Xiaofan Liang

Using Evolutionary Principle to Improve Genetic Algorithm

Minerva Schools at KGI

INTRODUCTION

Human beings have been long interested in manipulating and even creating evolution. For example, we know how to breed different rice plants to find the best hybrid that produces more rice or alter the environment to give ourselves comparative advantages over other species on Earth. What amaze us about the evolution are its complex mechanisms that automate the process of adaptation to a potentially better outcome. If human beings can learn to create these mechanisms, this will enormously improve societal efficiency. In this proposal, I will draw on the past research on the applications of evolutionary principles in algorithm designs, the limitations of those designs, and the inspirations from nature to suggest an improvement on the traditional genetic algorithm.

HOW HAVE EVOLUTIONARY PRINCIPLES BEEN USED IN ALGORITHM DESIGN[1] [2]

Computer scientists are one of the groups that enjoy leveraging evolutionary mechanisms to develop algorithms that solve complex problems. For example, the Genetic algorithm (in short, GA) developed in the 1970s is a successful early attempt to map evolutionary concepts to optimization and searching problems. After taking in a population of potential solutions that are coded into chromosome-like string characters, GA models after evolution to select individuals based on fitness values (relative to the outcome) and produce a new generation that comes from the more fitted parents (e.g. through crossover and mutation). This process goes on until the

[1] #Algorithm: Analyze how well algorithms have adopted evolutionary principles and design an algorithm that draw in evolutionary principles and addresses the limitations to for better optimization.
[2] #Optimization: Evaluate how well genetic algorithms at performing optimizations and design solutions for better optimization

algorithm reaches the optimal (Shiffman, n.d.). It targets problems that usually have complex fitness landscape that is hard to optimize in a linear way.

This computational implementation of evolution raises interesting questions regarding how well evolutionary concepts work in the artificial world and vice versa. It's very clear that in nature, things don't work in discrete order: different individuals in the species experience various stages of selection and reproduction at the same time. Also, the fitness value of species evolves through reproductive rate rather than being assigned, which points to important distinctions between natural evolution and algorithm design. Evolution in nature achieves balance or optimality in an emerging way that is subjected to multiple dynamical processes (e.g. eco-relation or randomness), while the algorithm is intentional, goal-directed and exact in which we define the modes of selection through the fitness function (e.g. stabilizing selection means setting the average values as the better fit). These differences limit traditional genetic algorithm in the following ways: 1) GA can easily converge to a locally optimal solution due to the loss of diversity in GA population, 2) unit of evolution in GA has little agency at constructing the environment and interactions (which means things like niche construction and phenotypic plasticity don't apply), 3) GA do not scale with complexity since computer has limited memory and space to host the growing population after each generation, 4) the quality of GA depends very much on the parameters people set for it, such as the crossover and mutation rate, which is not very self-adaptive within the operation and 5) the parameters for GA are fixed, which make it not adaptive across various types of inputs and problems (Kumar, Husian, Upreti & Gupta, 2010).

GA's "brothers" and "sisters" in the evolutionary algorithm family (see Table 1) address part of these limitations. Different evolutionary algorithms target different variables as units of evolution and address different problem space that GA alone cannot produce the optimal outcome (Streichert, 2017).

Family of Evolutionary Algorithms	**Unit of evolution / Variables**	**Problem Space**
Genetic Algorithm	General binary coding (like genes)	Find the optimal solutions among the population of potential solutions
Genetic Programming	Computer programs	Which computer program is the best for a certain problem type
Evolutionary Programming	Whole species	Optimize the numerical parameters within a fixed structure of program
Evolutionary Strategy	Vectors of real numbers	Which species will win based on payoff matrix
Learning Classifier Systems	Behavioral Rules	Which classifier leads to better learning outcome.

Table 1: Family of Evolutionary Algorithms

Researchers had included interactional elements such as cooperation and competitions into groups of genetic algorithms to facilitate a better search for the optimal solution and avoid the local convergence (Dimou & Koumousis, 2003; Potter & Jong, 1994). In traditional GA paradigm, this is hard to achieve because to allow such substructure of interactions to emerge, the fitness function and parameter values cannot be fixed. Coevolutionary Genetic Algorithms (CCGAs) and Competitive Genetic algorithms (CGAs) under the learning classifier systems solve this by finding a set of interaction rules in which individual species fitness depends on how well their cooperative or competitive performances have contributed to the larger problem. For

example, Potter and Jong (1994) uses CCGA to optimize functions in which each parameter for the function represents a suboptimal problem (a species). For each subproblem, he implemented a GA and all GAs will evolve together to give the best combinations of parameter values for the whole function. In the case of Competitive Genetic Algorithm, Casas (2015) conceptualizes improving the performance of a sorting network as a predator-prey model: one species represents the current sorting network, while the other represents the type of inputs to the network. While the sorting network evolves to better handle different types of input data, the types of inputs evolve to make it more difficult to be handled.

All of these efforts have largely expanded the flexible usage of genetic algorithms at finding optimality by allowing (defining) interactions between algorithms and leave room for parameters values for GAs to emerge in a coevolutionary relationship. Inspired and supported by these inventions, I would like to address further the limitation of scaling complexity of GA regarding limited computational resources and fixed parameters (for mutation, crossover, etc.) using the competition paradigm to improve traditional GAs.

RESEARCH DESIGN

The objective of this new competitive genetic algorithm is to to make the traditional GA more efficient at using computational resources (e.g. memory and space) with self-adaptive parameters values that are optimal for different inputs. It still aims to solve optimization and searching problem as traditional GAs do. My new algorithm works in the following steps: 1) the input data that comes in will be broken down into random subsets. 2) there will be multiple GAs and I will assign each GAs with a range of reasonable parameter values and a random subset. 3) a resource limitation will be defined (e.g. memory space) and the distribution of resources

depends on how close each GA's answer is to the optimal outcome. The better the performance, the more resources this GA can get (while less for others) and the faster this GA can perform. 4) If a GA has completed its own subset, it can start running for the subset from the slowest GA, and 5) the algorithm stops when it processed all the inputs. Essentially, all GAs run in parallel, but the key is you don't need to run through all the data for each GA to figure out which is the best set of parameters. The competition paradigm allows the GA with better parameter values for this particular sets of input work more than those that are less fit. The parameters that are up for change include but are not limit to population size, mutation rate, crossover rates, caliper for defining how optimal an outcome is and generations.

Therefore, my testing hypothesis is that this new CGA will consume less memory (or space) and better average performance with different types of inputs. The research design will be to run the new CGA algorithm and the traditional single GA on two functionally identical computers and feed them with various kinds of input sets and different sizes of inputs. Evaluate 1) the computational resources each of them takes when the sizes of inputs scale and 2) the performance given different types of inputs. The specific technicality regarding what input types should be considered, what specific problem they should solve and how much resources (e.g. memory) should be given, are still up for further design.

EXPECTED RESULTS AND INTERPRETATIONS

Theoretically, the new algorithm should take less time or less memory to complete, and its average performance is more adaptable (better than GA) to different types of input or problem. The reason is that the competition paradigm distributes computational resources to those GAs with better fit parameter values for this particular set of inputs without being

subjected to fixed parameters like the traditional single GA. This further addresses traditional GA difficulty of the tradeoff between diversity and complexity, which is having to maintain a diverse population to find the global optimal but this population costs computational resources and thus hard to scale. By restricting the computational resources to put in while allowing parallel GAs to run with different parameters, it creates a more diverse population than a single GA.

Alternative reasons for a better performance from the new algorithm could come from the specific problem type or specific datasets I feed to test both algorithms. Therefore, the robustness of the new algorithm also needs to be tested to make sure the performance don't vary too much or find out if it varies, what is the problem space that it works the best.

JUSTIFICATION

So, why should people care about this study? If we look back at 2008 financial crisis, many didn't know it's this one algorithm that crashed the whole Wall Street (Salmon, 2009). The main reason is that people use this algorithm (function/equation) to minimize risks, but ignored how this algorithm's parameters is designed based on biased dataset during the boom time. Of course, this algorithm will not account for "unprecedented risks" when everything collapsed. The essence of this study and all those it has drawn upon is to make algorithms flexible and resilient to different inputs so that such tragedy could be avoided. The key to this resilience is to prevent pre-set human biases at setting parameter values for the algorithm, but allow it to evolve when interacting with different inputs and problem sets. This resilience and adaptability are what I am trying to study by improving the traditional GA algorithm. From this aspect, human still has a lot to learn from evolution to achieve greater potentials.

Reference:

Casas, N. (2015). *A review of landmark articles in the field of co-evolutionary computing*. arXiv. Retrieved from https://arxiv.org/pdf/1506.05082.pdf

Dimou, C., & Koumousis, V. (2003). Competitive genetic algorithms with application to reliability optimal design. *Advances In Engineering Software*, *34*(11-12), 773-785. http://dx.doi.org/10.1016/s0965-9978(03)00101-7

Genetic algorithm. *En.wikipedia.org*. Retrieved 20 April 2017, from https://en.wikipedia.org/wiki/Genetic_algorithm

Kumar, M., Husian, M., Upreti, N., & Gupta, D. (2010). Genetic Algorithm: Review and Application. *International Journal Of Information Technology And Knowledge Management*, *2*(2), 451-454. Retrieved from http://www.csjournals.com/IJITKM/PDF%203-1/55.pdf

Potter, M., & Jong, K. (1994). A Cooperative Coevolutionary Approach to Function Optimization. In *International Conference on Parallel Problem Solving from Nature*. Davidor Y. Retrieved from http://cs.gmu.edu/~mpotter/pubs/ppsn94.pdf

Salmon, F. (2009). *Recipe for Disaster: The Formula That Killed Wall Street*. *Wired.com*. Retrieved 20 April 2017, from https://www.wired.com/2009/02/wp-quant/

Shiffman, D. *Chapter 9: The Evolution of Code*. *Natureofcode.com*. Retrieved 20 April 2017, from http://natureofcode.com/book/chapter-9-the-evolution-of-code/

Streichert, F. (2017). *Introduction to Evolutionary Algorithms* (1st ed., p. 6). Tübingen: University of Tuebingen. Retrieved from http://www.ra.cs.uni-tuebingen.de/mitarb/streiche/publications/Introduction_to_Evolutio

nary_Algorithms.pdf