1-1-2013

On the Evolvability of a Hybrid Ant Colony-Cartesian Genetic Programming Methodology

Sweeney Luis
Ryerson University

ON THE EVOLVABILITY OF A HYBRID ANT COLONY-CARTESIAN GENETIC PROGRAMMING METHODOLOGY

by

Sweeney Luis

B.E. in Information Technology, University of Mumbai, India, 2009.

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Canada, 2013

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

ON THE EVOLVABILITY OF A HYBRID ANT COLONY-CARTESIAN GENETIC PROGRAMMING METHODOLOGY

Sweeney Luis

M. Sc. in Computer Science, 2013

Ryerson University, Toronto, Canada

Abstract

In this thesis, we design a method that uses Ant Colonies as a Model-based Search to Cartesian Genetic Programming (CGP) to induce computer programs. Candidate problem solutions are encoded using a CGP representation. Ants generate problem solutions guided by pheromone traces of entities and nodes of the CGP representation. The pheromone values are updated based on the paths followed by the best ants, as suggested in the Rank-Based Ant System (AS_{rank}). To assess the evolvability of the system we applied a modified version of the method introduced in [1] to measure rate of evolution which considers variability and neutrality as the major influences in the evolution of a system. Our results show that such method effectively reveals how evolution proceeds under different parameter settings and different environmental scenarios. The proposed hybrid architecture shows high evolvability in a dynamic environment by maintaining a pheromone model that elicits high genotype diversity.

Acknowledgements

I am ever grateful to my supervisor Dr. Marcus V. dos Santos for his support, guidance and direction whose efforts have made this thesis possible. Marcus, I am very thankful to you for the countless number of hours and effort you have put in reviewing this work and the meetings we have had bringing this work to life. Also for making this graduate program experience enjoyable and meaningful.

I thank Dr. Alireza Sadeghian, Dr. Eric Harley and Dr. Isaac Woungang who were part of my examination committee for taking the time out of their busy schedules to read and provide their vital feedback on this thesis. Their insights have been an invaluable contribution to this work.

I would like to thank my family especially my parents for their support and belief in me. Mom and Dad you mean the world to me and I am so very appreciative for all that you have done for me. I would also like to thank my sister who has been a role model to me.

I would like to thank Vincent Koo, Dheeraj Peddi and Subir Biswas my fellow graduate peers who have made this journey one to remember. The weekly scrum meetings with Graham Holker, Mahsa Mostowf, Mohammad Islami, Alexey Adamsky and Olivia-Linda Enciu who have shared their knowledge and ideas which have been very helpful. Lastly I would like to thank all the graduate students in ENG251 (CI2) especially Leyla Vakilian and Ervis Sofroni who have been excellent companions.

Sweeney Luis

Ryerson University

January 2013

Dedication

To my family.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The work presented here belongs to the research area known as Evolutionary Computation (EC), a subfield of Machine Learning (ML) that involves optimization algorithms inspired either on the behavior of animals or on biological evolution. In particular, this thesis deals with the design of a hybrid methodology that extends Ant Colony Optimization to Cartesian Genetic Programming as a Model-based search.

In [2] C. Darwin proposed the theory that all living organisms are descendants of a common ancestor. Organisms produce offspring not all of which survive. Each offspring has its own unique characteristics that differentiates it from others. If these characteristics are beneficial they help the organism survive the process of *natural selection*. In simple terms natural selection is the survival of the fittest where the inferior organisms of the same species eventually die out. These characteristics are inheritable by the offspring of the organism.

Inspired by the Darwinian theory, EC extended the concepts and terminology of natural evolution to computation. The goal of every EC system is to find a solution to a problem by maintaining a population of candidate solutions called *individuals* that are modified iteratively by means of evolutionary operators. Most EC strategies follow a similar Evolutionary Algorithm (EA) where a population of individuals is maintained for a number of iterations called *generations*. At the end of each generation the individuals of the population are evaluated to find the highly fit individuals, *i.e.,* the candidate solutions that currently best solve the problem. In every new generation the

population comprises of individuals more fit than individuals from previous generations. New individuals are created by the application of evolutionary operators that modify individuals of the current generation, thus creating individuals for the next generation. This process is continued until the solution is found or some other termination criteria are met.

The key differences between the various EC strategies are how individuals are represented, evaluated and the search strategies employed. *Genetic Programming* (GP) is an EC methodology where an individual is a computer program represented as a *tree*. Each node of the tree can either be a function or a terminal. These functions and terminals are predefined at the start. A function as the name suggests is a function that takes in one or more terminals to produce an output. A terminal can either be a variable or a constant in the system. All leaf nodes of the tree must be terminals. An individual is evaluated by recursively executing all nodes beginning from the root node traversing down till all leaf nodes are executed. GP employs the use of genetic operators as a search strategy to find individuals of higher fitness. A genetic operator creates a new individual by varying the individual(s) in a way that the new individual created resembles the original individual but is not the same. The original individuals are called the parents and the new individual the child.

Cartesian Genetic Programming (CGP) is a form of GP where an individual of the system is an *indexed graph* addressed on the Cartesian co-ordinate system. The graph is made up of a number of connected nodes where each node is made up of a number of inputs and a function. The functions of the system are predefined at the start of the run. The input to each node can either be an input node predefined at the start of the system or the output of another node that has been previously defined. Every individual has a combination of one or more output nodes. An individual is evaluated by executing the nodes connected from the input nodes to the output node(s) of the individual. Like GP, CGP uses genetic operators as a search strategy.

Model-based search (MBS) [3] is a methodology of optimizing the search strategy of any Machine

Learning (ML) system by encoding candidate solutions to a problem in a probabilistic model. To begin the model is initialized uniformly and is sampled using a sampling algorithm to create a population of individuals. A *learning* algorithm is employed on the population to capture dependencies amongst best solutions and encode them in the model. The process of creating solutions from the model and learning from the solutions generated is repeated until a solution is found or another termination criterion is met.

Ant Colony Optimization (ACO) is a swarm optimization methodology that takes inspiration from the foraging behavior of ants and explicitly uses probability distributions of problem solutions. In ACO artificial ants roam the problem space, leaving pheromone traces in the solution instances they find along the way. Each ant samples this pheromone to navigate the path to the next tentative problem solution. The pheromone model is updated after each iteration, rewarding good paths by depositing an additional amount of pheromone value and unfitting paths have their pheromone value reduced by the process of evaporation.

1.1 Motivation

The motivation for exploring the hybridization of ACO and CGP arises from two particular features of these methodologies, namely, the capability of Cartesian Genetic Programming to evolve complex structures using a fixed length representation and the success of Ant Colony Optimization in handling both static and dynamic problems.

The research done in CGP has gained momentum due to its fixed length structure and graph representation. Having a fixed length representation, CGP is able to evolve highly complex structures. The graph representation used is more general than the tree representation. There are a number of redundancies that can be present in the CGP representation individual which gives rise to neutrality

in the system. The presence of neutrality has shown to be beneficial in the evolvability of problem solutions.

In applying ACO to any problem the concept of maintaining a pheromone model and learning from the candidate solutions remains the same. The difference lies in the learning mechanism, *i.e.,* the method in which the pheromone model is updated. ACO has been successfully applied to a number of static and dynamic problems and can be applied to any combinatorial problem [4]. Artificial ants have memory which is highly beneficial in building solutions, evaluating solutions and retracing the path taken. The CGP representation is a suitable structure for a pheromone model. Ants can travel from one node to another guided by the pheromone levels on the trails represented in the model.

1.2 Objectives

The objectives of this thesis are listed below:

1. Develop an appropriate probabilistic model and learning algorithm to support the hybridization of ACO and CGP.

2. Quantitatively assess the evolvability of the proposed hybrid approach using the Rate of Evolution (R_e) as a measure that helps better understand the dynamics of the system during a run.

3. Evaluate the performance of our system using experiments that cover a number of static and dynamic tests, and test the influence of various system parameters on the evolvability and then compare our results to Cartesian Ant Programming (CAP).

1.3 Methodology

Machine Learning=representation+optimization+evaluation, [5]

In the following we provide an overview of our methodology by briefly describing how we developed each of the components appearing in the right-hand side of the equation above.

The *representation* that we use in the AC-CGP system is the representation used in CGP. CGP differentiates between the genotype and phenotype of an individual. In EC, *genome or genotype* refers to the encoding used to represent an individual and the *phenome or phenotype* is the visual representation of an individual. In CGP the *genome* of an *individual* is an integer string of fixed length which maps to a directed acyclic graph (the individual's *phenotype*) that can be executed as a computer program. Each integer in the genotype of the individual represents either an input, function or an output node. Each genome is generated by an artificial ant by sampling the pheromone model. We design a tabular pheromone model that holds a pheromone value for every entity at each node of the individual. An entity is either an input or function of the system. The higher the pheromone value the greater is the probability of the entity being selected at that position of the genome. Since our CGP representation is of fixed size the pheromone model is also of fixed size and is scalable to represent any sized CGP individual. The pheromone model is initialized at the start of the run by assigning all entities an initial pheromone value to ensure every entity has an equal probability of being selected. We use the *roulette wheel* selection algorithm where all possible entities at each point of the genome could be visualized as having a portion of area assigned to it on a roulette wheel proportional to its pheromone level. The wheel is spun and the entity on which the wheel stops is selected at that position of the individual's genotype.

To *evaluate* an individual first we map the individual's genotype, *i.e.,* the integer string to its equivalent phenotype, *i.e.,* the indexed graph. The output of the system is a combination of one or

more nodes in the graph which are sampled by the artificial ant and represented in the genotype of the individual. We then proceed to execute all nodes connected to the output node that returns the output of the individual. The output of each individual helps us evaluate its *fitness*. Fitness can be defined as the quantitative measure of the quality of an individual. In other words the closer an individual is to solving the problem, the higher is its fitness. The fitness of an individual is calculated using a fitness function.

Optimization is achieved by running our AC-CGP system for a number of iterations and at the end of each iteration we use the Rank-based Ant System (AS_{rank}) learning algorithm to update the pheromone model. In AS_{rank} after every iteration the ants are ranked according to the fitness of individuals they create. We select n ants that created individuals of highest quality from the current iteration to update the pheromone model in accordance to their fitness. Each of the entities selected by the ant has its pheromone level increased by depositing an additional level of pheromone. The higher the fitness of the individual the larger is the level of pheromone deposited. The pheromone update is such that for every iteration the model is at least capable of creating an individual of maximum fitness from the previous iteration.

To test the performance of our system we choose the symbolic regression problem as a benchmark. We select a number of symbolic regression problems to assess the performance of our system for *average fitness* and *evolvability*. To quantify evolvability we observe the number of *nonsynonymous* and *synonymous* changes in the population over the iterations. Evolvability as defined in [1] is the ratio of the nonsynonymous substitution to the synonymous substitution rates. We perform an experiment to understand the effect of various system parameters on evolvability and convergence. In this experiment we alter one of the parameters of population size, initial pheromone level and number of *ranked-ants* at a time, keeping the rest the constant. We then test how our system performs under different environmental conditions. These environmental conditions include Random Evolution where a target expression is not given, Fixed Target Evolution where a single

target is assigned at the start of the run and Moving Target Evolution where the target is changed at different stages of the run. Lastly we simulate the work done in Cartesian Ant Programming (CAP) and compare our results for the Fixed Target Evolution and the Moving Target Evolution experiments.

1.4 Results and Contribution

Analyzing the results, we observed that our system is adaptable to both static and dynamic environments. In our test in varying the system parameters, we found a lower level of initial pheromone and a bigger population size helps in faster convergence. Also convergence is independent of the number of ranked ants that update the pheromone table. Our results also showed that AC-CGP is on par with CAP with regards to average fitness of the evolved population. AC-CGP, however, showed better adaptiveness when faced with a dynamic environment by maintaining a highly diverse genotype population.

The central contribution of this work was the successful introduction of a hybrid optimization algorithm that combines Cartesian Genetic Programming with ant colonies. We also extended the method to measure the rate of evolution introduced in [1] to evolutionary algorithms employing a model-based search. This helps us better understand the dynamics of the system during a run and serves as a guide to the types of problems an evolutionary system can be applied to.

1.5 Structure of this Thesis

In Chapter 2 we begin by providing a brief introduction to the field of Evolutionary Computation. We then review Genetic Programming and Cartesian Genetic Programming. Next we review

Ant Colony Optimization along with two different learning schemes Rank-based Ant System and MAX-MIN Ant System which is pertinent to our study. Lastly we give an account of quantifying evolvability for a evolutionary computational methodology.

Chapter 3 explains the methodology of our system. We begin by stating our AC-CGP algorithm and providing an explanation of it. We then provide an explanation of all the key components of our system beginning with the CGP representation. We proceed to describe our pheromone model and demonstrate how a CGP genome is created by sampling the pheromone model. We next explain how an individual is evaluated from its phenotype. The last component of our system is the pheromone update process. Here we describe how the pheromone table is updated using Rank-based Ant System. We then provide a method to quantify evolvability for a evolutionary computation system that incorporates a model-based search for its learning algorithm using the rate of evolvability concept. To conclude this chapter we provide a description of our benchmark symbolic regression experiments chosen to evaluate our system.

Chapter 4 presents the results of the experiments described in Chapter 3. We begin by showcasing our results for a Single Evolutionary Run. We then show the impact various parameters have on evolution by varying a single parameter from the population size, number of ranked ants and the initial pheromone level keeping the rest same. We next show how our system fares in different environmental conditions, namely Random Evolution, Fixed Target Evolution and Moving Target Evolution. To end this chapter we display the results found in comparing our system to CAP for the Fixed Target Evolution and Moving Target Evolution experiments.

Chapter 5 discusses our results. We then list our contributions and discuss possible future directions of our work.

Chapter 2

Literature Review

In the work presented here we propose a method for evolving computer programs that combines the problem representation used in Cartesian Genetic Programming (CGP) and the probabilistic techniques for searching the solution space used in Ant Colony Optimization (ACO). In this chapter we review the historic works done in Evolutionary Computation. We give a background of Genetic Programming and Cartesian Genetic Programming. We give an account of Ant Colony Optimization and two different learning schemes pertinent to our work. To end this chapter we give a background of the work done in quantifying evolvability in evolutionary computation.

2.1 Evolutionary Computation

Evolutionary Computation (EC) [6] is a branch of computer science that takes inspiration from the Darwinian theory of *natural evolution*. An EC system comprises of a *population* of candidate solutions called *individuals* that attempt to solve a definite problem. Through a number of *generations* these individuals are varied in an attempt to improve the *fitness* of the individual. Fitness is a quantitative measure of the quality of an individual. Individuals are varied by applying evolutionary operators to these individuals which results in creating new individuals that resemble their predecessors but are not the same. The process of generating individuals and varying them is continued until an individual that solves the problem is found or another *termination criterion* is

met.

Algorithm 1: An evolutionary algorithm

```
1: Create an initial population of individuals.
2: repeat
3:    Evaluate the population of individuals.
4:    if solution is found then
5:       stop.
6:    else
7:       Select individual/s using a search operator.
8:       Apply evolutionary operator to individual/s.
9:    end if
10: until Termination criteria are met.
```

Algorithm 1, depicts a basic evolutionary algorithm and almost all EC strategies follow this algorithm. The key difference between the various evolutionary computation techniques is how solutions are represented and the search strategy used to explore or exploit the search space. The most popular representations used are linear representation, trees and graphs. Where as the widely used search strategies employ the use of evolutionary operators or a model-based search.

2.2 Genetic Programming

Genetic Programming (GP) [7] is a branch of EC that enables a computer to solve problems automatically. The GP system maintains a population of computer programs known as *individuals* that solve the problem at hand. An individual is represented as a program tree with no distinction between its genotype and phenotype. Genotype is the underlying genetic coding of an individual where as the phenotype are its visible characteristics.

As a preparatory step to solving any GP problem, GP requires the definition of 5 parameters [8]:

- *Terminal set:* consists of variables and constants needed to solve the problem.

- *Function set:* consists of methods that take one or more *terminals* as inputs to produce an output.

- *Fitness Function:* measures the quality of an individual and assigns the fitness value to that individual.

- *Run parameters:* are the control parameters needed for the GP system; they include the *population size*, probability of selecting a *genetic operator* and maximum size of an individual.

- *Termination criteria:* this is the stop condition for the GP system. It is usually set to a maximum number of generations the system is allowed to run or the system is terminated once a solution is found.

A GP system begins with its parameter settings which include the population size, the number of generations and the ones listed above. An initial population of individuals is created randomly, where each individual is represented as a tree made of a number of nodes. A node can be either a function or a terminal. The creation of an individual begins by selection of the *root node* and iteratively selecting nodes (*left-to-right*, *top-to-bottom*) until all leaf nodes selected are terminals.

After all individuals of the population are created these individuals are evaluated. Evaluation is done either using the bottom up or top down approach. The bottom up approach involves executing all function nodes from the leaf nodes and moving up until the root node is reached. The top down approach takes place by recursively executing function nodes beginning from the root node and executing the nodes connected to it.

A new generation is created by using a *selection* algorithm that selects the best individuals of the current generation and places them in the next generation. The remaining individuals of the population are created by selecting one or more individuals from the current population at a time, applying genetic operators on them and placing these new individuals in the next generation. Since

the inception of EC, *reproduction*, *mutation* and *recombination* have been the most popular genetic operators. Genetic operators work on individuals in the population with a goal of creating newer individuals of higher fitness which can be incorporated into the next generation. Reproduction selects an individual from the current population and duplicates it into the population of the next generation. Mutation selects an individual, alters a node or an entire subtree of the individual thereby creating a new individuals. Recombination takes two or more individuals, labels them as the parent individuals; child individuals are created by combining different sub-trees of the individuals selected. This process of population creation, evaluation, selection and application of genetic operators is continued until a solution is met or another termination criterion is met.

2.3 Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) introduced by Miller in [9] is a form of GP where the individuals are represented as a graph addressed on the Cartesian co-ordinate system which can be executed as a computer program. CGP distinguishes between the genotype and phenotype of an individual. In CGP, the genotype of an individual is a string of integers of fixed-length which map to the phenotype which is an executable graph.

Prior to the start of any CGP system, we require the definition of the topology of the graph (*i.e.* number of rows and columns), input set, function set, number of outputs, number of input connections to a node and the level-back parameter. The genotype which is a string of integers represents each node of an individual and its output connections. A node of the graph is made up of one or more inputs and a function, each of which is represented by an integer in the genome. A node takes as input any of the inputs from the input set or the output of a previous node dependent upon the level-back parameter. The maximum number of inputs to a node is equivalent to the maximum

arity of the function set. The genotype is mapped to its equivalent phenotype to evaluate each individual. Evaluating an individual can be done in a number of ways: i) a feed forward execution of all nodes from the input nodes to the output nodes; ii) recursively executing connected nodes from the output nodes to the input nodes; or iii) using a two pass scheme which involves marking the connected nodes moving from the output nodes to the input nodes in the first pass and in the second pass we execute the marked nodes from the input nodes to the output nodes. A CGP system follows the similar algorithm as an EA as depicted in Algorithm 1, where every next generation contains the best individuals from the previous generation and the remaining population is populated with individuals by applying various genetic operators on individuals from the current generation.

CGP supports many-to-one mapping wherein many genotypes could map to the same phenotype which results in the presence of neutrality in an individual. The fixed-length representation used in CGP ensures that the bloating effect (*i.e.* uncontrollable exponential growth of individuals) is avoided. There is a possibility for a number of redundancies to be present in the CGP genotype, namely, *node* redundancy, *functional* redundancy and *input* redundancy. Node redundancy is the presence of unused nodes in the genome of an individual, *i.e.,* nodes that are not connected to the output node and hence do not affect the fitness of an individual. Functional redundancy occurs when a function implemented by a number of nodes can be expressed with a smaller number of nodes. Input redundancy occurs when the input nodes defined at the start of the system is not used by the individuals.

2.4 Ant Colony Optimization

Ant Colony Optimization (ACO) [10] is a swarm optimization technique inspired by the foraging behavior of some ants species. An ant moves from source to destination in the pursuit of food

guided by the pheromone levels on the available paths the ant can travel. Every ant deposits a layer of pheromone on the path it selects which evaporates over time. If a path is constantly being used by several ants, the pheromone level deposited on that path is larger. Therefore, there is a greater probability that an ant that comes along this path will choose this particular path. Similarly, the lower the pheromone levels on a path the smaller the probability that an ant will choose that path.

For artificial ants a pheromone model is maintained that holds the pheromone values at each node for the paths possible for the ant to travel from that node. The pheromone model is initialized at the start of the run and is updated at the end of every iteration. Pheromone update is the process in which good solutions are rewarded by adding to the level of pheromone on paths chosen to reach that solution and evaporating the pheromone level on paths that did not yield good solutions.

2.4.1 Rank-based Ant System

Rank-based Ant System (AS_{rank}) [11] is a ranked-based version of the Ant System (AS) [12]. In AS_{rank} after every iteration all ants are sorted according to their tour lengths. These ants are ranked according to their contributions and are weighted. Only the best ranked ants are allowed to update the pheromone model.

2.4.2 MAX-MIN Ant System

The MAX-MIN Ant System (MMAS) [13] is an improvement over the traditional AS that employs a greedier search strategy that further exploits the search space. In MMAS after each iteration the ant with the best solution is labeled the *iteration-best ant* and the ant with the overall best solution is called the *global-best ant*. Only the *iteration-best ant* is allowed to add pheromone to the model and to avoid stagnation of the pheromone trails they subjected to the process of

evaporation. Pheromone evaporation is a decrease in the level of pheromone on a trail caused due to the non-selection of that particular trail over a period of time. It is named as the MAX-MIN Ant System because the pheromone levels are bound to the interval $[\tau_{min}, \tau_{max}]$.

2.5 Model-based Search

One disadvantage of conventional evolutionary algorithms, is that one has no control on how genetic operators identify and exploit regularities in the search space. That is, such methods only implicitly use the probability distributions elicited by genetic operators to search the problem space.

Model-based Searches applied to Evolutionary Algorithms are known as Estimation of Distribution Algorithms (EDAs) [14, 15]. EDAs maintain a probabilistic model of the best individuals found and sample this model to create new individuals. Conventional EDA are designed to handle fixed-length genome representations, such as those used in genetic algorithms [16]. Estimation of distribution algorithms for genetic programming (EDA-GP) [17], also called Probabilistic Model Building GP, extend conventional EDAs to GP style tree representations. In EDA-GP, the design of the solution representation is tightly coupled with the design of the probabilistic model, as both have an impact on the algorithm's computational complexity and on the types of dependencies amongst solutions the algorithm is able to capture. For instance, Probabilistic Incremental Program Evolution (PIPE) [18] is an EDA-GP methodology where in solutions to a problem are encoded in a probability model called the *probabilistic prototype tree (ppt)*. PIPE is based on a representation that supports one-to-one genotype-to-phenotype mapping and its probabilistic model is able to capture univariate dependencies amongst candidate solutions. The works introduced in [19, 20] also used a one-to-one mapping representation scheme but introduced more sophisticated probabilistic models able to capture multivariate dependencies in GP style trees, although with a considerable

increase in the algorithm's computational complexity.

ACO is a form of MBS where in artificial ants execute a learning algorithm on the population of solutions and encode candidate solutions in a *pheromone model* through the process of *deposition* and *evaporation*. ACO has been successfully applied to a number of ML problems, static problems like the *Traveling Salesman Problem* (TSP) [21] and dynamic problems like *Job Scheduling in Grid Environment* [22]. Drawing from its success and application to a number of fields in computer science, ACO has been employed to a number of GP methodologies. Inspired on the Ants system introduced in [23], Ant programming (AP) [24] extended ACO to GP style tree representations. The AP system builds and modifies the programs in accordance to the pheromone model referred as *pheromone tree*. The pheromone tree is composed of a number of pheromone tables for each tree node containing the pheromone values for the possible functions and terminals for the corresponding node. Using a similar approach, but based on a different pheromone model, Dynamic ant programming (DAP) [25] introduced a dynamic tabular pheromone model which holds the pheromone values for the possible functions and terminals at each node and uses a *tabu list* restricting the selection of a non-terminal that has already been selected, thereby enabling the system to create diverse individuals. The pheromone table size changes dynamically in each iteration as nodes with low pheromone levels are deleted. This results in the system creating programs of smaller size on average. Our method is similar to Cartesian Ant Programming [26] where a pheromone model is sampled to create the genome of a CGP individual and used the Max-Min Ant System (MMAS) [13] as the learning algorithm to update the pheromone model.

2.6 Evolvability

Evolution has been studied in great depth from the biological point of view. In this section we highlight the computer science perspective of evolution and a way to quantify evolution using the *Rate of Evolution* (R_e) concept.

An evolutionary system can be termed successful if its population has the ability to evolve, *i.e.,* the ability of an individual to create children with higher fitness than itself [27]. The evolvability of an evolutionary system depends on various factors such as the individuals representation, genotype-to-phenotype mapping, selection pressure, evolutionary operators, size of population, genome redundancy, robustness to genetic operators *etc*. A reason why the study of evolvability is beneficial is because evolvability helps one understand the dynamics of a system and how a system is able to evolve complex structures for a problem solution [28].

Evolvability in an EC system is both hard to observe and quantify. In [29] M. Bedau and N. Packard mentioned that evolution does not take place by mere changes in the system but adaptive sustained changes. They went on to show how the presence of observable evolutionary activity in the system can serve as a guide in understanding evolvability in an evolutionary system. They defined evolutionary activity as the rate at which genetic changes are incorporated into the system. In [30] they provided a way of quantifying the evolutionary activity for a system using the *total cumulative evolutionary activity* or the *mean cumulative evolutionary activity* that takes into account the diversity in the system.

In [31] Y. Jin and J. Trommler proposed a method of measuring evolvability that is independent of the system's fitness function. Their method starts by calculating the genotypic and phenotypic distance between all individuals of the population. Evolvability is then calculated as the product of the interquartile range (IQR) of the genotypic and phenotypic distances. The IRQ measure

considers an individuals innovation ability and its robustness which are two key ingredients to evolvability considered in their work.

Hu and Banzhaf in [1] showed that *neutrality* and *variability* are key to the success of any EC system. They used the Rate of Evolvability, defined as the ratio of the *nonsynonymous to synonymous substitution rates* $(R_e = k_a/k_s)$, as a measure to evaluate evolvability. Changes brought about in the genome of an individual that brings about a change in the phenome with a change in its fitness is known as a *nonsynonymous* change, where as if the phenome of an individual remains the same along with its fitness it is called a *synonymous* change. Observing the changes to R_e over the run of an evolutionary system gives us an insight into the evolvability of the system.

Chapter 3

Methodology

The central hypothesis put forth in this work is that the CGP representation combined with an ACO algorithm that elicits redundancies imparts better evolvability of problem solutions. In our approach, artificial ants traverse paths representing candidate solutions to a given problem guided by pheromone traces stored in pheromone model. The model is updated after each iteration by rewarding the most fit programs of the previous iteration. Such rewarding is achieved by increasing the pheromone level of the entities that yielded the best solutions.

In this chapter we begin by describing the high-level algorithm underlying our approach, here called Ant Colony-Cartesian Genetic Programming (AC-CGP). Then we explain the key methods involved in detail. Next we provide a method of quantifying evolvability by measuring the rate of evolution. To end this chapter we provide a description of the experiments performed to evaluate our system.

3.1 The AC-CGP Algorithm

Algorithm 2 shows a high-level description of the algorithm underlying AC-CGP. As in all evolutionary algorithms, first the problem-dependent system parameters are initialized, *viz.* population size, number of iterations, and the parameters defining a cartesian program (the representation), such as, the inputs, outputs, functions, node connections, number of rows and columns.

The pheromone model is then created of size dependent upon the topology of the graph and initialized with an initial pheromone value. The reasoning behind initializing the pheromone values of all entities to the same value is to ensure all entities have an equal probability of selection.

In every iteration an ant is assigned to create the genotype of an individual by sampling the pheromone model. The individuals are mapped to their equivalent phenotype, evaluated and fitness assigned. Ants are then ranked according to the fitness of the individuals they create. At the end of each iteration only the best ranked ants of the current iteration and the ant which created the best individual (*i.e.* individual of highest fitness) update the pheromone table in accordance to the fitness of individuals they create.

The termination criterion set for our system is the number of iterations set at the start of the run.

Algorithm 2: AC-CGP

```
1:  Select appropriate CGP parameters
2:  Create pheromone table and initialize pheromone values
3:  Set number of ants = n
4:  Set iteration = 0
5:  repeat
6:     for i = 0 → n do
7:        Ant[i] creates a program by sampling the pheromone table
8:        Add individual to current population
9:        Evaluate the individual and calculate its fitness
10:    end for
11:    Rank the ants according to the fitness of individuals they create
12:    if iteration = 0 then
13:       Assign generation-best ant as the best ant
14:    else
15:       if fitness(generation-best ant) ≥ fitness(best ant) then
16:          Assign generation best ant as best ant
17:       end if
18:    end if
19:    The best n- ranked ants update the pheromone table in accordance to Equation 3.3
20:    iteration ← iteration + 1
21: until Termination criteria are met
```

3.2 Cartesian Genetic Programming Representation

As mentioned earlier, in CGP an individual's genotype is an integer string of fixed length that maps to an executable graph. A CGP system requires the representation of the individuals to be done at the start of the run with prior definition of the following parameters $\{\mathrm{G}, n_i, n_o, n_n, \mathrm{F}, n_f, n_r, n_c, \mathrm{l}\}$.

Where G : genotype (fixed set of integers)

n_i : program inputs.

n_o : number of program output connections.

n_n : number of node input connections.

F : node function.

n_f: number of functions.

n_r : number of nodes in each row.

n_c : number of nodes in each column.

l : level back parameter.

A CGP system starts with the declaration and definition of its inputs and functions required to solve a given problem. Each of the inputs and functions are labeled with an integer number. This integer number is used to represent that particular input or function in the genotype of the individual. The topology of the graph is then decided by selecting the appropriate number of rows (n_r) and columns (n_c). The topology of the graph depends on the problem to be solved. A node in the graph can have a maximum of n_n inputs which is typically set to the maximum *arity* of the functions in F. A node can have as its input, an input from the input set (n_i) or the output of a node from a previous column depending upon the level back parameter. The level back parameter l defines the number of previous columns output the current node can take as its input which can be any number from *zero* to a full level back (*i.e.* $0 \leq l < n_c$). A node is not allowed to take the output of another node as its input if the node is in the same column or in a column after it, this is done to avoid cycles

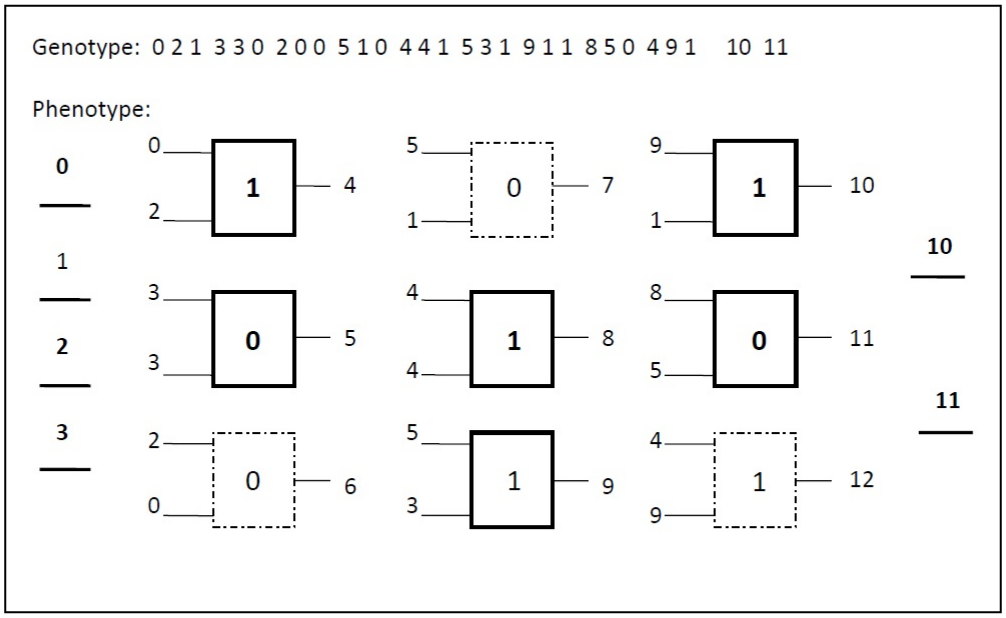


Figure 3.1: CGP: genotype-phenotype mapping

in the graph. These parameters once set at the start do not change for the number of iterations the system is allowed to run. The genotype size is fixed and can be calculated as shown in Equation 3.1.

$$G_{size} = n_r * n_c * (n_n + 1) + n_o \tag{3.1}$$

Figure 3.1 is an example depicting a CGP genotype and genotype-phenotype mapping for a system, where $n_r = n_c = 3$. The system has four inputs $n_i = \{0, 1, 2, 3\}$ and two functions $n_f = 2$, $F = \{0, 1\}$. Each node has $n_n = 2$, number of input connections and each individual has two output nodes $n_o = 2$ (Node 10 and 11 are the output nodes for this example). The system has full level back $l = 2$, *i.e.,* a node can have as its input the output from any of the previous nodes of another column or from the input set n_i. The nodes highlighted in Figure 3.1 are the *used nodes*, *i.e.,* the nodes that contribute to the *fitness* of the individual and the rest are *redundant*. These nodes are termed *redundant* as they are not connected to the output nodes and hence have no say in the *fitness*

Table 3.1: Representation of the pheromone model

Node	Input 1					Input 2					Function				Output Node					
	1.0	**X**	**0**	**1**	**2**	**1.0**	**X**	**0**	**1**	**2**	**+**	**-**	$\times$	$\div$	**1.0**	**X**	**0**	**1**	**2**	**3**
0	τ_d	τ_d	0	0	0	τ_d	τ_d	0	0	0	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d
1	τ_d	τ_d	τ_d	0	0	τ_d	τ_d	τ_d	0	0	τ_d	τ_d	τ_d	τ_d						
2	τ_d	τ_d	τ_d	τ_d	0	τ_d	τ_d	τ_d	τ_d	0	τ_d	τ_d	τ_d	τ_d						
3	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d	τ_d						

of the individual.

3.3 Pheromone Model

Structurally the pheromone model is an array of tables storing pheromone values for each entity in a cartesian program, as shown in Table 3.1. Each table is structurally similar to a CGP genome, *i.e.,* it stores pheromone values for all available inputs and functions needed in generating a node of the program. All pheromone values are initialized at the start to τ_d for all the available inputs and functions for a node. This initialization is done to ensure all entities have an equal probability of selection. The inputs that are unavailable to a node have their pheromone value set to zero to prevent them from being selected and generating syntactically incorrect programs.

After each iteration the pheromone table is updated and each entity (input or function) for every node has its pheromone value either increased or decreased. We employ the strategy used in AS_{rank} [11], where the pheromone values are updated by the best-ranked ants of the iteration.

Table 3.1 shows an example of a pheromone model for a symbolic regression representation. In the example, $n_i = \{1.0, X\}$, $n_f = 4$, $F = \{+, -, \times, \div\}$, $n_n = 2$, $n_r = 1$, $n_c = 4$, $l = 3$, and one output node $n_o = 1$. These inputs and functions are available for all nodes which are initialized to τ_d at the start. The output from each node can be used as input for future nodes, hence nodes have predecessor nodes made available by initializing their pheromone levels to τ_d and unavailable

nodes have their pheromone values set to zero.

3.4 Genome Creation

Each ant creates an individual's genotype by sampling the pheromone model. As mentioned in Section 2.3 the genotype of an individual in a CGP system is a string of integers of size given by Equation 3.1 where each integer represents either an input or function predefined at the start.

We employ the feed forward method in creating the genotype of an individual. The graph is addressed on the Cartesian co-ordinate system, and we begin from the top-leftmost node moving top-to-bottom and left-to-right. The ant samples the inputs and the functions required for each node of the individual. The maximum number of inputs each node is constant and depends upon the maximum *arity* of a function in the function set F and the number of available inputs depends upon the level-back parameter. The last n_o integers of the genotype by convention refer to the output nodes of the individual which can be any of the nodes in the graph or an input.

The ant samples the pheromone table and selects the appropriate input or function available, for each position in the genotype. This is done by converting the pheromone values of available entities to a probability and using a *roulette wheel* selection scheme. The probability p_i that the ant selects a particular input or function i is given by Equation 3.2, where n_n is the number of possible inputs or functions.

$$p_i = \frac{\tau_i}{\sum_{j=1}^{n_n} \tau_j} \tag{3.2}$$

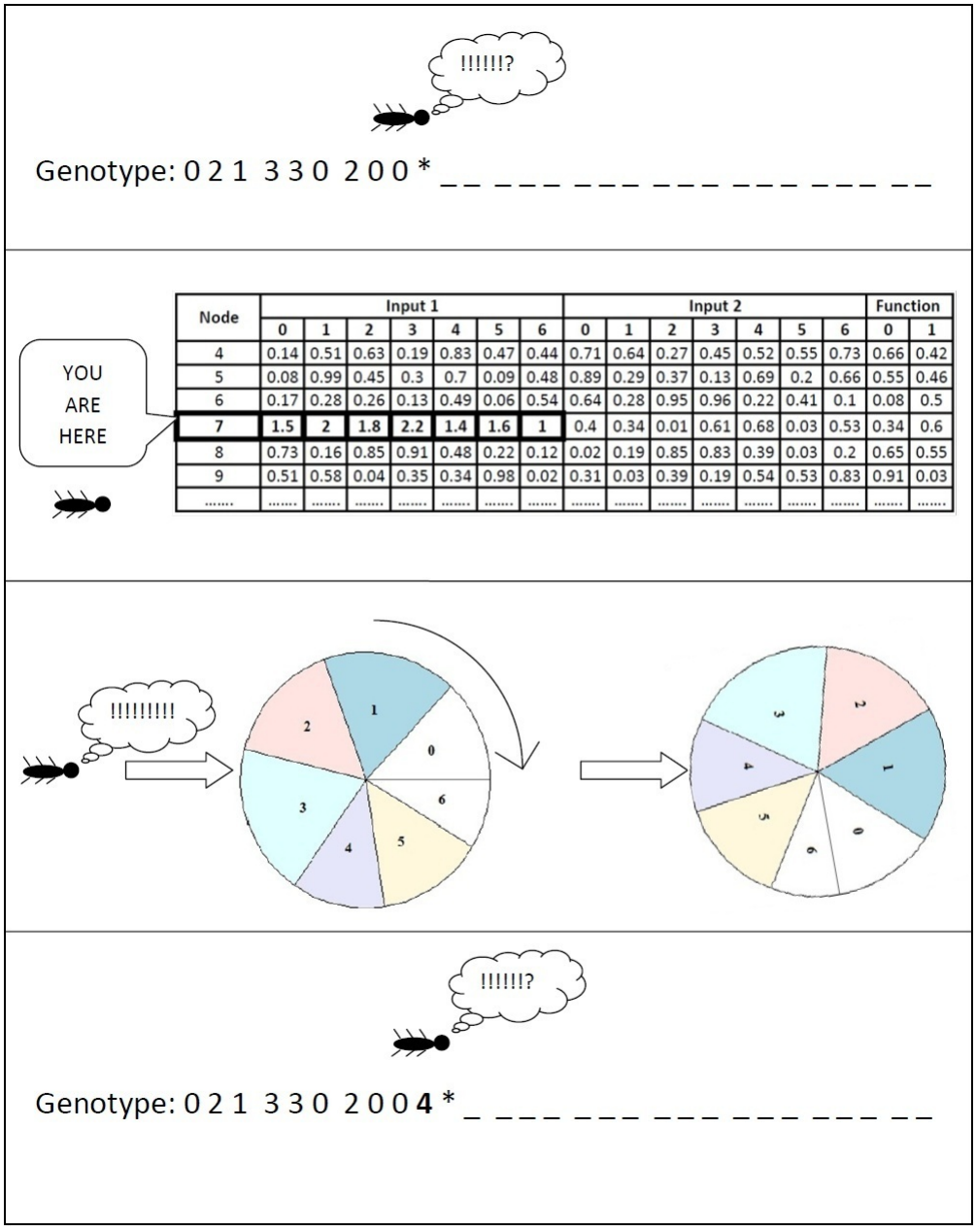
Roulette wheel selection is a selection operator that draws on the concept of a roulette-wheel as used in a casino. Every entity can be imagined to have a portion of space assigned to it on an

imaginary roulette-wheel proportional to its pheromone level. The wheel is spun and the entity on which the roulette-wheel stops is selected. It is imperative that the cumulative probabilities of all entities normalize to 1. The greater the probability of the entity being selected, the larger is the portion on the roulette-wheel assigned to it and the more likely is it for that entity to be selected.

We illustrate how an entity of the genome is selected in Figure 3.2. In that figure an imaginary ant, called "Tiny", is diligently attempting to create a genotype for an individual based on the same system parameters mentioned in Section 3.2. Suppose Tiny has completed selecting the entities for the first three nodes and is in the process of selecting the first input for the fourth node, *i.e.,* Node 7, as the inputs to the system are also labeled as nodes namely Node (0, 1, 2 and 3). So Tiny has arrived at Node 7 and has to select the first input. Tiny looks up the pheromone table and finds his current location. The pheromone values of the entities are converted to a probability according to Equation 3.2 and represented on a roulette wheel. The portion of area an entity occupies on the roulette wheel is proportional to the amount of pheromone, so Node 3 has the larger space on the wheel as it has the largest amount of pheromone. Similarly, Node 6 has the least amount of space allotted to it as it has the smallest amount of pheromone. The wheel is spun and the entity that the wheel lands on is selected which is Node 4 in this case. So Node 4 is selected as the first input to Node 7 and Tiny moves on to complete the rest of the genotype.

3.5 Individual's Evaluation

As mentioned in Section 2.3, there are a number of ways to evaluate an individual. From the mentioned techniques we use the two pass scheme to evaluate an individual. Evaluation is done using two passes. In the first pass, we begin from the output nodes and mark all connected nodes moving towards the input. In the second pass, we execute the connected nodes beginning from the



Node	Input 1							Input 2							Function	
	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1
4	0.14	0.51	0.63	0.19	0.83	0.47	0.44	0.71	0.64	0.27	0.45	0.52	0.55	0.73	0.66	0.42
5	0.08	0.99	0.45	0.3	0.7	0.09	0.48	0.89	0.29	0.37	0.13	0.69	0.2	0.66	0.55	0.46
6	0.17	0.28	0.26	0.13	0.49	0.06	0.54	0.64	0.28	0.95	0.96	0.22	0.41	0.1	0.08	0.5
7	**1.5**	**2**	**1.8**	**2.2**	**1.4**	**1.6**	**1**	0.4	0.34	0.01	0.61	0.68	0.03	0.53	0.34	0.6
8	0.73	0.16	0.85	0.91	0.48	0.22	0.12	0.02	0.19	0.85	0.83	0.39	0.03	0.2	0.65	0.55
9	0.51	0.58	0.04	0.35	0.34	0.98	0.02	0.31	0.03	0.39	0.19	0.54	0.53	0.83	0.91	0.03
.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....

Figure 3.2: Genome Creation

input nodes and move towards the output node.

First Pass: Marking Connected Nodes

Th output nodes from the individuals genotype are identified. As mentioned earlier the output nodes are represented at the end of the genotype. The first output node is identified in the phenotype of the individual and marked as visited. The inputs to the output node are identified as the next nodes to be marked. We then travel backwards identifying the inputs as nodes to be marked and marking them as visited till we reach the program's inputs. We do the same each output node present in the system.

Second Pass: Executing Connected Nodes

After all connected nodes are marked as visited, we start executing them, beginning from the system's inputs moving towards the output nodes. Nodes are executed by identifying the node function and passing to it the node's inputs. The values of all inputs and node outputs are saved. We execute nodes from system inputs moving towards the output node as some nodes take the output of another node as its input. These input values have been calculated and saved. We proceed until all marked nodes are executed. The values from the output nodes calculated are then presented as the individual's output.

3.6 Pheromone Model Update

With a goal of creating better individuals every new iteration, the pheromone table is updated at the end of every iteration. The update is a form of employing a greedy search methodology increasing

the probability of selecting the best entities in the next iteration that were selected in the current iteration by increasing their pheromone values.

As mentioned earlier, we employ the Rank-based Ant System (AS_{rank}) that selects the best ranked ants from the current iteration to update the pheromone model. After every individual of the iteration is evaluated, the ants are ranked according to the quality of the individuals they generate. The greater the fitness of the individual the higher is the ant ranked. Out of the total number of ants in each iteration only the $(n-1)$ *best-ranked ants* and the *best ant*, *i.e.,* the ant that produced the best solution so far (this ant could be from the current iteration or from a previous iteration) update the pheromone table.

The pheromone table is updated according to Equation 3.3.

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{r=1}^{n-1} (w - w_r)\Delta\tau_{ij}^{r}(t) + w\Delta\tau_{best} \tag{3.3}$$

where:

- $\tau_{ij}(t)$ is the current pheromone level of input(or function) i at node j at iteration t;
- w is a constant weight assigned at the start of the experiment;
- r is the rank of the ant;
- $w_r = w/(n-r)$;
- $(w - w_r)$ is the weight calculated such that higher ranked ants deposit a larger quantity of pheromone;

- $\Delta\tau^r_{ij}(t) = \begin{cases} \text{fitness of } r\text{th-best ant}, & \text{if ant selects input or function } i \text{ at node } j. \\ 0, & \text{otherwise.} \end{cases}$

- $\Delta\tau_{best} = \begin{cases} \text{fitness of best ant}, & \text{if best ant selects input or function } i \text{ at node } j. \\ 0, & \text{otherwise.} \end{cases}$

3.7 Measuring Rate of Evolution

The nonsynonymous to synonymous substitution ratio k_a/k_s is a concept used to measure genetic substitution in molecular biology. This measurement has been used in [1] to quantitatively assess evolvability in Linear Genetic Programming(LGP). A change in the genome of an individual that brings about a change in the phenome with a change in its fitness is known as a *nonsynonymous* change, whereas if the phenome of an individual remains the same along with its fitness it is called a *synonymous* change. We use the same terminology and a similar approach in the work presented here. One key difference, is that in the context of Ants (and Estimation of Distribution Algorithms) changes are brought about by probabilistic sampling of the pheromone table rather than by the application of genetic operators, which is the case in [1].

In our simulated studies we measure the k_a/k_s ratio by observing the changes brought about in the *best-ranked* ants. To determine the value of nonsynonymous (and synonymous) change we compare each individual I and individual J that were brought about by the N best-ranked ants of iteration t and $t-1$, respectively. The value of nonsynonymous change $m^I_{ak}(t)$ on each gene k of each individual I from iteration t is calculated as follows: if gene k did not change, *i.e.,* $I_k = J_k$, then $m^I_{ak}(t) = m^I_{sk}(t) = 0$. Otherwise, if change was *silent*, *i.e.,* individuals I and J have same fitness, then $m^I_{ak}(t) = 0, m^I_{sk}(t) = 1$. If change was not silent, then $m^I_{ak}(t) = 1, m^I_{sk}(t) = 0$.

We compute the number of nonsynonymous substitutions $M_a(t)$ and the number of synonymous

substitutions $M_s(t)$ as follows:

$$M_a(t) = \sum_{i=1}^{N} m_{ak}^i(t), \quad M_s(t) = \sum_{i=1}^{N} m_{sk}^i(t)$$

Like in [1], we keep a record of all changes to each gene during the iterations of the algorithm. We compute such accumulated numbers of nonsynchronous $c_{ak}(t)$ and synchronous $c_{sk}(t)$ changes in gene k up to iteration t, as follows: initially $c_{ak}(0) = c_{sk}(0) = 0$, for all genes k in the genome. We update these values for each gene k and individual I brought about by the N best-ranked ants of iteration t, as follows:

$$c_{ak}(t) = c_{ak}(t-1) + m_{ak}^I(t), \quad c_{sk}(t) = c_{sk}(t-1) + m_{sk}^I(t)$$

And we compute the *potential* of a gene k being changed nonsynonymously or synonymously (also called the *sensitivity* of a gene) as follows:

$$n_{ak}(t) = \frac{c_{ak}(t)}{c_{ak}(t)+c_{sk}(t)}, \quad n_{sk}(t) = \frac{c_{sk}(t)}{c_{ak}(t)+c_{sk}(t)}$$

We add up the sensitivities of all genes in the representation to obtain the total nonsynonymous and synonymous sensitivities $N_a(t)$ and $N_s(t)$.

$$N_a(t) = \sum_{k=1}^{N} n_{ak}(t), \quad N_s(t) = \sum_{k=1}^{N} n_{sk}(t)$$

Finally, we compute the nonsynonymous and the synonymous substitution rates k_a and k_s of iteration t as

$$k_a(t) = M_a(t)/N_a(t), \quad k_s(t) = M_s(t)/N_s(t)$$

which enables us to obtain the rate of evolution R_e in iteration t:

$$R_e(t) = k_a(t)/k_s(t) \tag{3.4}$$

3.8 Measuring Diversity

A key ingredient in the success of any evolutionary algorithm is the ability to maintain diversity in the population. Loss of diversity in the early stages of the run usually results in the system converging to a local optimum. Diversity in a system is influenced mainly by the system parameters like population size, genetic operators and the learning mechanism [32].

In [33], Shapiro proposed a method to measure diversity loss in EDAs. His method takes into account the empirical frequency of occurrence of components in an individual. We use this method to measure diversity in our system as given by the equations below:

$$\nu_i^A = \frac{1}{N}\sum_{\mu}\delta(x_i^\mu = A) \tag{3.5}$$

where:

- ν_i^A is the empirical frequency at which component i takes value A.
- N is the population size.
- δ is an indicator function; 1 if TRUE, 0 if FALSE.
- x_i^μ is component i of population member μ.

$$\upsilon = \sum_i \frac{1}{|A|} \sum_a \nu_i^a (1 - \nu_i^a) \tag{3.6}$$

where:

- υ is the diversity measure.
- $|A|$ is the number of values component x_i can take.

3.9 Experimental Design

We begin by testing how effectively the Rate of Evolution (R_e) reflects the dynamics of the underlying algorithm. Then we test how our AC-CGP system behaves under different environmental conditions. Finally, we compare the evolvability of our method with that of Cartesian Ant Programming [26].

In Sections 3.9.1, 3.9.2 and 3.9.3 we detail the experimental setup for these three tests.

3.9.1 Rate of Evolution Under Different Parameter Settings

In this section we describe the experimental setting we have designed to study the influence of different parameter settings on factors related to the rate of evolution in a symbolic regression problem.

In this experiment the target expression is the polynomial $x^4 + x^3 + x^2 + x$. The training set, *i.e.,* the fitness cases, consists of 40 equidistant example points in the interval $[-2.0, +2.0]$. Fitness of an

individual is computed as the inverse of the accumulated error between the actual example points and the values output by the individual. Formally, let the output of the ith training example be o_i. Let the output of individual g on the ith example from the training set be g_i. Then, for a training set of $n = 40$ examples the fitness f_g of g is calculated as follows:

$$f_g = \frac{1}{1 + \sum_{i=1}^{n} |o_i - g_i|} \tag{3.7}$$

The parameters chosen for this experiment are shown in Table 3.2.

3.9.2 Varying Environmental Scenarios

The environment is defined by the target our system is set out to achieve. We simulate three different environmental scenarios i) random evolution, where no target is defined, ii) fixed target, where the target is set at the start of the iteration and does not change for the entire run, iii) moving target, where the target changes at different stages of the run.

The parameters chosen for the experiments are shown in Table 3.2.

Random Evolution (RE)

Random evolution in the context of this work means that there is no target or a *fitness* for our system to converge upon. At the end of every iteration we randomly select n ants and label them the *n-best ranked ants* of the iteration. We use these *n best-ranked ants* to introduce variation in the *pheromone table* according to the evaluation of the expression that each individual's phenotype maps.

Table 3.2: Parameter values for experiments SR, RE, FTE, and MTE

Parameter	**Experiment**			
	SR	**RE**	**FTE**	**MTE**
n_i	$\{x, 1.0\}$	ditto	ditto	ditto
F	$\{+, -, \times, \div\}$	ditto	ditto	ditto
Number of Runs	100	ditto	ditto	ditto
Number of Iterations	1000	ditto	ditto	ditto
Number of Individuals	50	50	100	100
τ_d	1.0	ditto	ditto	ditto
Number of best-ranked ants	10	ditto	ditto	ditto
# of equidistant fitness cases	40	ditto	ditto	ditto
Interval	$[-2.0, 2.0]$	ditto	ditto	ditto

Fixed Target Evolution (FTE)

In this experiment, we set our target as the expression $x^5 - 2x^3 + x$. After every iteration the n *best-ranked ants* update the pheromone table in accordance to their fitness.

Moving Target Evolution (MTE)

In the moving target experiment, we have a moving target which is designed by increasing the degree n of the polynomial $\sum_{i=1}^{n} x^i$. In this experiment as the target keeps changing it becomes difficult for the system to converge to an optimal solution given the few iterations we let it run. The purpose for this experiment is to study the adaptiveness of our system to change.

We start our target for the first 200 iterations at $i = 3$ where the target expression is $x^3 + x^2 + x$. From iteration 200 to 500 $i = 4$ and iteration 500 onwards $i = 5$. We use an enhanced version of the AS_{rank} update method that includes evaporation. We incorporate evaporation to balance the effects of depositing a large amount of pheromone when the target is small which hinders exploration of alternative paths when the target is changed. For this experiment we set the evaporation rate to 0.2.

3.9.3 AC-CGP vs CAP

Our approach differs from Cartesian Ant Programming (CAP) [26] in two key aspects: the learning algorithm and the method of updating the pheromone model.

CAP uses Max-Min Ant System (MMAS) [13] as the learning algorithm, where the *best ant* updates the pheromone model at the end of the iteration and the unused pheromone trails are subjected to evaporation. In MMAS, the pheromone model is updated according to the equation below:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}^{best}(t) \tag{3.8}$$

where:

- $\tau_{ij}(t)$ is the current pheromone level of input(or function) i at node j at iteration t;
- ρ is the evaporation rate;
- $\Delta\tau_{ij}^{best} = \begin{cases} f_{best-ant} & \text{if } \textit{best ant} \text{ uses that path } ij. \\ 0 & \text{otherwise.} \end{cases}$

In regards to the method of updating the pheromone model, after evaluating the individual, CAP traverses the model beginning from the output nodes and proceeds backwards towards the input nodes, only updating the model for the nodes that appear in the individuals phenotype. As such, only the pheromone values of the used nodes are updated. In our method, however, we update the pheromone model in a forward manner beginning from the input nodes and moving towards the output nodes, updating the pheromone values for all nodes that appear in the individual's genotype.

Previous works have shown that the ability to capture node redundancies is of great importance for the evolvability of problem solutions [34, 35]. Therefore, the hypothesis we aim to verify

is that our approach indeed presents better evolvability than CAP. To verify this hypothesis we compared the evolvability of our approach versus CAP in the symbolical regression mentioned in Section 3.9.1 and in the moving target experiment mentioned in Section 3.9.2. We compare results obtained for both setups for the Fixed Target Evolvability (FTE) and Moving Target Evolvability (MTE) experiments. The parameters chosen for this experiment are given in Table 3.2. We used an evaporation rate of $\rho = 0.1$ in these particular experiments.

Chapter 4

Results and Discussion

4.1 Single Evolution Run (SE)

In this experiment we study the evolution of our system in tackling the symbolic regression expression $x^4 + x^3 + x^2 + x$ for a single successful run. We run the experiment with 50 individuals and observe how our system evolves at different stages of the run.

In our experiment, a successful run is one that has reached the target fitness within the number of iterations. We choose one of several successful runs to best understand the evolution process.

We choose a run in which at least 90% of the individuals reach the target fitness within the maximum number of generations. Figure 4.1(d) shows the best fitness and average fitness of the run, where the maximum fitness of an individual is 1.0. The fitness of an individual is calculated according to Equation 3.7. Analyzing the results for average fitness in Figure 4.1(d), we take note that from iteration 400-600 the system approaches maximum fitness and from iteration 600 on the system converges to the target with maximum fitness.

Figure 4.1(a) shows that at the start of the run the nonsynonymous rate (k_a) is high and gradually decreases as the run proceeds. This implies that variability is a driving force at the start of a run as the system creates individuals of better fitness than the previous iteration. This rate gradually decreases as the system is able to create a higher number of individuals with maximum fitness and

no more beneficial nonsynonymous changes are possible.

Figure 4.1(b) shows that at the start of the run the synonymous rate is low as the system imposes variation to be a driving force in creating highly fit individuals. As the system creates individuals of maximum fitness and converges to a solution, the number of synonymous changes increases to a maximum peak of the run. Neutral changes become a dominant factor and serves a purpose of preserving the accepted nonsynonymous changes. As the run continues the rate of synonymous changes continues to be high.

These changes are reflected in the rate of evolution R_e as depicted in Figure 4.1(c). R_e is high at the start of the run and gradually decreases as the run continues. R_e at its maximum value during the run remains slightly about 1.0 which is similar to what one observes in natural evolution.

4.2 Rate of Evolution Under Different Parameter Settings

In this section we describe the experiments we have designed to study the influence of different parameter settings on factors related to the Rate of Evolution. In each experiment we change one parameter and keep the remaining parameters the same. We plot the measures of k_a, k_s, R_e and the average fitness to study how changes on a given parameter influence these factors. The exponentially weighted moving average method, with a smoothing factor 0.1, is used to smooth the curves.

4.2.1 Population Size

In this experiment we alter the population sizes to 50, 100 and 200. Having a larger population size increases the probability that the system will better exploit the search space and is more acceptable

to adaptive change, which would imply that the system converges to an optimal solution faster than with a smaller population size. This is reflected in our results found in Figure 4.2(d). Observing the results of the substitution rates (Figure 4.2(a) and 4.2(b)) and rate of evolution (Figure 4.2(c)), we see they follow a similar trend.

The nonsynonymous substitution rate with a higher population size is at its maximum for a fewer number of iterations than that with a smaller population size, as seen in Figure 4.2(a). The nonsynonymous substitution rate continues to gradually decrease as the system converges to an optimal solution, canceling out the deleterious and harmful substitutions that adversely effect the system's convergence of the solution. The synonymous substitution rate is at a run time low at the start of the run and continues to increase as the importance of generating synonymous changes increases as the run progresses. The synonymous substitution rate is at a run time maximum value when the system converges to the optimal solution with maximum fitness. We understand this to be a protective measure of the system to preserve the generated solution. The system continues to accept synonymous changes and reject deleterious nonsynonymous changes as the run progress.

The effect of these substitution rates is reflected on the rate of evolution R_e in Figure 4.2(c). As the run progresses the value of R_e continues to descend until it reaches *zero* evolution. Zero evolution implies that the system is incapable of imposing any more adaptive changes and continues to create very similar individuals.

4.2.2 Number of *ranked-ants*

In this experiment, we test our system with different number of *ranked-ants* 5, 10 and 20 (Figure 4.3) that update the pheromone model at the end of each iteration. Our results found in Figure 4.3(d), show that for this experiment for the different number of *ranked-ants* the rate of conver-

gence, *i.e.,* the average fitness over the number of iterations is similar. This result reflects and supports the argument made in [36] which states that given a fixed number of ant tours per trail the number of optimal solutions found is independent of the number of ants.

Observing the nonsynonymous substitution rates in Figure 4.3(a), we take note that the system with fewer number of ranked ants, *i.e.,* 5 and 10 need a comparatively fewer number of iterations to generate the necessary adaptive changes needed to converge to an optimal solution. Also with 20 ranked ants the number of nonsynonymous changes throughout the run continues to be higher than that with 5 and 10 ranked ants. The system is more open to accept changes with a larger number of ranked ants as they do not adversely affect the average fitness of the system.

The synonymous substitution rate follows a similar trend for all the different number of ranked ants, where it continues to be at a minimum low at the start of the run and a maximum high when the system converges to the solution. The rate of evolution R_e shows us that with a fewer number of ranked ants 5 and 10, the system reaches *zero* evolution quicker following the scheme of natural evolution with R_e at the maximum of the run being close to 1.0.

4.2.3 Initial pheromone level

In this experiment we study the influence of the initial pheromone value on the rate of evolution R_e. The initial pheromone level is set to 0.1, then changed to 0.5 and 1.0. We analyze the effect on substitution rates, rate of evolution and average fitness.

From Figure 4.4(d) we see that the system having a lower initial pheromone level converges to a solution quicker. Having a lower value of τ_d the chances of selection of an unused node or an unused entity at a node decreases faster than with a higher value of τ_d. Our results reflect the findings in [25].

The effect of the initial pheromone level on the rate of evolution R_e can be analyzed from the substitution rates (Figure 4.4(a) and 4.4(b)) and the rate of evolution (Figure 4.4(c)). With a lower initial level of pheromone level the system needs fewer number of iterations to make adequate adaptive changes in order to converge to the solution. Observing R_e in Figure 4.4(c) in all conditions the maximum value is around 1.0 and proceeds towards *zero* evolution. With a lower initial pheromone level *zero* evolution is reached quicker than with a setup where the initial pheromone level is higher.

4.3 Rate of Evolution Under Different Environmental Conditions

4.3.1 Random Evolution (RE)

Analyzing the nonsynonymous rates in Figure 4.5(a) we see the rate is high, around 1.0, at the start of the run, and lowers down to 0.65 by the end of the run. This we understand is because the artificial ants start selecting the same entities at various parts of the genotype due to random fluctuations. This results in additional pheromone being deposited on those entity paths causing them to be repeatedly selected over future iterations.

The synonymous rates remain consistent, spread between 0.75 and 1.5. Throughout the run the synonymous substitution rates are higher than the nonsynonymous substitution rates which results in R_e remaining under 1.0 as seen in Figure 4.5(c). This implies that when the system has no target neutrality plays a marginally greater influence than the variability in the system.

4.3.2 Fixed Target Evolution (FTE)

From Figure 4.6(d) we observe the system reaches the target with maximum fitness after iteration 700. Observing the nonsysnonymous and synonymous substitution rates, plotted in Figures 4.6(a) and 4.6(b), respectively, we note that the nonsynonymous substitutions dominate the synonymous substitutions at the start of the run and the synonymous substitutions dominate the nonsynonymous substitutions after the system converges to an optimal solution and remains high until the end of the run. We can infer from the k_a/k_s ratio from Figure 4.6(c) that variability is the dominant factor at the start of the run when the system is aiming to reach the target. Once the system attains the target neutrality takes over and plays an important role in preserving the target that has been met. The k_a/k_s ratio eventually reaches *zero* evolution towards the end of the run.

4.3.3 Moving Target Evolution (MTE)

We run this experiment for 100 runs and plot the relevant measures in Figure 4.7. The accepted changes brought about in an individual that change the individual's phenotype known as the non-synonymous substitutions is depicted in Figure 4.7(a) and the accepted changes that does not alter the individual's phenotype, *i.e.,* the synonymous substitutions is depicted in Figure 4.7(b). In Figure 4.7(d) we plot the average fitness over the run. We see for all targets the system reaches the solution of maximum fitness.

When the target is changed we take note of the drastic changes brought about by the system in reaching the target. As shown in Figure 4.7(a), the system imposes a high level of variability in order to create individuals that meet the required fitness. Figure 4.7(b) shows that after the system has attained maximum fitness neutrality becomes a dominant part. As we can see in Figure 4.7(c) the ratio of ka/ks is greater than 1 when the target is changed, stabilizes to around 1 as the required

fitness is met and proceeds towards *zero* evolvability for the remainder of the iterations until the target is changed.

4.4 AC-CGP vs CAP

4.4.1 Fixed Target Evolution (FTE)

In this experiment we evolve the expression $x^5 - 2x^3 + x$ and plot our results in Figure 4.8. Analyzing our results found for rate of convergence in Figure 4.8(d), we take note that our approach for updating the pheromone model attains maximum fitness faster than the approach used in CAP. Updating the pheromone model in the manner of CAP results in faster convergence at the start of the run where only the nodes that contribute to the fitness of an individual have their pheromone levels positively altered in the model. The system is slower to attain maximum fitness towards the end of the run. In our approach we take note of the steady convergence rate throughout the run which results in attaining maximum fitness faster than the other approach.

Figure 4.8(a) shows that our method engenders a high number of nonsynonymous substitutions from the start of the run, and that number increases after the system converges to a solution. Using the method of updating the pheromone model as described in CAP, the system fixates on making a maximum number of nonsynonymous substitutions at the start of the run which results in the synonymous substitutions to be low at the start of the run and increase as the system converges to a solution of maximum fitness. In Figure 4.8(b) we plot the synonymous substitutions which shows us that using the AC-CGP approach results in making a larger number of substitutions throughout the run which increases after the system converges to a solution. The method used in CAP involves making a fewer number of substitutions compared to the AC-CGP approach, the

system makes most of the substitutions after the system has converged to a solution. The result of the substitutions is seen in the rate of evolvability R_e in Figure 4.8(c), the AC-CGP approach has a higher rate of R_e throughout the run where the system converges to *zero* evolvability towards the end of the run. Using the pheromone update method as used in CAP the system has a higher value of R_e at the start of the run and reaches *zero* evolvability quicker than the AC-CGP approach.

4.4.2 Moving Target Evolution (MTE)

In this experiment we create a moving target by increasing at regular intervals the degree n of the polynomial shown below:

$$\sum_{i=1}^{n} x^i$$

We start our target for the first 200 iterations at $i = 3$ where the target expression is $x^3 + x^2 + x$. From iteration 200 to 500 $i = 4$ and iteration 500 onwards $i = 5$.

We use an enhanced version of the AS_{rank} model update method that includes evaporation. We incorporate evaporation to balance the effects of depositing a large amount of pheromone when the target is small which hinders exploration of alternative paths when the target is changed. For this experiment we set the evaporation rate to $\rho = 0.2$.

We emulate CAP's methodology using MMAS as the learning algorithm and with an evaporation rate of $\rho = 0.1$.

In Figure 4.9(d) we plot the average fitness for both systems. We take note that the convergence of the solution is almost identical for both systems. Analyzing the substitution rates in Figure 4.9(a) and 4.9(b) we see that both systems follow a similar trend in evolvability from iteration 0

to 200 where the target is an expression of a lower degree. From iteration 200 to the end of the run, as the target is changed and the degree of the polynomial keeps increasing, we notice that the rate of nonsynonymous substitutions in CAP is greater than in AC-CGP. Also the number of synonymous substitutions made during this period in CAP is low compared to AC-CGP. Moreover, Figure 4.9(d) shows that immediately after the target change AC-CGP is able to produce solutions with higher fitness than CAP, which makes one wander if the AC-CGP pheromone model elicits a more diverse genotype than the CAP pheromone model. Figure 4.9(e) shows that there is strong evidence pointing in that direction. Genotype diversity, in this case, was calculated using the diversity measure introduced by Shapiro in [33] (Section 3 of that paper). For easy of reference, we reproduce Shapiro's formulation below.

Some systems that incorporate the MBS approach to EC strategies fail at finding an optimal solution due to loss of diversity in the system. Loss of diversity makes the system fixate on selecting the same entities over and over which results in failure to escape a local optimal solution reached. Shapiro used the empirical frequency of occurrence of entities in the individuals in the population to calculate diversity. First one calculates the empirical frequency ν_i^A at which component i takes value A.

$$\nu_i^A = \frac{1}{N} \sum_{\mu} \delta(x_i^\mu == A)$$

where:

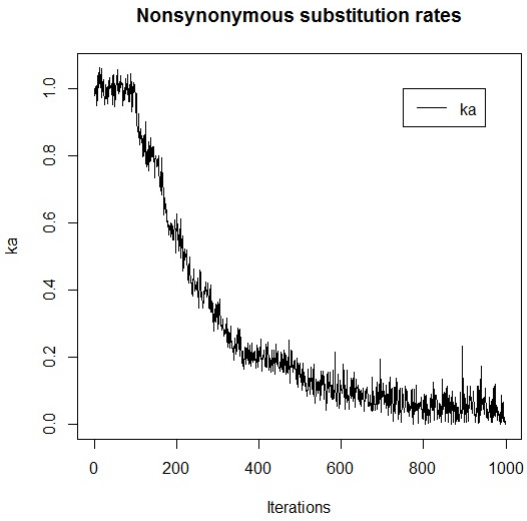
- N is the population size.
- $==$ denotes equality.
- δ is an indicator function; 1 if argument is true, 0 otherwise.

- x_i^{μ} is component i of population member μ.

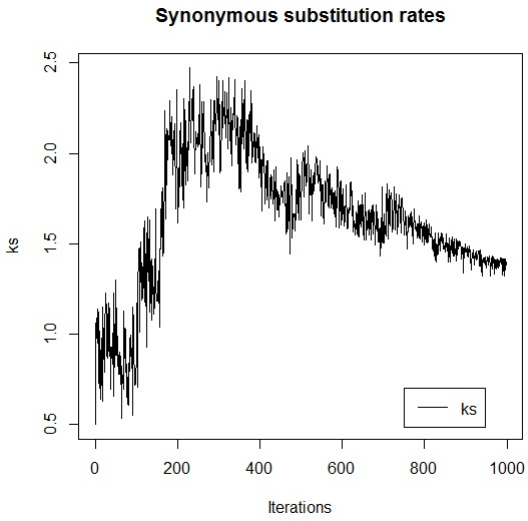
Then the diversity measure ν is defined as follows:

$$\upsilon = \sum_i \frac{1}{|A|} \sum_a \nu_i^a (1 - \nu_i^a),$$

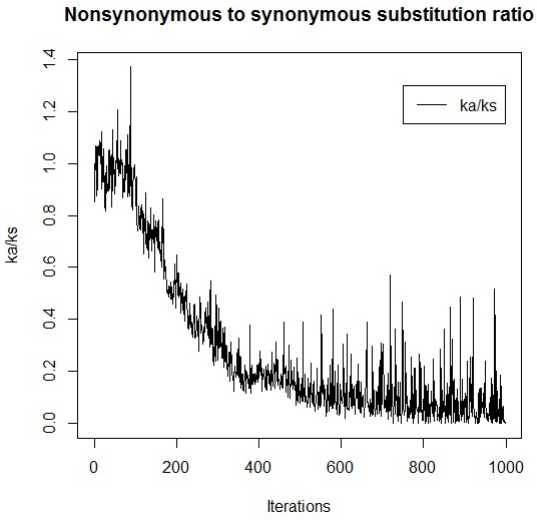
where $|A|$ is the number of values component x_i can take.



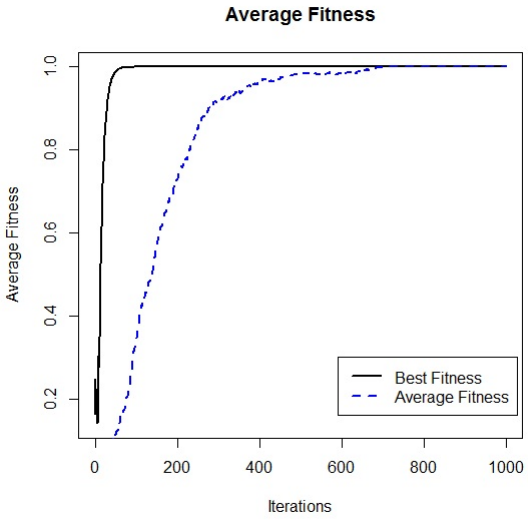
(a) k_a



(b) k_s

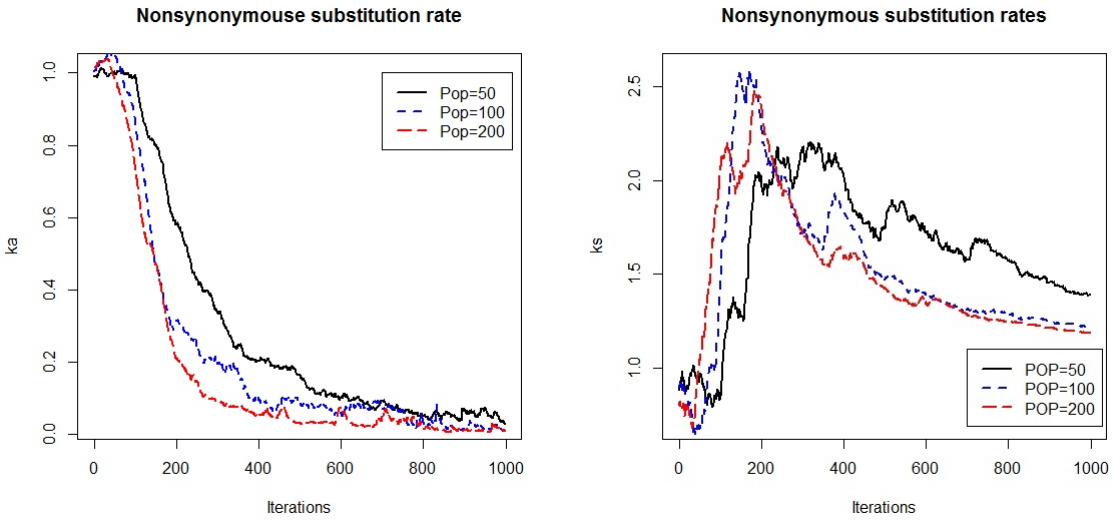


(c) $R_e = k_a/k_s$



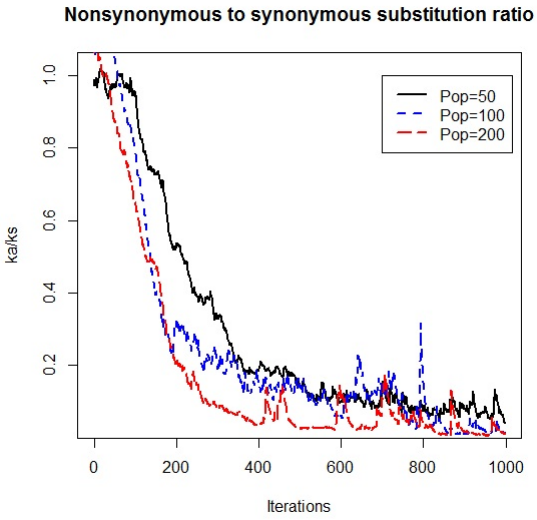
(d) Average fitness

Figure 4.1: A single evolution run

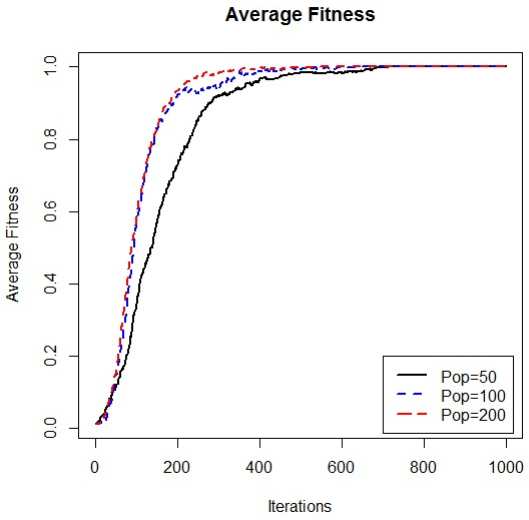


(a) k_a

(b) k_s

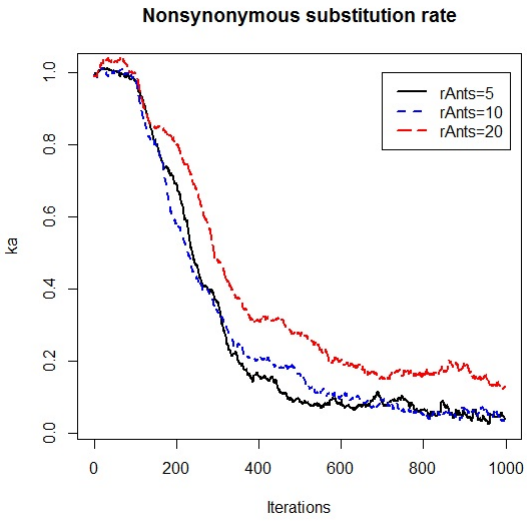


(c) $R_e = k_a/k_s$

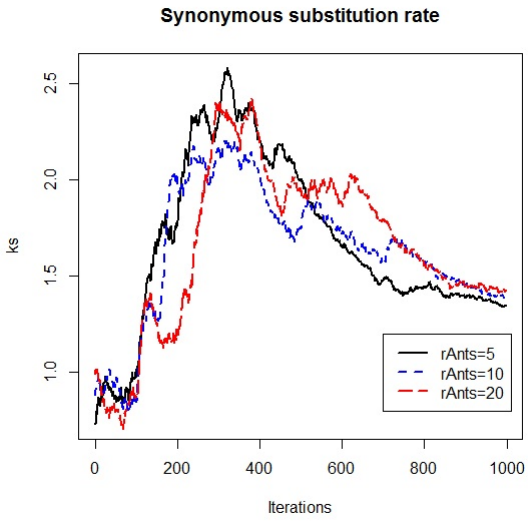


(d) Average fitness

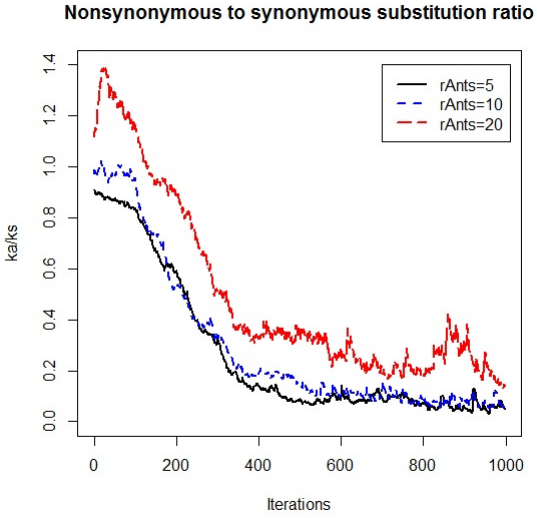
Figure 4.2: Evolution with different population sizes



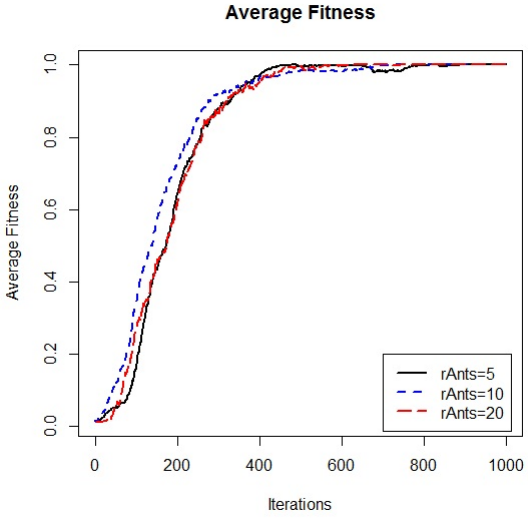
(a) k_a



(b) k_s

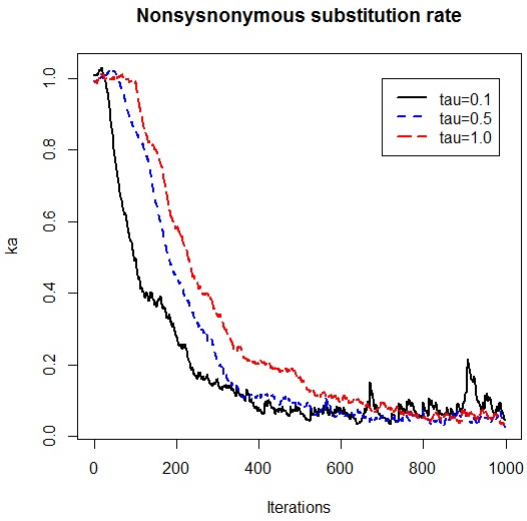


(c) $R_e = k_a/k_s$

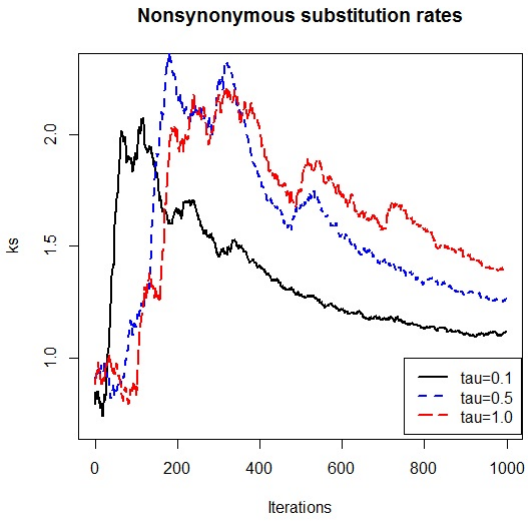


(d) Average fitness

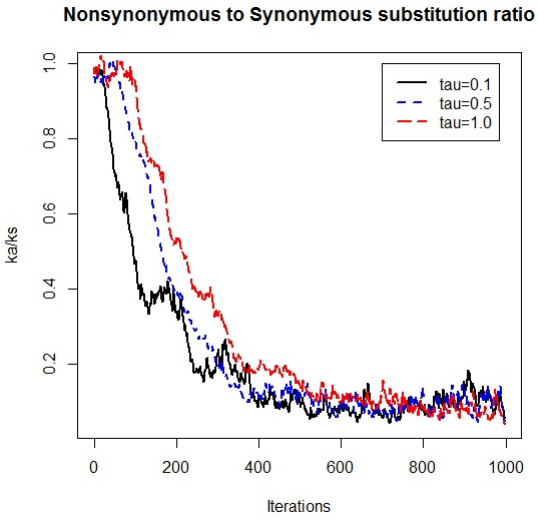
Figure 4.3: Evolution with different number of *ranked-ants*



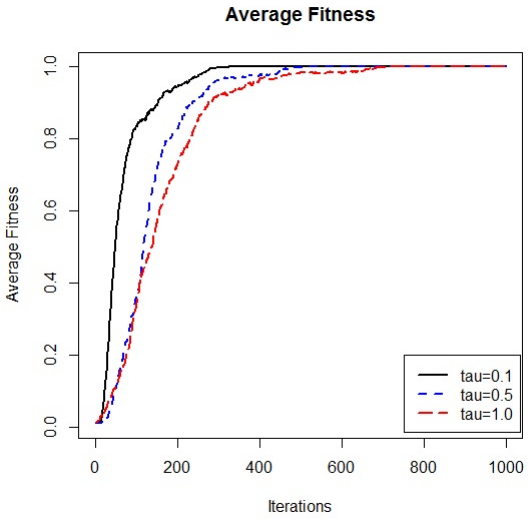
(a) k_a



(b) k_s

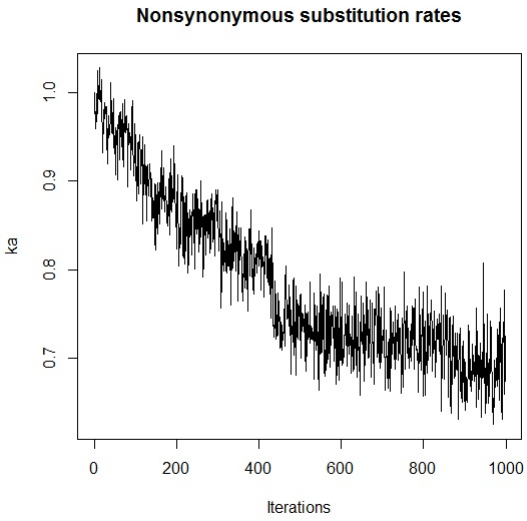


(c) $R_e = k_a/k_s$

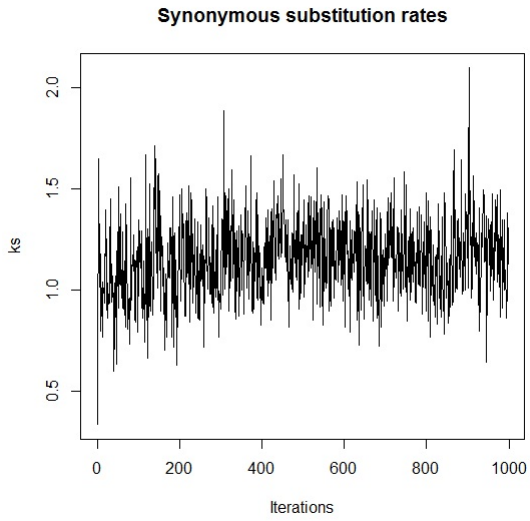


(d) Average fitness

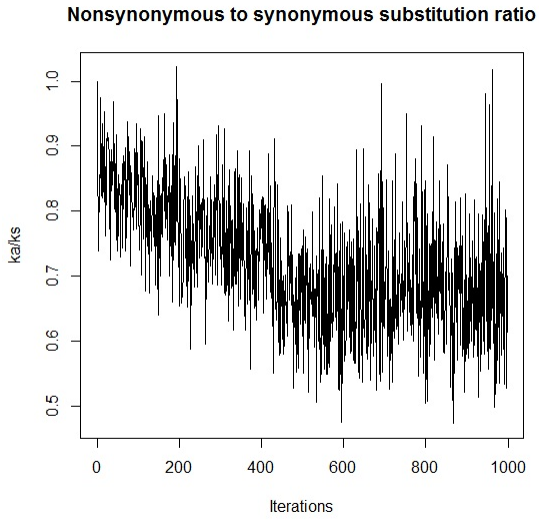
Figure 4.4: Evolution with different number levels of initial pheromone



(a) k_a

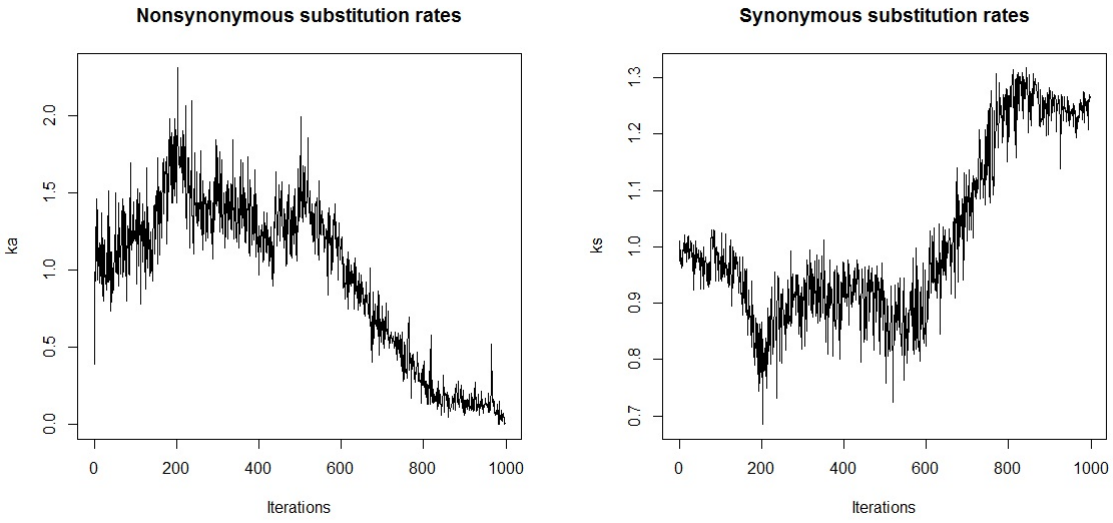


(b) k_s



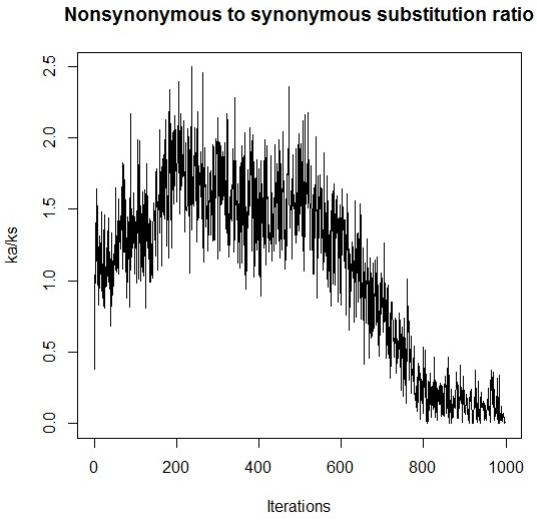
(c) $R_e = k_a/k_s$ ratio

Figure 4.5: A random evolution run

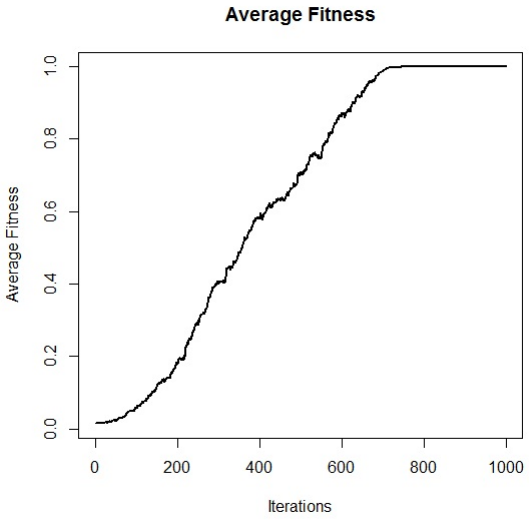


(a) k_a

(b) k_s

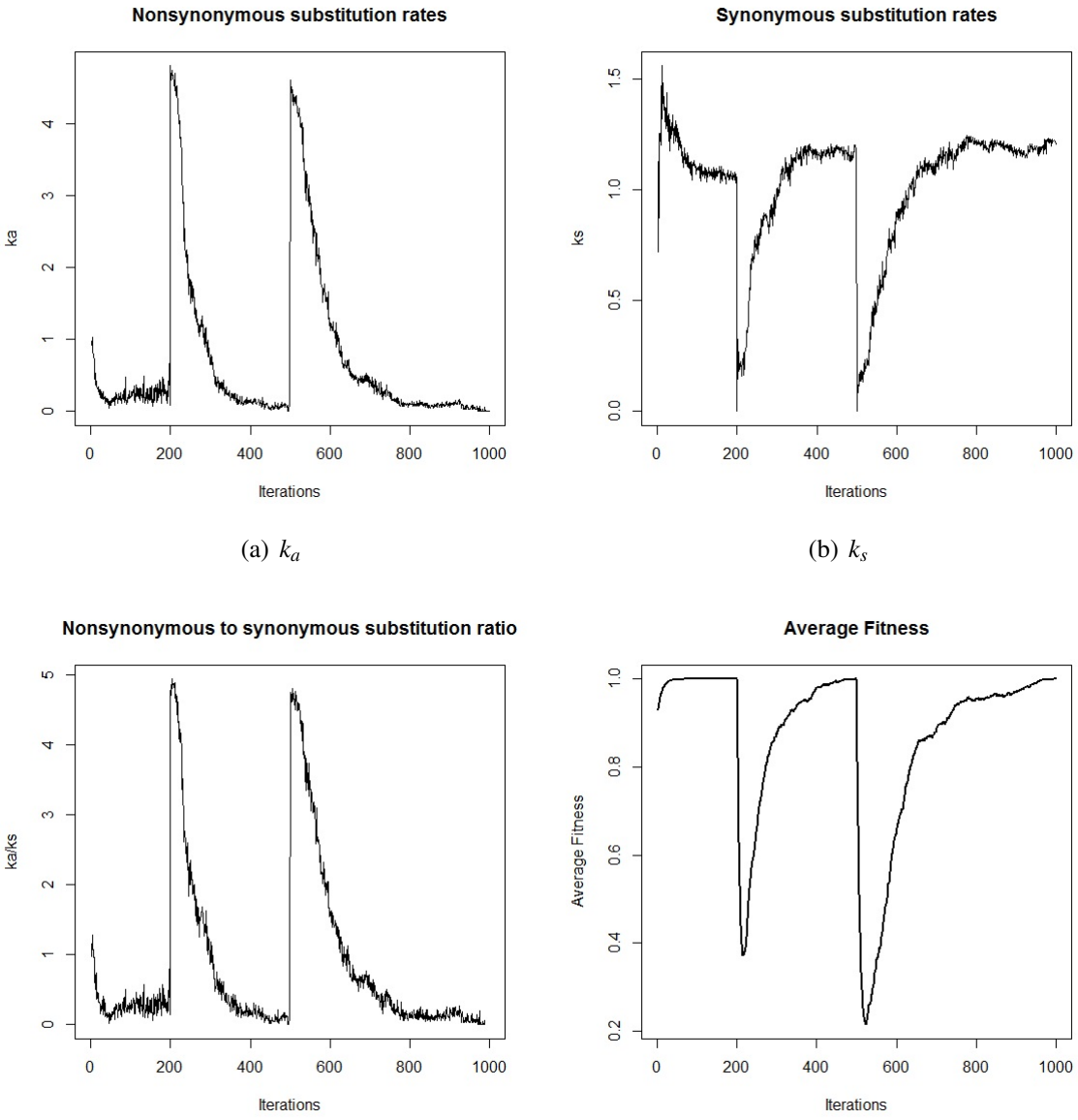


(c) $R_e = k_a/k_s$



(d) Average fitness

Figure 4.6: A fixed target evolution run



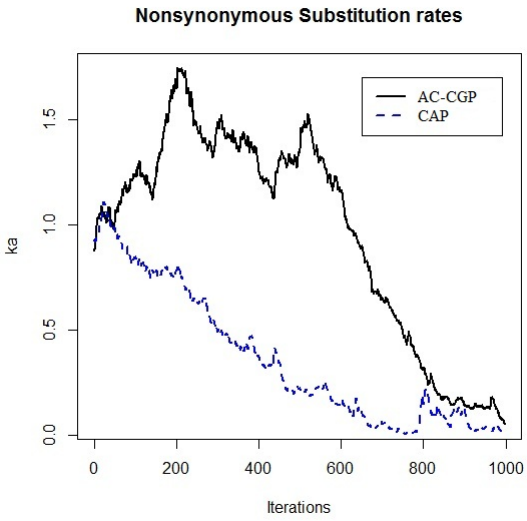
(a) k_a

(b) k_s

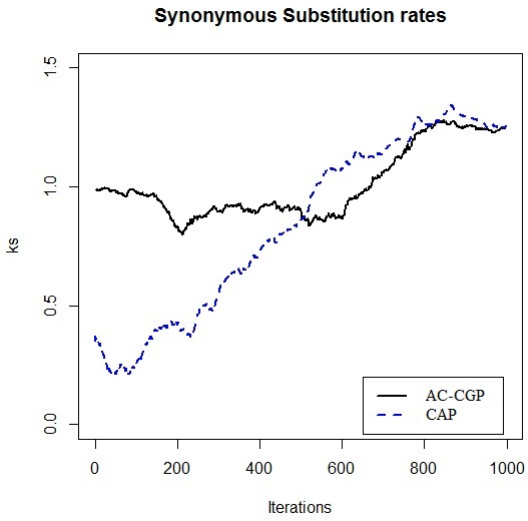
(c) $R_e = k_a/k_s$

(d) Average Fitness

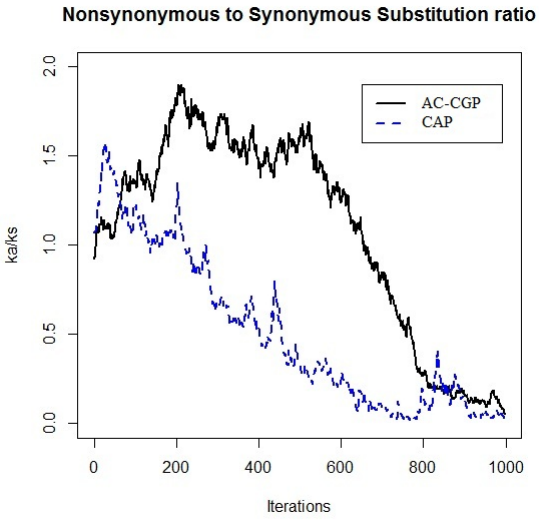
Figure 4.7: A moving target evolution run



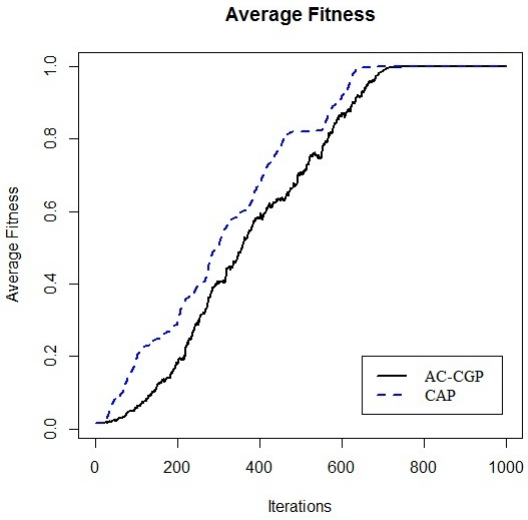
(a) k_a



(b) k_s

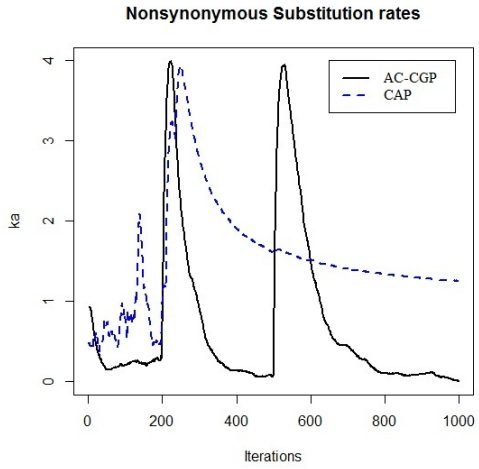


(c) $R_e = k_a/k_s$

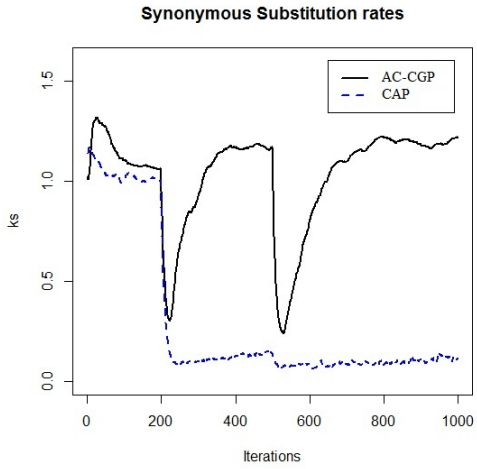


(d) Average fitness

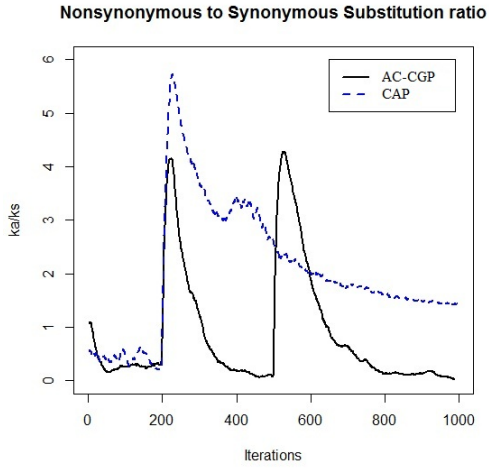
Figure 4.8: FTE: AC-CGP vs CAP



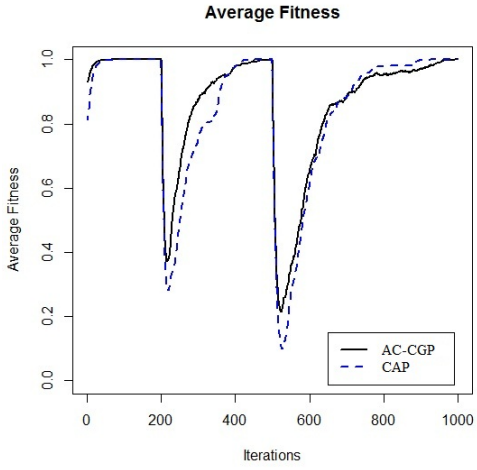
(a) k_a



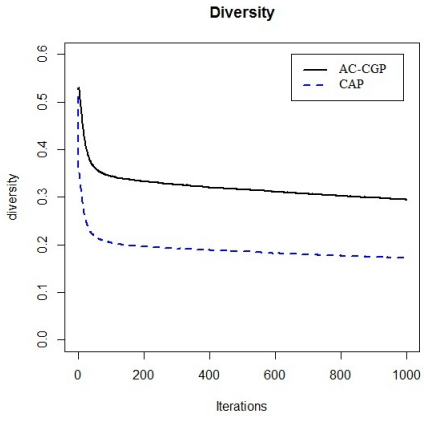
(b) k_s



(c) $R_e = k_a/k_s$



(d) Average Fitness



(e) Diversity

Figure 4.9: MTE: AC-CGP vs CAP

Chapter 5

Conclusions and Future Work

In this work we introduced a hybrid optimization algorithm that combines Cartesian genetic programming with ant colonies. We extended the measurement for rate of evolution used in [1] to evolutionary computation methodologies that make use of model-based search as the learning scheme. We tested the evolvability of our system under the three different environmental conditions, different parameter settings and compared it with CAP in two of the environmental conditions.

We set up the experiment to vary the different parameters of the system to observe the influence of these parameters on convergence and the evolution of the system. In regards to population size, we observe that the system with the population size of 200 converges to a solution of maximum fitness faster and the system needs to introduce a comparatively lesser rate of variability. Whereas the system with a population size of 50 takes longer to converge to a solution and the system needs to introduce a higher rate of variability for the system to reach a solution. This is identified with most EC strategies where a system with a larger population better explores the search space and converges to a solution faster [37]. In regards to the effect of varying the number of ranked ants from 5 to 10 and to 20 to update the pheromone table, we observe that the rate of convergence is independent of the number of ranked ants updating the pheromone table. However it impacts the rate of evolvability where a system that uses 20 ranked ants to update the pheromone table introduces a higher rate of variability in the system. These findings are synonymous to the results found for the AS_{rank} system where the convergence to a solution is independent of the number of ants that update the model [36]. In regards to the varying the initial pheromone level (ρ = 0.1, 0.5

and 1.0) we see that the system with an initial pheromone level of 0.1 converges to a solution faster and introduces a lower rate of variability. These results are agreeable to the findings in [25] where a system with a lower initial pheromone level converges to a solution faster than a system with a higher initial pheromone level.

Analyzing the results for the different environmental conditions we observe that our system is adaptable to both static and dynamic environments. In the random evolution environment we see that neutral changes are a major influence throughout the run whereas variability is a dominant factor only at the start of the run. The fixed target environment paints us a good picture of how our system evolves with respect to convergence. From the start of the run up to the system reaching a target solution, variability plays an important role in helping the system converge to the solution and after the system has attained the target, neutrality plays an important role in preserving the solution obtained. We simulate a dynamic environment in the moving target evolution experiment by changing the target solution at different stages of the run. We find that introducing evaporation eliminates the pheromone stagnation phenomenon. For every target we notice variability is a major influence when the target is changed and the system has a new target to achieve.

In our last experiment in comparison with CAP we use the two environment scenarios of a fixed target and a moving target to analyze our results in a static and dynamic environment. We find both systems are on par with the rate of convergence for both experiments. On evolvability in both experiments CAP introduces a far lesser amount of neutrality than our system. In the case of variability in the fixed target environment our system introduces a higher rate of variability over the entire run whereas in the moving target environment CAP has a higher rate of variability throughout the run whereas our system imposes a high rate of variability only when the target is changed. We use diversity as another measure to compare our system with CAP and find that our system showed better adaptiveness when faced with a dynamic environment by maintaining a highly diverse genotype population.

The system imposes variability as a driving force when it needs to attain an optimal solution, whereas neutrality helps the system preserve an optimal solution. Adaptiveness is a major characteristic of our system as changes in the environment causes the system to reflect those changes in the pheromone model, thus resulting in the creation of individuals that are highly fit for the environmental conditions.

5.1 Contributions

We have presented AC-CGP as a hybrid approach that uses Ant Colonies as a Model-Based Search to Cartesian Genetic Programming. We have introduced an enhanced version of the Rank-Based Ant System that incorporates pheromone evaporation that makes our system adaptable to a dynamic environment by counteracting the effects of pheromone stagnation.

We have also modified a method to measure evolvability for an evolutionary computation technique that uses a model-based search approach. Tracking the evolvability of a system gives a better insight into its dynamics and is a good measure of its effectiveness. It also helps in understanding the types of problems best suited for a system to tackle.

5.2 Future Work

Our results showed that AC-CGP is effective in tackling both static and dynamic problems. This paves the way for interesting future investigations.

In the work done here the application of ACO to CGP as a MBS has been successful, although there are several swarm optimization techniques which takes inspiration from other worldly species. The

application of alternative swarm optimization techniques to CGP or another suitable evolutionary computational algorithm is an area to be ventured into. The research done in ACO is ever growing and new learning schemes are being introduced regularly. It would be worthwhile applying a number of different ACO learning schemes to our system and conducting a comparative study to assess the performance of these learning schemes in different problem domains.

Our system showed positive results in a dynamic environment. To test this further one could conduct a few more extensive tests, to test the effectiveness of the system in dynamic environments. The work done in [38] gives a blueprint to the benchmark problems and tests that can be conducted to study the effectiveness of a system in a dynamic environment. We put down the following few tests one could conduct in studying the effectiveness of the system in a more variable dynamic environment. In the test we conducted, the dynamic environment was simulated by changing the problem at regular time intervals. One could test the system where the environment is changed at variable time intervals [39]. This will enable one to provide a better understanding how the system behaves for a problem where time is a factor. In a dynamic environment the fitness landscape of the problem varies. In the dynamic environment simulated in our study the fitness landscapes are related to one another. This paves a path to another possible way to test the system, where dynamic fitness landscapes are simulated that are independent of each other [40].

Due to time constraints we were unable to conduct an extensive study on the effect of the various system parameters on the evolvability of our system. One could conduct these tests and perform a more conclusive study to understand the effect of these parameters (population size, number of ranked ants and initial pheromone level) on the evolvability of our system.

There is a growing research done in the application of evolutionary algorithms to various fields of machine learning. The application of AC-CGP to the field of data mining as a classifier to evolve classification rules is an interesting avenue to be explored. Classification is an area in data

mining that deals with generalizing data into classes. Classification rules classify data into various classes by processing the dataset through a collection of rules in the form: *If* $< condition_1 >$ *and* $< condition_2 >$ ... *and* $< condition_n >$ *then* $< class >$ type rules. The predicate of each rule is a number of conditions each connected to each other using an AND function. The conditions are in the form of an attribute connected to one of its possible attribute values with an equality or inequality operator. The representation used in CGP is ideal to represent classification rules where each node takes two inputs the attribute and its corresponding attribute value whereas the function of the node would be the equality or inequality operator. ACO would be an effective learning scheme capable of handling data sets that are both static or dynamic in which additional data is added regularly which may or may not require the creation of new rules regularly.

Bibliography

[1] T. Hu and W. Banzhaf, "Neutrality and variability: two sides of evolvability in linear genetic programming," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, (New York, NY, USA), pp. 963–970, ACM, 2009.

[2] C. Darwin, *On the origin of species by means of natural selection; or, The preservation of favoured races in the struggle for life*. D. Appleton and Company, 1861.

[3] M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo, "Model-based search for combinatorial optimization: A critical survey," *ANNALS OF OPERATIONS RESEARCH*, vol. 131, Oct. 2004.

[4] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Scituate, MA, USA: Bradford Company, 2004.

[5] P. Domingos, "A few useful things to know about machine learning," *Commun. ACM*, vol. 55, pp. 78–87, Oct. 2012.

[6] K. De Jong, "Evolutionary computation: a unified approach," in *GECCO '08: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, (New York, NY, USA), pp. 2245–2258, ACM, 2008.

[7] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. A Bradford Book, 1 ed., Dec. 1992.

[8] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, Mar. 2008.

[9] J. Miller and P. Thomson, "Cartesian genetic programming," 2000.

[10] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization – artificial ants as a computational intelligence technique," *IEEE COMPUT. INTELL. MAG*, vol. 1, pp. 28–39, 2006.

[11] B. Bullnheimer, R. F. Hartl, and C. Strauss, "A new rank based version of the ant system — a computational study," *Central European Journal for Operations Research and Economics*, vol. 7, pp. 25–38, 1997.

[12] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B*, vol. 26, no. 1, pp. 29–41, 1996.

[13] T. Stützle and H. H. Hoos, "Max-min ant system," *Future Gener. Comput. Syst.*, vol. 16, pp. 889–914, June 2000.

[14] P. Larrañaga and J. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Boston, MA: Kluwer, 2002.

[15] H. Mühlenbein and G. Paass, "From recombination of genes to the estimation of distributions i. binary parameters," in *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, PPSN IV, (London, UK, UK), pp. 178–187, Springer-Verlag, 1996.

[16] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1 ed., Jan. 1989.

[17] Y. Shan, R. Mckay, D. Essam, and H. Abbass, "A Survey of Probabilistic Model Building Genetic Programming," pp. 121–160, 2006.

[18] R. Salustowicz and J. Schmidhuber, "New ideas in optimization," ch. From probabilities to programs with probabilistic incremental program evolution, pp. 433–450, Maidenhead, UK, England: McGraw-Hill Ltd., UK, 1999.

[19] K. Sastry, K. Sastry, D. E. Goldberg, and D. E. Goldberg, "Probabilistic model building and competent genetic programming," in *Genetic Programming Theory and Practise, chapter 13*, pp. 205–220, Kluwer, 2003.

[20] K. Yanai and H. Iba, "Estimation of distribution programming based on bayesian network," in *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003* (R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, eds.), (Canberra), pp. 1618–1625, IEEE Press, 8-12 December 2003.

[21] M. Dorigo and L. M. Gambardella, "Ant colonies for the traveling salesman problem," 1997.

[22] S. Lorpunmanee, M. N. Sap, A. H. Abdullah, and C. Chompoo-inwai, "An ant colony optimization for dynamic job scheduling in grid environment," 2007.

[23] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed Optimization by Ant Colonies," in *European Conference on Artificial Life*, pp. 134–142, 1991.

[24] O. Roux and C. Fonlupt, "Ant programming: Or how to use ants for automatic programming," *From Ant Colonies to Artificial Ants 2nd International Workshop on Ant Colony Optimization*, 2000.

[25] S. Shirakawa, S. Ogino, and T. Nagao, "Dynamic ant programming for automatic construction of programs," *IEEJ Transactions on Electrical and Electronic Engineering TEEE*, vol. 3, pp. 540–548, 2008.

[26] A. Hara, M. Watanabe, and T. Takahama, "Cartesian ant programming," in *SMC*, pp. 3161–3166, 2011.

[27] L. Altenberg, "The evolution of evolvability in genetic programming 1."

[28] P. Marrow, M. Heath, and I. I. Re, "Evolvability: Evolution, computation, biology," in *Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Program (GECCO-99 Workshop on Evolvability*, pp. 30–33, Morgan Kaufmann, 1999.

[29] M. A. Bedau and N. H. Packard, "Measurement of evolutionary activity, teleology, and life," 1996.

[30] M. A. Bedau, "Quantifying the extent and intensity of adaptive evolution," 1999.

[31] Y. Jin and J. Trommler, "A fitness-independent evolvability measure for evolutionary developmental systems.," in *CIBCB*, pp. 1–8, IEEE, 2010.

[32] J. Jayachandran and S. M. Corns, "A comparative study of diversity in evolutionary algorithms," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010*, pp. 1–7, IEEE, 2010.

[33] J. L. Shapiro, "Diversity loss in general estimation of distribution algorithms," in *In Parallel Problem Solving from Nature, volume 4193 of LNCS*, pp. 92–101, Springer, 2006.

[34] S. Harding, J. F. Miller, and W. Banzhaf, "Smcgp2: self modifying cartesian genetic programming in two dimensions," in *GECCO*, pp. 1491–1498, 2011.

[35] J. R. Woodward, "Complexity and cartesian genetic programming," in *The 5th annual UK Workshop on Computational Intelligence* (B. Mirkin and G. Magoulas, eds.), (London), pp. 273–280, 5-7 September 2005.

[36] M. Dorigo, L. M. Gambardella, P. Solving, F. Nature, H. m. Voigt, W. Ebeling, and I. Rechenberg, "A study of some properties of ant-q," in *Proceedings of PPSN IV Fourth International Conference on Parallel Problem Solving From Nature*, pp. 656–665, Springer-Verlag, 1996.

[37] S. Gotshall and B. Rylander, "Optimal population size and the genetic algorithm," 2002.

[38] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 6, pp. 1–24, 2012.

[39] T. T. Nguyen and X. Yao, "Dynamic time-linkage problems revisited," in *Proceedings of the EvoWorkshops 2009 on Applications of Evolutionary Computing: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG*, EvoWorkshops '09, (Berlin, Heidelberg), pp. 735–744, Springer-Verlag, 2009.

[40] J. Grefenstette, "Evolvability in dynamic fitness landscapes: a genetic algorithm approach," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3, pp. 3 vol. (xxxvii+2348), 1999.

Glossary

Ant Colony Optimization (ACO) is a swarm optimization methodology inspired by the foraging behavior of some ant species.

arity is the number of arguments needed for a function

best ant or global-best ant is the ant that created the individual of maximum fitness from the start of the run.

bloat or bloating effect is the exponential and uncontrollable growth of an individual.

candidate program solutions are tentative solutions to the underlying problem to be solved.

Cartesian Genetic Programming (CGP) is a Genetic Programming methodology where individuals are represented as graphs.

dynamic environment is one in which the problem space changes.

Estimation of Distribution Algorithms (EDA) a model based approach applied to Evolutionary Algorithms.

Evolutionary Computation (EC) is a machine learning methodology inspired from natural evolution.

evolvability is the ability of an individual or population to generate capable adaptive change in order to survive.

fitness is the evaluated value of an individual that indicates the quality of an individual.

fitness function is a function to evaluate the fitness of an individual.

Fixed Target Evolution (FTE) is a problem where a single target is defined at the start of the run.

Genetic Programming (GP) is a population based Evolutionary Computational (EC) methodology where individuals are represented as syntax trees. To begin, a random population of individuals is created that attempt to solve a problem and through a number of generations individuals are created by employing genetic operators on individuals of the previous generation. This process is continued until a solution is found or another termination criteria is met.

genome or genotype is the encoding used to represent an individual.

individual is the term used for a computer program in our system.

iteration-best ant ant that generated the individual of highest fitness in the current iteration.

many-to-one mapping is the property of an evolutionary system where multiple genotypes can map to the same phenotype.

MAX-MIN Ant System (MMAS) is an ACO learning scheme. See Section 2.4.2

Model-based search (MBS) is an optimized search strategy where candidate solutions are encoded in a probabilistic model.

Moving Target Evolution (MTE) a problem where the target changes at different stages of the run.

neutrality is the ability of an individual to be robust to changes.

nonsynonymous change a change in the genotype of an individual that affects its fitness.

nonsynonymous substitution rate is the average nonsynonymous substitutions for all individuals for an iteration.

phenome or phenotype is the visual representation of an individual. Individuals are evaluated based on their phenotype.

pheromone is a chemical substance secreted by some species.

pheromone deposition is the process of increasing the level of pheromone on paths used by ants that generated individuals of high fitness

pheromone evaporation is the process of decreasing the level of pheromone on path that are unused by ants or ones which did not yield high fitness individuals.

pheromone table is a table that holds pheromone values for all possible paths.

population is a pool of candidate solutions.

Random Evolution (RE) a problem where no target is defined.

Rank-based Ant System (AS_{rank}**)** is an ACO learning scheme. See Section 2.4.1

Rate of Evolution (R_e is the ratio of the nonsynonymous substitution rates to the synonymous substitution rates for an iteration.

roulette-wheel selection is a selection algorithm where the possible entities can be visualized as having a portion on the roulette-wheel proportionate to its probability of selection.

static environment is one in which the problem set does not change.

synonymous change a change in the genotype of an individual that does not effect its fitness.

nonsynonymous substitution rates is the average synonymous substitutions for all individuals for an iteration.

variability is the property of an individual to differ from another.