Heritage-Dynamic Cultural Algorithm for Multi-Population Solutions

Andrew William Hlynka and Ziad Kobti
School of Computer Science
University of Windsor
Windsor, Ontario, Canada
hlynkaa@uwindsor.ca and kobti@uwindsor.ca

***Abstract*—Multi-Population Cultural Algorithms (MPCA) define a set of individuals, each belonging to one of a set of populations that determines the shared goal, behavior or knowledge space of the individual. The design of MPCA extends from Cultural Algorithms (CA), which in turn improves on Genetic Algorithms (GA). To date, all examples of MPCA restrict individuals to belong to a single population at any given time, which can limit search potential and ability to simulate scenarios inspired by observations in real life. This article adopts ancestral "Heritage" as a new paradigm to extend MPCA. We introduce the "Heritage-Dynamic Cultural Algorithm" (HDCA) to allow easier definition of heterogeneous individuals and encourage greater search potential. As a test case, HDCA is compared directly with versions of MPCA, CA and GA against single-objective numerical optimization functions to demonstrate these intentions and inspire its use for new applications.**

Keywords—cultural algorithms; multi-population; evolution; migration; heritage

I. Multi-Population Evolutionary Algorithms

Evolutionary Algorithms (EA) are a category that carry much interest in artificial intelligence and computational science. Generally, algorithms defined within this area are inspired by biological evolution and the concept of improving oneself over many successive generations [1]. While not used explicitly to simulate natural evolution, the ability to improve by using the best-existing individuals makes these algorithms suited for search and optimization. Genetic Algorithms (GA) are an example, representing individuals as a set of memes that allow combining and altering of traits [2]. Three stages repeat during this process: 1) best-performing individuals are selected to help define new offspring, 2) new individuals are produced using selected parents as templates, and 3) mutation allows minor modifications in ways not yet seen in the population.

Through these three steps of "selection," "reproduction" and "mutation," GA are capable of improving themselves against many types of problems with no explicit knowledge about the problem itself. However, researchers proposed that using problem-specific knowledge would lead to better results. Such a GA that makes use of learned and pre-defined knowledge is known as a Cultural Algorithm (CA). To accomplish this, a global belief-space for the entire population acts as a representation of best-found knowledge of all the individuals, and distributes this knowledge during the reproduction and mutation phases [3]. The exact type of knowledge used in programming logic can vary; some resources define five categories to describe past outcomes in specific situations ("situational"), problem-specific intuition ("domain"), past environmental changes ("historical"), sampling sub-areas to predict unknown data ("topographical"), and storing dynamic ranges of likely optimal search locations ("normative") [4]. The performance of CA depends greatly on the logic defined, and unlike GA, a CA is typically problem-specific and must be rewritten for use elsewhere.

The concept of "Multi-Population" (MP) is a fairly new area of research, but existing work has already defined examples of MPGA [5] and MPCA [6], among others [7, 8]. As the name suggests, it evolves multiple versions of an EA running concurrently alongside each other. However, there is much variance among MPCA as to how these subpopulations are defined and how they communicate to improve each other's performance.

Some examples of MPCA include, but are not limited to, PARallel-co-operating Cultural Algorithm (PARCA) [6], Multi-population Cultural Algorithm Adopting Knowledge Migration (MCAKM) [9], Multi-population Cooperative Particle Swarm Cultural Algorithm (MCPSCA) [10], Multi-population Cultural algorithm with Differential Evolution (MCDE) [11], Transfer-Agent Multi-Population Cultural Algorithm (TAMPCA) [12] and Heterogeneous Multi-Population Cultural Algorithm (H-MPCA) [13]. These each have different designs and features, some combining CA with other algorithms, some using a static set of populations versus a dynamically changing list of populations, and most having different solutions of knowledge sharing between populations. To date, no standard exists to help new programmers.

MP allows greater flexibility in EA, but it still has a common restriction seen in all implementations: at the time of this writing, there are examples where individuals can change the population they belong to, but it is not possible for an individual to belong to multiple populations at once. That is, different goals and knowledge definitions cannot be combined with each other. In this paper, we introduce a new paradigm called "Heritage" that is to MP what MP was to EA. The following sections define "Heritage-Dynamic Cultural Algorithm" (HDCA) as an algorithm that utilizes Heritage, and compares it against MPCA, CA and GA with a simple test case to showcase differences between them.

II. The Concept of Artificial "Heritage" in MPCA

A. Motivation

The term "Heritage" has not yet been defined as an artificial concept for programming usage. The term suggests elements and connections being passed down to children, which makes it an appropriate term alongside the words "Genetic" and "Cultural."

In this context, the term is meant to represent an individual's weighted connection to multiple populations, making up a "Heritage-Tree" of influence. This defines the amount of influence each population has on the individual.

As a conceptual example, consider a set of countries C, each with their own cultures and beliefs stored in their own belief spaces B, such that for every c in C, there exists one and only one b in B (from here, corresponding pairs are labeled as c_i and b_i). As with MPCA, there exists a set of individuals P where every p in P belongs to one and only one population (p_{ik} – p_{ij} belong to c_i), and successive generations are influenced by their corresponding belief spaces (p_{ij} – p_{ik} are affected by b_i and only b_i, and vice versa). So far, this describes an example of MPCA without Heritage.

MPCA with Heritage can be initialized in this same way, but when the selection and reproduction processes occur, Heritage assumes it possible for two or more parents from different populations to produce an offspring. Along with traits that normally define an individual being passed down, ties to countries the parents belonged to is also passed down. For example, if p_{1a} and p_{2a} are selected to produce an offspring individual p_{ia}, then p_{ia} would belong to both countries c_1 and c_2. This means both b_1 and b_2 affect p_{ia} and vice-versa.

Conceptually, this follows observations in society. Children are influenced by the cultures of their heritage as passed down by their parents. In implementation, further details are required. How does a child know which conflicting culture to follow? What if changes occur in a country's modern culture that differ from how the child originally perceived it? Is a connection's strength consistent as it is passed down, or should it reduce to simulate loss of culture?

Fig. 1 shows a visual example depicting the difference between HDCA and MPCA from an individual's perspective.

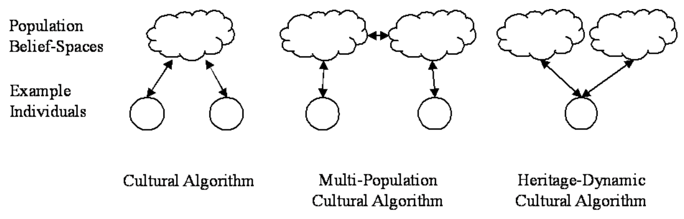
Population Belief-Spaces
Example Individuals
Cultural Algorithm
Multi-Population Cultural Algorithm
Heritage-Dynamic Cultural Algorithm

Fig. 1 – Visual difference between CA, MPCA and HDCA from an individual's perspective.

B. Artificial Heritage in Practice

To use Heritage requires that there be multiple distinct populations, for if there was only one then the purpose of combining traits of populations would be unnecessary. The general use of multiple populations implies some form of knowledge (or "culture") to be in use to differentiate them. Therefore, Heritage requires an MP-EA, and would be especially well-suited for a modified version of MPCA. This section describes such an example, here defined as "Heritage-Dynamic Cultural Algorithm" (HDCA), where the term MP is implied with the term "Heritage" and dropped for simplicity.

As described in the previous section, Heritage is stored in HDCA as a set of influence values each individual has with each population. This can be saved as a growing tree structure representing the history of each value, or to save on time and memory space, this information can be saved as an array of numbers of size equal to the number of populations, one array saved per individual.

These influence values are updated based on the influence values passed down from the parents of the individual. Further inspired by the real-life concepts inspiring it, these values are reduced in magnitude the further in the tree they occur. Additionally, these values are additive, so two ancestors with influence from the same population cause a child to have greater influence from that population. If using a set of values instead of a tree, the existing values passed down can be reduced by a factor before adding themselves to the new individual's Heritage, then normalizing all values to add up to 1.0 for usage and observation. It would be appropriate for an individual to be capable of having influence from new populations that did not affect its ancestors; the "mutation" stage of MPCA could be used to randomly modify an individual's Heritage with this in mind.

These influence values suggest that traits, knowledge, goals or behaviors from each population's belief space must be somehow compatible with each other, such that an individual can combine use of these elements accordingly. The exact definition and usage of knowledge in MPCA and CA is non-existent and dependent on the programmer and the problem being solved. Heritage offers an excuse to generalize knowledge to be comparable across different belief spaces. As a simple example, each belief space can store the best solutions found so far, and combining them can be a weighted average solution that the new individual uses. However, an alternative is to use the influence value and a statistical method (such as a "roulette wheel" approach) to choose which population to follow at a given time. This way, Heritage can be used in HDCA without major changes from existing MPCA.

As suggested so far, these influence values keep track of ties to a population's belief space. This means the population's belief space can update and change, and the individual would have no explicit memory of how the population used to be during their ancestor's time. Not only is this much easier for implementation and memory usage, this separates Heritage from the existing "historic" knowledge paradigm, which could still be used at the programmer's discretion.

TABLE I. COMPARISON OF HDCA AND RELATED EA

	GA	CA	MPCA	HDCA
Full name	Genetic Algorithm	Cultural Algorithm	Multi-Population Cultural Algorithm	Heritage-Dynamic Cultural Algorithm
Date first appeared	1975	1994	2002	2015
Basic structure	A set of solutions made of individual components combine traits into new solutions, producing better solutions through "selection," "reproduction" and "mutation."	Same as GA, but with a global belief-space that stores and spreads knowledge to be used during the reproduction of new individuals.	Multiple CA with communication channel between populations, either through individuals, belief-spaces, or a global belief-space.	CA with individuals that can be connected to multiple population belief-spaces. Connections are defined by ancestral Heritage.
Knowledge usage	Solution representation.	Solution representation, and situational, domain, historical, topographical, normative.	Solution representation, and situational, domain, historical, topographical, normative.	Solution representation, and situational, domain, historical, topographical, normative. (Heritage not a comparable knowledge type - it requires other knowledge to be appropriate to use)
Benefits	Easy to implement, easy to alter problem specifications.	Uses knowledge, likely to find better solutions than GA.	Able to represent multiple knowledge types / strategies concurrently, similar to parallel computing.	Fixes standardization issues of MPCA, can represent more complex individuals for different types of models.
Drawbacks	Relies on randomization rather than knowledge to solve a problem.	Knowledge is just as likely to find worse solutions, dependent of definition and programmer's knowledge of problem domain.	More reliant on knowledge definition than CA, and more difficult to implement due to undefined standards.	Individuals may require use of all population belief-spaces, resulting in worse time-complexity. Complexity of Heritage combinations not certain to reach better solutions.
Memory complexity	Number of individuals * size of solution format	Number of individuals * size of solution format + size of knowledge in belief-space	Number of individuals * size of solution format + size of knowledge * number of populations	Number of individuals * size of solution format * number of populations (Heritage set) + size of knowledge * number of populations
Time complexity	$O_1 = O$ (number of individuals) $O_2 = O$ (number made for new generation) $O(GA) = O_1 + O_2$	$O_3 = O$ (complexity of knowledge in belief-space) $O(CA) = O_1 + O_2 * O_3$	$O_4 = O$ (communication method between populations) $O_5 = O$ (number of populations) $O(MPCA) = O_1 + O_2 * O_3 + O_4 * O_5$	$O(HDCA) = O_1 + O_2 * O_3 * O_5$
Example problems	Scheduling, evolving structures, anything that can be evaluated with a fitness function with many parts.	Same as GA, plus social simulation.	Same as CA, particularly if multiple uses of knowledge from CA can be defined, plus more complex social simulation.	Same as MPCA, especially if different knowledge / strategies from MPCA can be combined, plus social simulation with heterogeneous individuals.

Since the use of Heritage allows a much greater variety of behaviors through combinations than existing MPCA, individuals in HDCA need not their own belief space or knowledge. They only require a current state representing an example solution to the problem, and a Heritage set of influence values to determine future solutions. This is a further generalization of MPCA made possible with Heritage.

Fig. 2 summarizes example code of HDCA for the purposes of the experiments in Sections III and IV.

C. *With or Without Heritage*

Table I provides a table to help compare HDCA, MPCA, CA and GA together. It shows that each successive algorithm is not necessarily more efficient, but the added complexity allows different representations, specifically for social simulation.

The purpose of Heritage is to extend the capabilities of MP-EA. In the same way MPCA is able to represent problem-solving capabilities not possible with CA alone, HDCA can represent problem-solving techniques not possible with MPCA alone. In situations where populations could be simplified as a combination of other populations, or cases where diversity is important, HDCA is appropriate. Heritage is also meant to be compatible with existing implementations of knowledge and strategies used in MPCA and CA.

This does not mean that HDCA would outperform MPCA, CA or GA for most traditional problems. Rather, the fact that HDCA would have individuals use combinations of population belief spaces complicates their behavior, and added complexity

```
public class HeritageAlgorithm{
  FitnessFunction f;
  ArrayList<double[]> individuals;
  ArrayList<double[][]> populationBeliefSpaces;
  ArrayList<double[]> heritage;

  public void Initialize(){
    Foreach individual i in individuals{
      Randomly assign solution values to define i;
      Randomly assign 1.0 to corresponding
        heritage set, to initialize belonging to
        one population;
    }
  }

  public void Update(){
    Foreach individual i in individuals{
      Update p in populationBeliefSpaces with goal
        parameter from i, fitness value of i
        against f, and i's influence on p from
        corresponding heritage;
      }
      Use top i's from individuals for "selection";
      Initialize ArrayList<double[]> newIndividuals
        from those chosen in "selection."
      newIndividuals are initialized heritage from
        parents. "Mutation" adds random population
        ties at rare intervals.
      Heritage influence values are used to
        determine how populationBeliefSpaces
        initialize solution values in
        newIndividuals;
      Update individuals and heritage accordingly,
        keeping top individuals from "selection."
    }
}
```

Fig. 2 – Pseudo-code for HDCA.

is typically a poor strategy in improving algorithmic design. What this added complexity does mean is a greater difference among all the individuals during the course of the algorithm – few individuals are likely to converge to the same solution, potentially improving search range. Additionally, the use of Heritage stores ties to previous populations longer when MPCA can lose such information quickly. In repeating dynamic environments where different populations perform better at different times (based on storing different types of information, goals or strategy), HDCA is hypothesized to have an edge thanks to its ability to retain this type of information.

III. Experiment Setup: Algorithms and Test Enivornment

A. *Single Objective Real-Parameter Numerical Optimization test functions*

The concept of Heritage and HDCA is new and untested, so the intention with introducing this paradigm is to test it against a simple, standardized testing environment. To accomplish this, test functions are borrowed from the CEC'14 Test Suite of Single Objective Real-Parameter Numerical Optimization Problems [14]. As the name suggests, these problems consist of mathematical functions with multiple input parameters and a single outcome value. With EA, this is a simple example of a problem that can quickly test a solution as a fitness function to guide the "selection" process of best-performing individuals.

The tests in this article are not comprehensive: a select number of functions were randomly chosen from the complete list from CEC's test suite. These range from "Unimodal," "Single Multimodal," "Hybrid" and "Composition" functions. The goal in these tests is to optimize the parameters to maximize the output of the functions. A search limit between 0 and 100 is used for each of ten parameters used as input. The exact functions used were Discus, High-Conditioned Elliptic, Ackely, Griewank, and Rastrigin, as well as Hybrid Functions 1 (Schwefel, Rastrigin, Elliptic) and 5 (Scaffer, HGBat, Rosenbrock, Schwefel, Elliptic), and Composition Functions 1 (Rosenbrock, Elliptic, Bent Cigar, Discus), 3 (Schwefel, Rastrigin, Elliptic) and 7 (Hybrid functions 1, 2 and 3) [14].

These functions would normally be tested as a static environment for the algorithm to optimize the input parameters. In addition to this, a simple dynamic version of the functions are also tested: in that scenario, the functions are multiplied by -1 every ten time-steps, to swap between positive and negative outputs and effectively switching the maximization problem to a minimization one. Even then, the algorithms seek to maximize the function output, reducing the ability to converge to a single solution over a long period of time.

B. *HDCA, H-MPCA, CA and GA*

HDCA is the example algorithm used in this experiment for the purposes of testing Heritage. MPCA is the target focus to compare against HDCA. However, there are very few examples in existing literature of MPCA being used against this otherwise common type of benchmark [13].

To represent the state-of-the-art of MPCA appropriate to compare with, Heterogeneous MPCA (H-MPCA) is used [13]. H-MPCA is described as an MPCA where each defined population explicitly seeks out unique goals (while not unusual to the concept of MPCA, this is one of the few examples that uses this as a requirement in implementation). The original paper introducing this algorithm uses very similar benchmark tests, with each population focused on optimizing a specific subset of parameters to the given function. Individuals perform a basic local-search for a small number of time-steps, afterwards the best-performing individuals in each population update a global belief-space with their parameter's values, and updating a new generation to be initialized with this global solution.

HDCA was designed to be similar to H-MPCA in this regard. HDCA divides the parameters to be optimized amongst each population, one population per parameter (alternatively, H-MPCA set multiple parameters per population). Each population has their own belief-space storing best-found solutions to update individuals. While H-MPCA does reduce the amount of new generations made through dedicating time for local-search, HDCA continuously replaces poor-performing individuals with a new generation, passing down combined Heritage of parent individuals to their children.

In addition to H-MPCA, CA is included in the test results. To implement a simple example that was also tested against a similar environment, the implementation used by Reynolds and Chung [15] is provided. This example uses situational and normative knowledge, represented by saving the best-found solutions so far and updating a range where better values are likely to be found based on existing individuals. Finally, a GA is included [2], here implemented by taking the best-performing 20% of individuals for the selection and reproduction process, with a 50% likelihood of individual parameters being altered by up to 5% the size of the given search space. The GA's values were arbitrarily chosen and not directly taken from another source, as GA is expected to perform poorly compared to the rest, and as there are far fewer implementation details to be concerned about with respect to the other EA.

Interestingly, very few resources directly compare MPCA with CA or GA. Instead, MPCA is commonly used for more complex problems that CA is assumed to be unsuited for.

IV. Results

Fig. 3 and 4 show an example simulation of the four algorithms against the static and dynamic Griewank function respectively. This is included to provide insight into the performance over time of each algorithm, which becomes important when discussing the performance of HDCA. Fig. 5 summarizes the normalized results from both the static and dynamic tests for the four algorithms. Each of the four algorithms ran against each of the ten benchmark functions ten times each, the average of which is used for the comparative results. For dynamic tests, the average shown is literally the average of each time-step taken over 100 time-steps, and for static the average of the first 10 time-steps and the end result after 100 time-steps are included. The algorithms are compared

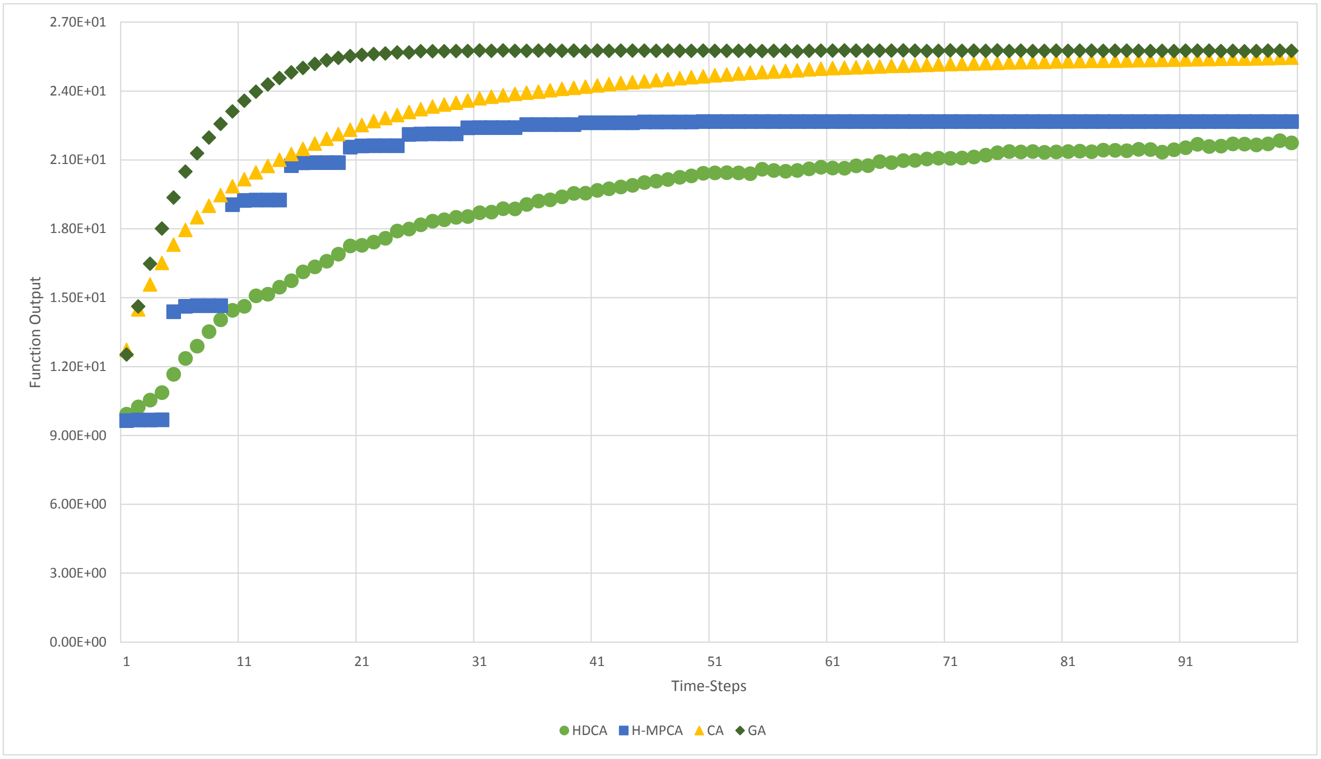

Fig. 3 – Example simulation of HDCA, H-MPCA, CA, and GA against the static Griewank function.

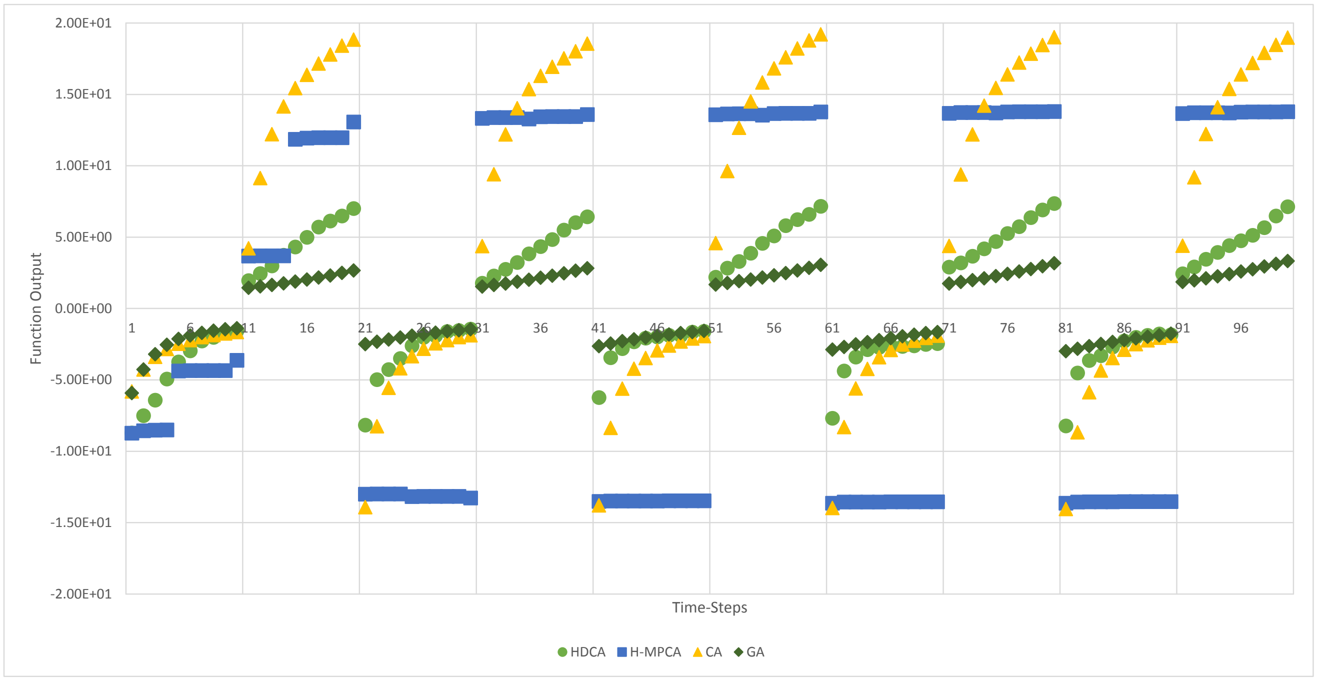

Fig. 4 – Example simulation of HDCA, H-MPCA, CA, and GA against the dynamic Griewank function.

with both the best-performing individual (a common property used to show an EA is capable of finding a solution) and the group's average (a sign of convergence and group learning, but also a sign of no more searching). The values provided are normalized between 0.0 and 1.0, representing the average results against the ten benchmark functions. Since the possible output of each function varies greatly in magnitude, the output values were compared with the best solution amongst the four algorithms for normalization, used for the average of all ten functions to show relative performance.

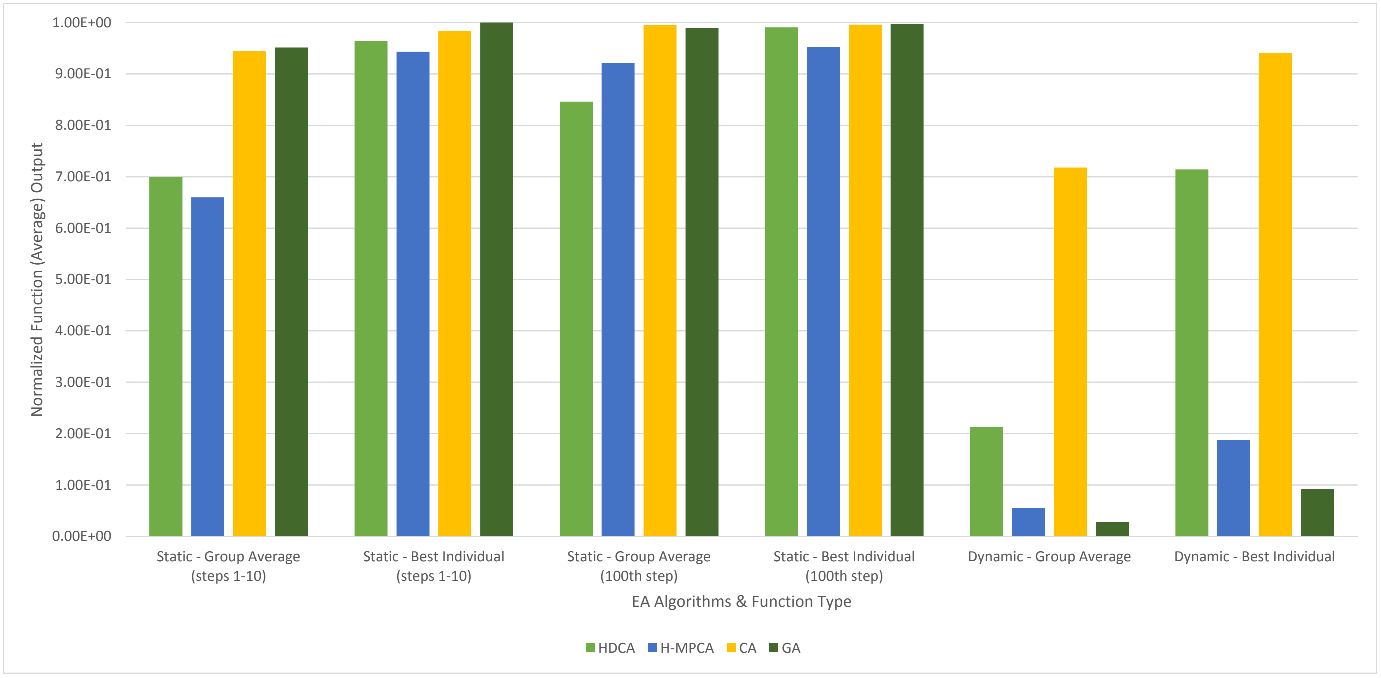

Fig. 5 – Summarized results against ten static and dynamic benchmark functions with HDCA, H-MPCA, CA, and GA.

The results after performing the test with static functions, such as commonly seen in other research articles, were surprising. GA gave the best performance, followed by CA, then H-MPCA and HDCA. This was true when recording both the single best individual in the algorithm and when observing the average of all individuals in the algorithm.

These static results are of the average taken from the first ten time-steps. Additionally recorded here is the state of each algorithm after 100 time-steps: the best-performing individuals barely improved, but the group average had. H-MPCA's static group average did surpass HDCA's, and CA's static group average barely outperformed GA's. The comparative results of the best-individuals remained similar, with HDCA, CA and GA all comparably similar, and GA being the best of the four. Also not recorded here is the standard deviation taken from the multiple runs of each benchmark function; HDCA had the largest range in deviation, followed by H-MPCA, while CA and GA remained fairly consistent. Some variation in performance occurred with specific benchmark functions, but no correlation presented itself.

The results from these static tests show that HDCA is generally an improvement over H-MPCA in both group evolution and individual evolution with limited time. H-MPCA does have potential to converge its individuals after longer periods of time, such that its group performance would beat HDCA. But the large difference between the group and individual averages in HDCA, both in the first ten time-steps and after 100 time-steps, suggest HDCA has greater search potential, preventing individuals from converging too quickly around a single solution. This matches with the algorithm's intended behavior, which led to it outperforming H-MPCA while using a similar knowledge structure in its belief-spaces.

As for the performance of CA against GA, this should be taken as an example showing that complicated intelligence does not always perform better. The design and usage of knowledge and strategy is crucial. This is also an example of the importance of testing all related algorithms for the target problem, no matter what the expected outcome might be.

The results from the dynamic-benchmarks were more interesting. GA was the worse of the four, converging during the negative phase and unable to break free to take advantage of the positive stages, and so its individuals stayed very near the 0.0 range. H-MPCA had similar problems, converging quickly near the beginning, and not showing any signs of change (for better or worse) after the first couple of environment updates.

HDCA and CA showed clear improvement over the other two, with CA being notably stronger. When analyzing individual function performance, performance in HDCA and CA is comparable during the negative stages, but CA exceeds the other algorithms during the positive stages. This is a sign that the specific knowledge usage in CA is appropriate for this previously-untested environment, as knowledge is able to quickly be replaced with recent and more relevant samples.

HDCA does show some strength in the negative stages, especially after successive repetition. Unlike the other algorithms, HDCA appears to accelerate its performance during the negative stages, as if it had learned or remembered where to find solutions appropriate for that period (see Fig. 4). This sign of improvement and self-learning is not so obvious during the positive stages, possibly because the negative stage occurred first in the simulation, but this nonetheless suggests that HDCA's other intended feature is true: HDCA is capable of retaining knowledge longer than other MPCA, making it better suited for changing problems with repetitive patterns. And again, the greater range between group average and best-performing individuals show the individuals in HDCA are hard-at-work searching for all possible solutions.

As a side note regarding execution time: these experiments were run on a Microsoft Surface Pro 3 with an Intel i3-4020Y 1.50GHz CPU, 4.00GB of RAM and Windows 8.1 64-bit OS. Most of the individual test functions were solved quickly, with the exception of the final function ("Composition07") taking the majority of time spent. The time spent for H-MPCA was 58 seconds, GA was 86 seconds, CA was 135 seconds, and HDCA was 180 seconds. The time performance of H-MPCA matches with the original author's intentions, improving on this example of GA due to not requiring the production of a new generation from existing individuals every time-step. HDCA's comparably poor performance can be explained by individuals accessing memory from multiple population belief-spaces.

Overall, CA showed itself to be the better choice of algorithm in this subset against both static and dynamic numerical optimization. HDCA provided evidence of its capabilities of memory retention and wider search ranges.

V. Conclusions and Future Work

Heritage is a suggested extension to MPCA to better allow the representation of complex combinations of population traits. For practical purposes, this allows individuals defined with Heritage to be more unique from one another, to retain ties that may prove useful in later stages of problem-solving, and have varied goals that lead to a wider search space.

The simple type of knowledge stored in HDCA for these experiments, essentially single solutions and their success rate stored in each belief space, is most comparable to H-MPCA and even GA. In theory, Heritage can be utilized with any knowledge that can be used in MPCA or CA, so it is possible to modify HDCA to use similar knowledge to what CA [15] used in Section IV. Whether or not this can match or exceed performance of CA is not verified, and is an appropriate next step in future experiments.

MPCA is normally used against complicated problems with greater restrictions or multiple goals. Future work would test HDCA against problems of similar complexity. But like MPCA, the use of Heritage can allow for representation and problem-solving that was not possible in prior versions of the algorithm, such as representing social networks and community detection, or representing complex but manageable definitions of individuals as a combination of populations. The authors of this paper are using this ability to combine behaviors from a limited set for easier implementation of character artificial intelligence for games and interactive experiences, and to better understand unknown details of migration of Native-Americans in history. The breadth and potential of Heritage requires extensive testing, experimentation and research in multiple fields.

The lack of standardization in MPCA has been a problem in implementation, and adding HDCA to the existing library can add further confusion. But Heritage requires many assumptions: populations each have unique belief-spaces, individuals can reproduce with each other across different populations, and individuals require no additional unique traits or behaviors other than what their Heritage set (combination) of population provides. If Heritage is not updated and begins with a single population, HDCA becomes a simple MPCA, but with many implementation questions answered. This type of standardization is important for the field of EA.

Acknowledgment

The work was partially supported by a research grant from NSERC Discovery and the University of Windsor.

References

[1] T. Back, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press. 1996.

[2] J. H. Holland, Adaption in Natural and Artificial Systems. MIT Press, Cambridge, MA. 1975.

[3] R. G. Reynolds, "An introduction to cultural algorithms." Proceedings of the Third Annual Conference on Evolutionary Programming. Sinapore. 1994, pp. 131-139.

[4] R. G. Reynolds, and S. M. Saleem, "The impact of environmental dynamics on cultural emergence." Perspectives on Adaptions in Natural and Artificial Systems. 2005, pp. 253-280.

[5] J. P. Cohoon, W. N. Martin, and D. S. Richards, "A multi-population genetic algorithm for solving the k-partition problem on hyper-cubes." ICGA, vol. 91. 1991, pp.244-248.

[6] J. G. Digalakis, and K. G. Margaritis, "A multipopulation cultural algorithm for the electrical generator scheduling problem." Mathematics and Computers in Simulation 60, 3. 2002, pp.293-301.

[7] A. Quintero, and S. Pierre, "Sequential and multi-population memetic algorithms for assigning cells to switches in mobile networks." Computer Networks, vol. 43, 3. Elsevier. 2003, pp.247-261.

[8] T. Blackwell, and J. Brake, "Multi-swarm optimization in dynamic environments." Applications of evolutionary computing. Springer. 2007, pp. 762-771.

[9] Y.-N. Guo, J. Cheng, Y.-Y. Cao, and Y. Lin, "A novel multi-population cultural algorithm adopting knwoeldge migration." Soft computing 15, 5. Springer. 2011, pp. 897-905.

[10] Y.-N. Guo, and D. Liu, "Multi-population cooperative particle swarm cultural algorithms." Seventh International Confernece on Natural Computation (ICNC), vol. 3. IEEE. 2011, pp 1351-1355.

[11] W. Xu, R. Wang, L. Zhang, and X. Gu, "A multipopulation cultural algorithm with adaptive diversity preservation and its application in ammonia synthesis process." Neural Computing and Applications 21, 6. Springer. 2012, pp.1129-1140.

[12] A. W. Hlynka, and Z. Kobti, "Knowledge sharing through agent migration with multi-population cultural algorithm." The Twenty-Sixth International FLAIRS Confernece. 2013.

[13] M. Raeesi, and Z. Kobti, "Heterogeneous multipopulation cultural algorithm." IEEE Congress on Evolutionary Computation (CEC). IEEE. 2013, pp 292-299.

[14] J. J. Liang, B. Y. Gu, and P. N. Suganthan. "Problem definitions nad 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. 2013.

[15] R. G. Reynolds, and C. Chung, "Knowledge-based self-adaption in evolutonary programming using cultural algorithms." IEEE International Conference on Evolutionary Computation. IEEE. 1997, pp. 71-76.