase = 10 and Rrange = 2. Example of a cones world environment is given in figure 8.

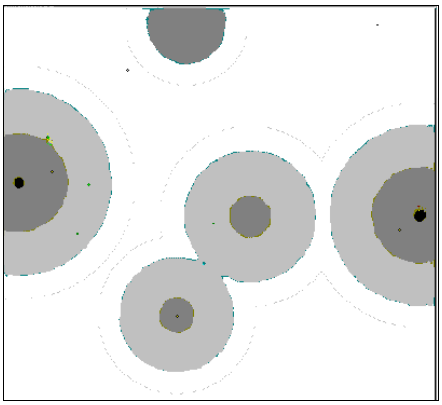


Figure 8: Example of a three dimensional landscape generated by the df1 function.

x □ (-1, 1) , y □ (-1, 1), with n = 2, H □ (3, 6) and R □ (10, 12)

Step2:

Specify the dynamics. Dimension $C_{j,i}$, height H_j and slope R_j of every cone j can be changed. These can be changed by the following logistics function,

$Y_i = A * Y_i\text{-}1 * (1\text{-}Y_i\text{-}1)$

A is a constant and Y_i is the value at iteration i

The generator will randomly generate cones each time it is called. Cones World Problem Generator was selected because it is able to generate test functions over a wide range of surface complexities and problem dynamics. Also it is able to evaluate the system in a more flexible and a systematic way.

3.2 Static Environment

In the static environment, once the cones are randomly generated they remain in that position. The cones do not move.

3.3 Dynamic Environments

Initially just like the static environment cones will be deployed on the landscape. Then after hundred and fifty steps the cones will be moved to different locations. Then after the next hundred and fifty steps (three hundred steps) not only the cones will be moved but also the heights of all the cones will be changed. This will be repeated till number of steps are six hundred. There are four different dynamic environments. When the environment changes the agents will start exploring from where they stopped.

The user is able to select or enter the following for the nine strategies.

- Particular strategy.
- Number of cones.
- Number of agents.
- Static/ dynamic environment.
- Number of pixels for a step (step size).
- The mutation probability for the strategy 5.

3.4 Strategy 1 (S1)- Hill Climbing Algorithm.

Implemented the Hill Climbing Algorithm. Once an agent is randomly placed on the landscape, the agent's current and surrounding pixel heights are passed to the algorithm as shown in figure 9. Note that the agent is at H_{11}.

The three basic rules for the heuristic algorithm are as follows:

1) Move the agent to the surrounding pixel with the highest height, higher than its current height.

2) If there are any surrounding pixels with height equal to its current height H_{11}, then the agent is randomly moved by one pixel within the allowed x and y cones space range.

3) Otherwise the agent remains at the current position.

These rules are repeated all the way to the top of the peak where the agent is located. If the peak is a plateau rather than a point (single pixel), then the agent will randomly wonder along the plateau at the same height in search of a higher pixel.

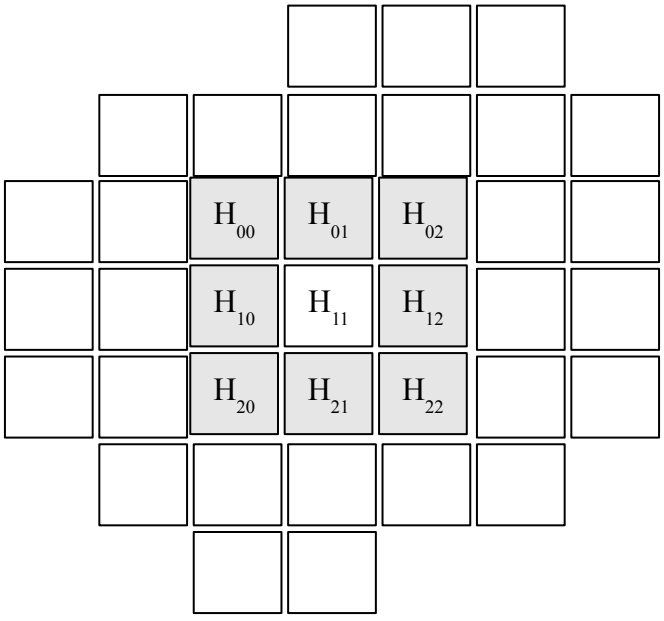


Figure 9: The surrounding heights of an agent.

The pseudo-code for strategy S1 is given in figure 10.

```
1 Each agent will check the surrounding height
2 If surrounding height > current height then
3 Move agent to the surrounding height.
4 Else
5  If surrounding height = current height then
6        Move agent randomly by one step within the allowed  x, y cones space range.
7  Else
8        Agent remains at the current position
9  Repeat till surrounding height <= current height.
```

Figure 10: Pseudo code for strategy S1.

3.5 Strategy 2 (S2) - Equal probability of Reputation for selecting a neighbour.

The agents who are within fifty pixels apart are grouped into networks. Initially all the agents reputation will be zero. Then each agent by using equal probability (50/50 probability of picking any neighbour) will pick a neighbour and will request for its height. If the height is greater than itself then will move one step towards the neighbour's location and increase the neighbour's reputation by one. Otherwise the agent will use hill climbing algorithm to decide on the next move and decrease the neighbour's reputation by one. If an agent does not have any neighbours then it will use hill climbing algorithm to decide on its next move. The pseudo code is given in figure 11.

```
1 For each agent, check within 50 pixels for another agent
2 If there is an agent within 50 pixels apart then
3        Add that agent to the current agent's network
4 End For
5 For each agent check whether the agent belongs to a network
```

```
6 If the agent belongs to a network then
7       By using equal probability of reputation select a neighbour.
8       Request for the neighbour's height.
9               If neighbour's height > current agent's height then
10                              Move the agent by 1 step towards the neighbour.
11                              Increase neighbour's reputation by 1.
12              Else
13                              Use Hill Climbing Algorithm (S1) to move
14                              Decrease neighbour's reputation by 1
15      Else
16                      Use Hill Climbing Algorithm (S1) to move
17 Repeat till user selects stop or till the number of iterations = number of iterations given
in the user settings menu.
```

Figure 11: Pseudo code for strategy S2.

3.6 Strategy 3 (S3) - Biased probability of Reputation for selecting a neighbour.

The agents who are within fifty pixels apart are grouped into networks. Initially all the agents reputation will be zero. For the first five iterations strategy S2 will be carried out. From the sixth iteration onwards, each agent will get all it's neighbours reputations, add them into an array. Using the reputations in the array a new array (Sa) is created that contains the sum of the reputations at the current and previous indices. Then a random number between 0 and 1 is generated where that number is multiplied by the total of all the neighbours reputations (Bp). The agent associated with the index in array (Sa) who's value is greater and closest to the result (Bp) as illustrated in figure 12 will be the agent's neighbour (in the case of the example in figure 12 it is agent at index 3). Then the agent

will request for its height. If the height is greater than itself then it will move one step towards the neighbour's location and increase the neighbour's reputation by one. Otherwise, will use hill climbing algorithm to decide on the next move and decrease that neighbour's reputation by one. If an agent does not have any neighbours then it will use hill climbing algorithm to decide on its next move.

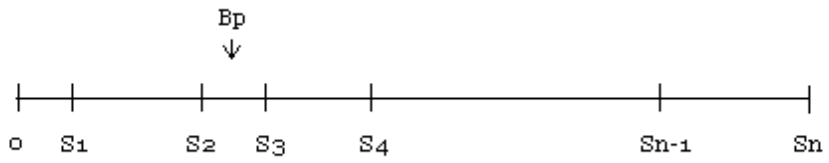


Figure 12: Number line showing how the biased probability is determined

The pseudo code is given in figure 13.

1 For each agent, check within 50 pixels for another agent
2 If there is an agent within 50 pixels apart then
3 Add that agent to the current agent's network
4 End For
5 For Number of iterations 1 – 5 do
6 Strategy S2
7 End For
8 For Number of iterations 6 - till user selects stop or till the number of iterations = number of iterations given in the user settings menu
9 Calculate the sum of reputations of all the neighbours
10 Store the first neighbour's reputation and id in an array
11 Then add the next neighbour's reputation to the previous neighbour's reputation and store that value in the same array.
12 Repeat till all the neighbours reputations are added to the previous neighbour's

```
  reputation.
13BiasedRep = ceiling (Rand * sum of reputations of all the neighbours)
14Check whether BiasedRep is less than or equal to any value in the array.
15Then get the id of the first value that is greater than or equal to it.
16The agent who has that id is this agent's neighbour.
17      Request for the neighbour's height.
18                  If neighbour's height > current agent's height then
19                        Move the agent by 1 step towards the neighbour.
20                        Increase neighbour's reputation by 1.
21                  Else
22                        Use Hill Climbing Algorithm (S1) to move
23                        Decrease neighbour's reputation by 1
24            Else
25                  Use Hill Climbing Algorithm (S1) to move
26End For
```

Figure 13: Pseudo code for strategy S3

3.7 Strategy 4 (S4) – S3 + Situational Knowledge of Cultural Algorithm.

In strategy 7, for the first ten iterations strategy 3 is carried out. Then at every tenth iteration the top 10% agents based on their reputations will be calculated and their ID, x-y coordinates and height will be stored in the belief space, which forms the global repository. Randomly the agents can either, pick an agent from the belief space or from the neighbourhood. If the agent (A) decides to pick an agent from the belief space, then by using biased probability of the agent reputation will pick an agent (B), followed by checking whether the height of B is greater than A. If it is, agent B's reputation is increased by one and will travel one pixel towards agent B. If not agent B's reputation

will be decreased by one and will select another agent based on biased probability of reputation from the neighbourhood. If the agent (A) decides to pick a neighbour from the neighbourhood the agent will perform strategy S3 as described in the previous section. The pseudo code is given in figure 14.

```
1 Nth Iteration = false (Nth Iteration = 10)
2 For Number of iterations 1 – till the user selects stop or  number of iterations = number
  of iterations given in the user settings menu do
3 If Number of iterations mod 10 = 0 then
4       Nth Iteration = true
5 Else
6       Nth Iteration = false
7 If Nth Iteration = true then
8       Evaluate all the agents reputations
9       Select the top 10% agents based on their reputations and store their id, x,y and z
        coordinates in the belief  space.(Global repository)
10      Randomly pick a neighbour or the belief space.
11              If belief space is selected then
12                      By using Biased probability of reputation select an agent
                        from the belief space.
13                      If selected agent's height > current agent's height then
14                              Move the agent by 1 step towards the neighbour.
15                              Increase selected agent's reputation by 1.
16                      Else
17                              Decrease selected agent's reputation by 1.
18                              By using Biased probability of reputation select an
                                agent   from the network.
19                                      If no neighbours then
20                                              Do Strategy S1
21                                      Else
22                                              Request for the neighbour's height.
```

```
23                                                  If neighbour's height > current agent's
                                                        height  then
24                                                      Move the agent by 1 step
                                                        towards the neighbour.
25                                                      Increase neighbour's
                                                        reputation  by 1.
26                                                  Else
27                                                      Use Hill Climbing Algorithm
                                                        (S1) to move
28                                                      Decrease neighbour's
                                                        reputation by 1
29              Else
30                  By using Biased probability of reputation select an agent from the
                    network.
31                                          If no neighbours then
32                                              Do Strategy S1
33                                          Else
34                                              Request for the neighbour's height.
35                                              If neighbour's height > current agent's
36                                                      height  then
                                                Move the agent by 1 step towards the
37                                                  neighbour.
38                                              Increase neighbour's
                                                reputation by 1.
39                                              Else
40                                                  Use Hill Climbing Algorithm
                                                    (S1) to move
41                                                  Decrease neighbour's
                                                    reputation by 1
42      Else
```

43 If Number of iterations 1 – 10 then

44 Do Strategy S3

45 Else

46 Follow the selected agent till the next Nth Iteration

47 Increase Number of iterations by 1.

Figure 14: Pseudo code for strategy S4

3.8 Strategy 5 (S5) - S4 + Mutation.

The user is given a choice whether to mutate an agent or not. If the user decides to mutate the agents, then the user can enter a mutation probability in the Mutation field of the Parameter settings. If the user decides not to mutate agents then zero should be entered.

Once a mutation probability (ex: 0.05) is entered, every ten iterations a random number between 0 to 100 is generated and if it falls between 0 to 5 (0.05 * 100), the agent will be mutated for the next ten iterations. As soon as the agent is mutated, random x-y coordinates will be generated to which the agent will travel towards. The agent will travel to its destination and while doing so will share and gather information with encountered neighbours along the way. If the destination x-y is reached prior to the expiration of the ten iterations the agent will remain at its destination until the time expires. If the random number is not within 0 and 5, that agent will not be mutated and will use S4 strategy as described in the previous section. The pseudo code is given in figure 15.

1 Nth Iteration = false

2 AmIMutated = false

3 For Number of iterations 1 – till the user selects stop or number of iterations = number of iterations given in the user settings menu do

4 If Number of iterations mod 10 = 0 then

5 Nth Iteration = true

6 Else

```
7       Nth Iteration = false
8 If Nth Iteration = true then
9       If Mutation > 0 then
10                  Get the mutation probability from the user settings menu.
11                  Random number generator will generate a number
12                      If that number*100 < user entered probability * 100 then
13                          Agent is mutated
14                      Else
15                          Agent is not mutated
16                      If agent is mutated then
17                          x,y coordinates will be randomly generated
18                          Agent will move by 1 step towards x,y
19                          AmIMutated is set to true
20                      Else
21                          Do all the steps after “If Nth Iteration = true
22              then” in strategy S4
23              Else
24                  Do all the steps after “If Nth Iteration = true
25              then” in strategy S4
26      Else
27              Follow the selected agent till the next Nth Iteration.
28 Increase number of iterations by 1.
```

Figure 15: Pseudo code for strategy S5

3.9 Strategy 6 (S6) - Neighbours and Domain Knowledge of Cultural Algorithm.

In Strategy 6 to Strategy 9 each agent has its own miniature map of 20 x 20 cells (10 x 10

pixels) denoted as MyMap. This is to reflect a miniature outline of the landscape forming 200 x 200 pixels. Each cell in MyMap is initialized with -1 and does not contain any cones.

At each step all the agents will check within 50 pixels apart for neighbours. If neighbours are found the agent will retrieve their maps and update it's own map. With the updated map the agent will select the closest location none of its neighbours have been (i.e. by determining the closest initialized value of -1) and travel toward it. On its way if the agent comes across a cone it will calculate the peak of the cone without climbing it and also will check for neighbours within 50 pixels and update it's map. At the end of each update of MyMap the agent checks if it's map is complete (i.e. no more -1s). With a complete map, the agent will find the highest height and travel to the location. The pseudo code is given in figure 16.

```
1If  (myMap is not complete) then
2       Get my height
3        If (has reached destination) or (No of steps >= 0) then
4                Store my height in the relevant x and y coordinates in myMap.
5                Get neighbours who are within 50 pixels.
6                If (no of neighbours >= 1) then
7                        Get the neighbours maps.
8                        Update myMap.
9                Out of all the -1's in myMap, the random number generator will
        select   the closest x and y coordinate.
10               Move agent by 1 step towards the selected location.
11               Clear has reached destination flag.
12       If (my height = 0) then
13               Calculate Domain knowledge
14               Move agent by 1 step towards the selected location.
15               Out of all the -1's in myMap, the random number generator will
        select   the closest x and y coordinate.
16               Store my height in the relevant x and y coordinates in myMap.
17      else
```

```
18          Calculate Domain knowledge
19          Move agent by 1 step towards the selected location.
20          Out of all the -1's in myMap, the random number generator will
        select the closest x and y coordinate.
21          Store my height in the relevant x and y coordinates in myMap.
22      Check whether myMap is complete.
23      If (myMap is complete) then
24          myMap complete flag = true.
25          Find the highest height in myMap.
26      Take 1 step towards the location of the highest height.
```

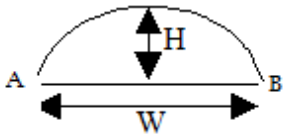
Figure 16: Pseudo code for strategy S6

3.9.1 Domain Knowledge

This strategy involves the agent estimating the peak of the cone without climbing to the top of the cone. First the agent finds an edge of the cone. This point is marked as A. Then the agent travels 20 steps along the edge of the cone to point B and will save each point along the way into an array. With the coordinates of point A and B the radius can be calculated using the formula below.

$$R = \frac{H}{2} + \frac{W^2}{8H}$$

Where:



W is length of the chord defining the length of the arc base.
H is the height measured from at the mid point of the arc base.

$$W = \sqrt{(X_A - X_B)^2 + (Y_A - Y_B)^2}$$

H above is the distance between the middle point in the array and the middle point between A and B.

$$H = \sqrt{(X_{AryMid} - X_{ABmid})^2 + (Y_{AryMid} - Y_{ABmid})^2}$$

$$X_{ABmid} = \frac{(X_A + X_B)}{2} \quad ; \quad Y_{ABmid} = \frac{(Y_A + Y_B)}{2}$$

Once the radius is calculated the agent travels a few steps towards the centre of the cone just enough to determine the slope of the cone. Finally the height of the cone is calculated as shown in the formula below.

$$Cone\ Height = Radius \times slope$$

3.10 Strategy 7 (S7) - Situational, Domain, Historical, and Topographical knowledge of Cultural Algorithm along with surrounding heights and the neighbours.

Each agent will get information from the Cultural Algorithm's knowledge sources such as, Situational knowledge, Topographic knowledge, Historical knowledge, domain knowledge, also from the neighbour's within 50 pixels and the surrounding area. Then the agent will update it's map with this information and will pick the closest location that none of its neighbour's have been nor a location given by the above mentioned knowledge sources. Then as the agent is travelling to the selected location, it will update the Historical knowledge and it's map. If the agent lands on a cone then will calculate the peak of the cone without climbing to the top (as explained in chapter 3.9.1), and store that value in the map and also will update it's map with the information given by the

neighbours. Once an agent has reached the selected destination it will check the above mentioned knowledge sources, neighbours and the local area and will select the closest location that none of its neighbour's have been nor a location given by the above mentioned knowledge sources. Once all the agents have taken a step, Situational knowledge, Topographic knowledge and Historical knowledge sources are updated. Also the agent's map is checked to determine if all the values in the myMap are filled. If it is, then will search for the highest value in myMap, and will update the Situational knowledge with the highest value and will set myMap complete flag to true. Then all the agents will travel to this location. The pseudo code is given in figure 17.

```
1If  (myMap is not complete)  then
2       Get my history list.
3       Get my height.
4       If (has reached destination) or (No of steps >= 0) then
5               Store my height in the relevant x and y coordinates in myMap
6               Get neighbours who are within 50 pixels.
7               If (no of neighbours >= 1) then
8                       Get the neighbours maps.
9                       Update myMap.
10              Check the local area.
11              If (No of steps >= 1) then
12                      update myMap with topographic knowledge.
13              Check Historical knowledge.
14              Update myMap with the surrounding values, situational knowledge and
                historical knowledge.
15              Out of all the -1's in myMap, the random number generator will
                select a particular x and y coordinate.
16              Move agent by 1 step towards the selected location.
17              Store the historical values in the history list
18              Store my height in the relevant x and y coordinates in myMap.
19              Clear has reached destination flag.
20      If (my height = 0) then
```

```
21              Calculate Domain knowledge
22              Out of all the -1's in myMap, the random number generator will
                select a particular x and y coordinate.
23              Move agent by 1 step towards the selected location.
24              Update myMap with the surrounding values, situational knowledge and
                historical knowledge.
25              Store the historical values in the history list
26              Store my height in the relevant x and y coordinates in myMap
27      else
28              Calculate Domain knowledge.
29              Out of all the -1's in myMap, the random number generator will
                select a particular x and y coordinate.
30              Move agent by 1 step towards the selected location.
31              Update myMap with the surrounding values, situational knowledge and
                historical knowledge.
32              Store the historical values in the history list
33              Store my height in the relevant x and y coordinates in myMap
34      Check whether myMap is complete.
35      If (myMap is complete) then
36              myMap complete flag = true.
37              Find the highest height in myMap.
38              Update situational knowledge.
39 else
40      Take 1 step towards the location of the highest height.
```

After all the agents have taken a step the knowledge sources will be updated.

Figure 17: Pseudo code for strategy S7

3.10.1 Situational Knowledge

Situational knowledge consists the highest height at that time step. This is the local best and can be the global best if that is the highest height in the landscape.

3.10.2 Historical Knowledge

At each step, the historical knowledge contains each agent's highest ten heights. Therefore when ever an agent queries the historical knowledge the agent will know the ten highest heights found until that time step. Then the agent will update its map with this information.

3.10.3 Topographic Knowledge

The landscape is divided into 16 cells. The Topographic knowledge consists of the highest height found by an agent in each cell. When ever a value is higher than the current value in that cell, that value will be replaced by the new found higher value.

3.10.4 Domain Knowledge

Domain knowledge is explained in chapter 3.9.1.

3.10.5 Neighbours

The agents who are within 50 pixels apart are neighbours. An agent will check the neighbours maps to find out the locations where the neighbours have been and will update its map. This is performed so that another agent does not have to go to a place where another has been. Then the entire landscape can be explored much faster.

3.10.6 Local area

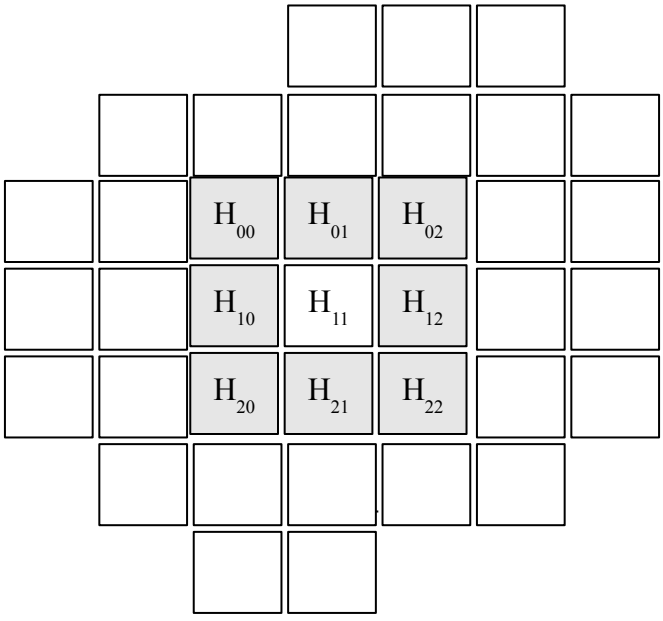


Figure 18: The surrounding heights of an agent.

The agent at H11 will check the local area which are coloured in gray to check whether the next step at any direction is higher than its current position. The agent will update its map with the values in the surrounding area.

3.11 Strategy 8 (S8) - Topographical Knowledge of Cultural Algorithm.

The landscape is divided into sixteen cells. Initially the agents will randomly be scattered on the landscape. There is a master topographical map with sixteen equal cells of 50 by 50 pixels each. If there are less than sixteen agents then each agent will be assigned to the closest cell according to their current location. For example:

If there are two agents that will fall into the same cell then the first agent will be assigned to that cell and the other agent will be assigned to the next closest cell. When ever an agent is assigned to a cell the master map is updated. But if there are more than sixteen agents then the first sixteen will be assigned to the sixteen cells and the others depending on their current locations and according to the master map will be assigned to cells.

Once an agent lands on a cell the agent will start exploring the cell and will check for agents within 50 pixels apart(neighbours) then will update its map with the value(s) given by the neighbours and also with its own explored locations. This way more than 1 agent would not explore any area that was explored by another. Then when it has finished exploring that particular cell the agent will check in the master map for cells that have not been explored by any agent. If there are any unexplored cells then that agent will be assigned to that cell.

If there are no more unexplored cells then the agents who have finished exploring their assigned cells will come to the middle of the landscape.

When all the agents are in the middle of the landscape, they will update their maps. Then will check for the highest height and all the agents will travel to the highest height. An agent's step = 10 pixels in this simulation.

With each agents map being 20 by 20 cells (10 by 10 pixels for each cell) and the topographical map being 4 by 4 cells (50 by 50 pixels for each cell) the worst case (T_{wc}) and best case (T_{bc}) exploration steps can be calculated as follows:

With number of agents (nAgents)>= 16

Steps to explore a topographical cell (S) = 5steps x 5steps = 25 steps

Worst case to assigned cell (wcs) is 28 steps (diagonal length = sqrt(20^2+20^2))

Worst case steps to come to the middle from a corner of the landscape (wcm) ≈ 14 steps

Worst case to the corner of the landscape from the middle (wcmh) ≈ 14 steps

Lowest number of agents in a cell (nApc) = floor(nAgents/16) (i.e. div. into 16 cells)

T_{wc} (nAgents) ≈ ceiling(S/ nApc) + wcs + wcm + wcmh

T_{bc} (nAgents) ≈ ceiling(S/ nApc) + wcm + wcmh

With number of agents (nAgents)< 16

T_{wc} (nAgents) ≈ S*(celling(16/nAgents)) + wcs + wcm + wcmh

T_{bc} (nAgents) ≈ S*(celling(16/nAgents)) + wcm

The pseudo code is given in figure 19.

```
1cntTopo = 0
2 If (cntTopo = 0) then
3       Initialize the Master Topo array
```

```
4       cntTopo = cntTopo + 1
5If (No of steps = 0) then
6       Get my x coordinate
7       Get my y coordinate
8       Assign a cell
9       Store my height in the relevant x and y coordinates in myMap array
10 Check whether the agent has finished exploring
11    Get neighbours within 50 pixels apart
12    If (no of neighbours > 0) then
13       Get the neighbours maps.
14       Update myMap with the values in the neighbours maps.
15 If (No of steps > 0) and (Not finished exploring) then
16       Move agent by 1 step in the cell
17       Store my height in the relevant x and y coordinates in myMap array
18 If (No of steps > 0) and (Finished exploring) then
19       Got a cell = false
20       Check Master Topo array to find whether there are any more unexplored cells
21       If (unexplored cells = true) then
22              Assign a cell
23              Move agent by 1 step in the cell
24              Store my height in the relevant x and y coordinates in myMap array
25              Finished exploring = false
26       Else
```

```
27          Finished exploring = true
28          Reached centre = false
29          Check whether the agent is in the centre of the landscape
30          If (reached centre) and (Not got highest height)
31                  Get neighbours who are within 50 pixels.
32                  Count number of neighbours
33                  If (number of neighbours = number of agents – 1) then
34                          Get the neighbours maps.
35                          Update myMap with the values in the neighbours maps.
36                  Get the highest height
37                  Got highest height = true
38                  Move agent by 1 step towards the highest height
39          Else if (Not reached centre) and (Not got highest height) then
40                  Move agent by 1 step towards the centre
41          Else if (got highest height = true) then
42                  Move agent by 1 step towards the highest height  till the highest
                    height is reached.
```

Figure 19: Pseudo code for strategy S8

3.12 Strategy 9 (S9) - Modified S8.

This strategy is very similar to strategy 8. The landscape is divided into sixteen cells. Initially the agents will randomly be scattered on the landscape. There is a master topographical map with sixteen cells. If there are less than sixteen agents then each agent will be assigned to the closest cell according to their current location. For example:

If there are two agents that will fall into the same cell then the first agent will be assigned to that cell and the other agent will be assigned to the next closest cell. When ever an agent is assigned to a cell the master map is updated. But if there are more than sixteen agents then the first sixteen will be assigned to the sixteen cells and the others depending on their current location and according to the master map will be assigned to cells.

Once an agent lands on a cell the agent will start exploring the cell and will check for agents within 50 pixels apart(neighbours) then will update its map with the value(s) given by the neighbours and also with its own explored locations. This way more than 1 agent would not explore any area that was explored by another. Then when it has finished exploring that particular cell the agent will check the master map for cells that are not been explored by any agent. If there are any unexplored cells then that agent will be assigned to that cell.

If there are no more unexplored cells and all the agents have finished exploring the assigned cells then will go through the maps of all the agents and find the location of the highest height and all the agents will travel to the highest height. The pseudo code is given in figure 20. The worst case (T_{wc}) and best case (T_{bc}) steps are same as S8.

1cntTopo = 0

2 If (cntTopo = 0) then

3 Initialize the Master Topo array

4 cntTopo = cntTopo + 1

5 If (No of steps = 0) then

6 Get my x coordinate

7 Get my y coordinate

8 Assign a cell

9 Store my height in the relevant x and y coordinates in myMap array

10 Check whether the agent has finished exploring

```
11    Get neighbours within 50 pixels apart
12    If (no of neighbours > 0) then
13        Get the neighbours maps.
14        Update myMap with the values in the neighbours maps.
15 If (No of steps > 0) and (Not finished exploring) then
16        Move agent by 1 step in the cell
17        Store my height in the relevant x and y coordinates in myMap array
18 If (No of steps > 0) and (Finished exploring) then
19        Got a cell = false
20        Check Master Topo array to find whether there are any more unexplored cells
21        If (unexplored cells = true) then
22                Assign a cell
23                Move agent by 1 step in the cell
24                Store my height in the relevant x and y coordinates in myMap array
25                Finished exploring = false
26        Else if (Not got highest height) then
27                Finished exploring = true
28                Store my height in the relevant x and y coordinates in myMap array
29                If (Finished exploring = true) then
30                        Get the neighbours maps.
31                        Find the location of the highest height
32                        Got highest height = true
33        If (Got highest height = true) then
```

34 Move agent by 1 step towards the highest height till the highest height is reached.

Figure 20: Pseudo code for strategy S9

The above mentioned strategies are implemented in Repast.

3.13 Repast: The environment

Repast (**Re**cursive **P**orous **A**gent **S**imulation **T**oolkit) is a simulation tool that consists of a set of API to build agent based simulations easily and rapidly. This is an object oriented tool and has many platforms such as, Java, Microsoft .Net and Python. It is an open source software that was developed by Sallach, Collier, Howe, North and others [Collier 2003] at the University of Chicago and later maintained by Argonne National Laboratory (ANL). Now it is maintained by non profit Organization ***R**epast **O**rganization for **A**rchitecture and **D**evelopment (ROAD)*. Even though it is mainly intended for developing agent based simulations it is also suitable for network analysis and visual descriptions of algorithms, concepts and programs. The Repast version that is built on the Java platform is Repast J 3.0. It is an integrated simulation development framework that provides some functionalities to develop and run simulations. These functionalities are,

- Graphical display features to demonstrate the simulation.
- To create charts and graphs to capture the outcome of the simulation visually.
- User settings menu which allows the user to initialize, run, stop or pause the simulation when its running.

Since Repast J is an open source software, the users can modify its behaviour any way they want. Some of the features of Repast are explained below.

3.13.1 Repast Setup:

The main function routine is given below in figure 21. The first routine that starts the environment is loadModel in the instantiated ***init*** object of class SimInit() that loads the object ***model*** of class ConesModel.

```
public static void main(String[] args) {
```

```
SimInit init = new SimInit();
ConesModel model = new ConesModel();
init.loadModel(model, "", false);
}
```

Figure 21: The main function routine

The basic Repast routines are listed below.

setup() - Implements uchicago.src.sim.engine.SimModel.setup. The cone space, agent space, scheduling and display is handled by this routine.

begin() - Implements uchicago.src.sim.engine.SimModel.begin. This routine calls buildModel(), buildSchedule(), buildDisplay() and displaySurf.display().

buildModel() - Cones model, agents and graph initialization and display are initiated by this function.

buildSchedule() - One of the most important steps in creating a dynamic environment is handled by this routine, here schedules for dynamic cones, moving agents and updating agent statistics (Graph and Histogram) are handled. Each item, that is Cones (if dynamic), agents and statistics all have their individual schedule that can be associated to the simulating computer's “tick” rate. In the schedule assigned to the agents, their behavioural algorithm can be called that determines their next movement.

buildDisplay() - The contour colour arrangement for each cone's gradient is serviced by this call. Each cone's height is limited from 1 to 100 and a colour gradient (RGB) has been assigned to each magnitude of height as shown in the table 1 below.

Table 1: Colour gradient assigned to each cone's height

Cone Height (some arbitrary units)	Colour gradient (RGB)
1 to 33	Aqua (0, 255, 255) to Green (0, 255, 0)
33 to 67	Green (0, 255, 0) to Orange(255, 255, 0)
67 to 100	Orange(255, 255, 0) to Red (255, 0, 0)
0	(0,0,0) white (no cones)

displaySurf.display()- Finally, this routine displays the cones and agents on the surface of the plot.

3.13.2 Repast User Interface

Repast Tool bar

Repast tool bar is given in figure 22.

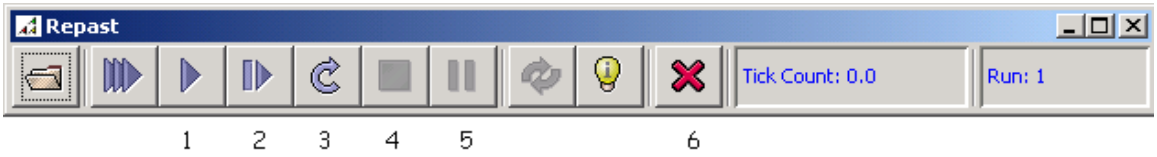


Figure 22: Repast tool bar

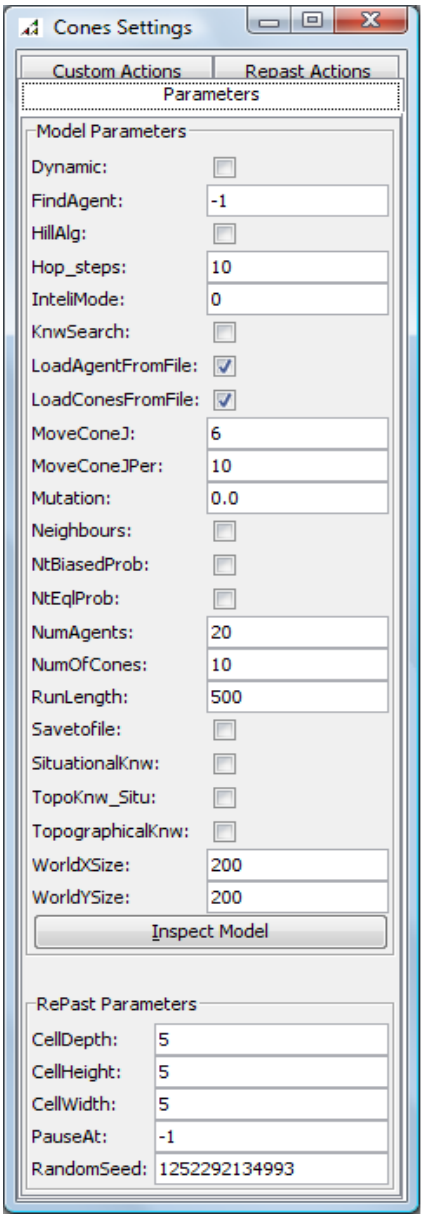
The repast tool bar consists of Play (1), Step (2), Initialize (3), stop (4), pause (5) and exit/close (6). Pressing Initialize as self described will initialize the repast simulation plus setup the cones and provide initial locations for the agents. Initialization is not required to be pressed prior to start of the simulation. Either pressing Play or Step will initialize the simulation prior to continuous or stepping through the simulation. Once play has been pressed the simulation can be stopped (4) or paused (5).

Cones Settings: The cones Settings menu provides the user selectable options to run the simulation. The cones settings used in the thesis are given in the figure 23.

Dynamic: The check box allows the user to select a moving landscape (moving cones) and unchecked is for a static landscape.

Find Agent : feature allows the user to enter the agents ID number (numeric) to identify

the location of the agent by changing the agent's colour.



Hop steps: The number of pixels to take a step.

HillAlg: Strategy 1 in Chapter 3.

KnwSearch: Strategy 7 in Chapter 3.

Mutation: The mutation probability for Strategy 5 will be entered here.

Neighbours: Strategy 6 in Chapter 3.

NtBiasedProb: Strategy 3 in Chapter 3.

NtEqlProb: Strategy 2 in Chapter 3.

RunLength: The number of steps that the simulation will run.

SituationalKnw: Strategy 4 in Chapter 3.

TopoKnw_Situ: *Strategy 9 in Chapter 3.*

TopographicalKnw: Strategy 8 in Chapter 3.

WorldXSize, **WorldYSize** - defines the size of the canvas.

Figure 23: Cones Settings

3.13.3 *Cones Model*

Cones model is the main class that initializes repast, ConesSpace, ConesAgent and statistics. In addition, this class also schedules the ConesSpace, ConesAgent and statistics of the agents (Histogram and Graph) for simulation.

3.13.4 Cones Space

Cones Space class generates the cones based on the paper [A Test Problem Generator for Non-Stationary Environments] The cones are generated in a Cartesian coordinates space where the x range is [-1, 1) and the y range is [-1, 1) based on the df1 function and then translated to a pixel range of width [0, 200) and height [0, 200). A sample of the cones space layout and colour gradient is shown below in figure 24.

3.13.5 Cones Agents

Agents will behave according to the selected strategy.

3.13.6 Statistics

The agent statistics are displayed in a graph and histogram form. Also beneath the histogram are time steps traversed by the agents to reach the peaks of the cones. To determine the time (number of steps) was divided into two counters. The reason was, because in the cones space and wider bases may have very smallslopes resulting in the agent hovering at a given height above ground level (pixel height = 0).

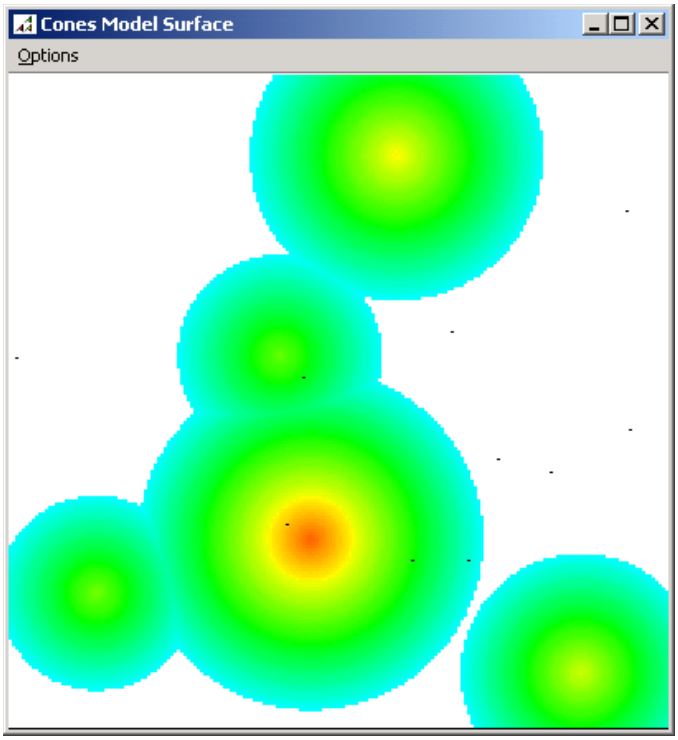


Figure 24: Landscape with 5 cones

The figure 25 below illustrates the scenarios:

(1) peak (2) peak (3) peak

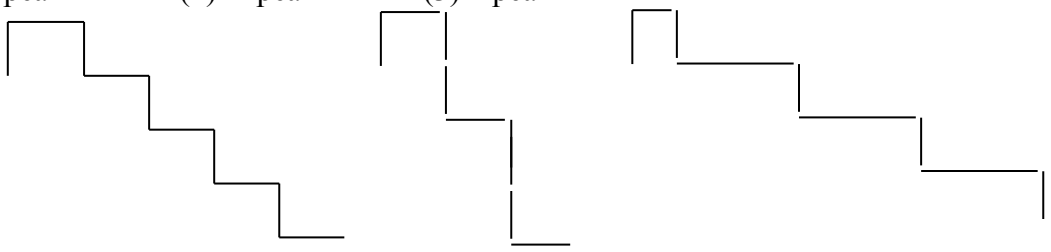


Figure 25: Scenarios of peaks

Scenario 1 has a slope of 1 vertical height unit per 1 horizontal pixel.

Scenario 2 has a slope of 2 or more vertical height units per 1 horizontal pixel.

Scenario 3 has a slope of 1 vertical height unit per 2 or more horizontal pixels.

With scenario 1 and 2 for every movement of the agent on the slope of the cone results in finding a height pixel until the peak is reached. However with scenario 3 after the agent climbs one vertical height unit the agent has to traverse horizontally to reach the next highest pixel at random. Therefore to determine the time (number of steps) to reach the peak two counters were used. The main counter keeps track of steps with an increasing height of the agent and this counter stops when the agent moves horizontally. The secondary counter keeps track of horizontal movement and adds its value to the main counter on detection of height followed by the secondary counter being reset to zero. Finally the time to reach the peak is the value in the main counter. The pseudo code is given in figure 26.

```
1       While(simulation running)
2       if (new height found)
3               main_ctr = main_ctr + 1 + secondary_ctr;
4               secondary_ctr = 0;
5       else
6               secondary_ctr = secondary_ctr +1,
7       loop
```

Figure 26: Number of steps taken to reach a peak

Graph

The graph in figure 27 contains the following,

- The light blue line shows the minimum cone's height
- The dark blue line shows the maximum height of the agents at that time step
- The red line shows the average height of all the agents.
- Black line shows the highest cone's height.

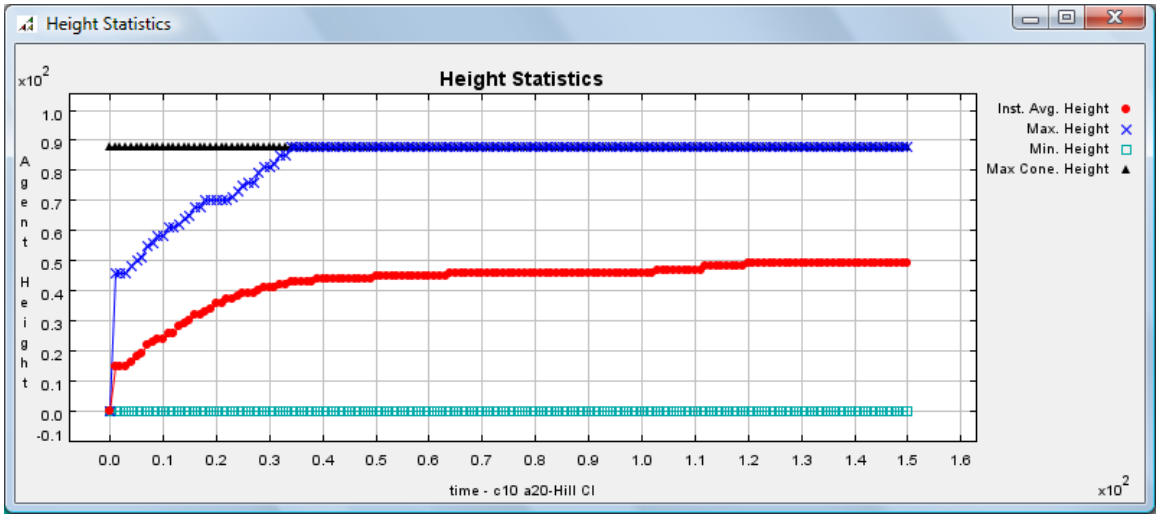


Figure 27: Example of a graph

Chapter 4: Experimental Setup

We performed experiments to find out which strategy is able to find the highest peak faster. Experiments were done on both the static and the dynamic environments. All the nine strategies explained in chapter 3 were tested in each of the following combinations.

- 10 cones and 20 agents in static/ dynamic environments with neighbours within 10 and 50 pixels.
- 10 cones and 10 agents in static/ dynamic environments with neighbours within 10 and 50 pixels.
- 10 cones and 5 agents in static/ dynamic environments with neighbours within 10 and 50 pixels.
- 5 cones and 20 agents in static/ dynamic environments with neighbours within 10 and 50 pixels.
- 5 cones and 10 agents in static/ dynamic environments with neighbours within 10 and 50 pixels.
- 5 cones and 5 agents in static/ dynamic environments with neighbours within 10 and 50 pixels.

The above mentioned six tests fall into two categories that are 10 cones and 5 cones.

For both categories, initially the cones and the agents were randomly deployed and the cone locations and the agents starting points were saved into files. Then for the next test in the same category the cones and the agents were loaded from the files. This was done for comparison purposes, since if the cones and the agents in a category are not in the same place for all the tests in that category we cannot compare the results and come to a conclusion on which strategy is the best. The results of the tests are given below,

4.1 Static Environment – 10 cones and 20 agents

Strategy S1: Hill Climbing Algorithm.

In this algorithm only one agent was able to find the highest peak as shown in figure 28. For more details see section 3.4. Agents average was about 50 pixels for 150 steps.

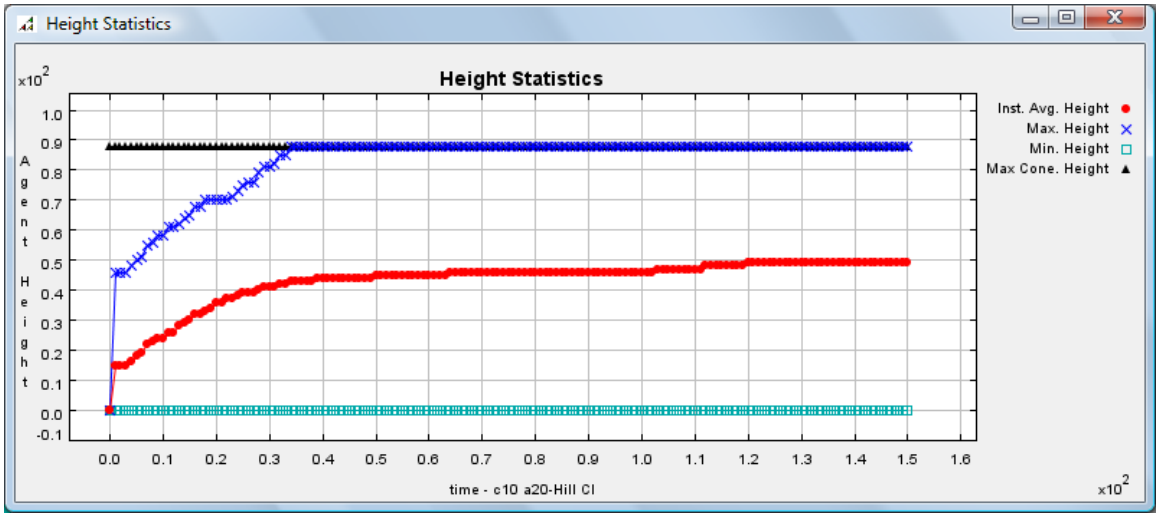


Figure 28: Strategy S1 – static - Hill Climbing Algorithm with 10 cones and 20 agents

Strategy S2: Equal Probability of Reputation for selecting a neighbour.

In this strategy initially all the agents will get a reputation of 0. Then each agent by using equal probability (50/50 probability of picking any neighbour) will pick a neighbour. For more details see section 3.5. In this strategy none of the agents were able to find the highest peak as shown in figure 29. Agents average was about 65 pixels for 150 steps.

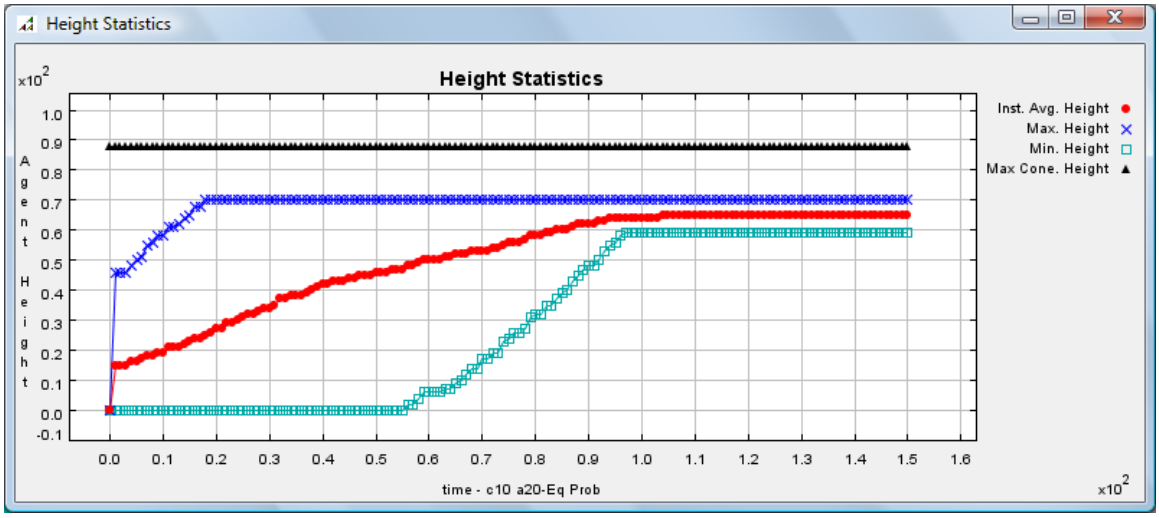


Figure 29: Strategy S2 – static - Equal Probability of Reputation for selecting a neighbour with 10 cones and 20 agents.

Strategy S3: Biased probability of Reputation for selecting a neighbour.

In this strategy for some agents to get reputations, equal probability of reputation strategy (S2) will run for the first five steps. Then each agent by using biased probability will select a neighbour. For more details see section 3.6. In this strategy none of the agents were able to find the highest peak as shown in figure 30. The average was about 68 pixels for 150 steps.

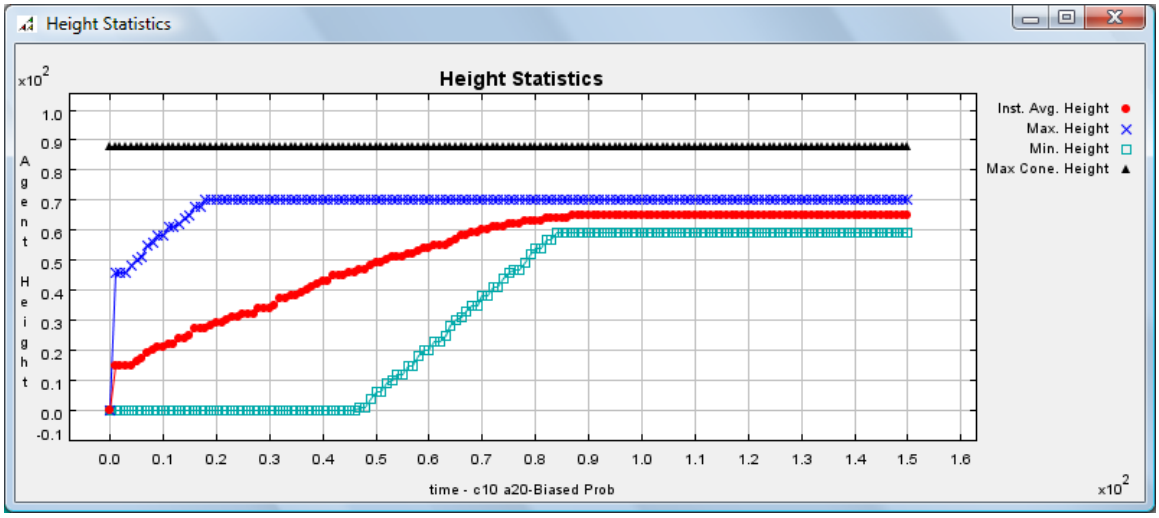


Figure 30: Strategy S3 – static - Biased Probability of Reputation for selecting a neighbour with 10 cones and 20 agents.

Strategy S4: Situational Knowledge of Cultural Algorithm

At every tenth iteration the top 10% agents heights are passed to the belief space which is a global repository. The agents can randomly by using equal probability, select whether to pick an agent from the belief space or from the neighbourhood. For more details see section 3.7. In this strategy none of the agents were able to find the highest peak as shown in figure 31. The average was about 39 pixels for 150 steps.

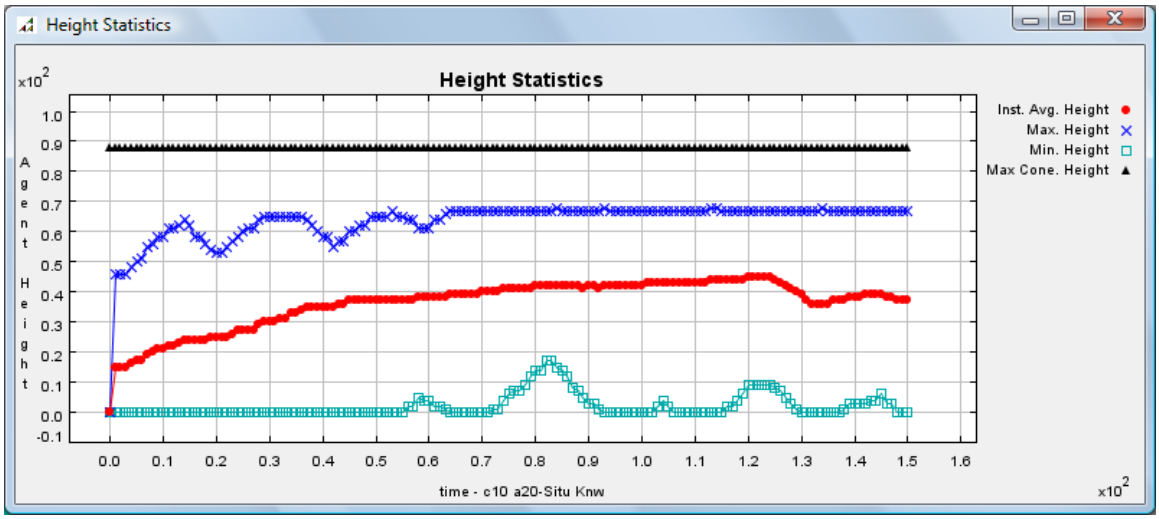


Figure 31: Strategy S4 – static - Situational Knowledge of Cultural Algorithm with 10

cones and 20 agents.

Strategy S5: Situational Knowledge and Mutation

The user can enter a mutation probability(ex: 0.05), then at every 10 iterations, the random number generator will generate a number and if its between 1-5, then that agent will be mutated, and will continue to be mutated until the next tenth iteration. If the number that the random number generates is not between 1-5, then that agent will use S4 strategy to move. That agent is not mutated. Once an agent is mutated, random x,y coordinates will be generated and the agent will continue to move towards that direction till the next tenth iteration. The test was carried out by entering 0.05 for the mutation probability. For more details see section 3.8. In this strategy none of the agents were able to find the highest peak as shown in figure 32. The average was about 65 pixels for 150 steps.

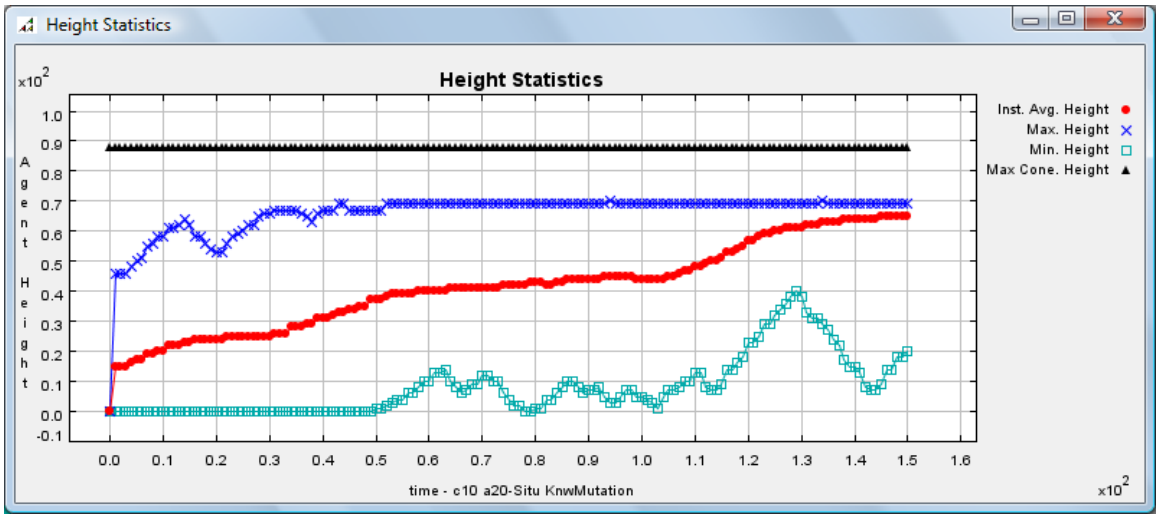


Figure 32: Strategy S5 – static - Situational Knowledge and Mutation with 10 cones and 20 agents.

Strategy S6 : Neighbours and Domain Knowledge of Cultural Algorithm

At each step all the agents will check within 50 pixels apart for neighbours. If there are neighbours then will get their maps and update it's own map. Then will select the closest

location that none of its neighbours have been and travel there. Then on their way when they come across a cone instead of climbing it they will calculate the peak of the cone. Domain knowledge is used to calculate the peak of a cone. For more details see section 3.9. In this strategy all the agents were able to find the highest peak as shown in figure 33. The agents took about 45 steps to reach the highest peak.

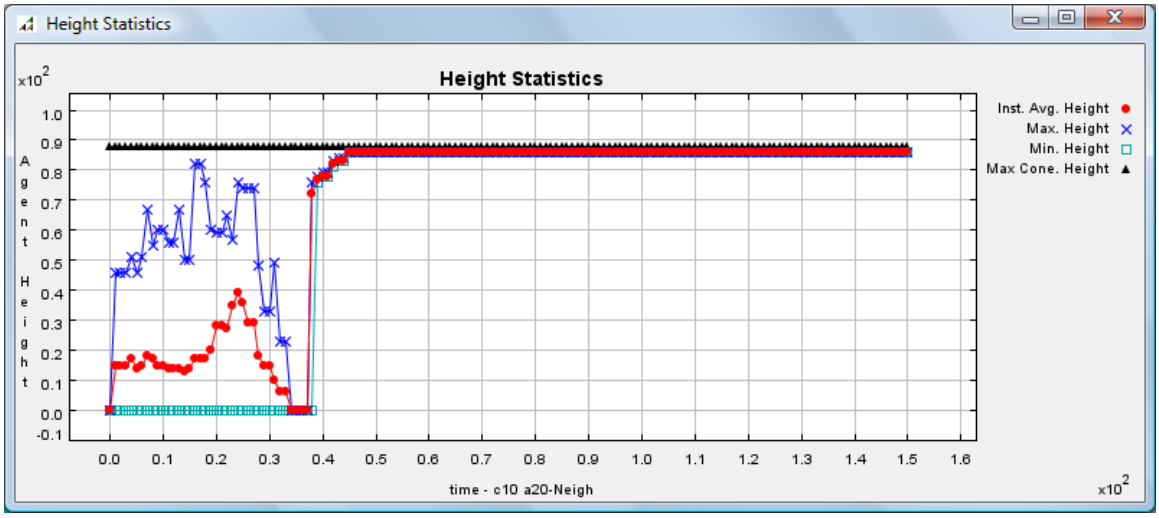


Figure 33: Strategy S6 – static- Neighbours and Domain Knowledge of Cultural Algorithm with 10 cones and 20 agents.

Strategy S7 : Situational, Domain, Historical, and Topographical knowledge of Cultural Algorithm along with surrounding heights and the neighbours.

Each agent will get information from the Cultural Algorithm's knowledge sources such as, Situational knowledge, Topographic knowledge, Historical knowledge, domain knowledge, also from the neighbour's who are within 50 pixels and the surrounding area. Then the agent will update it's map with this information and will pick the closest location that none of its neighbour's have been nor a location given by the above mentioned knowledge sources. After each step each agent will check its map to find whether it is complete. If it is complete then will look for the highest height in the map and will update the situational knowledge and set the map complete flag to true. Then all other agents will get the highest height from situational knowledge and everyone will travel to the

highest height. For more details see section 3.10. In this strategy all the agents were able to find the highest peak as shown in figure 34. The agents took about 47 steps to reach the highest peak.

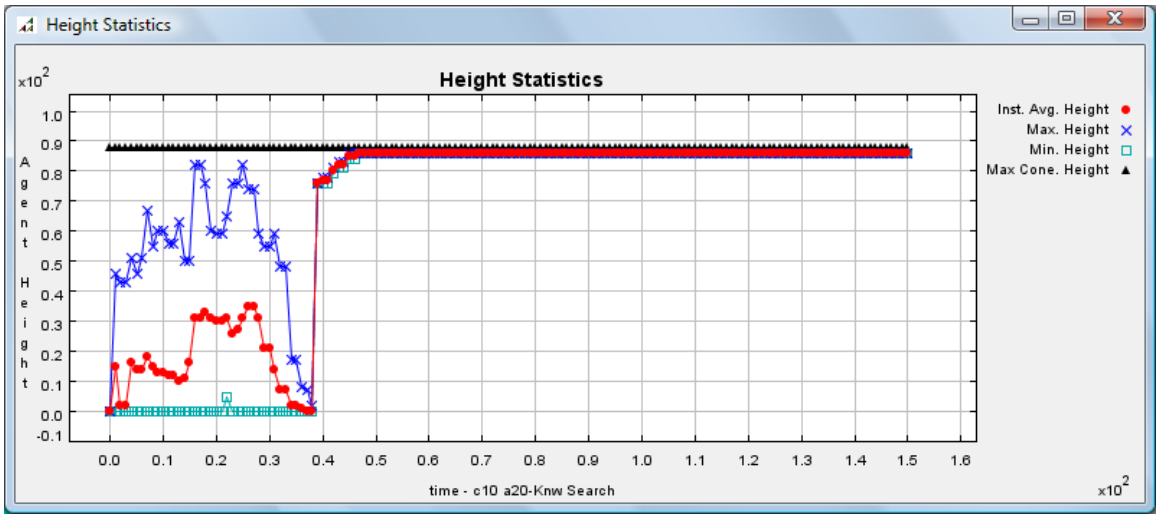


Figure 34: Strategy S7 – static - Situational, Domain, Historical, and Topographical knowledge of Cultural Algorithm along with surrounding heights and the neighbours with 10 cones and 20 agents.

Strategy S8: Topographical Knowledge of Cultural Algorithm

The landscape is divided into cells where a master topographical map has sixteen equal cells where each cell is 50 by 50 pixels. According to the locations of the agents they will be assigned to the nearest cell. Once an agent lands on a cell the agent will start exploring the cell and will check for agents within 50 pixels apart(neighbours) then will update its map with the value(s) given by the neighbours and also with its own explored locations. This way more than 1 agent would not explore any area that was explored by another. Then when it has finished exploring that particular cell the agent will check in the master map for cells that are not been explored by any agent. If there are any unexplored cells then that agent will be assigned to that cell. If there are no more unexplored cells then the agents who have finished exploring their assigned cells will come to the middle of the landscape. Once all the agents are in the middle, then they will update their maps and get

the highest height and travel to the highest height. For more details see section 3.11. In this strategy all the agents were able to find the highest peak as shown in figure 35. The agents took about 55 steps to reach the highest peak. This is within the worst case (T_{wc}) and best case (T_{bc}) steps with 1 agent.

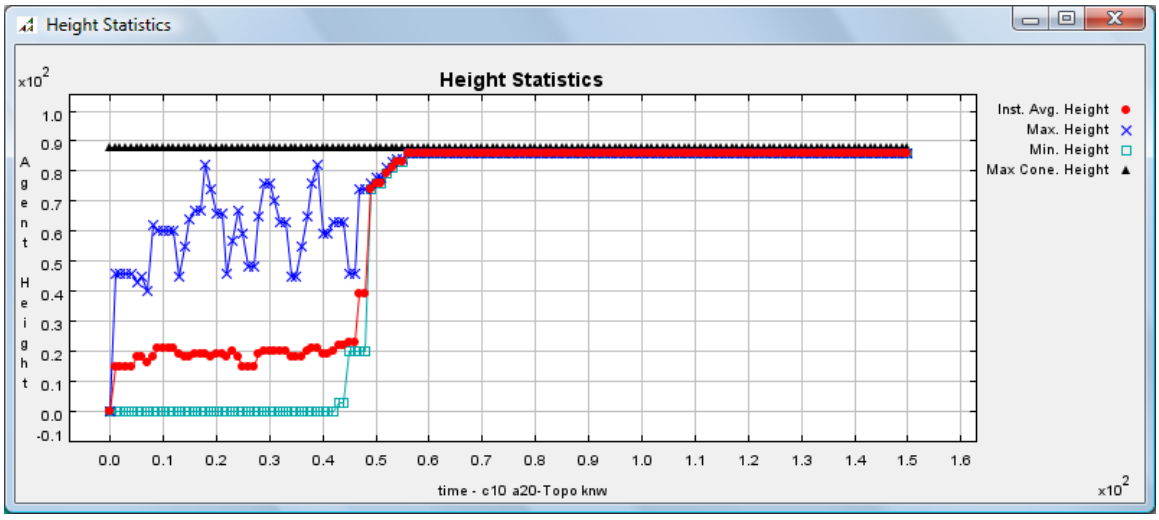


Figure 35: Strategy S8 – static - Topographical Knowledge of Cultural Algorithm with 10 cones and 20 agents.

Strategy S9: Modified S8

This strategy is very similar to strategy 8. Once all the agents have finished exploring their cells instead of travelling to the middle of the landscape as in strategy 8, they will calculate the highest height and travel to the highest height. For more details see section 3.12. In this strategy all the agents were able to find the highest peak as shown in figure 36. The agents took about 51 steps to reach the highest peak. This is within the worst case (T_{wc}) and best case (T_{bc}) steps with 1 agent.

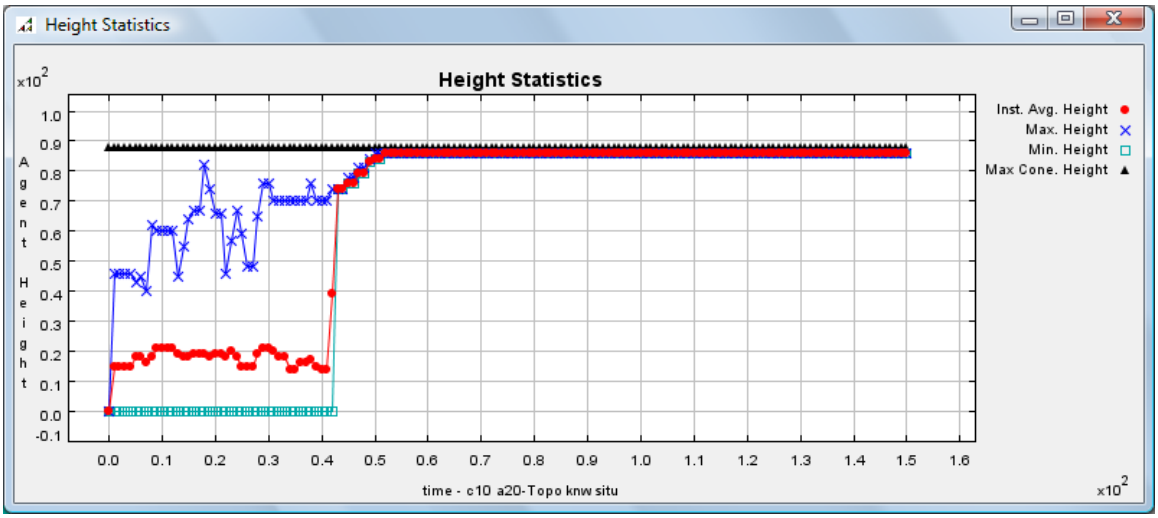


Figure 36: Strategy S9 – static - Modified S8 with 10 cones and 20 agents.

4.2 Dynamic Environments – 10 cones and 20 agents

Initially just like the static environment cones will be deployed on the landscape. Then after hundred steps the cones will be moved to different locations. Then after the next hundred steps (two hundred steps) not only the cones will be moved but also the heights of all the cones will be changed. This will be repeated for five hundred steps. Then at the five hundredth step the landscape that was loaded initially will be loaded again and then the next landscape (the landscape that was loaded at the hundredth step) will be loaded. When the environment changes the agents will start exploring from where they stopped.

Strategy S6 : Neighbours and Domain Knowledge of Cultural Algorithm

The performance of the agents are shown in figure 37 and the results of this test are given in table 2.

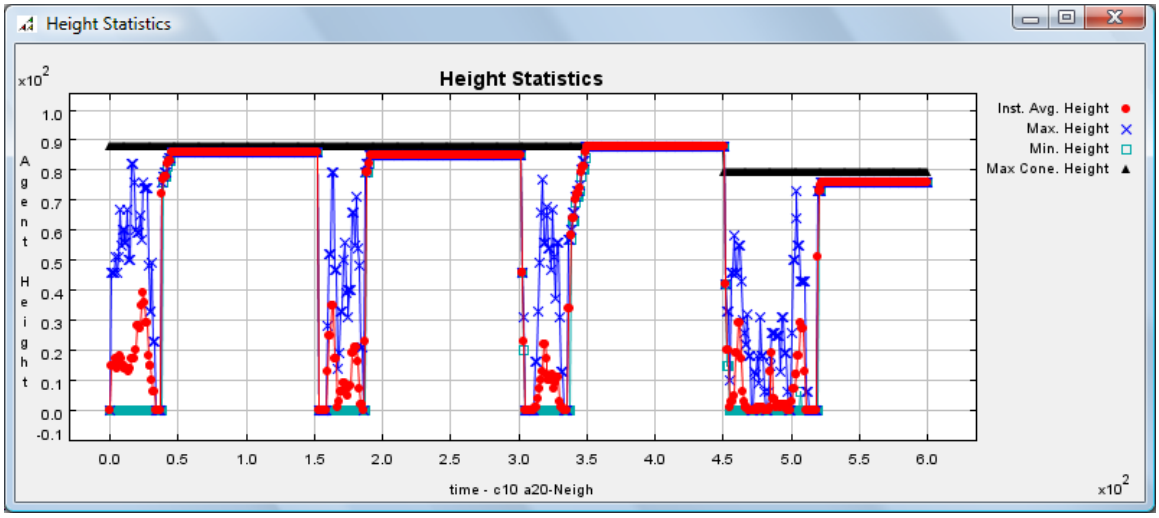


Figure 37: Strategy S6 – dynamic- Neighbours and Domain Knowledge of Cultural Algorithm with 10 cones and 20 agents

Strategy S7 : Situational, Domain, Historical, and Topographical knowledge of Cultural Algorithm along with surrounding heights and the neighbours.
The performance of the agents are shown in figure 38 and the results of this test are given in table 2.

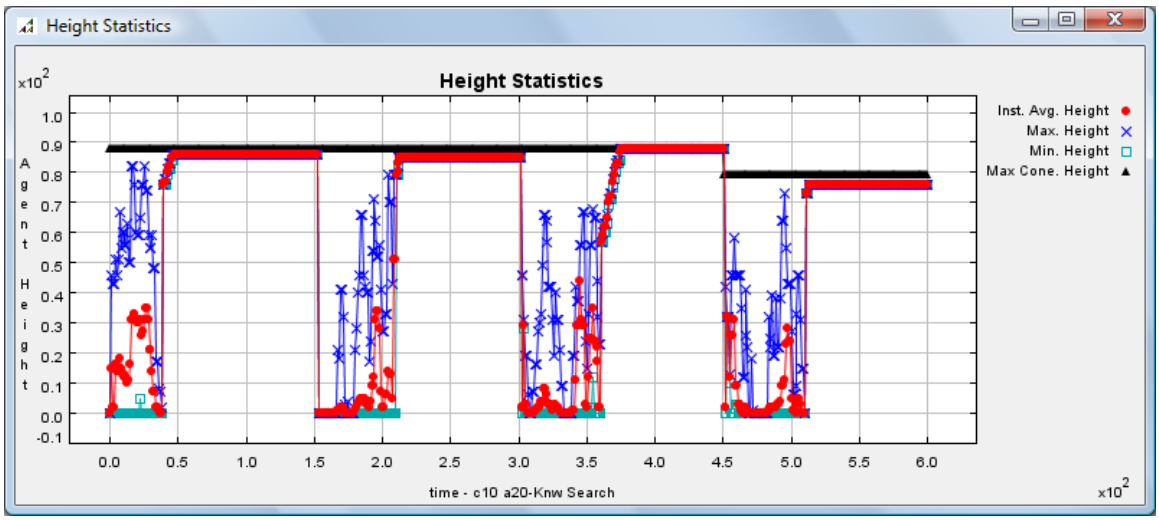


Figure 38: Strategy S7 – dynamic-Situational, Domain, Historical, and Topographical knowledge of Cultural Algorithm along with surrounding heights and the neighbours with 10 cones and 20 agents.

Strategy S8: Topographical Knowledge of Cultural Algorithm
The performance of the agents are shown in figure 39 and the results of this test are given in table 2.

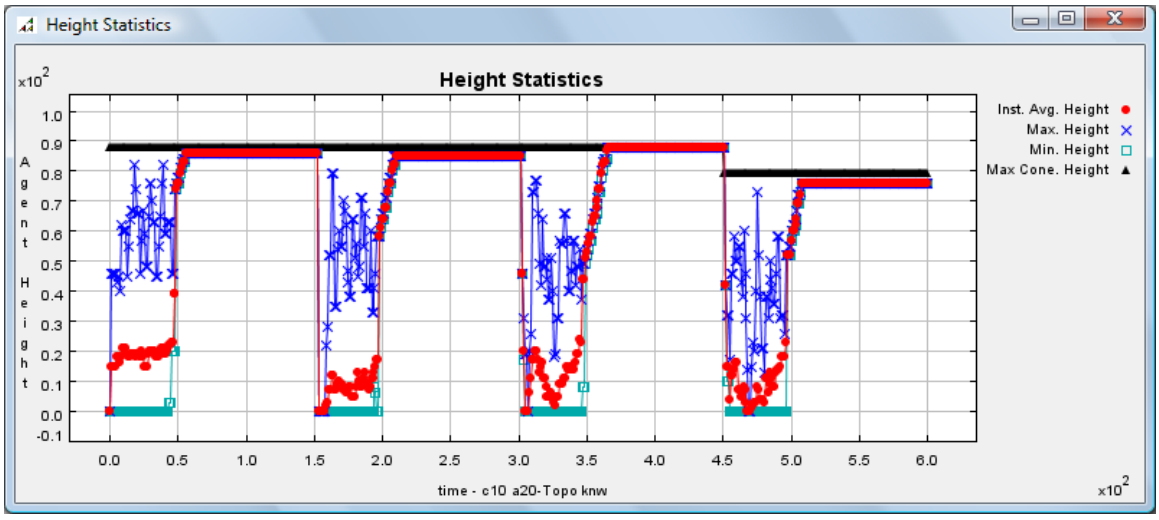


Figure 39: Strategy S8 – dynamic-Topographical Knowledge of Cultural Algorithm with 10 cones and 20 agents

Strategy S9: Modified S8
The performance of the agents are shown in figure 40 and the results of this test are given in table 2.

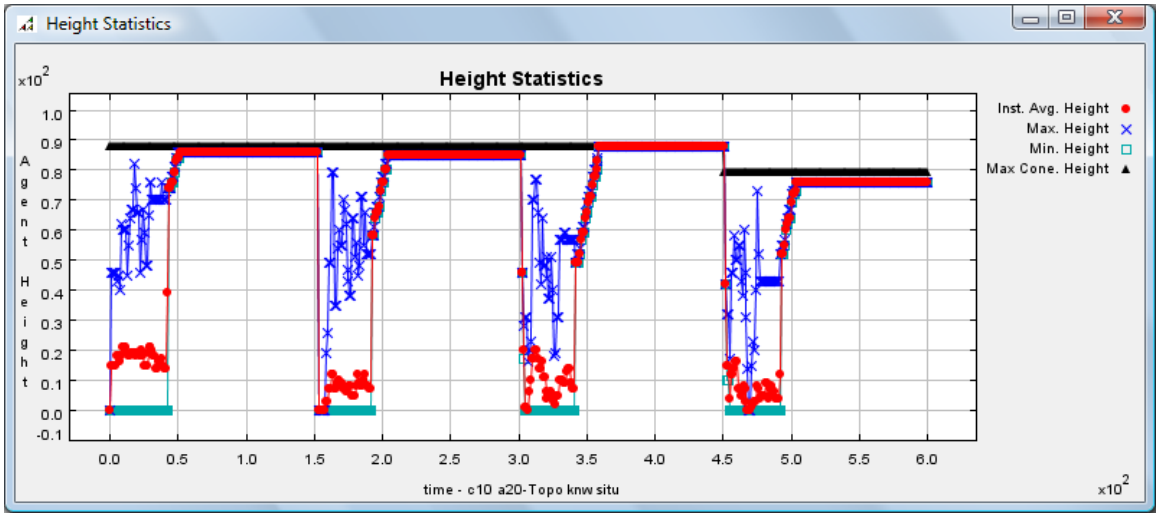


Figure 40: Strategy S9 – dynamic-Modified S8 with 10 cones and 20 agents.

Table 2: Results of the strategies with 10 cones and 20 agents with neighbours within 50 pixels

Strategy	**Static Env.**	**Dynamic Env.**				**Average of Dynamic (steps)**
		En1	En2	En3	En4	
S3(Best out of non knowledge strategies)	N/F	N/F	N/F	N/F	N/F	
S4	N/F	N/F	N/F	N/F	N/F	
S5	N/F	N/F	N/F	N/F	N/F	
S6	45	49	48	50	53	50.00
S7	47	49	52	55	52	52.00
S8	55	51	51	52	51	51.25
S9	50	50	50	51	50	50.25

Table 3: Results of the strategies with 10 cones and 20 agents with neighbours within 10 pixels

Strategy	**Static Env.**	**Dynamic Env.**				**Average of Dynamic (steps)**
		En1	En2	En3	En4	
S3(Best out of non knowledge strategies)	N/F	N/F	N/F	N/F	N/F	
S4	N/F	N/F	N/F	N/F	N/F	
S5	N/F	N/F	N/F	N/F	N/F	
S6	74	74	80	72	70	74.00
S7	76	76	110	70	70	81.50
S8	58	58	60	65	58	60.25
S9	52	52	50	60	50	53.00

4.3 Static and Dynamic Environments – 10 cones and 10 agents

The Table 3 given below contains the results of the tests. The first five strategies were run for 150 steps. The rest of the strategies were run till all the agents reached the highest peak.

Table 4: Results of the strategies of 10 cones and 10 agents with neighbours within 50 pixels

Strategy	**Static Env.**	**Dynamic Env.**				**Average of Dynamic (steps)**
		En1	En2	En3	En4	
S3(Best out of non knowledge strategies)	N/F	N/F	N/F	N/F	N/F	
S4	N/F	N/F	N/F	N/F	N/F	
S5	N/F	N/F	N/F	N/F	N/F	
S6	53	55	60	70	N/F	61.66*
S7	52	55	70	80	65	67.50
S8	79	75	80	90	80	81.25
S9	75	70	80	85	80	78.75

*- The average of all the valid tests

Table 5: Results of the strategies of 10 cones and 10 agents with neighbours within 10 pixels

Strategy	**Static Env.**	**Dynamic Env.**				**Average of Dynamic (steps)**
		En1	En2	En3	En4	N/F
S3(Best out of non knowledge strategies)	N/F	N/F	N/F	N/F	N/F	N/F
S4	N/F	N/F	N/F	N/F	N/F	N/F
S5	N/F	N/F	N/F	N/F	N/F	N/F
S6	138	138	N/F	140	130	136.00
S7	105	105	140	110	130	121.25
S8	80	80	90	100	90	90.00
S9	75	75	70	80	70	73.75

*- The average of all the valid tests

4.4 Static and Dynamic Environments – 10 cones and 5 agents

The Table 4 given below contains the results of the tests. The first five strategies were run for 150 steps. The rest of the strategies were run till all the agents reached the highest

peak.

Table 6: Results of the strategies of 10 cones and 5 agents with neighbours within 50 pixels

Strategy	**Static Env.**	**Dynamic Env.**				**Average of Dynamic (steps)**
		En1	En2	En3	En4	
S3(Best out of non knowledge strategies)	N/F	N/F	N/F	N/F	N/F	N/F
S4	N/F	N/F	N/F	N/F	N/F	N/F
S5	N/F	N/F	N/F	N/F	N/F	N/F
S6	105	105	95	95	90	96.25
S7	109	110	95	100	105	102.5
S8	120	120	130	140	125	128.75
S9	124	120	120	135	130	126.25

Table 7: Results of the strategies of 10 cones and 5 agents with neighbours within 10 pixels

Strategy	**Static Env.**	**Dynamic Env.**				**Average of Dynamic (steps)**
		En1	En2	En3	En4	
S3(Best out of non knowledge strategies)	N/F	N/F	N/F	N/F	N/F	N/F
S4	N/F	N/F	N/F	N/F	N/F	N/F
S5	N/F	N/F	N/F	N/F	N/F	N/F
S6	N/F	N/F	N/F	N/F	N/F	N/F
S7	280	N/F	N/F	N/F	N/F	N/F
S8	125	125	130	130	150	133.75
S9	122	122	120	130	130	125.50

4.5 Static and Dynamic Environments – 5 cones and 20 agents

The Table 5 given below contains the results of the tests. The first five strategies were run for 150 steps. The rest of the strategies were run till all the agents reached the highest peak.

Table 8: Results of the strategies of 5 cones and 20 agents with neighbours within 50 pixels

Strategy	**Static Env.**	**Dynamic Env.**				**Average of Dynamic (steps)**
		En1	En2	En3	En4	
S3(Best out of non knowledge strategies)	N/F	N/F	N/F	N/F	N/F	N/F
S4	N/F	N/F	N/F	N/F	N/F	N/F
S5	N/F	N/F	N/F	N/F	N/F	N/F
S6	45	45	45	50	65	51.25
S7	45	48	60	75	60	60.75
S8	56	55	60	70	55	60.00
S9	52	50	50	55	50	51.25

Table 9: Results of the strategies of 5 cones and 20 agents with neighbours within 10 pixels

Strategy	**Static Env.**	**Dynamic Env.**				**Average of Dynamic (steps)**
		En1	En2	En3	En4	
S3(Best out of non knowledge strategies)	N/F	N/F	N/F	N/F	N/F	N/F
S4	N/F	N/F	N/F	N/F	N/F	N/F
S5	N/F	N/F	N/F	N/F	N/F	N/F
S6	72	73	71	71	72	71.75
S7	75	74	105	65	73	79.25
S8	58	60	65	68	59	63
S9	53	56	52	58	50	54

4.6 Static and Dynamic Environments – 5 cones and 10 agents

The Table 6 given below contains the results of the tests. The first five strategies were run for 150 steps. The rest of the strategies were run till all the agents reached the highest peak.

Table 10: Results of the strategies of 5 cones and 10 agents with neighbours within 50 pixels

Strategy	**Static Env.**	**Dynamic Env.**				**Average of Dynamic (steps)**
		En1	En2	En3	En4	
S3(Best out of non knowledge strategies)	N/F	N/F	N/F	N/F	N/F	N/F
S4	N/F	N/F	N/F	N/F	N/F	N/F
S5	N/F	N/F	N/F	N/F	N/F	N/F
S6	56	55	60	70	75	65.00
S7	53	55	65	70	65	63.75
S8	79	70	75	85	72	75.50
S9	75	70	80	90	70	77.50

Table 11: Results of the strategies of 5 cones and 10 agents with neighbours within 10 pixels

Strategy	**Static Env.**	**Dynamic Env.**				**Average of Dynamic (steps)**
		En1	En2	En3	En4	
S3(Best out of non knowledge. strategies)	N/F	N/F	N/F	N/F	N/F	N/F
S4	N/F	N/F	N/F	N/F	N/F	N/F
S5	N/F	N/F	N/F	N/F	N/F	N/F
S6	135	140	N/F	130	132	134
S7	126	128	132	105	122	121.75
S8	83	85	86	95	90	89
S9	75	76	78	85	86	81.25

4.7 Static and Dynamic Environments – 5 cones and 5 agents

The Table 7 given below contains the results of the tests. The first five strategies were run for 150 steps. The rest of the strategies were run till all the agents reached the highest peak.

Table 12: Results of the strategies of 5 cones and 5 agents with neighbours within 50 pixels

Strategy	**Static Env.**	**Dynamic Env.**				**Average of Dynamic (steps)**
		En1	En2	En3	En4	N/F
S3(Best out of non knowledge strategies)	N/F	N/F	N/F	N/F	N/F	N/F
S4	N/F	N/F	N/F	N/F	N/F	N/F
S5	N/F	N/F	N/F	N/F	N/F	N/F
S6	105	105	90	90	85	92.50
S7	100	110	100	100	110	105
S8	120	120	130	140	130	130.00
S9	122	120	120	135	120	123.75

Table 13: Results of the strategies of 5 cones and 5 agents with neighbours within 10 pixels

Strategy	**Static Env.**	**Dynamic Env.**				**Average of Dynamic (steps)**
		En1	En2	En3	En4	N/F
S3(Best out of non knowledge strategies)	N/F	N/F	N/F	N/F	N/F	N/F
S4	N/F	N/F	N/F	N/F	N/F	N/F
S5	N/F	N/F	N/F	N/F	N/F	N/F
S6	N/F	N/F	N/F	N/F	N/F	N/F
S7	N/F	N/F	N/F	N/F	N/F	N/F
S8	125	126	128	130	128	128
S9	123	128	126	125	129	127

Chapter 5: Discussion

5.1 Simulation Statistics

Figures 41 and 42 compares the number of steps taken for all the agents to find the highest peak in both static and dynamic environments.

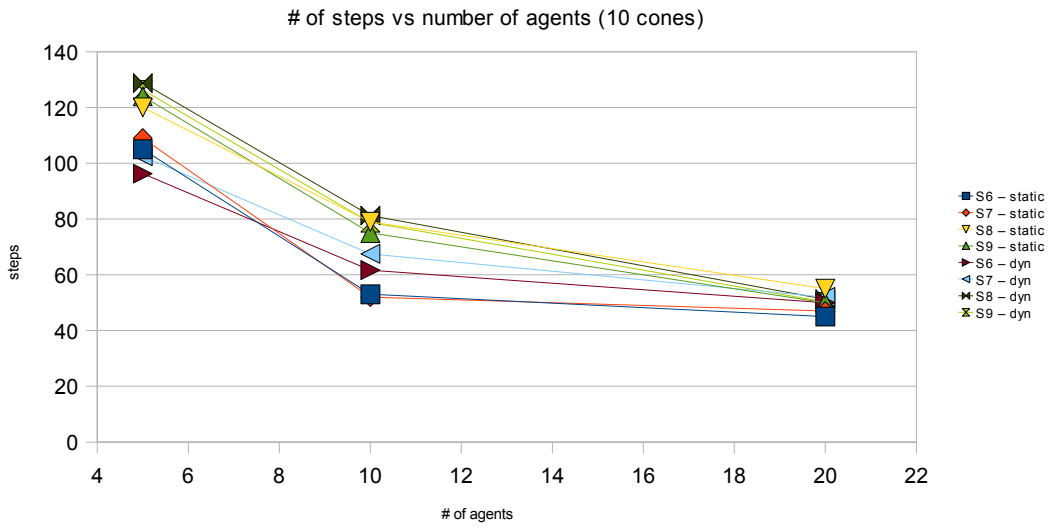


Figure 41: Shows the number of steps that all the agents required to find the highest peak for strategies S6 – S9 in both the static and the dynamic environments with 10 cones (neighbour size 50 pixels).

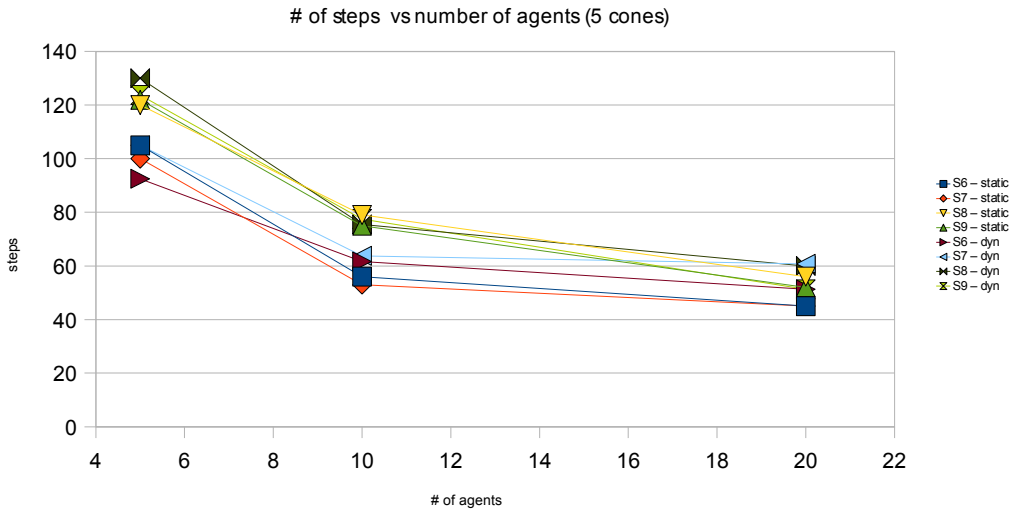


Figure 42: Shows the number of steps that all the agents required to find the highest peak for strategies S6 – S9 in both the static and the dynamic environments with 5 cones (neighbour size 50 pixels).

Figures 43 to 45 shows that the number of steps the agents took to find the highest peak for strategies S6 – S9 in both the static and the dynamic environments were not dependent on the number of cones.

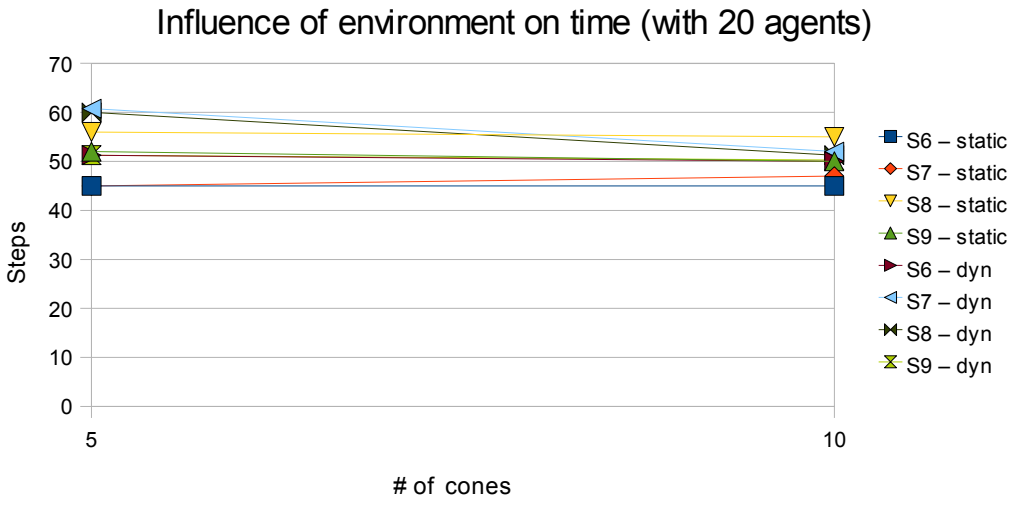


Figure 43: Shows the number of steps 20 agents are required to find the highest peak for strategies S6 – S9 in both static and dynamic environments with 5 and 10 cones (neighbour size 50 pixels)

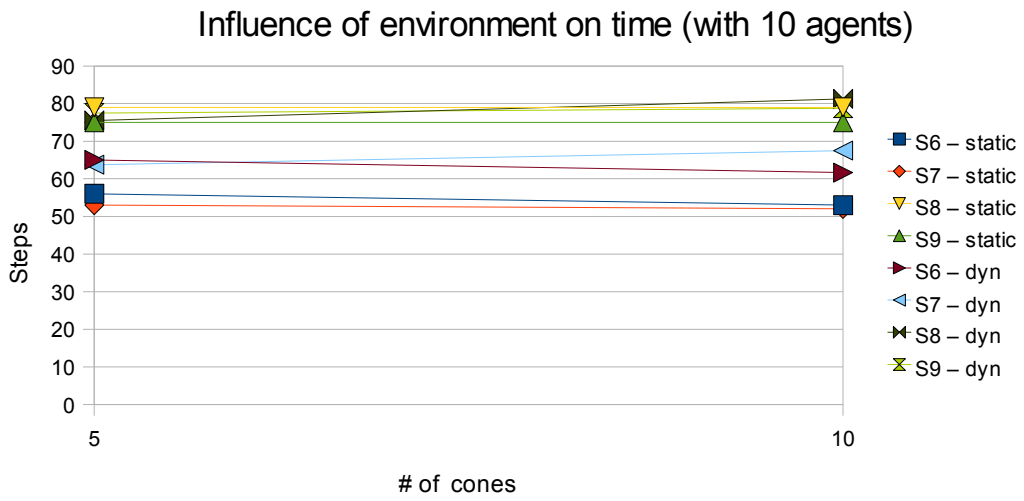


Figure 44: Shows the number of steps 10 agents are required to find the highest peak for strategies S6 – S9 in both static and dynamic environments with 5 and 10 cones (neighbour size 50 pixels).

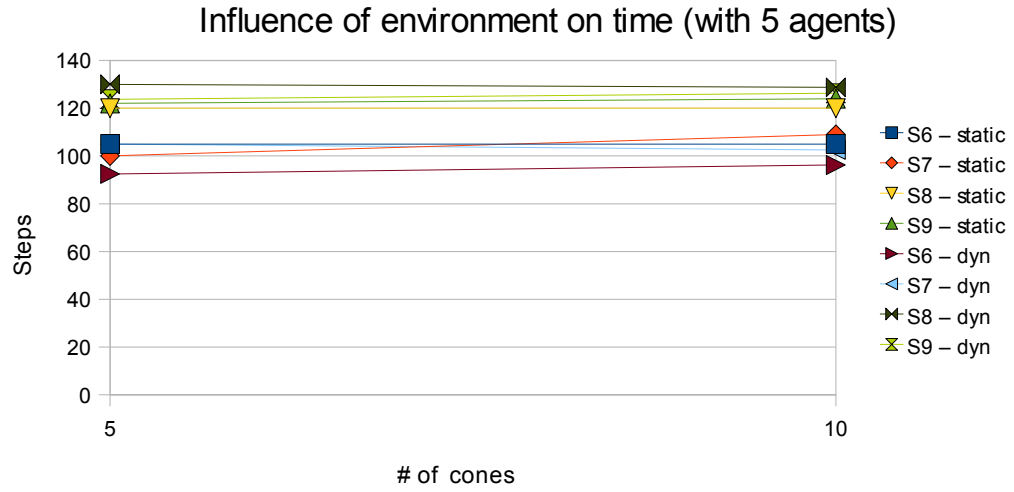


Figure 45: Shows the number of steps 5 agents are required to find the highest peak for strategies S6 – S9 in both static and dynamic environments with 5 and 10 cones (neighbour size 50 pixels).

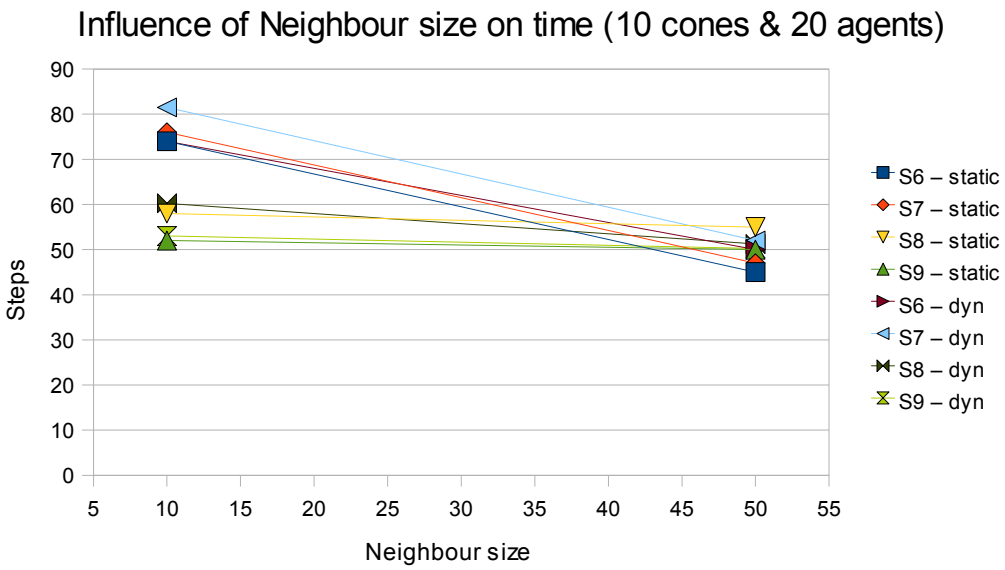


Figure 46: Shows the influence of the neighbour size on the simulation with 10 cones and 20 agents.

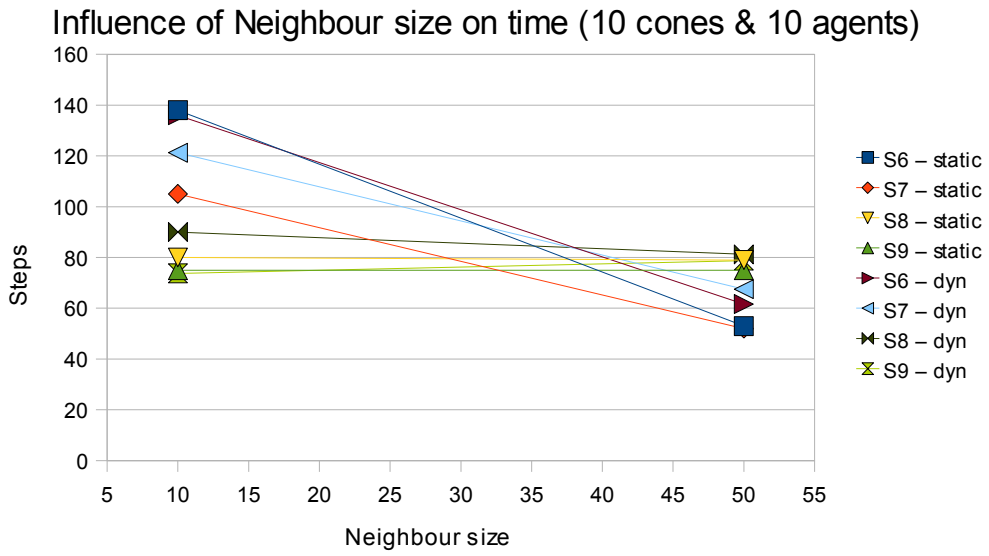


Figure 47: Shows the influence of the neighbour size on the simulation with 10 cones and 10 agents.

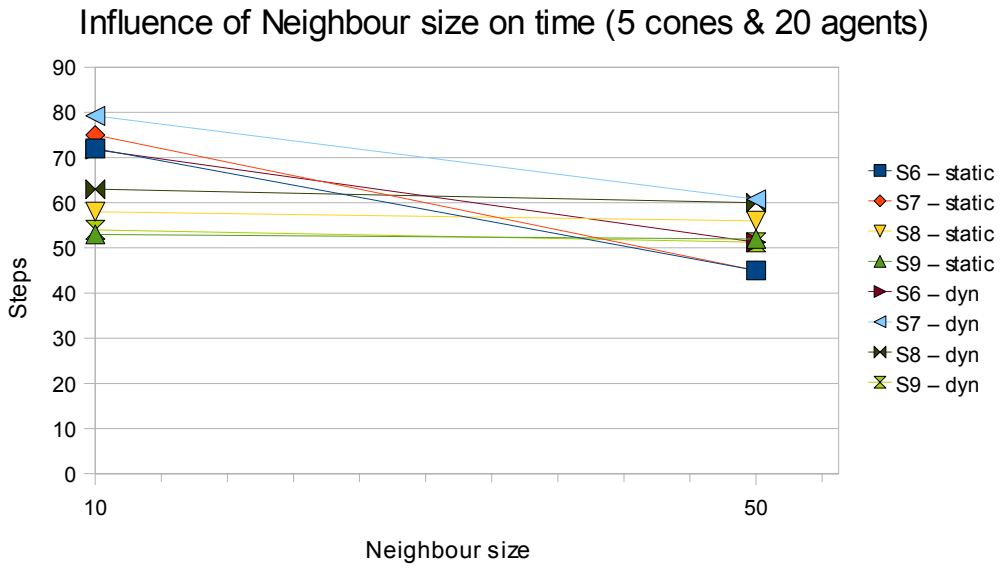


Figure 48: Shows the influence of the neighbour size on the simulation with 5 cones and 20 agents.

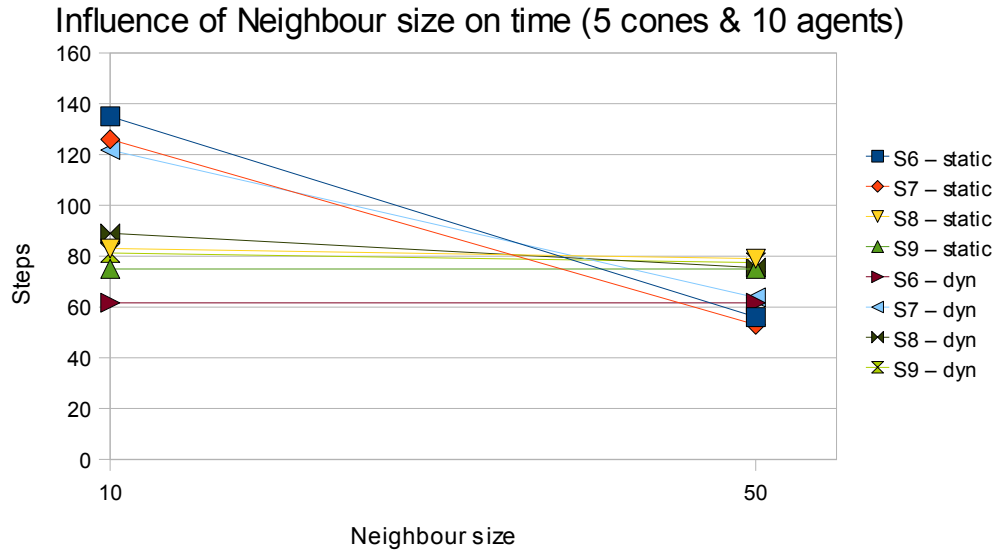


Figure 49: Shows the influence of the neighbour size on the simulation with 5 cones and 10 agents.

Note: Charts showing Neighbour size on time for 5 cones & 5 agents is not shown as the strategies S6 and S7 did not find any the cones within 150 steps.

5.2 Statistical Results

By analysing the test results that were presented in chapter 4 we can clearly observe that in strategies S1 to S5, all the agents were not able to find the highest peak. However in strategies S6 to S9 all the agents were able to find the highest peak.

Given the landscape canvas of 200 by 200 pixels, theoretically an exhaustive approach would take 40000 steps in total to determine the highest peak. In strategies S6 to S9, each agent has a map scaled one tenth the size of the landscape. The map (20 x 20 cells, where each cell represents and area of 10 x 10 pixels) allows each agent to identify explored areas and reconstruct the landscape of the canvas as it travels. Furthermore, this also reduces the exploration to no more than 400 steps (1%). However the drawback is that since only one pixel in each 10 x 10 pixel area is sampled there can be instances where two cones have close but unequal heights on the landscape in which case, the highest peak may not be found. On the other hand if a unique cell exists where all the pixels in the 10 x 10 cell have values greater than any other cell, then the highest peak will be found. Once all the agents reach the cell where the highest peak is located, Hill Climbing algorithm (S1) is used to reach the peak.

Based on the statistical data for S6 to S9 the most influence on time to find the highest peak is the number of agents. The statistics also shows that the number of cones have no effect on strategies S6 to S9. This is confirmed since the strategies are based on dividing the landscape irrespective of the number of cones. The goal is to find a strategy(s) in which all the agents are able to reach the highest peak (global optimum) the fastest.

It has been hypothesized in section 1.3 that domain based topographic exploitation strategies are better than the other strategies. From the statistics it can be observed that a topographical approach (S6 to S9) produced better and close results. With strategies S6 and S7 where the agents are going in search of unexplored agent map cells, the agents can consume many steps along the way towards unexplored cells. Furthermore, base on results in figure 46, 47, 48 and 49 the performance of S6 and S7 are heavily

dependant on neighbour size, where a larger neighbour size produced better results. With strategies S8 and S9, the agents are assigned to sections of the topographical map where the agents have minimal (if there are more than sixteen agents) or no overlap with another agent.

Although S8 and S9 did not always show better results than strategy S6 and S7 as in the case with neighbour size of 50 pixels, S8 and S9 had the least influence on neighbour size and showed better results than S6 and S7 with a small neighbour size of 10 pixels. Furthermore with S8 and S9 the landscape is explored in a more efficient "divide and concur" approach integrated with knowledge sharing.

Therefore based on the results, domain based topographic exploitation strategies, Strategy S8: Topographical Knowledge of Cultural Algorithm; and Strategy S9: Modified S8, provided overall better results than other presented strategies.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

Our main goal was to incorporate several Cultural Algorithms knowledge as a group or individually to socially motivated population based algorithms in which the entire population as a whole is able to find the global optimum. Therefore we incrementally developed a number of strategies of increasing complexity that are applied on an application neutral system which contains static/dynamic environments.

Strategy 1 - Implemented the Hill Climbing Algorithm. In this the agents are randomly scattered across the landscape. They will search the surrounding area for a higher location. When a higher location is found they will move there otherwise will continue looking for a higher location. This process will be repeated till the agents do not find a higher location in the surrounding area.

Strategy 2 – The agents within 50 pixels apart are grouped into social networks. An agent by using the equal probability of reputation will select a neighbour and inquire for its height. If the neighbour's height is greater than itself then will move by one pixel towards that location and increase the neighbour's reputation by one. Otherwise will decrease the neighbour's reputation by one and use the Hill Climbing algorithm to move.

Strategy 3 - The agents within 50 pixels apart are grouped into social networks. An agent by using the biased probability of reputation will select a neighbour and inquire for its height. If the neighbour's height is greater than itself then will move by one pixel towards that location and increase the neighbour's reputation by one. Otherwise will decrease the neighbour's reputation by one and use the Hill Climbing algorithm to move.

Strategy 4 – For the first ten steps strategy 3 will be executed. Then at the tenth step by using equal probability of reputation an agent is able to select a neighbour from its

neighbourhood or from the belief space of the Cultural Algorithm. If the belief space is selected then the agent will inquire from the selected agent for its height. If the height is greater than itself then will move by one pixel towards that location and increase that particular agent's reputation by one. Otherwise by using biased probability of reputation will select a neighbour and execute strategy 3.

Strategy 5 – The user is able to set a mutation probability. If the user decides not to mutate any agent then strategy 4 will be executed. If the user decides to mutate then will enter the mutation probability. When an agent is mutated that agent will go in the direction of a randomly generated location for the next ten steps.

Strategy 6 – At each step all the agents will check within 50 pixels apart for neighbours. If neighbours are found the agent will retrieve their maps and update it's own map. With the updated map the agent will select the closest location none of its neighbours have been (i.e. by determining the closest initialized value of -1) and travel toward it. On its way if the agent comes across a cone it will calculate the peak of the cone without climbing it and also will check for neighbours within 50 pixels and update it's map. At the end of each update of MyMap the agent checks if it's map is complete (i.e. no more -1s). With a complete map, the agent will find the highest height and travel to the location.

Strategy 7 – This is very similar to Strategy 6. Instead of only looking at the neighbours maps to decide on the next step the agents will use Cultural Algorithm's situational knowledge, historical knowledge, domain knowledge, topographic knowledge and the surrounding area to decide on the next step. Then once an agent's map is complete, The situational knowledge is updated with the highest height and myMap complete flag is set to true. Then all agents will go to the location of the highest peak since the entire landscape was explored at least by one agent.

Strategy 8 – The landscape is divided into cells where a master topographical map has sixteen equal cells where each cell is 50 by 50 pixels. According to the locations of the

agents they would be assigned to the nearest cell. Once an agent lands on a cell the agent will explore the entire cell and update it's map. Then when it has finished exploring that particular cell the agent will check in the master map for cells that have not been explored by any agent. If there are any unexplored cells then that agent will be assigned to that cell. If there are no more unexplored cells then the agents who have finished exploring their assigned cells would come to the middle of the landscape. Once all the agents have come to the middle, then they will update their maps and get the highest height and travel to the highest height.

Strategy 9 - This strategy is very similar to strategy 8. The landscape is divided into sixteen cells. Initially the agents would randomly be scattered on the landscape. There is a master topographical map with sixteen cells. Each agent will be assigned to the closest cell that is not been explored by another agent(if there are less than sixteen agents) according to their current location. Once an agent lands on a cell the agent will explore the entire cell and update it's map. Then when it has finished exploring that particular cell the agent will check in the master map for cells that have not been explored by any agent. If there are any unexplored cells then that agent is assigned to that cell. If there are no more unexplored cells and all the agents have finished exploring the assigned cells then will go through the maps of all the agents and find the location of the highest height. Then all the agents will travel to the highest height.

After performing extensive experiments on both static and dynamic environments, and carrying out simulation analysis as explained in the previous chapter (Chapter 5), we found that neighbour strategy (S6) and knowledge based strategy (S7) are highly dependent on the neighbour size and topographical knowledge strategy (S8) and modified topographical knowledge strategy (S9) were not dependent on the neighbour size. Therefore we came to the conclusion that domain based topographic exploitation strategies of CA play a significant role.

6.2 Future Work

Test these strategies in other benchmark functions.

To apply these strategies on multi dimensional landscapes and test the performance.

References

[Dorigo 2006] Dorigo, M.; Birattari, M.; Stutzle, T., Ant colony optimization, IEEE Computational Intelligence Magazine, Volume 1, Issue 4, Page(s):28 - 39, Nov. 2006

[Eberhart, 1995] Eberhart, R. C., and Kennedy, J. (1995). Particle Swarm Optimization. In the proceedings of the IEEE 1995 International Conference on Neural Networks, Perth, Australia, IEEE Service Center, Piscataway, NJ, 12-13.

[Morrison, 1999] Morrison, R.; and De Jong, K. (1999). A test problem generator for non-stationery environments. In Proceedings of the Congress of the Evolutionary Computing, 2047-2053.

[Chung, 1997] Chung, Chang Jin. 1997. Knowledge-Based approaches to self-adaptation in cultural algorithms. Wayne State University.

[Sycara, 1998] Sycara, K, P.; Multiagent Systems. American Association for Artificial Intelligence. 445 Burgess Drive, Menlo Park, California 94025. Page(s): 79-92, 1998.

[Poslad, 2007] Poslad, S.; Specifying Protocols for Multi-Agent Systems Interaction. Queen Mary, University of London. ACM Trans. Autonom. Adapt. Syst. 2, 4, Article 15, 24 pages. November 2007.

[Pavard] Pavard, B., and Dugdale, J.; An Introduction to Complexity in Social Science. GRIC-IRIT, Toulouse, France.

[McKelvey, 1997] McKelvey.; 'Quasi-Natural Organisation Science', Organisation Science, No 8, page(s): 351-380. 1997.

[McKelvey, 1999] McKelvey.; 'Complexity Theory in Organisation Science: Seizing the Promise or Becoming a Fad?', Emergence, Vol 1 No 1. Page(s): 5-32.

[Marion 1999] Marion, R.; The Edge of Organisation: Chaos and Complexity Theories if Formal Social Systems, Sage.

[Zhao, 2006] Zhao, Y., Hou, X., Yang, M., and Dai, Y.; Measurement Study and Application of Social Network in the Maze P2P File Sharing System. Proceedings of the First International Conference on Scalable Information Systems, Hong Kong. May 29-June 1 2006.

[Milgram, 1967] Milgram, S.; The Small World Problem Psychology Today, page(s) 62-67, June 1967.

[Watts 1998] Watts, D. J., and Strogatz, S. H,; Collective Dynamics of Small World Networks. Nature, 393:440-442. June 1998.

[Ahn, 2007] Ahn, Y., Han, S., Kwak, H., Moon, S., and Hawoong Jeong. Analysis of Topological Characteristics of Huge Online Social Networking Services. WWW 2007, May 8 – 12, 2007, Banff, Alberta, Canada.

[Backstrom, 2006] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg and Xiangyang Lan. Group Formation in Large Social Networks: Membership, Growth and Evolution. KDD 2006, August 20-23, 2006, Philadelphia, Pennsylvania, USA.

[Kumar, 2006] Ravi Kumar, Jasmine Novak and Andrew Tomkins. Structure and Evolution of Online Social Networks. KDD 2006, August 20-23, 2006, Philadelphia, Pennsylvania, USA.

[Jamali, 2006] Mohsen Jamali, and Hassan Abolhassani. Different Aspects of Social Network Analysis. IEEE 2006. Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence.

[Golbeck, 2006] Jennifer Golbeck and James Hendler. Inferring Binary Trust Relationships in Web Based Social Networks. 2006 ACM Transactions on Internet Technology, Vol. 6, No. 4, November 2006, Pages 497- 529.

[Goecks, 2004] Jeremy Goecks and Elizabeth D. Mynatt. Leveraging Social Networks for Information Sharing. CSCW 2004, November 6-10, 2004, Chicago, Illinois, USA.

[Boyd, 2004] Dana Michele Boyd. Friendster and Publicly Articulated Social

Networking. CHI 2004, April 24-29, 2004, Vienna, Austria. ACM

[Wikipedia, 2006] Was last modified on 27 January 2009 at 14:38.

[Marczyk, 2004] Marczyk, A., Genetic Algorithms and Evolutionary Computation, 2004.

[Saleem, 2001] Saleem, S, M., Knowledge Based Solution to Dynamic Optimization Problems using Cultural Algorithms.

[Maniezzo, 2000] Maniezzo, V. (2000). Ant Colony Optimization: An overview. http://www3.csr.unibo.it/~maniezzo/didattica/Vienna/ACOintro.pdf Retrieved on March 12, 2005.

[Holland 1975] *Holland, J. H.;* Adaptation in natural and artificial systems. Ann Arbor: The University of Michigan Press, 1975.

[Goldberg, 1989] D. Goldberg., Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wisely 1989.

[Windisch, 2007] Windisch, A.; Wappler, S.; and Wegener, J.; Applying Particle Swarm Optimization to Software Testing. GECCO 2007. Proceedings of the 2007 conference on Genetic and evolutionary computation. July 7-11, London, England, United Kingdom.

[Eberhart, 1995] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In Proceedings of the 6th International Symposium on Micromachine Human Science, pages 39–43, 1995.

[Kennedy, 1995] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In Proceedings of the IEEE International Conference on Neural Networks, volume 4, pages

1942–1948 vol.4. IEEE Press, 1995.

[Kennedy, 2001] Kennedy J., and Eberhart. R. C. and Shi, Y. (2001). Swarm intelligence. Morgan Kauffmann Publishers, San Francisco, CA.

[Reynolds, 1978] Reynolds, R. G.; On Modeling the evolution of hunter gatherer decision making systems, Geographical Analysis, 10(1), 31-46.

[Reynolds, 1994] Reynolds, R. G.; An Introduction to Cultural Algorithms. In

Proceedings of the 3rd Annual Conference on Evolutionary Programming, Sebald, A.V., Fogel, L.J. Ed. World Scientific Publishing, River Edge, NJ, 131-139.

[Reynolds, 2005] Reynolds, R. G.; Peng, B.; and Che, X.; Knowledge Swarms: Generating Emergent Social Structure in Dynamic Environments. Proceedings of the Agent 2005 Conference on Generative Social Processes, Models and Mechanisms, ANL/ DIS-06-5, ISBN 0-9679168-0, page(s) 747-770.

[Reynolds, Saleem, 2005] Reynolds, R. G.; Saleem, S. M.; The Impact of Environmental Dynamics on Cultural Emergence, in Perspectives on Adaptations in Natural and Artificial Systems. Oxford University Press, Page(s) 253-280.

[Xue, 2007] Xue, Z.; Guo, Y.; Improved Cultural Algorithm based on Genetic Algorithm IEEE International Conference on Integration Technology, 2007. ICIT '07. Page(s):117 - 122, 20-24 March 2007

[Coelho, 2006] An Efficient Particle Swarm Optimization Approach Based on Cultural Algorithm Applied to Mechanical Design, Coelho, L.; Mariani, V.C.; IEEE Congress on Evolutionary Computation, 2006. CEC 2006, Page(s):1099 – 1104, 16-21 July 2006.

[Peng, 2003] Peng, B.; Reynolds, R.G.; Brewster, J.; The 2003 Congress on Evolutionary Computation, 2003. CEC '03, Volume 3, Page(s):1965 - 1971, 8-12 Dec. 2003

[Peng, 2004]Peng, B.; Reynolds, R.G.;, Cultural algorithms: knowledge learning in dynamic environments, IEEE Congress on Evolutionary Computation, 2004. CEC2004. Volume 2, Page(s):1751 - 1758, 19-23 June 2004

[Peng, 2005] Peng, B.; Knowledge and Population Swarms in Cultural Algorithms for Dynamic Environments. P.hD Dissertation, Wayne State University, Detroit, Michigan.

[Morrison 1999] Morrison, R.; and De Jong, K.; A test problem generator for non-stationary environments. In Proceeding of the Congress on Evolutionary Computing, 2047-2053.

[De Jong, 1975] De Jong, K. A.; The Analysis of the behavior of a class of genetic adaptive systems, Doctoral dissertation, University of Michigan, Ann Arbor.

[Collier 2003] Collier, N., Howe, T., and North, M.; Onward and Upward: The Transition to Repast 2.0, Proceedings of the First Annual North American Association for Computational Social and Organizational Science Conference, Electronic Proceedings, Pittsburgh, PA USA ,June 2003

VITA AUCTORIS

NAME:	Viranthi R. Peiris
PLACE OF BIRTH:	Colombo, Sri Lanka
YEAR OF BIRTH:	1980
EDUCATION:	Bachelor of Science in Computing & Information Systems