Parallel Genetic Algorithms for Solving Non-Convex Optimization Problems

Eashan Singh
B.S. Computer Science – In Progress
University of Central Florida
Orlando, USA
eashan.singh1@knights.ucf.edu

Amelia Castilla
B.S. Computer Science – In Progress
University of Central Florida
Orlando, USA
accrazed@knights.ucf.edu

Samu Wallace
B.S. Computer Science – In Progress
University of Central Florida
Orlando, USA
samu.wallace@knights.ucf.edu

***Abstract*—Non-convex optimization problems refer to the optimization of an objective function which is non-convex – that is, a function which curves up and down, not fitting either of the clear definitions for convex or concave functions – whose global extrema we are trying to find. In this way, a non-convex function is expected to have several local minima/maxima, which makes it challenging for optimization algorithms to find the true global extremes. As such, there are many classes of algorithms for solving these problems, one of which being Genetic Algorithms (GA).**

Genetic Algorithms are inspired by and modeled after phenomena observed in biology, namely natural selection and genetics. They work by generating a population of 'candidate solutions' (set of variables to be input into the objective function) and then use evaluation, selection, crossover, and mutation operators in order to evolve the population over multiple generations into better and better candidate solutions, eventually settling on an optimized solution. The generation of these candidate solutions through the above steps can be computationally taxing, with limited scalability when using a single processor. This is where parallelization improves the performance of GAs, by allowing multiple populations to evolve concurrently.

In this paper, we build a Parallel Genetic Algorithm (PGA) using the "island model" for solving a non-convex optimization problem, specifically the Rastrigin function. We then benchmark the resulting performance of the PGA against its serial GA counterpart, as well as other algorithms such as Particle Swarm Optimization. Performance is measured in both runtime and solution accuracy. Since the parameters of a GA often have profound impacts in its performance, we also try differing hyperparameters (e.g., population size, crossover rate, and number of generations) and document performance changes as a result. As we observe performance differences, we also discuss possible causes for them.

Our benchmarks showed that ...

I. Introduction

A. Motivation

Non-convex functions have far-reaching applications in many fields, including mathematics, computer science (machine learning, image processing, signal processing, robotics motion, etc.), finance, material science, and more. As such, it is imperative to find the optimizations to these complex functions in as little time as possible through the use of increasingly powerful algorithms. This paper explores the power in the parallelization of the already powerful GA.

B. Background

Since the goal of our paper is heavily tied to benchmarking, we have chosen the Rastrigin function as our objective function to evaluate. The Rastrigin function is more popular in its ability to benchmark optimization algorithms than its practical applications; however, an optimization algorithm which performs well on the Rastrigin function can be expected to perform well with most other non-convex functions. Our PGA is also able to optimize any other non-convex (or convex) function.

$$f(\mathbf{x}) = An + \sum_{i=1}^{n} [x_i^2 - A\cos(2\pi x_i)] \tag{1}$$

Equation (1) shows the Rastrigin function, where A is a positive constant that scales the function's curvature and n is the number of dimensions. However, the equation alone does not fully convey the difficulty in optimizing such a function.

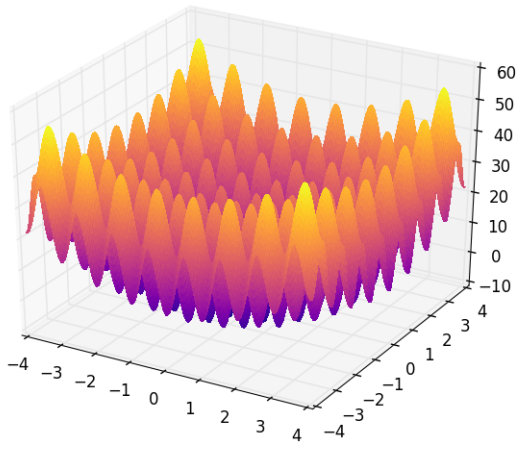

Fig. 1. Graph of the Rastrigin function.

The graph in Fig. 1 illustrates perfectly the clustering of local extrema and degree of ruggedness which makes the Rastrigin function so difficult to optimize. This is especially true when optimizing for the global minimum, which is located in a

large, flat valley surrounded by many small peaks (potentially untrue?). Unsuitable optimization algorithms may get trapped in these local minima, or may have trouble navigating the tight spacing of extrema and overshoot peaks. All of these challenges are what makes the Rastrigin function a fitting benchmark for our PGA.

C. Objectives and Research Questions

(potentially change subsection name)

(does the performance increase from parallelization warrant the multi-core power draw which may slow down other processes on the computer? does it warrant programming complexity? what is the optimal number of threads, depending on user CPU specifications? just how much better is it than serial computation? how does context-switching affect our runtimes? why use C++ for implementation? what is the scope of our paper? how is our implementation easily compatible with other non-convex functions?)

Our implementation of parallel genetic algorithms is designed using the ”island model”, where multiple separate GAs – each with unique hyperparameters and populations – run in parallel. In this way, our design will require a greater overall amount of computation compared to that of a single GA. Because of this, our paper aims to resolve if our particular PGA design takes more time, and if so, whether the results outweigh any runtime increase. We will then experiment with lowering population size, as our PGA should be able to create more varied individuals than a single GA, thus keeping solution accuracy high. We will also discuss where potential runtime growth was incurred, and where potential optimization gains were made.

II. Methodology

In this paper, we propose a parallel genetic algorithm (PGA) for solving non-convex optimization problems. The algorithm design used is the ”island model”.

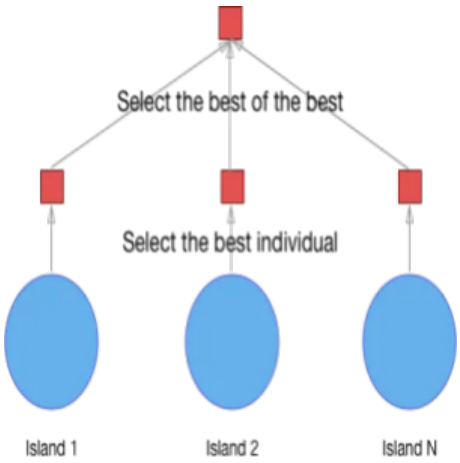

Fig. 2. Basic representation of the ”island model” PGA.

In this design, every island represents an individual genetic algorithm, each with their own populations and their own divergent mutations and crossover history. The GAs in this design are referred to as ”islands” because of their isolation; they do not depend on each other. Fig. 2 helps illustrate some of the benefits of a PGA, as it is clear to see that the pool of selected ”parent” individuals increases N-fold. However, it is also important that each island produces varied results, so that parallelization is not wasted on computing the same exact algorithm in separate processes. We do this by varying the following:

- How mutations are performed.
- How crossovers are performed.
- How many individuals are selected for crossover, as well as for parenting (generation size).
- The total number of generations to compute.

These variations help to address one of the main issues with genetic algorithms, in that they tend towards preliminary convergence to a subset of individuals that dominate others. Having GAs with differing parameters run in parallel and selecting from those should greatly mediate this issue.

The one aspect of the algorithm which stays constant is the fitness function, as all algorithms need to know what makes an individual successful.

III. Conclusion

Conclusion

References

[1] R K Nayak, B S P Mishra and Jnyanaranjan Mohanty, ”An overview of GA and PGA,” International Journal of Computer Applications 178(6):7-9, November 2017