11-27-2015

Using Heritage in Multi-Population Evolutionary Algorithms

Andrew William Hlynka
University of Windsor

Using Heritage in Multi-Population Evolutionary Algorithms

By

Andrew William Hlynka

A Thesis

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

Windsor, Ontario, Canada

2015

Using Heritage in Multi-Population Evolutionary Algorithms

by

Andrew William Hlynka

APPROVED BY:

M. Guarini
Department of Philosophy

S. Goodwin
School of Computer Science

Z. Kobti, Advisor
School of Computer Science

October 23, 2015

DECLARATION OF ORIGINALITY

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

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

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

ABSTRACT

Multi-Population Cultural Algorithms (MPCA) define a set of individuals that can be categorized as belonging to one of a set of populations. Not only reserved for Cultural Algorithms, the concept of Multi-Populations has been used in evolutionary algorithms to explore different search spaces or search for different goals simultaneously, with the capability of sharing knowledge with each other. The populations themselves can define specific goals or knowledge to use in the context of the problem. One limitation of MPCA is that an individual can only belong to one population at a time, which can restrict the potential and realism of the algorithm. This thesis proposes a novel approach to represent population usage called “Heritage,” which allows individuals to belong to multiple populations with weighted influence. Heritage-Dynamic Cultural Algorithm (HDCA) is used to test against different domains to examine the advantages and disadvantages of this approach.

DEDICATION

This is dedicated to my family for their perseverance and success as an example to strive for.

ACKNOWLEDGEMENTS

Thank you to Paul Anderson for providing the idea and resource for the subject of Chapter 5. Thank you to my family and friends who were willing to provide opinions and insight during the writing process.

TABLE OF CONTENTS

DECLARATION OF ORIGINALITY iii

ABSTRACT iv

DEDICATION v

ACKNOWLEDGEMENTS vi

LIST OF FIGURES x

LIST OF APPENDICES xi

LIST OF ABBREVIATIONS/SYMBOLS xii

CHAPTER 1 Heritage and Evolutionary Algorithms 1

1.1 Existing Evolutionary Algorithms 1

1.2 Limitations of Genetic, Cultural and Multi-Population Algorithms 3

1.3 The Introduction of Heritage as a Paradigm 6

1.4 Summary of Thesis Contributions 9

1.5 The Structure of This Thesis 10

CHAPTER 2 A Background Review 11

2.1 Evolutionary Algorithms 11

2.1.1 General Evolutionary Algorithms and Genetic Algorithms 11

2.1.2 Cultural Algorithms 15

2.1.3 Multi-Population Evolutionary Algorithms 19

2.2 Agent-Based Modeling 23

2.2.1 General Overview 23

2.2.2 Agent-Modeling in Evolutionary Algorithms 24

2.3 Weighted-Connections in Computer Algorithms 25

CHAPTER 3 Using Heritage with Multi-Population Cultural Algorithm 27

3.1 What is Heritage? 27

3.2 Why Use Heritage? 28

3.3 How is Heritage Defined? 32
3.3.1 Heritage Representation as a Tree 32
3.3.2 Heritage Representation as a Set 36
3.3.3 Heritage Functionality at the Action Stage 39
3.3.4 Dynamically-Generating Populations 41
3.3.5 Full Implementation Example of Heritage-Dynamic Cultural Algorithm 41
3.4 Really Need Heritage? 42
3.5 An Example Case of Using Heritage 43

CHAPTER 4 Heritage with Multi-Population Cultural Algorithm against Optimization Problems 46
4.1 Introduction 46
4.2 Definition of Algorithms 48
4.2.1 Heritage-Dynamic Cultural Algorithm 48
4.2.2 Heterogeneous Multi-Population Cultural Algorithm 50
4.2.3 Cultural Algorithm 51
4.2.4 Genetic Algorithm 52
4.3 Test Functions and Results 53
4.4 Discussion of the Results 56
4.4.1 General Performance Conclusions 56
4.4.2 Additional Insight and Test Variances with HDCA 61

CHAPTER 5 Using Heritage in Simulation with the Peloponnesian War 65
5.1 Heritage in a Simulation Context 65
5.2 The Peloponnesian War 66
5.3 Translating the War into a Computer Model 70
5.3.1 Background Insight and Preparation 70
5.3.2 HDCA Implementation 74
5.4 Simulation Results and Discussion 76

CHAPTER 6 Conclusions and Retrospective 81
6.1 Concluding Remarks of Previous Chapters 81
6.2 Future Improvements and Directions 82

REFERENCES 85

APPENDICES 89
Appendix A 89
Appendix B 92

VITA AUCTORIS 96

LIST OF FIGURES

Figure 1	A Typical GA (Holland 1992)	14
Figure 2	The Original CA (Reynolds 1994)	16
Figure 3	A Visual Example of Heritage (Tree)	35
Figure 4	Pseudo-code for Heritage (Tree)	35
Figure 5	A Visual Example of Heritage (Set)	38
Figure 6	Pseudo-code for Heritage (Set)	38
Figure 7	Pseudo-code for HDCA	42
Figure 8	Example results from dynamic function (Griewank)	55
Figure 9	Example results from static function (Griewank)	55
Figure 10	Normalized comparison of EA	56
Figure 11	Pseudo-code for HDCA in Peloponnesian War	76
Figure 12	Estimated population growth in Athens and Sparta	77
Figure 13	Estimated civil unrest in Athens and Sparta	77
Figure 14	Estimated overall alliances during the war	77

LIST OF APPENDICES

Appendix A	“The following is additional data from the results of experiments performed in Chapter 4.”	89
Appendix B	“The following is the program code (Java) of HDCA as used in Chapter 4.”	92

LIST OF ABBREVIATIONS/SYMBOLS

EA	Evolutionary algorithm(s)
GA	Genetic algorithm(s) (Holland 1992)
CA	Cultural algorithm(s) (Reynolds 1994)
MP	Multi-population
MPCA	Multi-population cultural algorithm(s)
MP-EA	Multi-population evolutionary algorithm(s)
HDCA	Heritage-dynamic cultural algorithm(s)
H-MPCA	Heterogeneous multi-population cultural algorithm(s) (Raessi and Kobti 2013)
PARCA	Parallel co-operating cultural algorithm(s) (Digalakis and Margaritis 2002)
MCGA	Multi-population genetic algorithm(s) (Da Silva and De Oliveira 2009)
MCAKM	Multi-population cultural algorithm(s) adopting knowledge migration (Guo et al. 2011)
MCPSCA	Multi-population cooperative particle swarm cultural algorithm(s) (Guo and Liu 2011)
MCDE	Multi-population cultural algorithm(s) with differential evolution (Xu et al. 2012)
TAMPCA	Transfer-agent multi-population cultural algorithm(s) (Hlynka and Kobti 2013)

CHAPTER 1

Heritage and Evolutionary Algorithms

1.1 Existing Evolutionary Algorithms

Evolutionary Algorithms (EA) are a set of algorithms related to evolutionary computation in the field of artificial intelligence (Back 1996). In their purest form, they represent a series of solutions which improve over time through evolution. This concept is inspired by biological evolution where species of living organisms evolve and improve with many generations. The benefit of utilizing this in computer programming is that computers can iterate through (simplified) generations of individuals at incredible speed: millions of years of evolution could be calculated within a day!

Typically, EA can be generalized in pseudo-code as being comprised of three main functions that help form a new generation of individuals from the previous generation. These are called *selection*, *reproduction* and *mutation* (Back 1996). Selection and reproduction select appropriate individuals to generate offspring as a combination of traits, and mutation allows controlled random variance to help discover elements of new solutions not represented by the parents. To help select the best performing individuals a fitness function (typically the problem that is to be solved) is used to rank the individuals numerically. With this in mind, it makes sense that the majority of research with EA relates to optimization problems.

There are many types of EA, each inspired by different aspects of nature. Some of these include Genetic Algorithms, Ant-Colony Optimization, Bees-Algorithm, Particle-Swarm

Optimization, and Cultural Algorithms (Back 1996). Of these, Genetic Algorithms (GA) and Cultural Algorithms (CA) are of the most interest in this study. GA use a basic structure of selection, reproduction, and mutation, defining individuals as a set of modulated memes that represent smaller parts of a larger solution (Holland 1992). By defining individuals in this manner it is easier to combine individuals in the reproduction stage to generate new individuals. CA are said to be an extension of GA, but utilize additional logic and knowledge in the individual's generation. This is accomplished with a "belief-space" that keeps memory of the best found knowledge so far discovered, which can quickly distribute knowledge without waiting for explicit sharing between individuals (Reynolds 1994).

The term "individuals" is often used for the solutions represented in each generation. In agent-based modeling, an agent (or "intelligent agent") is described as an autonomous entity which observes its environment and directs its activity to achieve its goals. A multi-agent system allows communication between agents to achieve such goals (Niazi and Hussain 2011). This description is not too different from how CA are defined, as such CA are especially well-suited to agent-based modeling where agents represent the individuals. Some work in the literature uses EA with agents (Gilbert and Terna 2000, Kobti et al. 2003).

While CA are generally considered an improvement over GA, they can still be limited in that the agents are uniform: other than the specific search space they occupy, their problem solving methods are typically identical amongst the population. To solve this problem, Multi-Population EA (MP-EA) have been introduced more recently, including MPCA (Digalakis and Margaritis 2002). This acts as multiple CA working in parallel, each as its own population with its own individuals and belief-space. By defining separate groups the agents can be defined to use separate strategies and knowledge sources as well as explore different search spaces or optimize different variables. As a drawback of MPCA being a new concept with less examples of it in existing work, very few studies currently use MPCA directly for agent-modeling and simulation. This is unfortunate as the ease of defining different strategies of communication and knowledge use makes them directly applicable to this type of model.

1.2 Limitations of Genetic, Cultural and Multi-Population Algorithms

The main benefit of GA is their ease of implementation and flexibility. Little to no explicit knowledge of the problem itself is needed for GA to be programmed: the only requirement is the format of an appropriate solution to the problem to help define the format of the individuals. The problem definition is treated as a “blind-box” for the fitness function to rank the individuals. However, this simplicity can come at the cost of optimization. While GA may be capable of reaching acceptable solutions with good speed, bare GA are not intended to always reach the best or most optimal solution.

It appears intuitive that CA would perform better than GA since they are meant to use explicit knowledge for problem solving. Studies show this to be true, but it leads to a difficult question of how exactly knowledge should be defined in a computer algorithm. There are any number of ways to define knowledge, and using that logic efficiently requires much insight from the programmer. Should the problem's criteria change slightly, a CA would need to be re-written from scratch, unlike the more flexible GA. The generalization of CA in the manner of GA to make them more adaptable for various problems is still under study. Also, as previously stated as the inspiration for MPCA, standard CA is uniform amongst its individuals for simplicity, which limits how logic can be defined and utilized.

MPCA are designed to be capable of utilizing multiple strategies and knowledge resources independently, while allowing separate goals. This can allow competitive, cooperative, or independent forms of CA to be run in parallel for the same problem domain while allowing the ability to share knowledge between populations. This can lead to faster convergence to solutions of optimization problems, to the discovery of better problems that a standard CA might miss, and is also better suited for complex problems that include multiple variables or parameters in the solution format. In a way this is a better representation of the complexity of interaction among evolutionary species as this was inspired by nature and by problem solving logic. However, one flaw which contradicts these inspirations is that agents are set to belong to one population at a time. They may switch between populations at the jurisdiction of the programmer, but to date no implementation exists that allows an agent to belong to more than one population at a time. This ensures the agents are still to some extent limited, since each agent mimics the behavior and logic of their population In reality one would see a much more diverse spectrum of behaviors, logics and goals in each individual, even if parts of them overlap.

This observation suggests that MPCA is still too simple, in the same way that CA is a simplified representation of human problem solving.

Another serious issue in MPCA is the wide potential of inter-population communication and lack of standardization. MPCA's strength is that it contains multiple populations which can share and communicate their discoveries with one another. But how do they make these exchanges? In existing work that define MPCA, examples have shown exchange through individual agents communicating with agents from other populations, like saying hello to a neighbor. Other examples limit agent communication to only other agents within the same population, similar to having students from the same school talk to each other, where sharing between populations occurs when an agent migrates from one population to another (moves to a "new school"). Some uses of MPCA skip agent communication and focus on inter-belief-space communication, which in theory might propagate the best discoveries more quickly. Even further, there exist examples where a single global belief-space is shared amongst all populations and controls how knowledge is distributed. These all bring up a variety of questions: do agents themselves maintain individual-belief-spaces or local knowledge in the way their populations do for when communication occurs? Do optimization the only concern to use MPCA or can it involve testing other factors of the problem-solving process? To what depth and detail is knowledge represented in these belief-spaces? If a global shared belief-space is used, what exactly is the difference between MPCA and CA?

MPCA is still a recent EA and the lack of standardization with both CA and MPCA reduce their applicability, where GA's simplicity makes it much easier to create and maintain for any problem. To put it simply, GA only requires extensive knowledge of the solution format, but CA and MPCA requires both extensive knowledge of the solution and the problem itself, which many researchers struggle with.

1.3 The Introduction of Heritage as a Paradigm

While the exact use and definition of MPCA is debatable, this thesis will attempt to solve the first problem of MPCA defined in Section 1.2. Currently, an agent in MPCA can only belong to one population at a time. To allow agents to be defined as belonging to more than one population at once, and therefore to take advantage of the benefits of these multiple groups, this text introduces a new heuristic approach called "Heritage."

The definition of Heritage in the context of this study would be described as a binary tree comparable to the classical "family tree." During the reproduction process of MPCA where the traits of two parent agents are combined into a child agent for the new generation, a single variable that defines the population that child belongs to would normally copy that of the parents (or should the parents be mixed, then of the stronger parent). By using a binary tree that history of populations is not lost, allowing generations far into the future to have a varied and diverse group of individuals as defined by their Heritage. Along with setting the population id, a numeric weight value is added to describe the amount of influence that population has on the agent's Heritage. This would allow two benefits: 1) in cases where knowledge discoveries and strategies cannot be easily merged between populations, a weighted-random selection process can be used to determine an agent’s immediate actions, and 2) the further down the tree a population resides, or the older the population is in the Heritage, the less effect it should have on the agent, which can be easily implemented by reducing all current influence values before extending the tree.

While seemingly creating further complexity to the MPCA model, using Heritage requires certain assumptions. Heritage must be used in a MP-algorithm, EA with only one population that is uniform in its agents would have nothing to benefit in seeking varied behavior in its individuals. Heritage as it is defined here can be used in other MP-EA, but MPCA is the main focus of this study as it purposely defines the concepts of knowledge in its structure. This definition of Heritage requires that it must be possible for new generations to be created through the mating of agents across different populations, otherwise an agent’s Heritage would only have a single population throughout its span. It also allows the assumption that individual agents themselves do not necessarily maintain individual knowledge through the generations: by defining an agent with a potentially unique Heritage, individuality can be seen in the Heritage alone even though the population’s belief-spaces maintain the knowledge being used. In this way the implementation may be reduced to a higher level that focuses on the populations and their belief-space definitions, even though the individual agents still do the work to solve the problem.

Aside from historic knowledge of whom an agent's ancestors' populations were, Heritage also represents symbolic ties to those older populations. Unlike historic knowledge that maintains a snapshot of the knowledge itself, Heritage is a dynamic link to populations, so as the populations change over time, so do the actions the agent carries out when consulting its Heritage. Heritage is not meant to replace existing concepts of knowledge in MPCA, but to be used alongside it. Like most EA, Heritage is based on observation of nature. In the animal kingdom, some species will show signs of devotion and loyalty to its family or community. In the human race, as people migrate to new homes, the cities and countries that make up their culture are an important factor in their way of life, even though their current home might hold the greatest influence. When countries change, even people with more distant ties to that country are affected. In this way, Heritage is a better representation of "culture" than that seen in most implementations of cultural algorithms.

The use of Heritage may still be questioned. While it makes sense through hypothetical scenarios, it may not necessarily be used effectively in most optimization or search problems. Our hypothesis is that it will not. The existing design of example MPCA show that removing traits that inspired the initial design from nature led to better performance. But in that vein, EA are not necessarily the best choice for optimization problems either. What is expected is that using Heritage will help maintain a greater diversity and make it more difficult to lose what might have been important information, which may be especially appropriate for stability in dynamic problems. More importantly, the use of Heritage can make it easier to implement and observe outcomes in problems related to the interactions and evolution of the individuals, the study of cultural evolution and social networks. Again, most EA are not the best choice for these types of problems, so why use them at all? Because they provide reasonable results and are easily understood, allowing insight into conclusions that might not be possible otherwise.

1.4 Summary of Thesis Contributions

In summary, the contributions of this thesis include:

- Enabling individuals in Multi-Population Evolutionary Algorithms to belong to more than one population at a time, allowing it to combine advantages (and disadvantages) of multiple groups at once, through a proposed paradigm called "Heritage."
- Defining the operations and rules for Heritage by example in Heritage-Dynamic Cultural Algorithm, as an example of a more standardized and flexible version of Multi-Population Cultural Algorithm.
- Testing the benefits of Heritage-Dynamic Cultural Algorithm in numerical optimization functions as a typical benchmark problem, and testing ease of use and capabilities with a simulation of historical politics.
- Showing that Heritage-Dynamic Cultural Algorithm does provide greater diversity in its population over related algorithms, that it had better learning properties in dynamic environments over related algorithms, and that the use of Heritage allows modeling of complex problems with greater ease.

1.5 The Structure of This Thesis

The following chapters will continue to explain the inspirations and details of Heritage, used in what is called a “Heritage-Dynamic Cultural Algorithm” (HDCA). This leaves out MP because it is assumed that HDCA would require MP. Chapter 2 provides an in-depth survey of existing work in EA, specifically MPCA, plus the concepts of weighted-links to populations and the use of EA for agent-based modeling. Chapter 3 explains in detail the structure and implementation of Heritage and HDCA. Chapter 4 tests HDCA against related EA algorithms in static and dynamic environments built around single-goal optimization functions. Chapter 5 provides an in-depth example of HDCA to model a simulation of the Peloponnesian War of Ancient Greece history. Chapter 6 summarizes the conclusions of each chapter and discusses further avenues of research.

CHAPTER 2

A Background Review

This chapter includes an in-depth overview of Evolutionary Algorithms (EA), specifically Cultural Algorithms (CA), Multi-Population Cultural Algorithms (MPCA) and their uses with agent-based modeling and simulation. This is included not only for its relevance and importance to understanding the contributions of this thesis, but also because there does not yet exist a summary of this nature elsewhere in available literature.

2.1 Evolutionary Algorithms

2.1.1 General Evolutionary Algorithms and Genetic Algorithms

EAs mimic the basic processes of evolution as understood by modern natural sciences, where the name originates. There are many common examples of algorithms that can be categorized as EA, including Genetic Algorithms, Ant-Colony Optimization Algorithms, Particle-Swarm Optimization and Cultural Algorithms, each of which are also inspired from different observations of nature (Back 1996). These types of algorithms are often associated with optimization or search problems, and are designed with both their inspirations and problem solving goals in mind.

Different types of EA typically share certain features. One is that a population of artificial creatures, referred to as “individuals,” represent the algorithm’s progress towards a solution, and the cycle of an EA involves the evolution of these individuals. These individuals usually represent solutions to the problem that algorithm is attempting to solve, symbolized as fixed length strings (Jones 1998).

In the first generation, the individuals are produced randomly. During each time-step, an EA evolves the current population into a new generation with three steps: “selection,” “reproduction” and “mutation.” First, the “selection” process occurs where the best-fit individuals are chosen by testing against the problem (a “fitness function”) and comparing the numeric results. The “reproduction” process breeds new individuals with those chosen during the selection process. This is done with crossover operations that combine the individuals, using individual elements from at least two individuals to create the new individual. A third process called “mutation” is completed at the same time as the reproduction process, randomly making slight modifications in the new individual’s parameters. Mutation is justified by observations of unusual occurrences in natural evolution. A child is never exactly like its parents: there are always slight differences that appear for the first time in the child, and sometimes children can have properties that make them outliers to the rest of their population. This also makes sense for problem solving – if not for mutation, an EA would converge quickly to a certain solution and never explore values outside of those randomly generated in the first generation. These processes of selection, reproduction and mutation occur indefinitely, until a satisfactory solution is reached or until a time limit has expired. Specific details of these steps, such as how many individuals to choose during the selection process or how many parents a new child should have, or exactly how individuals are selected and reproduced, are entirely up to the programmer, the specific algorithm, and the problem at hand.

While the exact solution of the problem may not yet be known, the fitness function must be capable of returning a value indicating how well the solution has performed so far. A simple example is a mathematical function F(x,y) with two variables x and y: the individuals would have two values that represent example values for x and y. The individuals that return the highest values of F(x,y) are kept for the selection process. The fitness function is treated as a blind-box component, such that the algorithm's programming logic does not know any details about the function when attempting to solve it.

There are three main implementation branches of EA: Genetic Algorithms, Evolution Strategies, and Evolutionary Programming (Jones 1998). This thesis focuses on Genetic Algorithm examples.

Genetic Algorithms (GA) are a popular EA to use, partly because of its simplicity and partly because of its adaptive nature to many problems. Holland (1992) is credited with the original design of the GA, the concept of which can be viewed in Figure 1. Individuals are described as genes with a set of chromosomes, which is where the name comes from.

As Figure 1 shows, the pseudo-code of GA is not too different from the description of a typical EA. Instead of simply choosing the top individuals during the selection process, the roulette wheel parent selection process ensures that even poor-performing individuals have a chance to be selected, albeit with less probability than more successful individuals (this also means there is no guarantee that the best performing individuals are used at all). The parents are purposely paired in two's, so new children would only have two parents in the reproduction process. Probability values based on their success helps combine the parents' genes for the creation of the child, with mutation performed afterwards. While certain details here are more specific than the previous description of EA provided, GA can still vary in implementation, and so the exact differentiating factors between a basic EA and GA are not clear, although EA can also classify more complicated algorithms other than GA.

```
GENETIC ALGORITHM

1. A population of u random individuals is
   initialized.
2. Fitness scores are assigned to each individual.
3. Using roulette wheel parent selection u/2 pairs
   of parents are chosen from the current
   population to form a new population.
4. With probability Pc, children are formed by
   performing crossover on the u/2 pairs of
   parents. The children replace the parents in the
   new population.
5. With probability Pm, mutation is performed on
   the new population.
6. The new population becomes the current
   population.
7. If the termination conditions are satisfied
   exit, otherwise go to step 3.
```

Figure 1 A Typical GA (Holland 1992)

The main limit of GA is its easy convergence to local optima of a problem and lack of problem-specific knowledge to solve the problem. But few researchers ever point out its benefit, that the lack of problem-specific knowledge makes it easy to apply to many problems with minimal alteration. Varying programmer-set variables can speed up convergence at cost of final solution fitness, and vice-versa. This simplicity can make it troublesome for researchers to understand how a solution was found, making higher-level understanding difficult with GA.

It is worth pointing out that the individual's representation in GA can be as simple as a string of binary values (Holland 1992), but can be implemented with strings of integer values or more complicated representations, including matrices (Michalewicz 1992) and trees (Zhou and Gen 1999, Syarif et al. 2002). While the use of trees as the chromosomes of an individual is meant for optimizing network trees purely for representing the solution format, this concept has loose ties to the inspiration behind the proposed contribution of this thesis in Chapter 3. GA are now a widely known EA, and while many publications exist with them using various extensions and novel problems, including wind turbine placement (Grady et al. 2005), vehicle routing (Ombuki et al. 2006) and bankruptcy prediction (Min et al. 2006), few research projects stand out as being important to the development of GA in the last ten years.

2.1.2 Cultural Algorithms

Robert G. Reynolds (1994) is credited for proposing the concept of Cultural Algorithms (CA). He describes evolution as a process of "dual inheritance," occurring both at the individual level, and being generalized into a group "mappa," otherwise called a "belief-space," which in turn influences evolution in future generations with group knowledge not yet known by all individuals.

The inspiration of CA is said to come from the concept of cultural evolution. Where some EA, such as GA, focus on the genetic level for evolution and progress, cultural evolution suggests that faster adaptation can occur through societies then through standard biological inheritance (Reynolds 1994). Interestingly, the concept of “cultural evolution” as a field of study in computer modeling appears to be non-existent at the time of this writing.

In its simplest form, CA can derive directly from GA, although forms of CA based on other EA can be produced. The traits saved in the belief-space can vary greatly, but again the simplest form is to take the best performing values of what the population has done so far. Pseudo-code for CA (Reynolds 1994) is provided in Figure 2.

```
CULTURAL ALGORITHM
begin
      t=0;
      Initialize Population POP(0);
      Initialize Belief Network BLF(0);
      Initialize Communication Channel CHL(0);
      Evaluate (POP(0));
      t=1;
      repeat
         Communicate (POP(0), BLF(t));
         Adjust (BLF(t));
         Communicate (BLF(t), POP(t));
         Modulate Fitness (BLF(t), POP(t));
         t = t+1;
         Select POP(t) from POP(t-1);
         Evolve (POP(t));
         Evaluate (POP(t));
      until (termination condition)
end
```

Figure 2   The Original CA (Reynolds 1994)

Comparing Figure 1 and Figure 2, the main differences in CA is the addition of a belief-space network and a communication channel between the population and the belief-space. Here, each time-step has the population's individuals communicate to the belief-space, the belief-space adjusting its generalized knowledge, then having the belief-space communicate back to all of the individuals. After this, the evolution process occurs in the same fashion as GA.

Five specific knowledge categories have been defined to describe types of knowledge that can be used (Reynolds and Saleem 2005): *Situational* knowledge storing well-performing past individuals for specific environment situations, *Domain* knowledge able to use problem-specific intuition to predict environment patterns, *Normative* knowledge storing dynamic ranges for finding optimized parameters, *Historical* knowledge storing sequences of past environmental changes from a global perspective, and *Topographical* knowledge that divides the landscape into sub-maps for sampling to predict unknown data. While Reynolds and Saleem provide fair contextual examples using a "Cone's World" problem (a simple maximization problem where the domain is a set of hills), the representation and usage of these can vary greatly based on the problem. Exactly which knowledge should be used at certain periods during the problem solving phase brings questions.

It intuitively makes sense that CA would lead to faster solutions than GA, since a belief-space shared with all individuals would help communicate knowledge of other solutions much faster than propagating knowledge through nearest-neighbor strategies in GA.

However, GA has no such requirement: GA's selection process does not state that only nearby (or else those with similar genetics) individuals can mate with each other. GA assumes individuals are globally accessible, and while their selection process can vary from a variety of methods including tournament selection, truncation selection, roulette-wheel selection and others (Thierens and Goldberg 1994), using positional comparisons between individuals is not common.

So if the increased speed of best-found solutions is not a benefit of CA over GA, what is the purpose of CA? CA has a belief-space that can hold knowledge more advanced than what genomes of individuals in GA can represent, with greater flexibility not limited to the individual. That CA can technically represent its population using GA, neural networks, swarm intelligence, or any other number of forms (Reynolds and Saleem 2005) gives it further flexibility, although this freedom can lead to more confusion during implementation. Most importantly, it is clear from these sources that the ability to use location-based information and selective knowledge dispersal allows the study of knowledge distribution and acceptance in a variety of domains, to be able to determine if certain knowledge types are more powerful than another or if a certain communication method is more realistic than another. The use of the belief-space, its main unique factor over other EA, represents elements of society that may not be fully understood, but is high-level enough to be implemented for study of other external elements despite not understanding those finer details. The belief-space itself can provide a generalized overview of the population as a whole rather than the best-performing individuals for review by researchers.

While seemingly of little meaning, these thoughts suggests that CA were not meant purely for the study of optimization problems, and begin to delve further into qualitative information for new types of problems, even though using functions that return comparable fitness values still helps greatly in testing. This mindset is part of the inspiration behind the developments in Chapter 3. Recent uses of CA focus on the specific design and uses of social structures and patterns (Ali et al. 2013, Reynolds et al. 2014), but significant developments in CA have slowed down in the past few years.

2.1.3 Multi-Population Evolutionary Algorithms

Multi-Population Cultural Algorithms (MPCA) are a natural extension of CA that are still relatively new in modern research. The concept is straight-forward: CA are defined as having a population of individuals with a global belief-space helping guide their evolution. Instead of a single population with a single belief-space, it could be beneficial to have more than one, for greater variety and capability of easier representations of different groups (two heads are better than one!). So MPCA consists of two or more sub-populations, each with its own individuals and its own belief-space. Note that some may prefer to describe MPCA as a set of "sub-populations" instead of a set of "populations;" "sub-populations" suggests that they are still part of a larger group or algorithm rather than being entirely separate, but since many other sources in this field use the same definition for both terms, they will be used interchangeably in this thesis with the same meaning.

While MPCA is the main focus of this section, it should be pointed out that the concept of Multi-Population (MP) has been applied to other EA as early as 1991, including GA (Cohoon et al. 1991), memetic algorithms (Quintero and Pierre 2003) and swarm algorithms (Blackwell and Brake 2004). Coincidently, research using MPCA specifically is fairly recent and sparse as of this writing.

MPCA implementations usually do not require extensive alteration of existing definitions of CA. The focus of designing a functional MPCA, and the new area of research which MPCA allows, is how these separate populations interact with each other. Should populations be completely separate or have a form of shared memory? Should they share all available information whenever possible, share the best available information, or share information expected to help respective neighbor populations based on their unique goals? Should they communicate knowledge between each other or should they migrate individuals that contain knowledge or behavior traits for the population to recognize over time? These vary based on what the purpose of the implementation requires, and these thoughts have led to a variety of different MPCA.

MPCA first appeared in 2002 from Digalakis and Margaritis, although their implementation was called a "parallel-co-operating cultural algorithm" (PARCA). The concept was still similar to later works that used the MPCA name, and involved using multiple sub-populations of CA to solve local parts of a scheduling problem. A master global-knowledge module would keep track of the sub-populations as they each evolved with their own unique goals. Each sub-population has access to different data sets, but communicate with each other to avoid redundant exploration through the exchange of their best individuals.

Other uses of MPCA are to specifically find and keep track of more than one solution to a problem (Alami et al. 2007), to speed up evolution (Guo et al. 2011) and to discover dominant uses of specific types of knowledge (Hlynka and Kobti 2013). While some work uses MPCA with a defined finite-set of sub-populations (Digalakis and Margaritis 2002), some use dynamically forming sets of sub-populations such that new populations can appear (Alami et al. 2007, Xu et al. 2012). Additionally, while some uses of MPCA share knowledge through the migration of individuals between sub-populations (Digalakis and Margaritis 2002, Da Silva and De Oliveira 2009, Mokom and Kobti 2014), others share knowledge directly in the sub-belief-spaces (Alami et al. 2007, Guo et al. 2011, Raeesi and Kobti 2012), and some even simplify MPCA to have a single global belief-space for all sub-populations (Guo and Liu 2011, Raeesi and Kobti 2013).

It is significant that there does not exist many examples of MPCA designed for traditional numerical optimization problems (Raessi and Kobti 2013), a common benchmark for algorithm comparison involving optimizing multiple parameters of a mathematical function. This may be due to MPCA still being a new concept in algorithmic research, as standard CA has been used in numerical optimization (Reynolds and Chung 1997, Coello Coello and Becerra 2004). It is difficult to confirm that having multiple populations can lead to faster or better solutions, although research with Heterogeneous Multi-Population Cultural Algorithm (Raessi and Kobti 2013) provides results that help with this argument.

It is a fair assumption for both MPCA and CA that problem-specific information does not necessarily improve performance, and that added complexity could hurt performance if not used correctly.

While things like quicker evolution, multiple solutions, solving multi-goal problems and maintaining diversity during search were common reasons cited for using MPCA, it is clear from these varied implementations that MPCA is plagued with lack of standardization, adding complexity and confusion to similar issues in CA. Many of these cited articles also like to "extend" their version of MPCA as its own algorithm: these include PARCA (Digalakis and Margaritis 2002), MCGA (Da Silva and De Oliveira 2009), MCAKM (Guo et al. 2011), MCPSCA (Guo and Liu 2011), MCDE (Xu et al. 2012), H-MPCA (Raessi and Kobti 2013) and TAMPCA (Hlynka and Kobti 2013), just to name a few (these terms can be found on the "Abbreviations" page of this thesis).

One might bring up whether or not the existence of MPCA is really necessary. In theory, could the functionality of MPCA be made in a single CA? In examples where there is a single large belief-space shared amongst several sub-populations, yes. It would make sense to have a numerical ID assigned to each individual to keep track of what population it belongs to, which would affect communication and goals for the individual, otherwise a single standard CA will carry out the same general functionality. In the case where there are multiple belief-spaces, the algorithm requires multiple separate CA. However, from a 'meta' point-of-view, a set of populations can be seen as a set of individuals, such that MPCA is really just a complex GA with smarter individuals (each individual being made up of a subpopulation of the MPCA). Truly, the main reason to use MPCA is for organization of complex interactions between individuals that would normally be difficult to implement with CA or GA alone. Moreover, multiple belief-spaces can be observed along with the output data for conclusions. Because of the nature of the structure, it is easier to understand the interactions of these complex individuals using MPCA.

As discussed in Chapter 1, an individual can migrate between populations, but cannot belong to two or more populations concurrently. This missing element is the basis of the contributions in Chapter 3.

2.2 Agent-Based Modeling

2.2.1 General Overview

Agent-based modeling is a computational model consisting of multiple autonomous agents, which themselves are simple individuals able to sense parts of their local environment or other local agents and act upon their discoveries to achieve goals. Typically "agent-based modeling" is concerned with the agent framework themselves and their collective behavior to deduce qualitative understanding, where the term "multi-agent system" refers to a system with multiple agents applied to more traditional problems of practical significance (Niazi and Hussain 2011). The exact definition and use of agents can vary greatly based on the problem. It is even possible to have an agent defined as a series of sub-agents that reach the larger agent's conclusion and action (Russell and Norvig 2010).

2.2.2 Agent-Modeling in Evolutionary Algorithms

GA, CA, MPCA and other similar EA are often specifically defined as being a set of “individuals,” not “agents.” Many experts in the field will correct someone trying to define these EA as a set of agents. The term “agent” may be inaccurate for algorithms such as GA, where the individuals do not necessarily have a local environment or other individuals to sense to determine actions. GA will constantly use selection, reproduction and mutation, the only decisions to be made is which individuals to use at each generation and how to use them, decisions made by the global class and not by the individuals themselves.

However, the definition of CA in Chapter 2.1.2 suggests that decisions become an important part of the process. A belief-space is introduced, as well as more complex uses of knowledge. The purpose to store knowledge is to make some form of decision with it at various stages. Further, the belief-space is meant to spread knowledge more quickly, suggesting that local individuals could not simply spread knowledge to the rest of the algorithm otherwise, and that the concept of “local” individuals is in fact considered. These ideas lend themselves perfectly to the thoughts of agents and agent-based modeling. MPCA, meant to encourage diversity and multiple goals, seems even better suited for the concept of using agents as the individuals. The opposite can also be considered, to use GA, CA or other EA as elements of a single agent’s reasoning process, for example to have a GA-agent-based-model instead of an agent-based-model-GA.

A brief search revealed the existence of agent-based-models that use GA for the fundamental learning process of the individual agents (Gilbert and Terna 2000), although some existing work does refer to GA/MPGA using populations of agents (Chen and Yeh 2001). CA has significantly less results appear in existing literature that relates to agents, again using CA in the logical process of agent-based-models (Ostrowshi et al. 2002) and using CA with agents instead of individuals (Kobti et al. 2003, Peng and Reynolds 2004, Reynolds et al. 2014). Use of MPCA and agents is limited to a handful of articles at the time of this writing (Hlynka and Kobti 2013, Mokom and Kobti 2014). The reduction in use of agents as they become more relevant seems only related to the amount of work available in the EA, with little on MPCA due to it being in its infancy in modern research.

The purpose of this section is to demonstrate that the words “agents” and “individuals” being used interchangeably in this thesis is appropriate based on past work and the problems studied in later chapters.

2.3 Weighted-Connections in Computer Algorithms

An important feature of Chapter 3 will be the concept of using weighted connections in MPCA, as opposed to a binary variable that only allows an agent to belong to one population at a time.

Weighted connections exist in a variety of algorithms in computer science. Artificial Neural Networks are a common example, using weighted connections between components that update over time (Russell and Norvig 2010). More specifically in agent-modeling, Denton Cockburn (2012) proposed a Weight-Allocated Social Pressure System (WASPS) to improve specialization in an agent framework.

EA concepts have been used to improve neural networks (Yao and Liu 1997, Karaboga et al. 2007) and similar problems involving the design of networks (Juang 2004). One example of MPGA with weights given to the populations to determine direction to explore solutions was found (Chang et al. 2007). However, no examples exist in the literature to allow individuals in MPGA or MPCA to belong to more than one population at once through non-binary weight values.

CHAPTER 3

Using Heritage with Multi-Population Cultural Algorithm

3.1 What is Heritage?

Heritage is a new proposed heuristic approach to extend Multi-Population Cultural-Algorithms (MPCA). Similarly, the concept of Heritage as it is defined here can be applied to other MP algorithms and not strictly to CA.

According to the Merriam-Webster Online English Dictionary, the word “Heritage” can be defined as “something transmitted by or acquired from a predecessor” (Def. 2, July 20, 2015). This definition is not too different from how the concept of “culture” is defined in CA, as a form of information passed down from generation to generation. However, the term “Heritage” does not just suggest knowledge, but also “tradition,” where knowledge and behavior might be passed down based on ties to the source. For example, in human societies, cultural-Heritage refers to countries, clans or groups that the individual’s ancestors belonged to, and those groups still hold meaning to descendants that were not directly part of those groups. More importantly, in human-societies Heritage is additive, growing over time as a combination of ties to many groups, such that many unique Heritage trees exist among individuals, instead of a tie to a single group.

With this in mind, Heritage as it relates to MPCA is meant to be a method to represent individuals that belong to multiple populations at the same time. Currently, while individuals could migrate from one population to another, there exist no examples of MPCA that allow individuals to belong to multiple populations at a single instance.

Heritage is built by combining the Heritage of two parent agents to become the Heritage of their child, and after several generations individuals would have diverse cultural makeups. When agents act based on their Heritage, weight values keep track of which populations in the Heritage have the most influence in helping determine current behavior, strategies and goals. Aside from being a more complex yet manageable solution for agent-based modeling with MPCA, this concept could also take advantage of combining traits from multiple populations in search decisions for potential improvement against certain problems.

This concept is used in a new example algorithm called "Heritage-Dynamic Cultural Algorithm" (HDCA). Multi-Population is left out of the name because the existence of Heritage assumes the existence of multiple populations. Additionally, Heritage assumes that the definitions of the populations, whether it be through their objectives or methods to obtain those objectives, are unique from one another. In this way, HDCA is loosely related to concepts explored in Heterogeneous MPCA (Raessi and Kobti 2013).

3.2 Why Use Heritage?

Current research around MPCA does not allow individuals to belong to more than one population. This is a flaw that only exists in the concept of MP algorithms, and has no relevance to traditional single-population CA or GA. Because MPCA are still a recent concept in algorithmic research, the use of Heritage seems a natural evolution in the early stages of MPCA.

There are a handful of theoretical benefits to using Heritage. One is the representation of a new property inspired by natural behavior. Evolutionary algorithms are often inspired by specific traits in nature and perform with reasonable success. This would make them well-suited as general algorithms for agent-based models the require modeling of natural properties, but most evolutionary algorithms are left in their most basic forms for simplicity, leaving a lot of potential details overlooked. Heritage and MPCA do not satisfy all the potential influences in the world to represent cultures and societies, but as a generalization can do more than MPCA alone. Further, enough implementation details of MPCA are left open that they could be added later when an appropriate problem could make use of them.

A greater benefit of Heritage, and the greatest factor in its inspiration, is the creation of greater diversity. One of the reasons MPCA is utilized today is that the separate populations can act differently from each other, using diversity in ways that help keep track of different solutions or to help find new solutions in different spaces. However, individuals can only belong to one population at a time, only receiving influence from one belief-space at a time. The level of diversity is limited by the number of populations defined. This may have been by design, to help maintain simplicity and allow closer study of the populations as a whole instead of the individuals. But with Heritage, individuals would have unique combinations of existing populations to drive their behavior. This means that opposed to homogeneous individuals forming groups, individuals are unique from each other and have a more important role: the micro-level of evolution becomes as important as the macro-level.

Aside from being diverse, Heritage is something that is always part of the individual, even if the influence value becomes very small. This way, information (or in this case, ties to a certain population) is more difficult to lose. Standard CA and MPCA converge by only using the best individuals in the reproduction of a new generation, but HDCA uses evolution both to create new generations and to change weight values of Heritage nodes. This added complexity makes it difficult to suggest that HDCA can lead to better or faster solutions (storing potentially insignificant information may confuse which strategy to use), but the greater diversity it does maintain can prove significant. In dynamic environments where repeated patterns are likely to occur, Heritage's ability to retain older combinations in some form might make it more likely to perform favorably.

Another aspect of belonging to multiple populations is the potential for finding combinations of populations that work well together, as a simplified form of feature selection for reducing parameters used in an optimization problem. MPCA is well-suited to representing different objectives amongst different groups or populations. Combining the goals of these groups in some manner has been studied in Heterogeneous MPCA (Raeesi et al. 2014), meant to improve performance by testing strategies of dividing multi-dimension problems into sub-problems per population. By using Heritage, dimension combinations would be found naturally during the evolution process instead of during pre-processing, poor-performing combinations would be less likely to affect future generations while better combinations thrive. Heritage also allows weighted influence amongst these goals, adding further depth in potential combinations.

This section describes many intuitive reasons to make Heritage a favorable addition to the library of evolutionary algorithm paradigms, but it also promises a more generalized method to represent MPCA. Certain assumptions must be made to use Heritage. By assuming Heritage, agents must be capable of producing offspring together regardless of their corresponding population. It no longer makes sense to keep populations as separate physical groups with controlled migration of agents or knowledge between them. The concept of “population” is reduced to the population’s belief-space, as originally intended in early forms of MPCA and CA, to be a hypothetical example of knowledge distributing quickly to influence its new generations, and to represent the population as a whole. Further, the definitions of individuals themselves had always been dependent on the target problem, posing questions such as whether individuals should have their own logic and belief-space outside that of the population. Now, Heritage itself would define the individual, with the individual’s goals and past knowledge being stored exclusively in the population’s belief-spaces, the only local knowledge being the agent’s current state and surrounding environment. Then the aspects of the populations themselves and how multiple populations differentiate themselves can be focused on. Not meant to replace other methods of knowledge representation, Heritage does not interfere with how populations describe storing and utilizing problem solving components. While the individual is given greater focus than other forms of MPCA, this description also makes it easier to observe and design the populations.

Heritage in MPCA can remove open decisions about how to control communication and migration between different populations, and how to differentiate individuals from populations. While not meant to become a standard component of MPCA, the inclusion of Heritage helps bring a generalization that MPCA and CA have lacked, potentially being a benefit to programmers looking to implement algorithms in this family.

3.3 How is Heritage Defined?

This section discusses how to implement Heritage in a Heritage-Dynamic Cultural Algorithm (HDCA).

3.3.1 Heritage Representation as a Tree

The concept of Heritage can be best related to the common idea of a family tree. A typical family tree can be traced from the child to its parents, from the parents to their parents, and so on. A family tree can also grow into a complex web when additional children or siblings not directly related to the original child's lineage are included. For the sake of representing Heritage, these additional components are not necessary, so the tree is simplified to a linear tree.

During the reproduction process of MPCA, traits of two or more parents are combined to create a new child. With Heritage saved in each individual, it can also be passed down from the parents to the children. So each individual contains at least two references of a programmer-defined node type, and this would be built like a typical tree in computer programming. In a simple implementation, the programmer may only need to consider two parents per child, and therefore only needs two references. But for flexibility, this can also be expanded to using a linked-list reference, which ultimately acts as a list of lists.

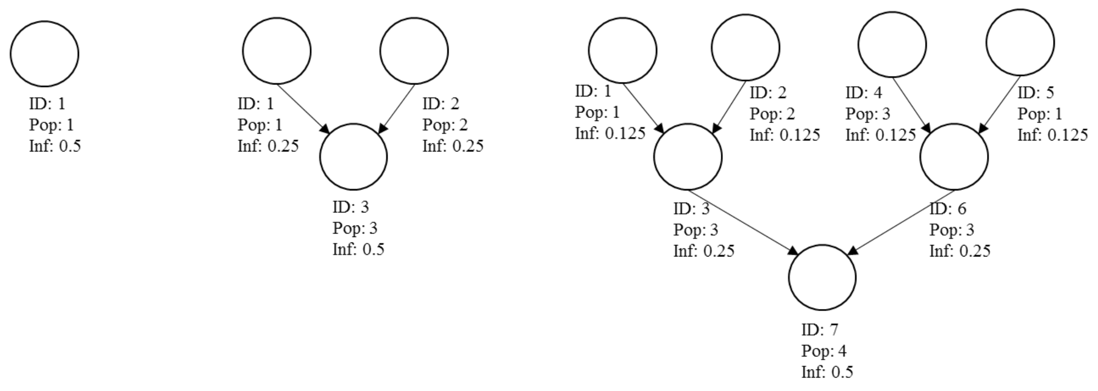
A node contains two additional variables besides references to other nodes. It also contains information on the current population with a numeric ID, and a real value that represents the strength of influence the node would have on children. When initialized, a agent or node with no existing Heritage still has a current population and influence value initialized.

Exactly how influence values are defined can be up to the programmer. It may be worth considering using the performance from the fitness function to help determine this number. But there is a natural occurrence of Heritage in real life that inspires Heritage as a paradigm: when a node is farther in the past of the tree, it should automatically have less influence. So when the influence of a node of the current child is calculated, it should use the number of branches it took to reach it as a decreasing factor. This can be calculated in real-time at run-time when a child is accessing its Heritage, but to do so often can require large computation. It may be worth storing a set to quickly store the influence values after a single calculation for quick reference (this is explained in further detail in Chapter 3.3.2). Influence is additive, so if a node with the same population occurs more than once, all instances add together to give that population a greater influence on the individual. This also allows populations a chance to gain dominance should they appear often in the past, instead of assuming the most recent population has the greatest effect on the individual.

The current population of a node can be determined in different ways. It can be determined based on past Heritage using the influence values, or it can be randomized. The act of randomizing the current node of an agent to state population can be seen as part of the mutation process of EA. If randomized, it better ensures that all populations can be represented somewhere in the individuals, but if mutation is not used to determine current population than this would be an appropriate method for modeling opportunities of population extinction.

Figure 3 shows a diagram example of a Heritage tree growing, and how the influence values change accordingly in future generations. Figure 4 provides example pseudo-code that describes how Heritage can be stored and updated in an example agent.

The use of a list of lists can grow to a limitless size, and can be effected quickly by hardware limits. If a child is limited to only having two parents, each generation would have Heritage growing by approximately $O(n^2)$ at each time-step, where n is the number of generations. From the previous description, nodes far in the past would always have a very low influence on the current generation, so it may be acceptable to check and delete older nodes that have influence below a certain threshold. This then allows for flexibility based on the intentions of the problem.



Popul-ation	1	2	3	4
Influence	0.5	0	0	0
Norm. influence	1.0	0	0	0

Popul-ation	1	2	3	4
Influence	0.25	0.25	0.5	0
Norm. influence	0.25	0.25	0.5	0

Popul-ation	1	2	3	4
Influence	0.25	0.125	0.625	0.5
Norm. influence	0.167	0.083	0.417	0.333

Figure 3 A Visual Example of Heritage (Tree)

```
HERITAGE_TREE
begin
      Node
              List<Node> parents = null;
              Int population_id = 1;
              Double influence = 1.0;
      End Node
      Update_Heritage(List<Node> parents)
              Node.parents = parents;
      End Update_Heritage
      Get_Influence(int pop_id, double degree)
              Double influence = 0.0f;
              If (node.population_id == pop_id)
                    Influence +=
                          node.influence * degree;
              For each (p in node.parents)
                    Influence +=
                          p.Get_Influence(pop_id,
                          degree*0.5);
              End loop
              Return influence;
      End Get_Influence
end
```

Figure 4 Pseudo-code for Heritage (Tree)

The purpose of storing Heritage explicitly as a tree is to be able to track the origins of Heritage. In certain problem environments it is helpful to know this type of information to recognize patterns from past experiences. Further, the Heritage tree itself could result in patterns that help guide appropriate Heritage in future generations. It could show certain combinations of parents / Heritages working well and others not. Heritage is not meant to replace historic knowledge, but using Heritage in this way could help add to historic information to make decisions. The exact usage of Heritage is open for many types of applications. But in cases where this type of historical pattern information is not of benefit, storing Heritage as a set instead of a tree can save memory and still provide general information on influence values.

3.3.2 Heritage Representation as a Set

To use Heritage as defined in this thesis with evolutionary algorithms, the major criteria is to have an influence value corresponding to each existing population. The simplest form of this is a set or array of real values, normalized between 0.0 and 1.0, such that the sum of all the values add up to 1.0. Every individual would have their own set indicating the influence populations have on them.

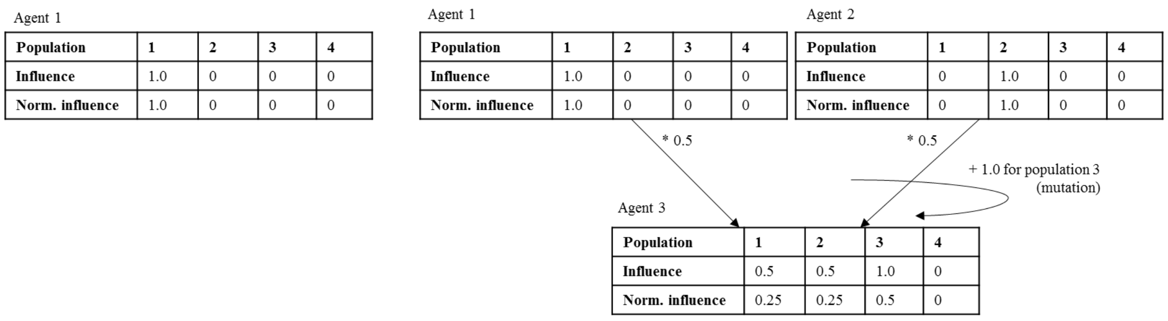
At the beginning of an HDCA, it is assumed the set of individuals will be initialized to belong to a single population. This means their Heritage set has one value of 1.0 and many 0.0's. When Heritage is combined at the reproduction stage for a new individual, the Heritage from the parents are combined and added together. If the Heritage was originally normalized in the parents, than the new sum of the values in the child's

Heritage should be 2.0. After being added, to represent the decrease in Heritage over time, the values are decreased by a certain percentage (to match the similar strategy of Chapter 3.3.1, the valued are decreased by half as an example). Similar to Chapter 3.3.1, the programmer may or may not choose to use mutation to add value to the influence from a specific population, or may add value based on the existing influence values in the Heritage. In this case, the Heritage values would need to be normalized again, such that all values in the set add up to 1.0. Normalizing the values at the end of the reproduction stage helps ensure consistency and makes it easier to compare the values for further study during a simulation.

Figure 5 shows a diagram example of Heritage sets developing over multiple generations, and Figure 6 shows example pseudo-code to help guide how such a set would be programmed.

Compared to a tree, storing Heritage as a set can be beneficial for computation. The memory usage for each individual would be consistent, at O(k), where k is the number of populations. Accessing the influence values is also more straight forward.

As described in Chapter 3.3.1, the programmer may wish to use both a tree and a set to represent Heritage: the set could be a place to store the total influence values in the tree, and the tree would act as a secondary source in instances where the origin of Heritage points were required. But in some cases the use of this type of historic information may not be needed, and using a set alone would suffice. In such examples, the focus would not be about the evolution and history of Heritage for individuals, but the combinations of existing populations to set and reach appropriate goals. In this case, the definition of the populations themselves and their role in HDCA is crucial.



Agent 1

Population	1	2	3	4
Influence	1.0	0	0	0
Norm. influence	1.0	0	0	0

Agent 1

Population	1	2	3	4
Influence	1.0	0	0	0
Norm. influence	1.0	0	0	0

Agent 2

Population	1	2	3	4
Influence	0	1.0	0	0
Norm. influence	0	1.0	0	0

Agent 3

Population	1	2	3	4
Influence	0.5	0.5	1.0	0
Norm. influence	0.25	0.25	0.5	0

Figure 5   A Visual Example of Heritage (Set)

```
HERITAGE_SET
begin
      Double[] Set = new Double[population.count];
      Update_Heritage(List<Set> parents)
              Set += parents.set;
              Set *= 0.5;
              Set += 1.0 for mutation value;
              Set.Normalize();
      End Update_Heritage
      Get_Influence(int pop_id, double degree)
              Double influence = 0.0f;
              Influence = Set[pop_id];
              Return influence;
      End Get_Influence
end
```

Figure 6   Pseudo-code for Heritage (Set)

3.3.3 Heritage Functionality at the Action Stage

Heritage can define combinations of populations that help define an agent. The majority of focus in an implementation of HDCA is how those populations should be defined. Unlike traditional MPCA, a population is no longer a subset of individuals and their collective belief-space – it is only the belief-space.

Once an individual has their Heritage defined, they must use that Heritage to determine actions in the environment, or else define how to represent a solution to the user. This is where population belief-spaces come in. Populations may store goals unique to one another that guides the individual on what to focus on. They may store historic, topographic, situational, domain or normative knowledge that provide clues on optimizing those goals. But Heritage defines a combination of populations. This forces a standardization of the definition of a population's belief-space, such that the conclusions made from one population can be merged with the conclusions of a second population.

As an example, suppose two populations each had a unique goal to be solved in a multi-objective problem, where the problem has one parameter to optimize. Each population might store knowledge of the best found parameter value so far. If an individual had Heritage with a combination of these two populations, it may act by going towards the average of the two best-found values provided by the two populations. If the influence of one population is greater than another, than this might cause the individual to move closer to one population's preferred search area. In other cases where population goals did not conflict with each other, the actions might be able to co-exist. As a second example, if two populations were trying to optimize separate parameters of a function, the individual could use knowledge from both populations in improving those separate parameters, and using them together only when grading the final solution using a fitness function.

In situations where populations are defined in a way where their actions cannot be merged, the influence values can be used in a statistical roulette-wheel approach to picking a single population to follow for a given time-step. Implementers could take advantage of this scenario: if a population is chosen to be followed and immediately leads to worse results, then the influence of that population in that individual’s Heritage could be reduced to make it less likely to be picked again.

In addition to a population having influence on an individual, the individual should have similar influence on the population. That is to say, an individual with a population that has dominating influence on its Heritage should also have greater influence on that corresponding population compared to individuals with more varied Heritages that are less dedicated. This is a representation of certain individuals having the priorities of the population having a greater effect on that population. This is not a requirement of HDCA, but if chosen to be used, the influence values stored in the individual’s Heritage could be passed along with the solution to the population’s belief-space. By storing this alongside found solutions in the belief-space, influence values would play in the decisions of future generations when they determine how to act.

3.3.4 Dynamically-Generating Populations

All usages of Heritage so far in this chapter assume the population belief-spaces to be well-defined at the start of the problem. Depending on the simulation or problem, the programmer may wish to make these elements more flexible. Perhaps a population's goals would change over time, based on the success or failure of other populations. Perhaps populations could go extinct or die out. Perhaps new populations could be born as a grouping of commonly-found population combinations from existing Heritages.

While this level of complexity will not be studied in great detail in this thesis, this is an example of how the explicit study of cultural evolution could benefit from the representation of Heritage. More interactions and situations could be represented in ways that a traditional MPCA was not capable of.

3.3.5 Full Implementation Example of Heritage-Dynamic Cultural Algorithm

This section includes a complete pseudo-code example of HDCA, as seen in Figure 7. It assumes Heritage is defined as it is in Figure 4 or Figure 6. Keep in mind that details such as population belief-space definition, agents chosen during selection and ways of generating new agents in reproduction can vary greatly based on the problem and the programmer.

```
HDCA
begin
      Initialize list_of_agents;
      Initialize population_belief_spaces;
      Initialize Heritage in list_of_agents;
      Until termination
              Select best from list_of_agents;
              Initialize new generation of agents
                    new_agents;
              For each n in new_agents
                    New_agents.UpdateHeritage(2 or
                          more parents from best);
                    Use New_agents.GetInfluence()
                          to reach new solution;
              End loop
              Replace worst from list_of_agents
                    with new_agents;
      Repeat loop
end
```

Figure 7 Pseudo-code for HDCA

3.4 Really Need Heritage?

MPCA already has many variations that utilize different concepts (see Chapter 2.1.3). Is it necessary to define Heritage and HDCA as yet another form of MPCA?

In the case of statistically choosing a population to follow at each time, is this not identical to representing migration in MPCA? For the most part, yes it is. MPCA already has examples that allow individuals to migrate from one population to another. Doing this often could be similar to pretending to be part of many populations and picking one to be at for an instance. But this would also require the individual to retain its own belief-space of knowledge, such that it can carry it over to other populations. With HDCA, it is not necessary for an individual to retain this type of information; the Heritage itself and its relation to multiple population belief-spaces makes up the theoretical belief-space of the individual. Otherwise, using Heritage is a more standardized form of migration and knowledge transfer that otherwise did not exist with MPCA. In the concept of merging the strategies of multiple populations to determine an action of an individual, MPCA is not capable of this in any existing form.

Once more, many people would translate Heritage as just a form of historic knowledge, but this is inaccurate. While Heritage as a tree does retain historic knowledge as to how the Heritage was formed, the strength in Heritage is the ability to access multiple population belief-spaces at once. As those belief-spaces update, changes can be seen accordingly in the individual, whereas historic knowledge would be static and cut off from updates not collected by the individual directly. Heritage is not necessarily a form of knowledge as defined in other CA, as it decides how to use knowledge defined in the population belief spaces; if there was no other knowledge defined, Heritage would be meaningless. In this way, Heritage and historic (or other) knowledge can and should co-exist.

3.5 An Example Case of Using Heritage

The inspiration behind Heritage and HDCA was the desire to model a complex system of multiple groups, and to have varied individuals defined in a simple manner. This section describes the “True King” problem, a potential game environment that can make use of HDCA in a multi-agent system.

The “True King” world is defined as the following: suppose there exist a set of *n* villages that make up a kingdom ruled by a benevolent king. The new player is a man who wants to become the new king, and must convince the villages that he is a worthy alternative. Individual people who live in each village are allowed to move from one village to another. The village as a whole has a shared belief-space that represents their favor with the player. Each village may have different desires to be met in order for them to side with the player. If the people of the kingdom had to make a unanimous decision about whether or not to follow him, would he be crowned the true king?

In this example, the villages make up the different sub-populations of a larger system, and the individuals living in each village are agents. In a traditional CA, this might be modeled as one giant system, with a global belief-space storing the unanimous decisions of the entire kingdom, but it would be difficult to implement the unique individuals and obtain information about different villages. In a traditional MPCA, the villages could be modeled separately, but individuals would be simplified to belonging to a single village at a time. This might be acceptable, but intuitively if a village was against someone and a person who used to belong to that village heard, that person would also have an instinct to be against that person, which in turn could have small effect on his current village.

With HDCA, it becomes simpler to implement this system of agents. Individual agents do not need to retain their own knowledge. Their actions would be decided by the collective strategies of their Heritage populations, and their results would update their populations to make global decisions. Defining individuals to be able to move from place to place while retaining these ties becomes understandable. And having unique agents with different points of view becomes easier to manage without writing every individual from scratch.

In this situation, the fitness function value is the overall acceptance of the player in the kingdom, and the problem is really based around how one optimize their actions to modify the opinions of the villagers. This shows how HDCA can be suited for modeling rather than strict problem-solving. Alternatively, other examples could have the agents represent solutions with HDCA guiding their evolution with varied search. As an off-topic example, the villagers in the kingdom may be trying to maximize their happiness, with each village having their own unique desires such that individuals with multiple villages in their Heritage have more varied goals. This then guides them as a simple human simulation. Such an example could be used to model an existing city to determine if certain traits or interests can be properly satisfied based on the resources in the area.

So far all examples of HDCA suggest ties with populations or groups, as if belonging to a society. These populations could also be used for other categorical representations, like people with common interests or with literal families with common desires.

CHAPTER 4

Heritage with Multi-Population Cultural Algorithm against Optimization Problems

4.1 Introduction

Numerical optimization problems are a common method to test algorithms against each other in speed and performance. They usually involve optimizing the output of a mathematical function by altering the values of multiple input parameters - for example, optimizing function F(x,y) by finding ideal values of x and y. Simple problems involve single objectives, but more complex problems may involve multiple objectives, where it is mandatory to fulfill all defined criteria (for example, optimize function $F_1(x,y)$ while keeping $F_2(x,y) > 0$).

It may be possible to check all possible solutions if the parameter values have finite precision, but more interesting search algorithms can find near-optimal solutions in a much shorter span of time without checking all possibilities. While such optimization functions may loosely represent real-world problems, they are used in research fields as a general basis to test multiple algorithms against each other in a non-biased environment. Under real-world circumstances, a near-optimal solution may be accepted if it can be achieved more quickly.

Multi-Population Cultural Algorithms (MPCA) have not been tested thoroughly against these generalized mathematical problems (see Section 2.1.3). Standard Cultural Algorithms (CA) and Genetic Algorithms (GA) have more examples of published work with optimization problems. This chapter tests Heterogeneous-MPCA (Raeesi and Kobti

2013), CA (Reynolds and Chung 1997) and GA (Holland 1992) against Heritage-Dynamic CA (HDCA), as closely related evolutionary algorithms. This uses modified test functions taken from the CEC 2014 special session on single objective numerical optimization (Liang et al. 2013), updated to be tested as both static and dynamic environments. The dynamic versions of these functions are tested by reversing the output values of the function at every 10 time steps. As an example, F(1,2) = 5 becomes F(1,2) = -5, then becomes F(1,2) = 5, and so on. These are tested to optimize the maximum value of the chosen function, which is irrelevant in the case of the dynamic test. Unlike the functions explicitly described in CEC 2014's list, these functions are not rotated or shifted.

Although the basis of CA is to utilize knowledge over time to improve performance, the definition of said knowledge can become a detriment to the algorithm if used incorrectly. Under this logic, it is possible GA can outperform the advanced algorithms that are dependent on that extra knowledge. It is expected the HDCA will not perform better than any of these algorithms in the static tests for these same reasons, knowing that the use of additional knowledge through ties with multiple populations may only further confuse the algorithm's logic. However, it is also expected that HDCA may show signs of promise in the dynamic test functions, as it may be better suited to retaining knowledge than the other algorithms in this test. Additionally, the representation of HDCA may show information about specific combinations of parameters having greater influence on the function than others, a feature difficult to see in other algorithms used here.

4.2 Definition of Algorithms

4.2.1 Heritage-Dynamic Cultural Algorithm

For these tests HDCA is defined with the simplistic model, as a set instead of a tree. Searching the origins of obtained Heritage is not used for this specific case.

HDCA requires multiple populations, which here are defined by goals. If the function in the test has n parameters, then n populations will be initialized, each with a goal of optimizing a single parameter. A set of agents (100 agents are used here for all algorithms) are initialized and distributed equally among the populations. At the start each agent wishes to optimize only one parameter of the problem. Each agent is represented as a set of parameter values that together make a solution when entered into the function (randomly initialized), and each agent has its own Heritage set of influence values describing its ties to the populations from 0.0 to 1.0. The belief-spaces for each population are represented as a set of best parameter values (for its specific goal parameter) found so far, with an influence value representing its source and the fitness score it received to describe its success. This helps represent both how a population influences an individual and how the individual influences the population.

After each time-step, the belief-spaces of all populations are updated with all parameter values found in the previous time-step with respect to the population's goal. For example, if population 1 is optimizing the first parameter, it will collect all values of the first parameter from all agents with Heritage containing ties to population 1. Each value is collected along with the influence value the population has on said agent, plus a calculated fitness value returned from the function (the agent's entire solution is used for calculating the fitness, although only the one parameter is saved in the population's belief-space).

The top *n* percentage among the agents (the top 10% were used, an arbitrary value) are kept during the reproduction process into the new generation, both their solutions and their Heritage. They are also used exclusively for the selection process before reproduction. The remaining 90% of the new generation are a combination of two random parents from the top *n* of the previous generation. Unlike traditional MPCA or CA, the passed-down traits are the parents' Heritage rather than their solutions. The values from the parents' Heritage are halved and passed down, potentially summed if elements of their Heritage are shared, to represent influence decreasing over time. The most successful parent based on the fitness of their solution decides the current population of the child, equivalent to the population of the greatest influence. The influence values of the new Heritage set are normalized between 0.0 and 1.0 before being given to the new agent.

The parameter solution values of the new agent are taken from the belief-spaces based on the Heritage values it has been given. A statistical roulette-wheel selection process takes values from the solutions in the population's belief-space, where "value fitness" X "influence it had on the whole population" equates to the chances of it being chosen. Additional mutation allows variance in these parameter values to help find new solutions, where 50% of the parameter values will be shifted by up to 5 units (in a search space of

100). Any parameters not part of the new agent's Heritage are randomized, representing such knowledge as unknown to the agent.

For example, suppose agents A and B belong to populations 1 and 2 respectively at the start of the algorithm. Agent A only belongs to population 1, so its solution is 1.0 * fitness of A, where population 1 having the greatest influence on A means A has the greatest possible influence on population 1. A and B have a child called C, which has a Heritage of populations 1 and 2. Agent A performs better than B, so C belongs primarily to 1 and has greater influence from 1 than 2. C's solution to population 1's parameter is taken from 1's belief-space (here, only from A), and C's solution to population 2's parameter is taken from 2's belief-space (here, only from B). The influence values represent how the agents are represented in the belief-spaces used to pass down information to future generations. C will update 1 with a 0.75 influence and 2 with a 0.25 influence, and C can have greater likelihood of passing down traits by being more dedicated to a population.

4.2.2 Heterogeneous Multi-Population Cultural Algorithm

H-MPCA from (Raeesi and Kobti 2013) is described as a class of MPCA where each population has unique goals from one another. In this way, it is not too different from how HDCA is defined. Currently, this is the most appropriate version of MPCA in the field to use against single-objective optimization problems, as others have been tested in different environments.

Each population is evenly assigned the parameters of the problem, instead of a single population for every parameter (this defines 40% the number of parameters as the number of populations, such that every population has 2-3 parameters to focus on as described in the original paper). Agents are initialized randomly. During most time-steps, they explore possible solutions through local search, reducing the search distance during each step as a simple form of simulated annealing. Every 5 time-steps, the search distance is re-initialized, and the best performing individuals (calculated by taking their parameter values with respect to their population with the best found values of other parameters stored in a globally-shared belief-space) update the belief-space, consisting of only the best individual parameters so far and their fitness value.

Like the original paper, explicit knowledge used is simplistic and limited to storing the best found parameters so far. Unlike the original paper, offspring are generated in a simpler method by keeping the best found individuals and taking the combination of two parents from the best found. The original algorithm was tested against differential-evolution algorithms, how H-MPCA would test against traditional CA (and whether or not such a comparison is appropriate) is uncertain.

4.2.3 Cultural Algorithm

Using a traditional CA or GA against the proposed extensions may be unnecessary, but since H-MPCA has not been formally tested against CA, and both HDCA and H-MPCA are ideally meant to be improvements of CA, it seems an appropriate opportunity. To use the purest form of CA without major alterations, this chapter uses one of the earliest forms of the algorithm used in this type of problem (Reynolds and Chung 1997).

The algorithm is defined by randomly initializing a population of individuals each representing a full solution, and at each time-step generating new offspring. In the example provided by the text, situational and normative knowledge are used, and so they are used here. Situational knowledge is represented as a set of best individual solutions found so far, and normative knowledge is an interval range where the best parameter values are likely to be found. Both are stored in the population's belief-space. The best performing agents update the belief-space at the beginning of each time-step, and new offspring equal in number to the previous generation use the population's belief-space to define their solution. The old and new generations are combined, the best half remaining while the worst half is removed.

4.2.4 Genetic Algorithm

GA is defined well in (Holland 1992), but is simple enough to implement from scratch. An initial population of agents are generated with random solutions. The top n (20%) are selected for reproduction, offspring being combinations of two parents with individual parameters from parent 1, parent 2, or an average of both. Mutation (of 50% likelihood) modifies individual parameters by up to 5 units in a 100-large search space. The top individuals remain in the new generation. A large mutation rate is used to help speed up search since best solutions are kept through older individuals, but the search radius is small enough that improvements due to mutations are gradual. Note that the values used in this example are arbitrary and not optimized since this is the least relevant algorithm in this test (but the results of GA turn out to be quite surprising).

Unlike H-MPCA and CA, where the parameter values are taken directly from the publications cited in Chapters 4.2.2 and 4.2.3, the parameters in both GA and HDCA are arbitrary. This is because of the huge range of possibilities for these values, making up an infinite number of variations. This is especially true for HDCA, H-MPCA and CA, which require the definition of strategies and goals that may be defined with new functions rather than numeric values. In a way, this makes the use of these evolutionary algorithms for optimization, and the study of optimizing the algorithms themselves, a very difficult and abstract concept. Chapter 4 is meant as an example comparison only, and better results for all the algorithms used is possible.

4.3 Test Functions and Results

The test functions are borrowed from the CEC'14 Test Suite of Single Objective Real-Parameter Numerical Optimization problems (Liang et al. 2013). In this suite, there are four categories of functions, named "Unimodal," "Simple Multimodal," "Hybrid," and "Composition."

The experiments here are not comprehensive, as a general overview of how HDCA performs is the purpose of these tests and there is not enough room in this chapter for further data. A handful of functions from the suite are randomly chosen from each category. The search space for each input parameter is 0 to 100. Applied rotations and shifts to the functions as described from the suite are not used. In addition to testing static performance, a dynamic environment where the function's value is inverted (F(x) = 1 becomes F(x) = -1, and so on) every 10 time-steps is also tested to observe HDCA's features. The number of parameters to optimize is kept at 10, again to not overcrowd this chapter, but tests with 5 and 20 parameters are also done and briefly summarized to give an indication of change in performance in the algorithms.

The functions used here are Discus Function (unimodal, non-separable, one sensitive direction) and High-Conditioned Elliptic Function (unimodal, non-separable, Quadratic ill-conditioned), Ackley (multi-modal, non-separable), Griewank (multi-modal, non-separable) and Rastrigin (multi-modal, separable, many local optima), Hybrid Functions 1 (Schwefel's, Rastrigin's and Elliptic Functions) and 5 (Scaffer's, HGBat, Rosenbrock's, Schwefel's and Elliptic Functions), and Composition functions 1 (Rosenbrock's, Elliptic, Bent Cigar, Discus and Elliptic Functions, multi-modal, non-separable, asymmetrical, different properties around different local optima), 3 (Schwefel's, Rastrigin's and Elliptic Functions, multi-modal, non-separable, asymmetrical, different properties around different local optima) and 7 (Hybrid functions 1, 2 and 3, multi-modal, non-separable, asymmetrical, different properties around different local optima and different properties for different variable subcomponents), for a total of 10 test functions.

Figure 8 and Figure 9 shows two sample graphs of Griewank's function with the four algorithms, one graph for the dynamic version and a second graph for the static. Figure 10 shows an additional graph showing a normalized average of the algorithms' performance against the ten functions to better compare them. Appendix A is a multi-page chart that details numerical data from the tests. The algorithms were run ten times and the average "group average" and the average "best performing individual" is recorded, as well as standard deviation. The average of every 5 time-steps is recorded to show signs of improvement during each stage. Every 10 time-steps the dynamic function changes from negative to positive stages and vice versa, the results changing accordingly. Only the first 50 time-steps are recorded to save space. Static data is also recorded, first the average of the first ten 10 time-steps and the final results after 100 time-steps.

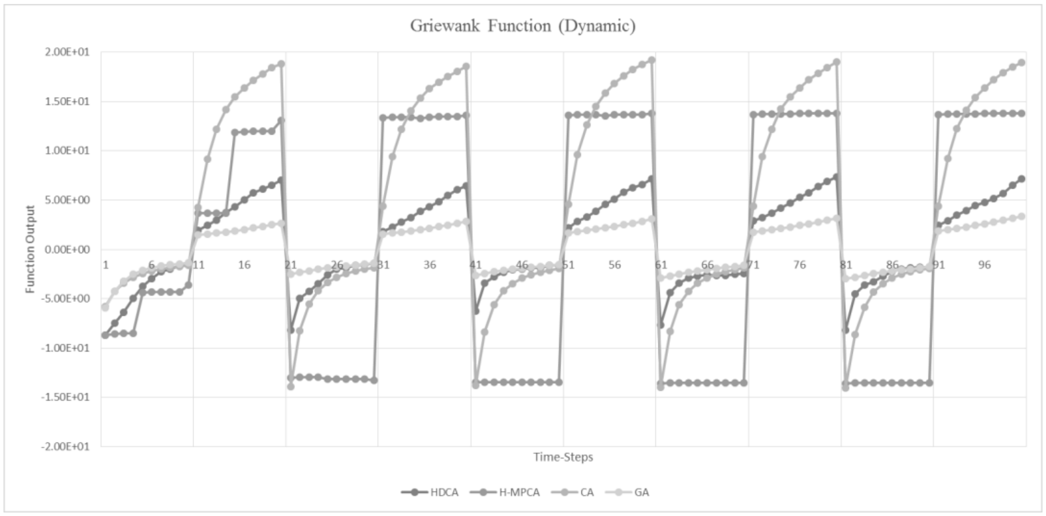


Figure 8   Example results from dynamic function (Griewank)

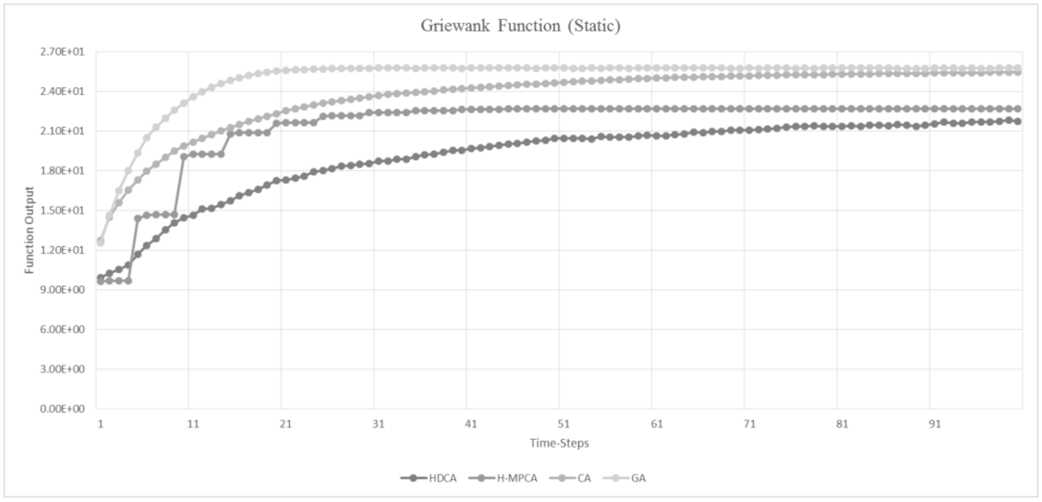


Figure 9   Example results from static function (Griewank)

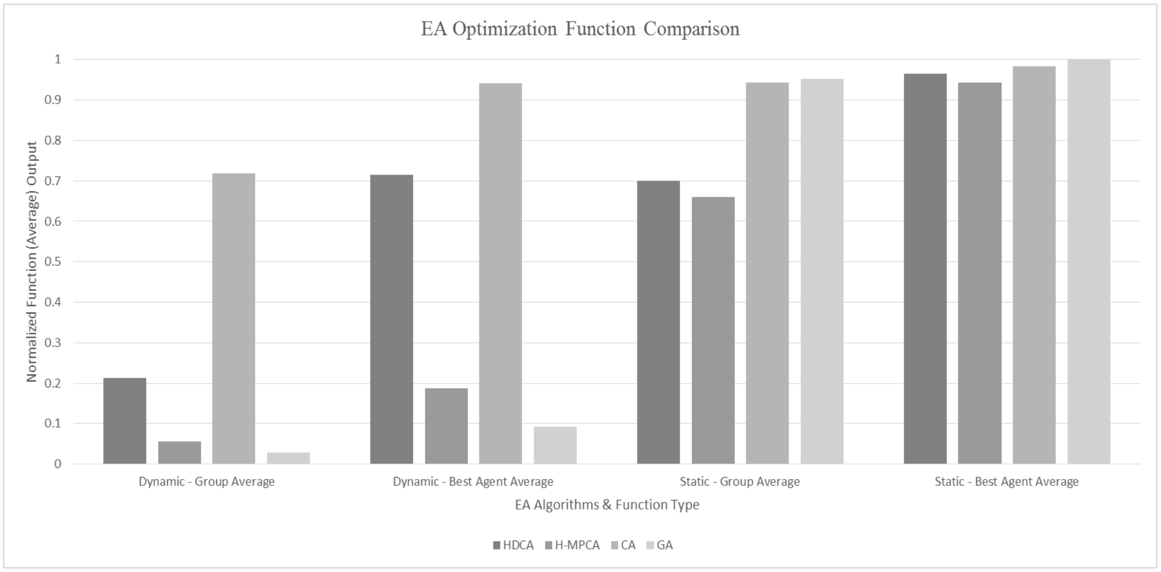


Figure 10 Normalized comparison of EA

4.4 Discussion of the Results

4.4.1 General Performance Conclusions

Both the average performance of the group as a whole and the best individuals of the group are recorded, as well as standard deviation.

Briefly, the results of the static functions show GA outperforms the other functions in nearly all instances in evolving the group as a whole towards the maximum optimal value. Depending on the problem and the usage, the individual finding an optimal value is the goal rather than the whole group, but GA also shows itself as having individuals reach that optimal value more quickly in every case. CA almost matches GA in both group and best-individual evolution on average, but typically does not have an individual find the optimal solution even when GA, H-MPCA and HDCA do. HDCA has individuals finding optimal values less often than GA but more often than the other algorithms. H-MPCA has individuals finding optimal values only for two functions (Discus in the first 10 time-steps, and also Ackley by the end of 100 time-steps), and does not show outstanding performance anywhere else. The largest standard deviation for the group and individual is usually H-MPCA and sometimes HDCA. Minor improvement continues to occur after the first 10 time steps for all algorithms. The difference in performance for the best-performing individual for each algorithm is minimal enough to allow any of them to be a fair substitute for each other.

While CA shows the most promise with group convergence in this traditional form of optimization, it may seem unusual that GA's individuals generally outperformed CA. Notably CA rarely has any individual reach the optimum value of the function, even when the other algorithms do after 100 time-steps. Both GA and CA are older algorithms, and that H-MPCA did not outperform them despite being a recent development seems contradictory. H-MPCA was not originally tested against GA and CA when conceived (Raeesi and Kobti 2013). This is an example where over-complicating the evolution process in EA does not necessarily lead to improvement.

For HDCA, testing against static optimization functions shows that high values are generally found by individuals quickly (sometimes second only to GA during the first 10 time-steps), but the group as a whole generally shows the weak performance, only outperforming H-MPCA on average (individual functions would have varied results as to which outperformed the other) and by a slight margin. HDCA also shows large standard deviation based on the runs collected, showing its performance can vary. The range between the group average and best individual seems greater in HDCA than the other algorithms. This all matches with the intended design of maintaining diversity in its individuals to better extend the search space instead of converging everyone too quickly. Given that all algorithms were able to find good solutions quickly, there was little room for improvement by expanding the search space through the diversity Heritage provides.

Using dynamic optimization functions show that CA is the better performing algorithm on average for all functions. Specifically, CA continuously shows better performance when the dynamic function is in its positive stage. The dynamic nature of the functions mean that doing well during the positive stage can mean doing poorly in the negative stage, and vice versa. A safer algorithm might be one that is slow to converge to a maxima, for never reaching a local maxima means never being at the worst possible point during a stage change. This safer approach accurately describes GA, which shows better results than H-MPCA. H-MPCA occasionally performs well during the positive stage as a group and sometimes for individuals, but always does poorly in the negative, and due to converging too quickly after the first few environment changes and not overcoming the local maxima it found.

HDCA in the dynamic functions shows itself to perform reasonably and is second to CA overall, but shows promise particularly during the negative stages. This is sometimes true observing the group’s behavior and often significantly true observing the best performing individual. The only explanation for this is that the negative stage occurs first in the cycle, and HDCA may do better in keeping memory of those better solutions even in later instances when those solutions are no longer important. When looking at a graph (see Figure 8) it is clear that HDCA does improve more quickly in the negative stages than the other algorithms. As to exactly why HDCA does not perform as competitively in the positive stages, this may also be because the experience from the negative stages is dampening the ability to find those new solutions. The change in the second half of each stage is recorded in Appendix A to see which algorithms show the most improvement. HDCA, CA and GA each share maximum improvement depending on the stage and comparative solutions found. Generally, CA shows better group improvement, HDCA and GA show better individual improvement, but finding good solutions early usually means limited improvement is possible and removes best-performing algorithms from this area of thought. If an algorithm finds the best solution quickly, it cannot improve in the time-steps following immediately afterward, and this type of growth can seem irrelevant for certain instances, for example why CA does not show the best improvement consistently in every case.

HDCA's interesting property is the sign of improvement in certain functions, gaining dominance in the negative stages later on during the simulation. This is more noticeable in Rastrigin's, Griewank's and especially Ackley's functions (all multi-modal functions), where HDCA did not surpass the others at first but did in the second and successive occurrences of the negative stages. Further running showed HDCA outperforming CA on average for both group and individuals if the first few stages are ignored. This shows more apparent learning that the other algorithms were less susceptible to, and H-MPCA in particular was weak with. While not reaching these results as quickly as the others, HDCA becomes only second to CA over most of the functions if those initial learning-generations are ignored.

The algorithms' completion time were measured in seconds to give an estimate of their practical performance. The four algorithms were each run against 10 functions, 10 times each, as they were to obtain the data for this chapter. The hardware used was a Microsoft Sufrace Pro 3 with an Intel i3-4020Y 1.50GHz CPU, 4.00 GB of RAM and Windows 8.1 64-bit OS. It is worth mentioning that most of the functions were solved quickly, but the final function ("Composition07") took the majority of time spent for the entire test. The times for completion were as follows: GA was 86 seconds, CA was 135 seconds, H-MPCA was 58 seconds, and HDCA was 180 seconds. That H-MPCA is so efficient, even more so than GA, matches with the authors' original intentions for the algorithm (Raeesi and Kobti 2013) despite the solutions found being less favorable. Since HDCA was not intended to be optimal the results are not surprising, but anyone wishing to use HDCA will want to keep this information in mind.

Overall, HDCA performs as expected, not performing better than all related EA but showing ability to retain information to allow better performing individuals rather than a better performing population as a whole. In situations where the goals or environment change, having a diverse population in this manner becomes desirable. CA shows surprising resilience in both static and dynamic functions by being able to converge its population quickly, but not necessarily always having individuals able to outperform the individuals in HDCA, and is least likely to have individuals find the global maximum in static functions. GA performs comparatively well in static environments but performs poorly in dynamic environments, although the static data suggests the algorithm converges to reasonable solutions within the time frame that the dynamic function used to set change. H-MPCA is the most recent state-of-the-art adaption of MPCA and CA, and yet HDCA on average outperformed it in static functions and significantly bested it in dynamic function, for converging too quickly made H-MPCA a poor choice for that type of environment.

4.4.2 Additional Insight and Test Variances with HDCA

One aspect of the design of HDCA in this example was that the influence a population had on an agent's Heritage also defined the influence that agent had on the corresponding population. This means that more successful solutions found by an individual may not automatically be used in successive generations if that individual had a large Heritage made up of many populations, that is to say if the agent's "dedication" towards that population was low. Other researchers might suggest that this is a flaw that may hold back the success of the algorithm against optimization problems.

Experiments did test the algorithm again with a modified HDCA that ignored the influence the agent had on a population: while the population would still store multiple solutions, the fitness score of the parameter would be the only value used in the randomized roulette-wheel selection process when generating a new individual's solution. In theory, this would ensure that better-performing parameters would be chosen more often, which was not a guarantee in the previous design. While the results of this test are not outlined in detail to save space, the outcome was surprising. In dynamic functions, the group's overall average was consistently worse than the results from the original design in Chapter 4.3, although the "Best in Group Average" was split evenly, sometimes improving and sometimes worsening on average, with no clear relation between functions. It is possible the results from the best performing individual can be accredited to statistical variance, as the changes were small enough to fit within the standard deviation range from previous results, but the group average decreasing in every instance is significant. Similar results occurred against static functions, only the Hybrid01 and Composition03 functions showed increase in the group averages; the rest showed decrease in both the first 10 time-steps and at the 100th time-step. In both cases the results did not change the algorithm's ranking against H-MPCA, CA or GA, but the general decrease is against the hypothesis. This could be a sign that agents searching for multiple goals at once can lead to misguided results against other agents dedicated to optimizing fewer goals, but further analysis is required to make certain conclusions.

The decision to have dynamic test functions change every 10 time-steps was based on the original design specifications of H-MPCA, which normally requires part of its functionality to occur every 5 time-steps. After seeing that H-MPCA performed poorly under the dynamic functions, the tests were repeated with a quicker speed of change, to change the function every 5 time-steps instead of 10. The hypothesis was that the CA's overall performance would not be as strong, since the CA would not be at high levels for as long, giving chances for other algorithms that may or may not converge faster to shine.

Specifically, HDCA might have performed better in comparison to the other algorithms. Ultimately this did not turn out to be true. While the difference in average overall performance between HDCA and CA reduced slightly, CA's overall group average and best-in-group average was still the best of the four algorithms.

While not recorded in depth, the experiments were repeated for parameter sizes of 5 and 20. The optima that exist may change with different parameter numbers, so the only relevant information was the comparable data of each algorithm with each other. There did not appear to be any change for static or dynamic functions regardless of parameter size, and CA continued to be the dominant method.

One of the features of HDCA that the other EA in this chapter are not capable of is the ability to combine population goals into the Heritage of new individuals, and observing the Heritage influence values of that individual as a simple form of feature selection. This could help categorize which parameters, if any, have the greatest influence on the optimization functions being tested. To test this, code printed out the best performing individual of the group at the end of each run of 100 time-steps. Unfortunately, the results were disappointing, showing the top individual always had a Heritage made up of a single population. This could mean that the best individuals had parents of the same background, and is less likely to mean that a random individual from the very first time-step reached the best solution since varied improvement does occur in each step. This could also suggest the success of HDCA came from large dependence on randomized parameters for new generations, explaining why the group as a whole rarely mimics the performance of the best individuals, but this would be a poor conclusion seeing how well its best individuals performed against the other EA. After several runs, the single parameters of the best individual remain fairly consistent, a handful of parameters from each function appearing more often than others. If one had printed out more information than simply the one top individual, or increased the top individuals kept to allow greater variety in Heritage populations, or allowed for mutation to occur in Heritage and not just the solutions, then the results might provide more insight, but this requires further testing in future work.

As described in Chapter 4.2.4, it is possible that the parameters can be tested with variations to improve all of these algorithms, including HDCA. While not expected, this also means that HDCA may be able to outperform CA under the right conditions.

The code for HDCA, written in Java and as used for the experiments in this Chapter, can be seen in Appendix B.

CHAPTER 5

Using Heritage in Simulation with the Peloponnesian War

5.1 Heritage in a Simulation Context

One of the major influences of the design of Heritage is the desire to model complex and varied individuals in a simplified manner.

Connected individuals in varied environments with different goals, desires and strategies can be intimidating, and when such a model is meant to represent realistic parameters the implementation becomes all the more daunting. Heritage focuses on the definitions of the populations, with the individuals limited to being represented as combinations of those populations, which would make it simpler to control and change the behaviors of all individuals involved.

As described in Chapter 3, the populations in Heritage-Dynamic Cultural Algorithm (HDCA) would represent differing objectives, strategies or knowledge acquired. This well-suits requirements for agent-based models, where agents must make decisions based on the environment around them. It is also possible to create a system with no explicit environment, where the simulation observes the links between the individuals and how they react with each other. The merging of different strategies from an individual's Heritage, or else the randomized selection of a strategy, also has room for further testing and research from the designer.

It can be difficult to find appropriate test-cases to verify the use of a system design in simulation compared to solving a problem. This is partly because simulated models representing real-world scenarios are not always meant to use optimality in their actions. If this is the case, then evolving individuals using Evolutionary Algorithms (EA) where a fitness function plays a crucial part in the design appears less suitable. HDCA was initially meant for use with social systems, where an individual that could be categorized as a combination of features would react to other individuals based on the population belief-spaces from those features. For this type of simulation, much data is required to represent friendships over a period of time, for example, data collected from modern social networking websites and services. Alternatively, models to simulate political preferences and alliances within individual cities, across provinces or between countries over time could also be made with Heritage. Unfortunately, data sets of this nature are not easily accessible for use at the time of this writing, but as “big data” becomes available to researchers, HDCA may become more appropriate to representing these types of systems.

This chapter is meant to show the use of HDCA in a simulation context by in-depth example. For this case study, an example is generated using available information on the Peloponnesian War, a conflict in Greece involving many cities circa 430 BC.

5.2 The Peloponnesian War

The Peloponnesian War was a conflict in Greece from 431-404 BC, fought between two groups led by Athens and Sparta. During this time, Sparta was growing its empire and influence while Athens' powerful navy, economy and size made it feared by some provinces for being too dominant. The war was a series of battles and internal rebellions that led to the downfall of Athens. Information of the events of this war is limited to accounts of the time. The specific perspective of Thucydides is covered in great detail in the book *A History of Ancient Greece in its Mediterranean Context* from pages 254 – 272 (Demand 2013). This is the source of the information used to build the model in this chapter, the events of which are summarized in the following paragraphs.

As early as 450 BC, Athens was known for its naval fleet, giving it greater strength and providing a means of transportation for a strong economy. In the 430's, Athens knows it needs to keep control of the Aegean coast for timber supply, as well as the silver and gold mined in the area. Athens establishes Amphipolis along the route to these cities, but had to fight the local population to take it. Athens aligns with Corcyra and both countries battle against Corinth, a city originally part of Corcyra. Athens also asks for Potidaea's allegiance to be symbolized by bringing down its walls of protection against Macedonia (which was aligned with Athens) and rejecting Corinth. Potidaea refuses, and with support from Sparta, declares itself against Athens. Megara is sentenced by decree to be forbidden access to Athens for trade, which would later cause economic difficulties for Megara, a decree that Sparta said would be cause for war later on. Thebes attacks Plataea, where Plataea was an ally of Athens but also an easy target. Plataea survives this conflict and kills 180 Theban hostages, which was seen as unusually cruel. Afterwards, Plataea would receive greater support from Athens while Thebes receives support from Sparta.

At this point, war seems certain, and the account says that young men were excited for battle from the stories of glory in the past, and that Greece overall seems to favor Sparta.

In 431 Athens walls itself from other cities on land to protect itself from Sparta's forces. Peloponnesus attempts to invade Athens, but Athens would raid Peloponnesus, Megara and Aegina, repopulating them with citizens from Athens. Athens gains an alliance with Thrace and Macedonia. Sparta invades Attica, and Potidacia formally surrenders to Athens. In 430, 429 and 427, Athens gets three separate occurrences of a plague, exactly which disease it was is still unknown today but estimates suggest that approximately 25% to 33% of Athens' citizens were lost. Peloponnesus does not attack Athens out of fear of the plague, but focuses their forces against Plataea instead. Lesbos, seeing a chance in Athens' weakening, revolts against Athens by siding with Mytilene, and both rely on Sparta's support to defend against Athens. However, Sparta does not arrive in time and Athens defeats Lesbos and Mytilene, ultimately deciding not to kill off their entire population in mercy. Sparta overtakes Plataea and executes most of their citizens. Civil strife breaks out in Corcyra, and Athens' plague has led to untimely replacement of important leaders and unusually large inheritances of wealth being collected by family members.

In 425 Athens fortifies their ally Pylos for it being close in vicinity to Sparta. Sparta in turn sets up a base at Sphakteria but loses control of it to Athens; local Spartan citizens are kept as hostages by the Athenians. Sparta offers to makes peace with Athens, but Athens declines; alternatively, Sparta tries to send food to their hostages under Athens for their survival, and Athens reacts by taking the hostages back to Athens for security. Athens tries to expand their empire into Boeotia, takes Delium, but due to poor organization loses heavily to Thebes. Athens also loses to Sparta the important Amphipolis, a significant source of timber, silver and gold. In 423 Athens and Sparta agree on a peace treaty, but an attempted revolt of Scione from Athens ruins this trust, and Athens destroys Scione for their actions. Athens tries to retake Amphipolis from Sparta but fails, with important figures being lost in the battle. Athens and Sparta call a truce again in 421, and agree to refrain from attack each other for seven more years.

During this truce, Athens focuses its forces on Scione, Melos and Mytilene, and then prepares to attack Sicily for resources. Attempts by Athens to gain nearby allies were fruitless. Around 414 Athens attempts an attack on Syracuse, but fails, and then fails a second time. Due to superstition, their forces waited too long to retreat from Sicily and most were killed. In 410 Athens defeats Cyzius in battle at sea. Sparta again asks for peace, but negotiations fail. Sparta concentrates on strengthening Ionia instead of Hellespont, which would later be seen as a strategic mistake to be acted upon in the future. During this time, several cities under Athens' control begin to revolt; Athens is able to subdue some of them but the revolts continue to grow.

The events after 410 are not included in the simulation of this chapter, but in summary, Athens would win multiple victories after 410 to show its dominance, and recover cities they had lost. The general that led Athens to these victories is not re-elected as general after 406 and is exiled from the city. During a naval battle between Sparta and Athens,

Athens is victorious but due to weather conditions is unable to complete the fight. In outrage some of Athens' top naval commanders were executed, which decreased moral in the city. Sparta took Hellespont after not having done so before in 410, which was Athens' main source of grain. Athens attempted to fight back but is defeated. Facing starvation and disease, Athens surrendered in 404 BC. Instead of being decimated, Sparta took Athens as a city under them.

5.3 Translating the War into a Computer Model

5.3.1 Background Insight and Preparation

Tracing the events outlined in Section 5.2 requires an in-depth understanding of geography, politics, and events leading up to the war, and can be a great area of further research by devoted experts in the field of Ancient Greek history. For those not well-versed in the subject matter, the references to different cities, provinces and countries can be overwhelming. It can be difficult to find resources that confirm the alliances these cities had before the war and the countries they belonged to during this period, which may differ from modern maps of the region.

It was decided that a computer model of the Peloponnesian War designed with HDCA would be a general model of the political climate, to determine alliances of cities as a whole towards either Athens, Sparta or neutral. Individuals would be modeled to make an estimate of the population of each city, representing the varied Heritage at work that might cause a city to have unexplained influence from another city. Another numeric factor called ‘mutation’ would be recorded, to represent civil unrest at times of large death counts. As a factor, this might help explain why certain decisions were carried out in situations that conflict with descriptions of other data. While individuals are modeled, the environment and resource management is not, and updates are made once every year as a broad representation. Therefore, this is not accurately defined as an agent-based model because there is no explicit environment for the individuals to traverse and interact with. The time period is kept between 450 BC and 410 BC: the resource (Demand 2013) cited years as early as this and including these in this simulation helps initialize factors in the world before conflict begins over a decade later. The last few years of the war were ignored for simplicity and because they have greater reliance on food resources which is not included in this model.

To begin, a class called “Population” was made to represent both cities (and groups of countries) and the individuals (people) living in those countries. A list of Population representing cities and a list of Population representing people are initialized. With Heritage in mind, Population can store a local Heritage set (see Chapter 3.3.2) that represents uni-directional hierarchical relationships. In the case of cities, Heritage represents the ownership of a city, for example city *A* belonging to country *B* would have *B* in the set, but *B* would not have *A* in the set. In situations where a city overtakes another city as part of them, this is also represented in the Heritage set. In the case of people, Heritage represents the cultural Heritage of where that person (and their ancestors) lived. For example if person *X* moves from city *A* to city *B*, both *A* and *B* would be in *X*’s Heritage, with *B* holding greater weight.

There are certain keywords of events in the resource that repeat enough to be used in the design of the model. These include occurrences such as attacking one another, strengthening bond through trade or alliances, taking over a city, killing individuals from a city, and the result of civil strife by various circumstances. These are further simplified down to functions to update Heritage, update Friendship, update Side and update Mutation, where each makes up the knowledge stored in each entity of Population.

Aside from Heritage representing links to cities, Friendship is stored in a similar way as a set of weighted numeric values. Friendship represents the relationship a city has with another city based on past events and not direct inheritance. An example would be if one city attacks another city, then the Friendship level between them would decrease. Side is a single integer that defines which side a city is on, either -1 for Athens, 1 for Sparta or 0 for neutral. Mutation is an extra variable that represents civil strife from accounts and from a decrease in population, and while it does not have any effect on the other variables, it can stand as an explanation to certain actions occurring in later years that contradict the rest of the data. Only Heritage relates to the people, but all four are kept in the knowledge of the city's belief-space. As defined by HDCA, individual people are represented as a combination of the cities, which in turn helps update the cities to represent complex interactions with each other.

When certain recorded events occur during the war, they can be broken down to updating these four main variables. "Attacking" involves reducing the Friendship level between two cities, and with an assumption that the armies fought and died in battle is equivalent to 50% of the smallest population size of each city, these individuals are killed. “Takeover” is a separate event from Attacking as it could occur willingly or not, but kills off 50% of the city’s population, replaces that same amount with individuals from the new city overtaking the old city, and the Heritage of the individuals and the city itself are updated accordingly. An event function called “Update Friendship” simply updates the friendship level directly between two cities, to represent a variety of events which symbolize changes in trust. Another event function called “Kill” kills individuals that belong to a specific city by a factor passed as a parameter. These functions are called accordingly each year based on written accounts of events to help the simulation occur. Killing individuals from a specific city was based explicitly on their ID indicating which city they belonged to, but sending individuals out to attack was based on any who had high friendship with each other to symbolize the sharing and supporting of armies.

At the end of each year regardless of other events, the code updates the individual people, by increasing the population by 10% and decreasing the population by 4% each year (these factors are based on informal sources online that suggest that the average age of death was 25, thus statistically $1/25$ of the population would die each year, and that the average household had 5 children in the lifetime of the parents, which suggests the number of deaths * 5 per two parents would make up the annual birth rate). When generating new individuals, two parents are randomly selected from the existing individuals in the previous generation, and their Heritage is combined to make up the Heritage of a new individual. To allow for mutation and migration, the new individual is randomly chosen to belong to a population and bring its old Heritage with it, with a greater preference to belong to a city that does not differ greatly from the old cities and their current side on the war. After new individuals are generated, the Friendship, Side, and Mutation values stored in each city's belief-space are updated based on the Heritage of the individuals belonging to those cities. Here the individuals may have a stronger influence to the city they belong to, but the varied Heritage to other cities allows the Friendship levels from other cities to affect each other. Additionally the Friendship values of cities allows cities to affect the values with each other in equal measure as with the individuals. The Side to lean towards in the war is based on the Friendship levels, and the Mutation value is based on a rate of change in population size for each city.

While not included, one additional factor that could have been considered was the spatial location of cities in relation to each other for affecting migration of new children and friendship values.

5.3.2 HDCA Implementation

A set of cities with a name and index are initialized, 37 in total, 16 of which are overarching countries or provinces that the smaller cities belonged to. Smaller cities are initialized to have a Heritage set that includes weight to the country they belong to. A second set of individuals representing people are initialized, with one individual per every 1000 people, for simplicity. Using existing evidence (Demand 2013), Athens had a large population of roughly 315,000 and Sparta had roughly 16,000. In this model this is reduced to 315 and 16 individuals respectively. These two cities make up the leading and unflinching symbols of both sides of the war, and they will not change sides. With lack of evidence, other cities are estimated to have approximately 10,000, represented as 10 individuals in this model. These individuals are initialized to have a Heritage containing their city which will grow in future generations, plus a variable set to specify the current city they belong to.

A year index is initialized to 450, and a loop repeats until the year is decremented to 410. During each year, events are written based on whatever is provided in the resource (Demand 2013). For example, in the year 434 BC, Corcyra aligns with Athens and causes a conflict between Corinth and Athens, resulting in both Corcyra and Athens attacking Corinth. This is summarized in code by 1) updating the side Corcyra belongs to, 2) reducing the Friendship value between Athens and Corinth, and 3) having both Corcyra and Athens use their forces to attack Corinth. Similar event summaries occur for the years between 436 and 410.

After all events are carried out for a year, the entire population of individuals updates by reproducing new individuals and killing off others to simulate birth and death, and to propagate the generation of varied Heritage sets. The Friendship values of cities, including which side they ultimately side with during different years of the war, are updated with the Heritage makeup of their individuals and of cities they have close friendships with.

This implementation can be seen summarized in the pseudo-code of Figure 11.

```
HDCA
begin
      Initialize list_of_agents;
      Initialize population_belief_spaces;
      Initialize Heritage in list_of_agents and
              population_belief_spaces;
      Until termination (from 450 to 410)
              If event occurred, play out event
              Update individuals based on
                    birth/death rate;
              Update friendship values based on
                    events and individuals.
              Update mutation based on size change.
      Repeat loop
end
```

Figure 11 Pseudo-code for HDCA in Peloponnesian War

5.4 Simulation Results and Discussion

To trace the events of the war in this simulation, the Friendship sets, the Side, and the Mutation values of all 37 cities are exported into a 2D array for each year, for a total of 40 matrices. For statistical purposes, the simulation is run 10 times and the average is used in the observations discussed in this section. This would take up many pages and is not included here (the data comes out to over 1,500 lines, which would require a minimum of 30 pages in this thesis). However, Figure 12 and 13 show the simulation's estimates of the population size of Athens and Sparta and their mutation rates over time, and Figure 14 shows an estimate of how the other cities felt of these two sides during the course of the war.

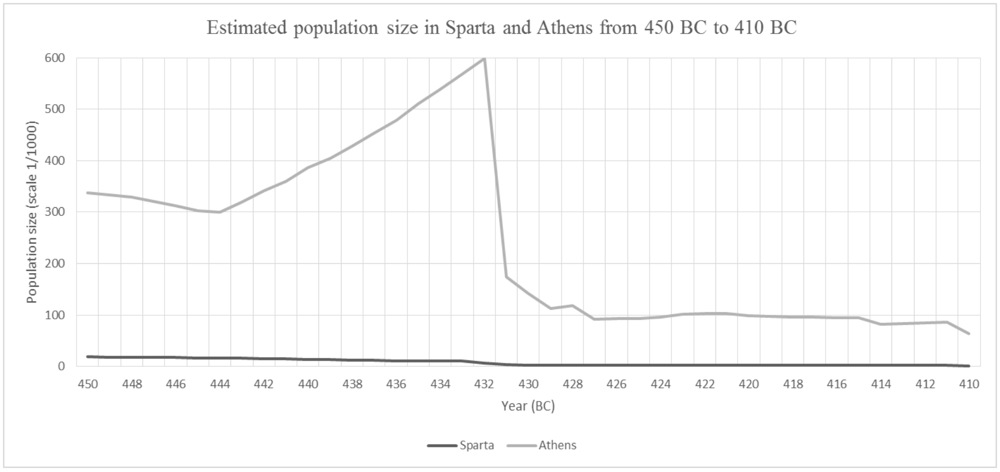


Figure 12 Estimated population growth in Athens and Sparta

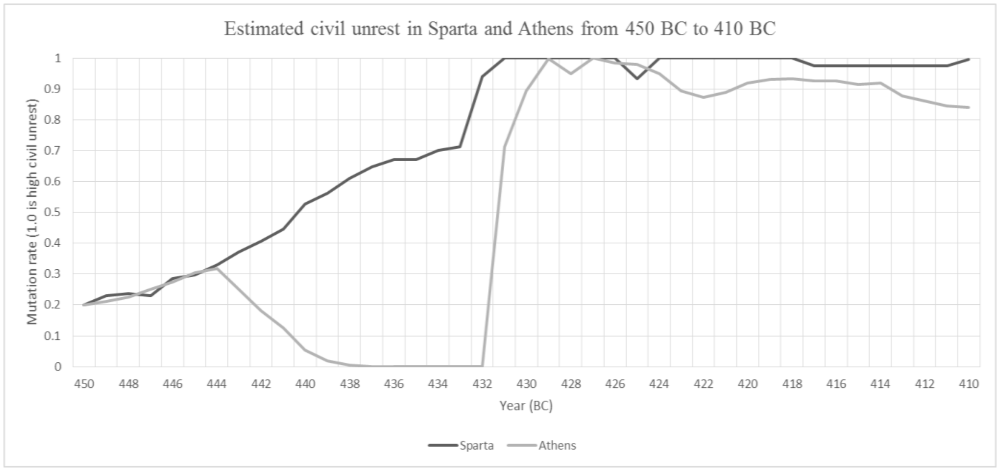


Figure 13 Estimated civil unrest in Athens and Sparta

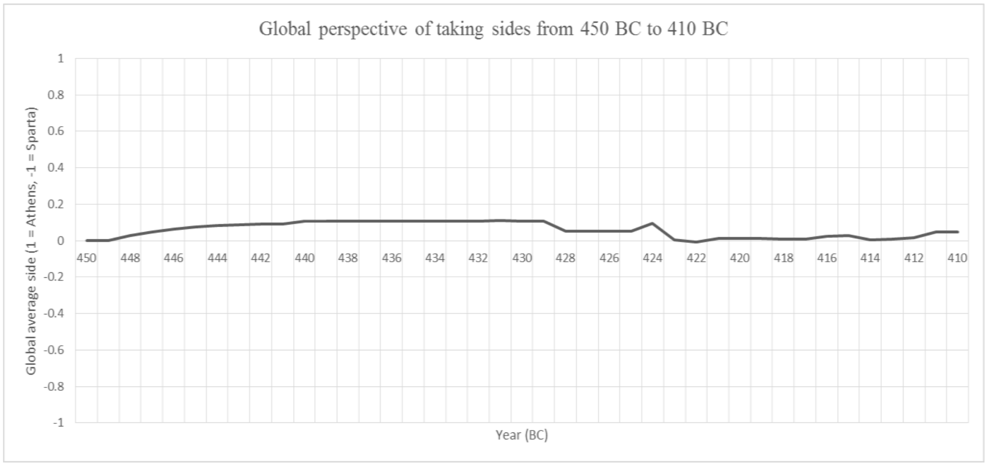


Figure 14 Estimated overall alliances during the war

To verify whether or not HDCA was successful in representing this model, certain statements from the accounts (Demand 2013) are compared to the data to see if they conflict or complement each other. Additionally, the experiments looked for inconsistencies that did not appear correct to general logic.

Figure 12 shows an estimate that the population size of Sparta was mostly static during the time frame of the simulation, only decreasing around 431 BC as conflicts began. In comparison, the size of Athens' population was much larger and changes to its size are easier to see. It decreases slightly and at a constant rate at first, but after 444 BC shows a constant rate of growth; there is no reason for this other than the city's size, but this does not explain the decrease that happened previously. A sharp decrease in population size occurs in 431 BC, at the time when conflicts were in full effect. By 427 BC the size of Athens stabilizes a bit, which can be explained by the occurrence of fewer fights (in comparison to the years prior) and the use of Athens' other cities to represent it during the war, an assumption made that is better suited with the use of Heritage. However, the plague that hits Athens three times (in 430, 429 and 427 BC) do not appear to be represented properly in these results, suggesting an error in implementation.

Figure 13 shows civil unrest in the two main cities, here a term describing the change in population size, where greater levels of unrest represent a decrease in population size from the previous year, and low levels of unrest represent an increase in population size. That Athens shows a general decrease (to levels of 0.0) of unrest when Sparta shows a fairly consistent increase suggests that the size of Athens made it more likely that new individuals would live there, and this unrest in Sparta is likely similar to what occurred in other cities. Athens and Sparta both rise to high levels of unrest when conflicts occur, and here effects of the plague on Athens are visible. Consider Sparta's attempts to call a truce with Athens in 425, 423 and 421: Athens' high unrest in 425 may relate to its decline of a truce in 425, and its decrease in unrest in 423 and 421 may explain why Athens was more open to a truce. However, this does not explain Sparta's motivation, where it shows unrest at maximum levels during most of this time (this may be attributed to the statistical occurrence of more people coming to live in Athens over time).

Figure 14 shows a summary of the other cities and how they sided with Athens or Sparta during this time period. This is represented as an average value where all cities have equal representation. Favor appears to be mostly on the side of Athens until 423 BC, where the favor between Athens and Sparta is mostly neutral. For the entire time period, favor for Athens was not especially high, suggesting that both sides had significant influence, a sign that the simulation behaves appropriately. The decrease around 423 could have resulted from Athens starting battles and losing, which in the implemented code means the target city would have reduced trust for Athens. The lower levels of influence of Athens from 423 BC onwards also corresponds to when they agree to peace treaty offers from Sparta, suggesting again that the simulation behaves appropriately and that this may have been a factor during this event.

In comparison, could it be possible to make a model like this with Multi-Population Cultural Algorithm (MPCA)? This would mean individuals could in theory still store past history of where they lived when they migrate, but this would not connect to current feelings in each city. The evolution of Friendship values between cities seems dependent largely on those individuals and their migration, and storing past thoughts of a city's beliefs would add further complexity than connecting to the city's most recent belief. It seems impossible to implement such a model without using weighted connections between the cities to represent Friendship and without using the individuals to connect them, which ultimately leads to what Heritage is all about.

In conclusion, the implementation of HDCA for the Peloponnesian War had some issues with larger cities gaining population size and smaller cities decreasing, an occurrence that may be realistic but may also be remedied in code should one desire in future implementations. The significant size of Athens compared to the other cities made it more difficult to compare with Sparta for those reasons. Otherwise, initial results of this simulation suggest that factors like change in population size and influence on other cities can effect political choices such as conflict or calling for peace. This is a big assumption, and not much else can be said with these results without more data from the real time period, but HDCA appears to be acceptable here as a simulation algorithm.

CHAPTER 6

Conclusions and Retrospective

6.1 Concluding Remarks of Previous Chapters

Chapter 3 discussed in detail the implantation and reasoning of Heritage for use with existing Multi-Population Evolutionary Algorithms such as Multi-Population Cultural Algorithms (MPCA). Heritage is a weight-based component passed down through successive generations of individuals to allow variety in them, as opposed to uniform status of belonging to a single population at any time. This is described by example in the use of Heritage-Dynamic Cultural Algorithm (HDCA).

Chapter 4 uses HDCA against MPCA, Cultural Algorithm (CA) and Genetic Algorithm (GA) with Single-Objective Real Optimization Problems, both in traditional static versions and un-traditional dynamic versions. Against our hypothesis, GA outperformed all of these “improved” algorithms in the static problems, but overall CA performed well in both static and dynamic functions to be the winner of the four. HDCA showed an interesting property of learning in dynamic functions that the other algorithms did not. Due to this and its ability to have varied individuals instead of individuals with strict singular goals, HDCA generally outperformed MPCA (technically H-MPCA, a most recent form of MPCA that was tested against similar problems) in static functions and at a greater level in dynamic functions, on both a group-evolution and best-individual level. When comparing the best-individuals, HDCA even compared favorably with CA, making HDCA an appropriate algorithm for these types of problems and worth the effort to optimize further.

Chapter 5 used HDCA in a detailed test-case of the Peloponnesian War to show its capabilities for simulation modeling. Heritage was a useful way to represent the relationships of the cities with each other and the people with the cities, and a way to use the individuals to represent the interactions and relationships between the cities. Implementing any model to simulate real events can be challenging, and both due to implementation problems and lack of further information on the events, the resulting simulation both provided minimal insight. Regardless, this in-depth example shows how HDCA can be used to build a simulation model and that Heritage is appropriate for specific types of models to accomplish this, a type of problem Evolutionary Algorithms typically do not focus on.

6.2 Future Improvements and Directions

Initially, it was intended for HDCA to be tested in an agent-based model. The difference between an agent-model and a simulation as seen in Chapter 5 is that an agent typically explores and reacts to an environment in a spatial perspective, not just other individuals. For HDCA to be appropriate for an agent-model, the model must require individuals that can have a combination of pre-defined properties that affect the decisions and actions of the individual, and the model must have time-based data to allow for successive generations and updating Heritage. After months of research, it still proved too difficult to find an appropriate test-case example to verify HDCA and obtain the respective data. However, HDCA has great potential as an appropriate algorithm to maintain complex agent models in a simplified manner. Future opportunities may use HDCA in situations where verification against real-world data is not significant to build a seemingly complex and realistic world, for example crowds of dynamic and unique people in the background of computer games without the need to define each person meticulously.

In Chapter 3, it was explained that HDCA can represent Heritage as either a uni-directional “Family Tree” or as a simplified Heritage set of fixed size. The examples of HDCA for both Chapters 4 and 5 use a set instead of a tree, so this thesis does not include an explicit example implementation for a tree. This is partly because the benefit of using a tree over a set (the ability to trace the history of the Heritage’s generation) was not important for the problems addressed, for example, understanding *why* a concluding answer was reached rather than the answer itself. Researchers may wish to search for problems where that level of detail is helpful; otherwise, this theoretical implementation method may continue to go unutilized.

Some of the preceding might be taken to suggest that the uses for HDCA are few and uncertain. It could be predicted that “big data,” where data is collected everywhere at many instances of time from many people and places, will become a significant source of new problems where HDCA can be used. Future issues on privacy and maintenance need to be overcome to ensure this type of data is widely available to researchers interested in experimenting with new solutions and problems; otherwise, research and applications with this type of data will progress very slowly.

Other improvements can be made towards using HDCA for feature selection problems or classification problems. While this text has shown that HDCA has interesting properties of learning, there are other types of problems for knowledge and understanding that should be researched further. Specifically the evolution of Heritage and the ability to define new populations as combinations of existing Heritage sets. If this aspect is considered, HDCA can be an appropriate algorithm for the study of cultural evolution and the origin of cultural aspects in the same species. This relates to the study of using HDCA for simulation rather than optimization problems with fitness functions, a direction that Evolutionary Algorithmic research will continue to take in the future.

REFERENCES

ALAMI, J., EL IMRANI, A., AND BOUROUMI, A. 2007. A multipopulation cultural algorithm using fuzzy clustering. *Applied Soft Computing 7*, 2, 506–519. Elsevier.

ALI, M. Z., ALKHATIB, K. AND TASHTOUSH, Y. 2013. Cultural algorithms: emerging social structures for the solution of complex optimization problems. *International Journal of Artificial Intelligence*, vol. 11, A13, 20—42.

BACK, T. 1996. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press.

BLACKWELL, T. AND BRAKE, J. 2004. Multi-swarm optimization in dynamic environments. *Applications of evolutionary computing*, Springer, 489-500.

CHANG, P.-C., CHEN, S.-H. AND LIU, C.-H. 2007. Sub-population genetic algorithm with mining gene structures for multiobjective flowshop scheduling problems. *Expert systems with applications*, vol. 33, 3, 762—771. Elsevier.

CHEN, S.-H. AND YEH, C.-H. 2001. Evolving traders and the business school with genetic programming: A new architecture of the agent-based artificial stock market. *Journal of Economic Dynamics and Control*, vol. 25, 3, 363—393. Elsevier.

COELLO COELLO, C. A. AND BECERRA, R. L. 2004. Efficient evolutionary optimization through the use of a cultural algorithm. *Engineering Optimization*, VOL. 36, 2, 219—236. Taylor & Francis.

COHOON, J. P., MARTIN, W. N. AND RICHARDS, D. S. 1991. A Multi-Population Genetic Algorithm for Solving the K-Partition Problem on Hyper-Cubes. *ICGA*, vol. 91, 244—248.

DA SILVA, D. J. A., AND DE OLIVEIRA, R. C. L. 2009. A multipopulation cultural algorithm based on genetic algorithm for the MKP. *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, 1815—1816. ACM.

DEMAND, N. 2013. *A history of ancient Greece in its Mediterranean context.* Third edition. New York: Sloan publishing.

DIGALAKIS, J. G., AND MARGARITIS, K. G. 2002. A multipopulation cultural algorithm for the electrical generator scheduling problem. *Mathematics and Computers in Simulation 60*, 3, 293–301.

GILBERT, N. AND TERNA, P. 2000. How to build and use agent-based models in social science. *Mind & Society*, vol. 1, 1, 57—72. Springer.

GRADY, S. A., HUSSAINI, M. Y. AND ABDULLAH, M. M. 2005. Placement of wind turbines using genetic algorithms. *Renewable energy*, vol. 30, 2, 259—270. Elsevier.

GUO, Y.-N., CHENG, J., CAO, Y.-Y., AND LIN, Y. 2011. A novel multi-population cultural algorithm adopting knowledge migration. *Soft computing 15*, 5, 897–905. Springer.

GUO, Y.-N., AND LIU, D. 2011. Multi-population cooperative particle swarm cultural algorithms. In *Natural Computation (ICNC), 2011 Seventh International Conference on*, vol. 3, IEEE, 1351–1355.

HLYNKA, A. W. AND KOBTI, Z. 2013. Knowledge Sharing Through Agent Migration with Multi-Population Cultural Algorithm. In *The Twenty-Sixth International FLAIRS Conference*.

HOLLAND, J. H. 1992. *Adaption in Natural and Artificial Systems*. MIT Press, Cambridge, MA.

JONES, G. 1998. Genetic and evolutionary algorithms. *Encyclopedia of Computational Chemistry*, Wiley Online Library.

JUANG, C.-F. 2004. A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 34, 2, 997—1006. IEEE.

KARABOGA, D., AKAY, B. AND OZTURK, C. 2007. Artificial bee colony (ABC) optimization algorithm for training feed-forward neural networks. *Modeling decisions for artificial intelligence*, 318—329. Springer.

KOBTI, Z., REYNOLDS, R. G. AND KOHLER, T. 2003. A multi-agent simulation using cultural algorithms: The effect of culture on the resilience of social systems. *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, vol. 3, 1988-1995. IEEE.

LIANG, J.J., GU, B.Y., AND SUGANTHAN, P.N. 2013. Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization. *Computational Intelligence Laboratory*, Zengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore.

MICHALEWICZ, Z. 1992. Genetic Algorithms + Data Structures = Evolution Programs. Springer, Berlin.

MIN, S.-H., LEE, J. AND HAN, I. 2006. Hybrid genetic algorithms and support vector machines for bankruptcy prediction. *Expert systems with applications*, vol. 31, 3, 652—660. Elsevier.

MOKOM, F., AND KOBTI, Z. 2014. Improving artifact selection via agent migration in multi-population cultural algorithms. In *Swarm Intelligence (SIS), 2014 IEEE Symposium on*, IEEE, 1–8.

NIAZI, M. AND HUSSAIN, A. 2011. Agent-based computing from multi-agent systems to agent-based models: a visual survey. *Scientometrics*, vol. 89, 2, 479—499. Akademiai Kiado, co-published with Springer Science+ Business Media BV, Formerly Kluwer Academic Publishers BV.

OMBUKI, B., ROSS, B. J. AND HANSHAR, F. 2006. Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, vol. 24, 1, 17—30. Springer.

OSTROWSKI, D., TASSIER, T., EVERSON, M. AND REYNOLDS, R. G. 2002. Using cultural algorithms to evolve strategies in agent-based models. *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, vol. 1, 741—746. IEEE.

PENG, B. AND REYNOLDS, R. G. 2004. Cultural algorithms: Knowledge learning in dynamic environments. *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 2, 1751—1758. IEEE.

QUINTERO, A. AND PIERRE, S. 2003. Sequential and multi-population memetic algorithms for assigning cells to switches in mobile networks. *Computer Networks*, vol. 43, 3, 247—261. Elsevier.

RAEESI N, M. R., KOBTI, Z. 2012. A knowledge migration-based multi-population cultural algorithm to solve job shop scheduling. In *The Twenty-Fifth International FLAIRS Conference*.

RAEESI, M., AND KOBTI, Z. 2013. Heterogeneous multipopulation cultural algorithm. *In Evolutionary Computation (CEC), 2013 IEEE Congress on, IEEE*, 292–299. IEEE.

RAEESI N, M. R., CHITTLE, J. AND KOBTI, Z. 2014. A new dimension division scheme for heterogeneous multi-population cultural algorithm. *In The Twenty-Seventh International Flairs Conference*.

REYNOLDS, R. G. 1994. An introduction to cultural algorithms. *In Proceedings of the Third Annual Conference on Evolutionary Programming*, 131—139. Singapore.

REYNOLDS, R. G. AND CHUNG, C. 1997. Knowledge-based self-adaption in evolutionary programming using cultural algorithms. *Evolutionary Computation, 1997, IEEE International Conference on*, 71—76. IEEE.

REYNOLDS, R. G. AND SALEEM, S. M. 2005. The impact of environmental dynamics on cultural emergence. *Perspectives on Adaptions in Natural and Artificial Systems*, 253—280.

REYNOLDS, R. G., SALAYMEH, A., O'SHEA, J. AND LEMKE, A. 2014. Using agent-based modeling and cultural algorithms to predict the location of submerged ancient occupational sites. *AI Matters*, vol. 1, 1, 12—14. ACM.

RUSSELL, S. AND NORVIG, P. 2010. *Artificial Intelligence: A Modern Approach (3rd edition)*. Pearson Education, Inc.

SYARIF, A., YUN, Y. AND GEN, M. 2002. Study on multi-stage logistic chain network: a spanning tree-based genetic algorithm approach. *Computers & Industrial Engineering*, vol. 43, 1, 299—314. Elsevier.

THIERENS, D. AND GOLDBERG, D. 1994. Convergence models of genetic algorithm selection schemes. *Parallel problem solving from nature-PPSN III*, 119—129. Springer.

XU, W., WANG, R., ZHANG, L., AND GU, X. 2012. A multipopulation cultural algorithm with adaptive diversity preservation and its application in ammonia synthesis process. *Neural Computing and Applications 21*, 6, 1129–1140. Springer.

YAO, X. AND LIU, Y. 1997. A new evolutionary system for evolving artificial neural networks. *Neural Networks, IEEE Transactions on*, vol. 8, 3, 694—713. IEEE.

ZHOU, G. AND GEN, M. 1999. Genetic algorithm approach on multi-criteria minimum spanning tree problem. *European Journal of Operational Research*, vol. 114, 1, 141—152. Else

APPENDICES

Appendix A

The following is additional data from the results of experiments performed in Chapter 4.

	HDCA		H-MPCA		CA		GA	
Function – Dynamic (Time Range)	Group Average Avg.	Best in Group Avg.	Group Average Avg.	Best in Group Avg.	Group Average Avg.	Best in Group Avg.	Group Average Avg.	Best in Group Avg.
Discus (0-100)	9.81E+08 **± 1.63E+09**	3.49E+09 **± 1.16E+09**	-5.68E+07 ± 3.62E+07	2.21E+08 ± 1.87E+04	**4.01E+09** ± 1.15E+08	**4.92E+09** ± 7.63E+07	7.39E+07 ± 6.62E+07	2.57E+08 ± 7.50E+07
Discus (0-4)	-1.60E+09	-1.53E+05	-2.07E+09	-1.16E+05	**-1.33E+07**	-9.48E+05	-3.72E+07	**-7.29E+04**
Discus (5-9)	-8.24E+07	-1.09E+04	-7.37E+05	-8.01E+03	**-6.79E+05**	-5.99E+05	-1.63E+06	**-7.93E+03**
Discus (10-14)	3.42E+09	6.73E+09	5.00E+07	1.18E+08	**7.51E+09**	**9.77E+09**	1.01E+08	2.40E+08
Discus (15-19)	5.04E+09	8.58E+09	2.72E+08	4.83E+08	**9.93E+09**	**9.94E+09**	6.20E+08	9.78E+08
Discus (20-24)	-1.75E+09	**-7.77E+05**	**-3.70E+08**	-3.32E+07	-1.69E+09	-5.36E+06	-5.00E+08	-2.65E+08
Discus (25-29)	-2.76E+08	**-1.57E+04**	-3.69E+08	-2.88E+04	**-5.75E+05**	-3.14E+05	-6.88E+07	-1.25E+07
Discus (30-34)	2.46E+09	6.14E+09	3.92E+08	4.94E+08	**7.46E+09**	**9.69E+09**	1.27E+08	2.79E+08
Discus (35-39)	3.38E+09	8.26E+09	4.91E+08	4.94E+08	**9.88E+09**	**9.91E+09**	6.62E+08	9.84E+08
Discus (40-44)	-9.41E+08	**-3.19E+06**	**-3.76E+08**	-3.32E+07	-1.71E+09	-5.12E+06	-5.20E+08	-2.89E+08
Discus (45-50)	-2.19E+07	**-1.93E+04**	-3.61E+08	-4.44E+04	**-2.71E+05**	-2.18E+05	-7.65E+07	-1.87E+07
Elliptic (0-100)	2.13E+08 **± 3.90E+08**	9.71E+08 **± 3.16E+08**	1.28E+08 ± 7.97E+07	4.07E+08 ± 1.22E+08	**1.22E+09** ± 4.99E+07	**1.55E+09** ± 4.57E+07	2.02E+07 ± 2.03E+07	7.26E+07 ± 2.25E+07
Elliptic (0-4)	-5.06E+08	-2.07E+07	-9.26E+08	-5.31E+07	**-5.66E+07**	**-9.25E+06**	-8.21E+07	-1.83E+07
Elliptic (5-9)	-3.09E+07	-4.19E+06	-4.51E+08	-3.16E+07	**-4.48E+06**	**-2.12E+06**	-7.16E+06	-3.59E+06
Elliptic (10-14)	6.19E+08	1.83E+09	9.53E+08	2.59E+09	**2.25E+09**	**3.02E+09**	2.76E+07	6.03E+07
Elliptic (15-19)	1.01E+09	2.48E+09	2.90E+09	3.14E+09	**3.24E+09**	**3.28E+09**	1.53E+08	2.36E+08
Elliptic (20-24)	-4.70E+08	**-4.89E+07**	-3.05E+09	-2.00E+09	-6.83E+08	-8.21E+07	**-1.23E+08**	-6.91E+07
Elliptic (25-29)	-1.49E+08	-1.52E+07	-3.07E+09	-2.00E+09	**-9.99E+06**	**-3.90E+06**	-1.88E+07	-5.78E+06
Elliptic (30-34)	4.84E+08	1.40E+09	**3.09E+09**	**3.12E+09**	2.23E+09	3.00E+09	3.65E+07	7.53E+07
Elliptic (35-39)	8.52E+08	1.96E+09	3.11E+09	3.12E+09	**3.24E+09**	**3.28E+09**	1.75E+08	2.55E+08
Elliptic (40-44)	-3.33E+08	**-4.73E+07**	-3.12E+09	-3.07E+09	-6.85E+08	-9.12E+07	**-1.42E+08**	-8.19E+07
Elliptic (45-50)	-6.10E+07	-9.54E+06	-3.11E+09	-3.07E+09	**-1.18E+07**	**-4.87E+06**	-2.73E+07	-9.02E+06
Ackley (0-100)	3.61E-01 **± 1.25E+00**	**1.97E+00 ± 1.53E+00**	7.52E-02 ± 2.28E-02	1.69E-01 ± 3.72E-02	**4.30E-01** ± 2.43E-02	8.07E-01 ± 7.28E-02	3.27E-01 ± 4.02E-02	9.05E-01 ± 6.20E-02
Ackley (0-4)	-2.14E+01	-2.07E+01	-2.15E+01	-2.08E+01	**-2.12E+01**	**-2.06E+01**	-2.14E+01	**-2.06E+01**
Ackley (5-9)	-2.10E+01	-2.03E+01	**-2.09E+01**	-2.04E+01	**-2.09E+01**	-2.05E+01	-2.11E+01	**-2.02E+01**
Ackley (10-14)	2.15E+01	2.21E+01	2.12E+01	2.15E+01	**2.19E+01**	**2.22E+01**	**2.19E+01**	**2.22E+01**
Ackley (15-19)	2.16E+01	2.22E+01	**2.22E+01**	**2.23E+01**	2.21E+01	**2.23E+01**	2.20E+01	**2.23E+01**
Ackley (20-24)	**-2.12E+01**	**-2.04E+01**	-2.22E+01	-2.21E+01	-2.14E+01	-2.08E+01	-2.14E+01	-2.06E+01
Ackley (25-29)	**-2.04E+01**	**-1.88E+01**	-2.22E+01	-2.21E+01	-2.10E+01	-2.04E+01	-2.11E+01	-2.03E+01
Ackley (30-34)	2.10E+01	2.20E+01	**2.23E+01**	**2.23E+01**	2.19E+01	2.22E+01	2.19E+01	2.22E+01
Ackley (35-39)	2.15E+01	2.21E+01	**2.23E+01**	**2.23E+01**	2.21E+01	**2.23E+01**	2.20E+01	**2.23E+01**
Ackley (40-44)	**-2.08E+01**	**-1.87E+01**	-2.22E+01	-2.22E+01	-2.14E+01	-2.08E+01	-2.14E+01	-2.07E+01
Ackley (45-50)	**-1.97E+01**	**-1.69E+01**	-2.22E+01	-2.22E+01	-2.10E+01	-2.05E+01	-2.11E+01	-2.03E+01
Griewank (0-100)	6.60E-01 **± 1.30E+00**	4.50E+00 **± 1.26E+00**	3.73E-01 ± 5.58E-01	1.14E+00 ± 5.47E-01	**5.06E+00** ± 4.04E-01	**7.82E+00** ± 5.90E-01	3.75E-02 ± 1.82E-01	4.15E-01 ± 2.00E-01
Griew. (0-4)	-6.26E+00	-2.63E+00	-7.73E+00	-3.50E+00	-3.75E+00	**-1.96E+00**	**-3.61E+00**	-2.26E+00
Griew. (5-9)	-2.09E+00	-1.37E+00	-4.20E+00	-2.28E+00	-1.89E+00	**-1.32E+00**	**-1.59E+00**	-1.33E+00
Griew. (10-14)	3.10E+00	9.51E+00	5.33E+00	7.30E+00	**1.10E+01**	**1.55E+01**	1.66E+00	1.96E+00
Griew. (15-19)	6.07E+00	1.15E+01	1.22E+01	1.35E+01	**1.77E+01**	**2.06E+01**	2.34E+00	2.74E+00
Griew. (20-24)	-4.70E+00	-2.35E+00	-1.30E+01	-1.13E+01	-7.05E+00	-3.80E+00	**-2.18E+00**	**-1.84E+00**
Griew. (25-29)	-1.69E+00	**-1.06E+00**	-1.32E+01	-1.13E+01	-2.26E+00	-1.50E+00	**-1.60E+00**	-1.36E+00
Griew. (30-34)	2.78E+00	9.30E+00	**1.34E+01**	1.36E+01	1.11E+01	**1.56E+01**	1.77E+00	2.06E+00
Griew. (35-39)	5.43E+00	1.16E+01	1.35E+01	1.37E+01	**1.75E+01**	**2.01E+01**	2.48E+00	2.88E+00
Griew. (40-44)	-3.37E+00	**-1.81E+00**	-1.35E+01	-1.32E+01	-7.08E+00	-3.91E+00	**-2.30E+00**	-1.99E+00
Griew. (45-50)	-1.79E+00	**-8.89E-01**	-1.35E+01	-1.32E+01	-2.37E+00	-1.52E+00	**-1.73E+00**	-1.47E+00

	HDCA		H-MPCA		CA		GA	
Function – Dynamic (Time Range)	Group Average Avg.	Best in Group Avg.	Group Average Avg.	Best in Group Avg.	Group Average Avg.	Best in Group Avg.	Group Average Avg.	Best in Group Avg.
Rastrigin *(0-100)*	3.64E+03 **± 6.42E+03**	1.77E+04 **± 5.04E+03**	1.15E+03 ± 2.64E+03	4.21E+03 ± 2.78E+03	**2.03E+04** ± 1.55E+03	**3.14E+04** ±2.26E+03	1.37E+02 ± 8.98E+02	1.53E+03 ± 9.43E+02
Rastrig.(0-4)	-2.27E+04	-7.80E+03	-2.74E+04	-9.97E+03	-1.17E+04	**-4.34E+03**	**-1.06E+04**	-5.17E+03
Rastrig.(5-9)	-5.83E+03	-2.76E+03	-1.35E+04	-5.32E+03	-3.88E+03	**-1.46E+03**	**-2.71E+03**	-1.79E+03
Rastrig.(10-14)	9.42E+03	3.77E+04	1.71E+04	2.51E+04	**3.98E+04**	**5.79E+04**	3.01E+03	4.01E+03
Rastrig.(15-19)	2.12E+04	4.47E+04	4.38E+04	4.85E+04	**6.72E+04**	**7.86E+04**	5.91E+03	7.41E+03
Rastrig.(20-24)	-1.35E+04	-5.62E+03	-4.74E+04	-3.87E+04	-2.49E+04	-1.19E+04	**-5.34E+03**	**-4.26E+03**
Rastrig.(25-29)	-4.00E+03	**-1.62E+03**	-4.80E+04	-3.87E+04	-5.45E+03	-2.15E+03	**-2.93E+03**	-2.18E+03
Rastrig.(30-34)	1.31E+04	3.58E+04	**4.82E+04**	4.90E+04	4.16E+04	**5.94E+04**	3.52E+03	4.60E+03
Rastrig.(35-39)	2.37E+04	4.41E+04	4.83E+04	4.94E+04	**6.76E+04**	**7.83E+04**	6.55E+03	8.08E+03
Rastrig.(40-44)	-1.41E+04	-6.50E+03	-4.88E+04	-4.77E+04	-2.50E+04	-1.20E+04	**-5.93E+03**	**-4.74E+03**
Rastrig.(45-50)	**-2.57E+03**	**-9.08E+02**	-4.88E+04	-4.77E+04	-5.46E+03	-2.08E+03	-3.35E+03	-2.52E+03
Hybrid01 *(0-100)*	1.24E+66 **± 1.94E+66**	3.55E+66 **± 1.19E+66**	3.81E+65 ± 6.91E+65	1.22E+66 ± 7.22E+65	**3.97E+66** ± 1.27E+65	**4.90E+66** ± 9.03E+64	7.67E+64 ±4.42E+64	2.54E+65 ± 5.09E+64
Hyb01 (0-4)	-1.49E+66	**-1.07E+61**	-3.01E+66	-1.49E+63	**-1.19E+64**	-3.40E+62	-4.09E+64	-2.59E+61
Hyb01 (5-9)	-6.87E+64	**-7.74E+57**	-1.33E+66	-1.49E+63	**-6.69E+61**	-5.11E+61	-1.66E+63	-5.53E+58
Hyb01 (10-14)	2.21E+66	5.98E+66	3.04E+66	**9.27E+66**	**7.34E+66**	9.65E+66	9.80E+64	2.30E+65
Hyb01 (15-19)	3.41E+66	8.17E+66	8.69E+66	9.27E+66	**9.91E+66**	**9.91E+66**	6.09E+65	9.52E+65
Hyb01 (20-24)	-8.85E+65	**-1.66E+60**	-9.09E+66	-6.38E+66	-1.77E+66	-1.18E+64	**-4.72E+65**	-2.49E+65
Hyb01 (25-29)	-5.65E+64	**-9.45E+59**	-9.16E+66	-6.38E+66	**-5.33E+62**	-4.70E+62	-6.01E+64	-1.16E+64
Hyb01 (30-34)	3.81E+66	6.84E+66	**9.21E+66**	9.27E+66	7.32E+66	**9.60E+66**	1.21E+65	2.61E+65
Hyb01 (35-39)	5.19E+66	9.04E+66	9.26E+66	9.27E+66	**9.84E+66**	**9.89E+66**	6.54E+65	9.69E+65
Hyb01 (40-44)	-1.64E+66	-1.11E+64	-9.27E+66	-9.21E+66	-1.74E+66	**-3.91E+63**	**-5.18E+65**	-2.88E+65
Hyb01 (45-50)	-2.32E+65	**-7.56E+59**	-9.27E+66	-9.21E+66	**-7.82E+62**	-4.45E+62	-7.01E+64	-1.78E+64
Hybrid05 *(0-100)*	1.24E+71 **± 1.98E+71**	3.61E+71 ± 1.13E+71	3.38E+70 ± 1.81E+71	1.28E+71 **± 1.88E+71**	**3.99E+71** ± 1.15E+70	**4.91E+71** ± 7.70E+69	7.57E+69 ± 6.39E+69	2.51E+70 ± 7.20E+69
Hyb05 (0-4)	-1.38E+71	-6.44E+66	-2.99E+71	-2.11E+68	**-1.36E+69**	-2.66E+67	-2.82E+69	**-4.87E+64**
Hyb05 (5-9)	-8.41E+69	**-5.16E+63**	-1.51E+71	-2.11E+68	**-1.60E+67**	-1.58E+67	-1.69E+68	-8.09E+63
Hyb05 (10-14)	3.30E+71	5.68E+71	2.94E+71	8.73E+71	**7.40E+71**	**9.75E+71**	1.03E+70	2.35E+70
Hyb05 (15-19)	4.18E+71	7.95E+71	8.04E+71	8.73E+71	**9.91E+71**	**9.92E+71**	5.88E+70	8.84E+70
Hyb05 (20-24)	-2.68E+71	-1.22E+70	-8.49E+71	-4.78E+71	-1.70E+71	**-1.68E+68**	**-4.57E+70**	-2.40E+70
Hyb05 (25-29)	-9.44E+70	**-1.61E+64**	-8.60E+71	-4.78E+71	**-2.49E+67**	-1.85E+67	-5.56E+69	-1.06E+69
Hyb05 (30-34)	3.91E+71	7.00E+71	**8.66E+71**	8.73E+71	7.48E+71	**9.81E+71**	1.10E+70	2.49E+70
Hyb05 (35-39)	4.90E+71	8.56E+71	8.71E+71	8.73E+71	**9.94E+71**	**9.96E+71**	6.48E+70	9.82E+70
Hyb05 (40-44)	-1.55E+71	**-1.36E+67**	-8.72E+71	-8.60E+71	-1.73E+71	-5.76E+68	**-5.07E+70**	-2.76E+70
Hyb05 (45-50)	-1.98E+70	**-1.08E+65**	-8.73E+71	-8.60E+71	**-2.99E+67**	-1.87E+67	-6.71E+69	-1.42E+69
Composition01 *(0-100)*	2.96E+65 **± 3.73E+65**	7.04E+65 **± 2.58E+65**	6.58E+64 ± 2.01E+65	2.03E+65 ± 2.10E+65	**7.96E+65** ± 2.44E+64	**9.81E+65** ± 1.80E+64	1.50E+64 ± 1.10E+64	4.98E+64 ± 1.32E+64
Com01 (0-4)	-3.09E+65	-9.62E+59	-5.71E+65	-3.27E+62	**-3.75E+63**	-2.37E+62	-5.99E+63	**-1.41E+59**
Com01 (5-9)	-1.65E+64	**-1.32E+56**	-2.48E+65	-3.27E+62	**-7.20E+61**	-4.84E+61	-3.50E+62	-1.34E+58
Com01 (10-14)	5.38E+65	9.65E+65	5.48E+65	1.65E+66	**1.48E+66**	**1.94E+66**	1.91E+64	4.66E+64
Com01 (15-19)	8.42E+65	1.41E+66	1.56E+66	1.65E+66	**1.98E+66**	**1.98E+66**	1.17E+65	1.77E+65
Com01 (20-24)	-2.34E+65	**-7.12E+62**	-1.62E+66	-1.29E+66	-3.58E+65	-4.00E+63	**-9.01E+64**	-4.43E+64
Com01 (25-29)	-6.30E+63	**-6.44E+58**	-1.63E+66	-1.29E+66	**-1.12E+62**	-4.49E+61	-9.69E+63	-1.52E+63
Com01 (30-34)	6.40E+65	1.47E+66	**1.64E+66**	1.65E+66	1.51E+66	**1.97E+66**	2.15E+64	4.82E+64
Com01 (35-39)	9.89E+65	1.88E+66	1.65E+66	1.65E+66	**1.99E+66**	**1.99E+66**	1.22E+65	1.88E+65
Com01 (40-44)	-3.05E+65	**-2.35E+59**	-1.65E+66	-1.62E+66	-3.50E+65	-1.04E+63	**-9.48E+64**	-4.90E+64
Com01 (45-50)	-3.11E+64	**-1.42E+59**	-1.65E+66	-1.62E+66	**-1.78E+62**	-1.14E+62	-1.14E+64	-1.78E+63
Composition03 *(0-100)*	1.51E+64 **± 2.06E+64**	3.64E+64 ± 1.22E+64	3.56E+63 ± 1.39E+64	1.15E+64 **± 1.50E+64**	**3.97E+64** ± 1.35E+63	**4.88E+64** ± 9.95E+62	8.47E+62 ± 9.33E+62	2.73E+63 ± 1.01E+63
Com03 (0-4)	-1.54E+64	-2.51E+59	-2.84E+64	-5.69E+61	**-1.59E+62**	-1.31E+61	-2.90E+62	**-1.31E+59**
Com03 (5-9)	-7.00E+62	**-1.34E+56**	-1.26E+64	-5.69E+61	**-5.26E+60**	-3.72E+60	-1.55E+61	-5.17E+56
Com03 (10-14)	2.95E+64	6.97E+64	2.87E+64	8.74E+64	**7.25E+64**	**9.59E+64**	1.04E+63	2.52E+63
Com03 (15-19)	4.49E+64	8.48E+64	8.20E+64	8.74E+64	**9.89E+64**	**9.92E+64**	6.39E+63	9.71E+63
Com03 (20-24)	-1.11E+64	**-1.33E+58**	-8.60E+64	-6.05E+64	-1.72E+64	-3.73E+61	**-5.09E+63**	-2.74E+63
Com03 (25-29)	-9.02E+62	**-1.14E+58**	-8.64E+64	-6.05E+64	**-5.62E+60**	-5.16E+60	-7.21E+62	-1.61E+62
Com03 (30-34)	2.24E+64	4.74E+64	**8.67E+64**	8.74E+64	7.44E+64	**9.59E+64**	1.39E+63	2.98E+63
Com03 (35-39)	3.80E+64	7.63E+64	8.72E+64	8.74E+64	**9.87E+64**	**9.89E+64**	7.04E+63	1.02E+64
Com03 (40-44)	-1.48E+64	**-5.64E+60**	-8.73E+64	-8.67E+64	-1.76E+64	-8.26E+61	**-5.69E+63**	-3.23E+63
Com03 (45-50)	-3.40E+63	**-8.04E+57**	-8.73E+64	-8.67E+64	**-4.17E+60**	-3.05E+60	-9.17E+62	-2.38E+62

	HDCA		H-MPCA		CA		GA	
Function – Dynamic (Time Range)	Group Average Avg.	Best in Group Avg.	Group Average Avg.	Best in Group Avg.	Group Average Avg.	Best in Group Avg.	Group Average Avg.	Best in Group Avg.
Composition07 (0-100)	1.40E+66 **± 1.79E+66**	3.61E+66 **± 1.15E+66**	3.62E+65 ± 6.86E+65	1.22E+66 ± 8.13E+65	**3.98E+66** ± 1.24E+65	**4.90E+66** ± 8.64E+64	8.59E+64 ± 6.23E+64	2.79E+65 ± 7.07E+64
Com07 (0-4)	-1.68E+66	**-7.15E+61**	-2.73E+66	-3.41E+62	**-2.13E+64**	-2.38E+63	-3.81E+64	-8.50E+61
Com07 (5-9)	-1.27E+65	**-2.33E+59**	-1.13E+66	-3.41E+62	**-3.39E+62**	-2.28E+62	-1.76E+63	-3.11E+59
Com07 (10-14)	2.43E+66	6.02E+66	2.68E+66	8.75E+66	**7.58E+66**	**9.79E+66**	1.03E+65	2.43E+65
Com07 (15-19)	4.08E+66	9.44E+66	8.10E+66	8.75E+66	**9.91E+66**	**9.93E+66**	6.11E+65	9.15E+65
Com07 (20-24)	-1.17E+66	**-1.25E+62**	-8.51E+66	-5.34E+66	-1.70E+66	-3.64E+63	**-4.89E+65**	-2.74E+65
Com07 (25-29)	-8.05E+64	**-2.73E+59**	-8.62E+66	-5.34E+66	**-5.70E+62**	-4.76E+62	-6.44E+64	-1.52E+64
Com07 (30-34)	2.69E+66	5.83E+66	**8.69E+66**	8.75E+66	7.46E+66	**9.71E+66**	1.25E+65	2.68E+65
Com07 (35-39)	4.34E+66	7.68E+66	8.75E+66	8.75E+66	**9.92E+66**	**9.94E+66**	6.67E+65	1.01E+66
Com07 (40-44)	-1.58E+66	-4.26E+64	-8.75E+66	-8.74E+66	-1.72E+66	**-4.32E+63**	**-5.35E+65**	-3.03E+65
Com07 (45-50)	-1.30E+65	**-1.13E+60**	-8.75E+66	-8.74E+66	**-8.41E+62**	-5.24E+62	-8.64E+64	-2.08E+64

	HDCA		H-MPCA		CA		GA	
Function – Static (Time Range)	Group Average Avg.	Best in Group Avg.	Group Average Avg.	Best in Group Avg.	Group Average Avg.	Best in Group Avg.	Group Average Avg.	Best in Group Avg.
Discus (0-9)	7.15E+09 **± 7.54E+08**	9.99E+09 ± 1.74E+07	6.91E+09 ± 3.17E+08	**1.00E+10** ± 4.80E+06	**9.80E+09** ± 7.56E+07	9.90E+09 **± 5.56E+07**	9.67E+09 ± 8.31E+07	**1.00E+10** ± 5.30E+03
Discus (100)	9.48E+09 **± 1.54E+08**	**1.00E+10** ± 5.75E+03	**1.00E+10** ± 4.24E+03	**1.00E+10** ± 4.24E+03	9.99E+09 ± 5.77E+06	9.99E+09 **± 5.77E+06**	9.90E+09 ± 2.11E+07	**1.00E+10** ± 2.29E+02
Elliptic (0-9)	2.18E+09 **± 2.17E+08**	3.17E+09 ± 8.32E+07	2.00E+09 ± 1.80E+08	2.95E+09 **± 1.99E+08**	**3.13E+09** ± 5.24E+07	3.24E+09 ± 4.52E+07	3.09E+09 ± 4.69E+07	**3.27E+09** ± 3.92E+07
Elliptic (100)	3.07E+09 ± 7.57E+07	3.33E+09 ± 2.39E+07	3.08E+09 **± 2.23E+08**	3.08E+09 **± 2.23E+08**	**3.35E+09** ± 2.91E+06	**3.35E+09** ± 2.99E+06	3.32E+09 ± 7.24E+06	**3.35E+09** ± 5.80E+05
Ackley (0-9)	2.17E+01 ± 2.56E-02	2.22E+01 ± 3.27E-02	2.20E+01 ± 2.91E-02	2.22E+01 ± 3.41E-02	**2.21E+01** ± 1.51E-02	2.22E+01 ± 2.62E-02	2.19E+01 ± 2.27E-02	**2.23E+01** ± 2.70E-02
Ackley (100)	2.18E+01 **± 3.77E-02**	**2.23E+01** **± 2.33E-02**	**2.23E+01** ± 1.70E-02	**2.23E+01** ± 1.57E-02	**2.23E+01** ± 6.56E-03	**2.23E+01** ± 5.90E-03	2.21E+01 ± 1.88E-02	**2.23E+01** ± 3.68E-03
Griewank (0-9)	1.21E+01 ± 6.07E-01	1.77E+01 ± 9.59E-01	1.31E+01 **± 1.17E+00**	1.75E+01 **± 1.31E+00**	1.71E+01 ± 4.56E-01	2.01E+01 ± 6.82E-01	**1.90E+01** ± 6.17E-01	**2.13E+01** ± 7.31E-01
Griewank (100)	2.17E+01 ± 8.58E-01	2.48E+01 ± 5.56E-01	2.27E+01 **± 1.17E+00**	2.27E+01 **± 1.17E+00**	2.55E+01 ± 1.70E-01	2.56E+01 ± 2.17E-01	**2.58E+01** ± 1.52E-02	**2.60E+01** ± 5.38E-05
Rastrigin (0-9)	4.48E+04 ± 2.86E+03	6.87E+04 ± 4.41E+03	5.04E+04 **± 4.07E+03**	6.60E+04 **± 4.97E+03**	6.50E+04 ± 2.51E+03	7.70E+04 ± 3.34E+03	**7.13E+04** ± 2.02E+03	**8.02E+04** ± 2.18E+03
Rastrigin (100)	8.61E+04 ± 2.77E+03	9.63E+04 ± 2.36E+03	8.41E+04 **± 4.24E+03**	8.41E+04 **± 4.21E+03**	9.79E+04 ± 3.29E+02	9.84E+04 ± 4.25E+02	**9.90E+04** ± 1.63E+02	**1.00E+05** ± 0.00E+00
Hybrid01 (0-9)	6.52E+66 **± 1.24E+66**	9.99E+66 ± 6.81E+63	6.41E+66 ± 3.72E+65	9.83E+66 **± 1.58E+65**	**9.83E+66** ± 8.92E+64	9.94E+66 ± 5.06E+64	9.68E+66 ± 6.47E+64	**1.00E+67** ± 5.57E+63
Hybrid01 (100)	9.31E+66 ± 1.93E+65	**1.00E+67** ± 7.42E+59	9.55E+66 **± 3.82E+65**	9.83E+66 **± 1.58E+65**	**9.99E+66** ± 8.19E+63	9.99E+66 ± 8.02E+63	9.90E+66 ± 2.39E+64	**1.00E+67** ± 1.50E+51
Hybrid05 (0-9)	7.07E+71 **± 1.11E+71**	**1.00E+72** ± 4.02E+68	5.96E+71 ± 6.12E+70	9.83E+71 **± 1.96E+70**	**9.78E+71** ± 1.15E+70	9.90E+71 ± 7.52E+69	9.67E+71 ± 7.38E+69	**1.00E+72** ± 3.85E+68
Hybrid05 (100)	9.48E+71 ± 1.40E+70	**1.00E+72** ± 3.35E+59	8.66E+71 **± 1.00E+71**	9.83E+71 **± 1.96E+70**	**9.99E+71** ± 4.25E+68	9.99E+71 ± 4.25E+68	9.90E+71 ± 1.64E+69	**1.00E+72** ± 0.00E+00
Composition01 (0-9)	1.42E+66 ± 1.61E+65	**2.00E+66** ± 5.01E+63	1.28E+66 ± 7.89E+64	1.96E+66 ± 3.07E+64	**1.96E+66** ± 1.58E+64	1.98E+66 ± 1.19E+64	1.93E+66 ± 1.25E+64	**2.00E+66** ± 2.94E+59
Composition01 (100)	1.88E+66 ± 3.13E+64	**2.00E+66** ± 2.37E+58	1.92E+66 **± 7.17E+64**	1.96E+66 **± 3.07E+64**	**2.00E+66** ± 2.03E+63	**2.00E+66** ± 2.03E+63	1.98E+66 ± 3.19E+63	**2.00E+66** ± 0.00E+00
Composition03 (0-9)	7.18E+64 ± 1.15E+64	9.99E+64 ± 1.22E+62	6.05E+64 ± 5.33E+63	9.76E+64 ± 1.22E+63	**9.74E+64** ± 1.43E+63	9.86E+64 ± 1.23E+63	9.67E+64 ± 8.76E+62	**1.00E+65** ± 2.36E+61
Composition03 (100)	9.47E+64 ± 1.59E+63	**1.00E+65** ± 1.75E+57	9.12E+64 **± 6.91E+63**	9.76E+64 **± 1.22E+63**	**1.00E+65** ± 2.65E+61	**1.00E+65** ± 2.28E+61	9.90E+64 ± 2.52E+62	**1.00E+65** ± 0.00E+00
Composition07 (0-9)	7.32E+66 ± 3.95E+65	9.99E+66 ± 1.65E+64	5.85E+66 ± 4.86E+65	9.69E+66 ± 1.96E+65	**9.81E+66** ± 8.71E+64	9.91E+66 ± 6.30E+64	9.67E+66 ± 8.78E+64	**1.00E+67** ± 1.63E+63
Composition07 (100)	9.43E+66 ± 2.28E+65	**1.00E+67** ± 3.21E+58	8.89E+66 **± 7.76E+65**	9.69E+66 **± 1.96E+65**	**9.99E+66** ± 3.54E+63	9.99E+66 ± 3.46E+63	9.90E+66 ± 2.21E+64	**1.00E+67** ± 1.50E+51

Appendix B

The following is the program code (Java) of HDCA as used in Chapter 4.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Random;

public class HeritageAlgorithm{

    OptimizationFunction f;                      //fitness function

    ArrayList<double[]> population = new ArrayList();
    //agents, each a set of parameters that form the solution

    ArrayList<ArrayList<double[]>> populationTypes = new ArrayList();
    //x by 3 (solution parameter, influence value, score), belief spaces

    ArrayList<double[]> heritage = new ArrayList();

    int size;
    int numOfParameters;
    int max = 100;
    int min = 0;

    public HeritageAlgorithm_Op02(int size, int numOfParameters){
        this.size = size;
        this.numOfParameters = numOfParameters;
        Initialize();
    }

    public void Initialize(){
        //initialize population

        Random rand = new Random();
        for (int i = 0; i < size; i++){
            double [] newAgent = new double[numOfParameters];
            for (int j = 0; j < numOfParameters; j++){
                newAgent[j] = rand.nextDouble() * (max - min);
                //every parameter between 0 and 100
            }
            population.add(newAgent);
        }
        for (int i = 0; i < numOfParameters; i++){
            populationTypes.add(new ArrayList<>());
        }

        //initialize agents to belong to population
        for (int i = 0; i < size; i++){
            double [] newHeritage = new double[numOfParameters];
            for (int j = 0; j < numOfParameters; j++){
                newHeritage[j] = 0;
            }
            newHeritage[rand.nextInt(numOfParameters)] = 1;
            heritage.add(newHeritage);
        }

    }
```

```
public void SetFitness(OptimizationFunction newF){
    f = newF;
}

public void Update(){
    //update population "belief spaces"
    for (int i = 0; i < numOfParameters; i++){
        populationTypes.get(i).clear();
    }
    for (int i = 0; i < size; i++){
        for (int j = 0; j < numOfParameters; j++){
            if (heritage.get(i)[j] > 0){
                double[] newSolution = new double[3];
                newSolution[0] = population.get(i)[j];
                newSolution[1] = heritage.get(i)[j];
                newSolution[2] = f.CalculateValue(population.get(i));
                populationTypes.get(j).add(newSolution);
            }
        }
    }

    //sort
    ArrayList<double[]> sortedPopulation = population;
    Collections.sort(sortedPopulation, new HDCAComparator());

    //choose top n for "selection"
    double n = 0.1;
    ArrayList<double[]> newPopulation = new ArrayList();
    ArrayList<double[]> newHeritage = new ArrayList();
    ArrayList<double[]> sortedHeritage = new ArrayList();
    for (int i = 0; i < size; i++){
        for (int j = 0; j < size; j++){
            if (sortedPopulation.get(i) == population.get(j)){
                sortedHeritage.add(heritage.get(j));
                break;
            }
        }
    }
    for (int i = 0; i < size*n; i++){
        newPopulation.add(sortedPopulation.get(i));
        newHeritage.add(sortedHeritage.get(i));
    }

    //combine top n for "reproduction"
    Random rand = new Random();
    for (int i = (int)(size * n); i < size; i++){
        double [] newAgentHeritage = new double[numOfParameters];
        int parent1 = rand.nextInt((int)(size));
        int parent2 = rand.nextInt((int)(size));
        for (int j = 0; j < numOfParameters; j++){
            newAgentHeritage[j]
            = sortedHeritage.get(parent1)[j]*0.5 + sortedHeritage.get(parent2)[j]*0.5;
        }

        if (f.CalculateValue(sortedPopulation.get(parent1)) >=
            f.CalculateValue(sortedPopulation.get(parent2))){
            int largestHeritage = 0;
            for (int j = 0; j < numOfParameters; j++){
                if (sortedHeritage.get(parent1)[j] >
                    sortedHeritage.get(parent1)[largestHeritage])
                    largestHeritage = j;
            }
            newAgentHeritage[largestHeritage] += 1;
        }
        else{
            int largestHeritage = 0;
            for (int j = 0; j < numOfParameters; j++){
                if (sortedHeritage.get(parent2)[j] >
                    sortedHeritage.get(parent2)[largestHeritage])
                largestHeritage = j;
        }
        newAgentHeritage[largestHeritage] += 1;
    }

    //normalize heritage values
    double sumHeritage = 0;
    for (int j = 0; j < numOfParameters; j++){
        sumHeritage += newAgentHeritage[j];
    }
    for (int j = 0; j < numOfParameters; j++){
        newAgentHeritage[j] = newAgentHeritage[j] / sumHeritage;
    }

    //add new heritage to set
    newHeritage.add(newAgentHeritage);

    //define parameter values for agent
    double[] newAgent = new double[numOfParameters];
    for (int j = 0; j < numOfParameters; j++){
        if (newAgentHeritage[j] > 0){
            double sumHeritageParameter = 0;
            for (int k = 0; k < populationTypes.get(j).size(); k++){
                sumHeritageParameter
                += populationTypes.get(j).get(k)[1] * populationTypes.get(j).get(k)[2];
            }
            double choiceHeritageParameter = rand.nextDouble() * sumHeritageParameter;
            sumHeritageParameter = 0;
            for (int k = 0; k < populationTypes.get(j).size(); k++){
                if (choiceHeritageParameter < (populationTypes.get(j).get(k)[1] *
                    populationTypes.get(j).get(k)[2]) + sumHeritageParameter){
                    newAgent[j] = populationTypes.get(j).get(k)[0];
                    break;
                }
                else{
                    sumHeritageParameter += populationTypes.get(j).get(k)[1] *
                        populationTypes.get(j).get(k)[2];
                }
            }
        }
        else{
            newAgent[j] = rand.nextDouble() * (max - min);
        }
    }
    //add new agent to set
    newPopulation.add(newAgent);
}

//modify some for "mutation"
for (int i = (int)(size * n); i < size; i++){
    if (rand.nextDouble() < 0.5){
        for (int j = 0; j < numOfParameters; j++){
            newPopulation.get(i)[j] += ((rand.nextDouble() * 10) - 5);
        }
    }
}

//fix to ensure all parameters are between 0 and 100
for (int i = 0; i < size; i++){
    for (int j = 0; j < numOfParameters; j++){
        newPopulation.get(i)[j] = Math.max(min, newPopulation.get(i)[j]);
        newPopulation.get(i)[j] = Math.min(max, newPopulation.get(i)[j]);
    }
}

//replace old population with new one
population.clear();
population = newPopulation;
heritage.clear();
```

```
        heritage = newHeritage;
    }

    public double [] PrintTopAgent(){
        ArrayList<double[]> newPopulation = population;
        Collections.sort(newPopulation, new HDCAComparator());

        double average = 0;
        for (int i = 0; i < size; i++){
            average += f.CalculateValue(population.get(i));
        }
        average = average / (double)(size);

        double averageHeritageSizes = 0;
        for (int i = 0; i < size; i++){
            for (int j = 0; j < numOfParameters; j++){
                if (heritage.get(i)[j] > 0)
                    averageHeritageSizes += 1;
            }
        }
        averageHeritageSizes = averageHeritageSizes / (double)(size);
        double best = f.CalculateValue(newPopulation.get(0));
        double [] returnValues = {average,best};

        return returnValues;
    }

    class HDCAComparator implements Comparator<double[]>{
        public int compare(double[] a, double[] b){
            double aValue = f.CalculateValue(a);
            double bValue = f.CalculateValue(b);
            if (aValue > bValue)
                return -1;
            else if (aValue < bValue)
                return 1;
            else
                return 0;
        }
    }

}
```

VITA AUCTORIS

NAME:	Andrew William Hlynka
PLACE OF BIRTH:	Windsor, ON
YEAR OF BIRTH:	1992
EDUCATION:	Sandwich Secondary School, LaSalle, ON, 2010
	University of Windsor, B.CS. [Honours], Windsor, ON, 2014
	University of Windsor, M.Sc., Windsor, ON, 2016