Parallel Genetic Algorithm for Regression

Paulo Santos and Maria Fidalgo

Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa
{fc47806,fc49034}@alunos.fc.ul.pt

Abstract. The first sentence of an abstract should clearly introduce the topic of the paper so that readers can relate it to other work they are familiar with. However, an analysis of abstracts across a range of fields show that few follow this advice, nor do they take the opportunity to summarize previous work in their second sentence. A central issue is the lack of structure in standard advice on abstract writing, so most authors don't realize the third sentence should point out the deficiencies of this existing research. To solve this problem, we describe a technique that structures the entire abstract around a set of six sentences, each of which has a specific role, so that by the end of the first four sentences you have introduced the idea fully. This structure then allows you to use the fifth sentence to elaborate a little on the research, explain how it works, and talk about the various ways that you have applied it, for example to teach generations of new graduate students how to write clearly. This technique is helpful because it clarifies your thinking and leads to a final sentence that summarizes why your research matters.

Keywords: Parallel Genetic Programming · Regression · Island Model.

1 Introduction

Genetic Algorithms (GAs) are metaheuristic searching algorithms where the main idea lies in following the same principles as Natural Selection and Evolution in Biology [11]. That is, the algorithms work around a *population* of potential solutions for the problem, the *individuals*. The population changes during the execution of the algorithm, mimicking the evolution of a real population of living beings, from generation to generation, where the fitter individuals are more likely to survive and reproduce.

The algorithm is composed by three main operations: Fitness, Crossover and Mutation. Fitness measures how good an individual is; Crossover (or sexual reproduction) generates a new solution based on two existing ones, simulating the breeding of two individuals; and Mutation (or asexual reproduction) produces random changes to an individual [7].

Due to the characteristics of the GAs, applications are often related to optimization [11], such as the Traveling Salesman Problem [5], classification [2], decision making [4] and prediction [3].

Prediction can be accomplished through Regression Analysis, which is the task of modeling a random variable Y as a function of a vector of random

variables X. This may be translated as the task of finding the mathematical expression best suited to explain Y. Regression models presuppose the existence of constants, called *parameters*, that are to be estimated from the data [10].

However, when using Genetic Algorithms for this type of problem, it is not the parameters that we want to estimate but the whole mathematical expression. Therefore, we will be looking for the one that minimizes the error between the value provided by the model and the actual value.

In this work, we explore different parallelizations of the Genetic Algorithm for regression, using the toxicity dataset [6]. We will start by introducing the algorithm and its operations in further detail. Next, we will present the several approaches studied:

- Sequential
- Adaptive Sequential
- Trivial Parallelization
- Island Parallelization

Finally, we will demonstrate the experimental evaluation and compare with each other.

2 Background

This is an optional section, as it depends on your project. In projects where a given specific knowledge is required to understand the article, you should give a brief introduction of the required concepts. In the case of genetic algorithms, you should present the basic algorithm in this section.

3 Approach

In this section, you should present your approach. Notice that an approach may be different than an implementation. An approach should be generic and ideally applied for different machines, virtual machines or languages. You should present algorithms or generic diagrams explaining the approach.

4 Implementation Details

4.1 Abstract Syntax Tree

The implementation of an Abstract Syntax Tree allows us to easily generate random mathematical expressions with an immutable tree where each node either represents a binary operator node or a constant node, which can be a variable or value.

Each tree contains a crossover operation which allows it to generate offspring by crossing itself with another tree in a single-point of crossing. This has been accomplished by randomly choosing and swapping two nodes from each tree.

Every tree may also mutate itself by randomly choosing one node and changing its inner value. A constant binary node could have its inner content changed with a new value or variable, while binary operator nodes would have their operation changed.

As aboved-mentioned, Abstract Syntax Tree allows us to generate random mathematical expressions. By using an expression builder[1] was possible to set values to variables of a given expression and obtain a result useful to calculate the fitness of the tree.

4.2 Regular Genetic Algorithm

The linear implementation of a regular genetic algorithm where each genetic algorithm operation has fixed ratios.

The fitness is measured using a linear regression **TODO:**

The population is sorted from best fitness to worst fitness using a default arrays sorting algorithm from Java.

The best individual from each generation is remains unchanged to the next generation. The rest of the population is replaced with generated offsprings by crossing parents using a Fitness Proportionate Selection, where the best individuals have a higher chance of being chosen for crossover.

Each offspring contains 10% chance of being selected to suffer a mutation.

4.3 Adaptative Genetic Algorithm

The linear implementation of a genetic algorithm where the mutation crossover rates are adaptative depending on the progression made by each operation in the last generation [8]. The measure fitness and sorting is implemented the same way has in the Regular Genetic Algorithm.

The best individual still remains from each generation unchanged to the following one. The new offsprings are generated by crossing parents using an Adapted Fitness Proportionate Selection. Since new offsprings are always generated by crossing two individuals, the crossover rate will influence the probability each individual has to be chosen. This has been attained by using the absolute value of normal probability density function and multiplying it by the sum of a third of the amount of population with the crossover rate. The crossover rate is an integer value between $-populationSize$ and $populationSize$.

$$\left|\left|\frac{1}{\sigma\sqrt{2\pi}} * e^{-\frac{(-\mu)^2}{2\sigma^2}}\right| * (popSize\,/\,3) + crossoverRate)\right| \,\%\, popSize$$

The mutation rate controls the probability of an individual to suffer a mutation. This value is a value between 0.05 and 1.0.

Both of these rates are updated according to the progressions they obtained for each individual. The chosen mutation rate offset is 0.05 and the crossover rate offset is $populationSize * 0.025$.

[1] exp4j - Expression Builder from String for Java

if $mutationProgress < crossOverProgress$ **then**
$mutationRate := min(1.0, mutationRate + mutationOffset)$
$crossoverRate := max(-popSize, crossoverRate - mutationOffset)$
if $mutationProgress > crossOverProgress$ **then**
$mutationRate := max(mutationOffset, mutationRate - mutationOffset)$
$crossoverRate := min(popSize, crossoverRate + mutationOffset)$

4.4 Parallel Versions Implementation

4.4.1 Parallel Population Sorting

Due to parallel mergesort ease of implementation and advantages of execution time [9], the sorting of the population by fitness has been implemented with a parallel mergesort with linear insertion sort when the amount of population to compute is smaller than an offset of 7. The population is splitten by half until the offset is reached and the insertion sort and merge algorithms' are computed.

4.4.2 ForkJoin

TODO:

4.4.3 Phaser

The Phaser Approach makes use of Phasers in order to introduce a synchronization point between all the Threads running.

First of, there has been made the decision of creating N threads proportional to the amount of available processors of the machine. A smaller value would not take advantage of all the processing power available, whereas a higher value would have an impact on the performance since the CPU scheduler would give each one of those $N > availableProcessors$ some share of CPU time.

Every Thread is responsible for a partial amount of the population. In order to maintain consistency in the population over the several generations and between different Threads it is required a synchronization point, the phaser.

The synchronization point has been introduced before and after the transition from the old population to the new one. This is a requirement since the following operations cannot happen until every single expression from the new population has been introduced in the current population.

It is also a requirement a synchronization point ahead and prior to the sorting of the population. Once every single individual has transitioned from the old to the new population, the Thread with $threadId := 0$ launches the Parallel Population Sorting algorithm. Once the algorithm finishes, every Thread gets through the synchronization point and a new generation starts.

The phaser runs the Regular Genetic Algorithm approach in the computation of each individual.

4.4.4 Island

In order to take advantage of the amount of available processors we've also introduced an implementation of an Island Model Genetic Algorithm [12].

Our approach takes advantage of the multiple available processors to create and hold the computation of the islands. Each island has its own population, with size $populationSize/amountIslands$, and is independent of other islands, doesn't require synchronization points. Occasionally, every 20 generations, the best individual of an island is sent to a random island using a *ConcurrentLinkedQueue*. Upon receiving a new individual, right before sorting the population, the last individual is replaced with the new one received.

A naive approach would be to create one island per available processor. Our implementation allows us to create a variable amount of islands between 1 and the amount of available processors.

If the specified amount of islands is smaller than the amount of processors then the remaining available processors will be evenly distributed through the islands, allowing these to contain inner parallelization. The inner parallelization allows the island to split the population with the amount of threads it has been assigned and compute each generation quickier. As in the Phaser approach, it has been used a phaser to create synchronization points in essencial operations, such as transitioning from the old population to the new one and sorting it, between the threads of an island. In this case the island is responsible for calling the Parallel Population Sorting algorithm.

When an island completes its computation, it redistributes the upcoming available threads with the rest of the islands starting by the end. This way we're able to take advantage of the processors once they're free. The re-destribution is implemented using a $ConcurrentLinkedQueue < Pair < Integer, Integer >>$ where the island sends a message to create a new thread to another active island containing $< islandId, newAmountThreads >$.

Every island runs the Regular Genetic Algorithm approach to compute each individual.

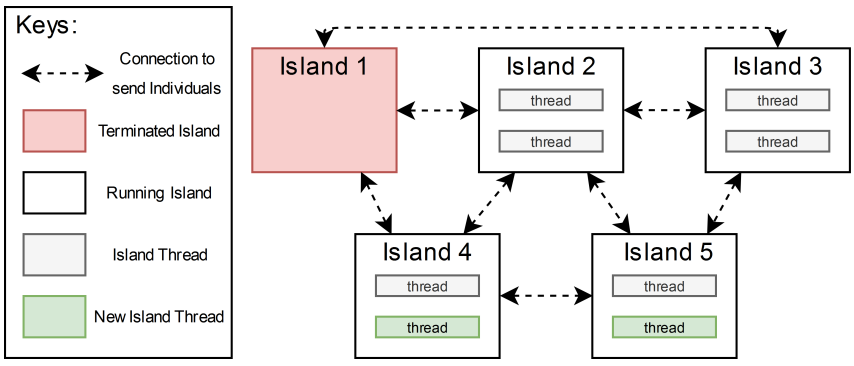

Fig. 1. Example of thread redistribution, with 8 processors and 5 islands, after island 1 has terminated.

5 Evaluation

5.1 Experimental Setup

TODO:
24 cores,
1000 pop,
500 iteracoes,
tamanho geracao inicial 2^{10},
tamanho maximo com crossover 2^{15}
dataset offset = 250 iteracoes.
dataset split = 100 = 2 partes.

In this section you should describe the machine(s) in which you are going to evaluate your system. Select the information that is relevant.

5.2 Results

+ Comparacao dos resultados da regular linear com a adaptative linear
+ Comparacao dos resultados da de ilhas com 6 ilhas, 12 ilhas, 18 ilhas e 24 ilhas
+ Comparacao de todas as implementacoes em termos finais de execucao com boxplot
+ Apresentacao em linha de tempo da linear, forkjoin e phaser
+ Comparacao em linha de tempo da linear, forkjoin, phaser, ilhas com 6

In this section you should present the results. Do not forget to explain where the data came from.

You should include (ideally vectorial) plots, with a descriptive caption. Make sure all the plots (Like Figure **??** are well identified and axis and metrics are defined.

5.3 Discussion

Here you should discuss the results on a high level. For instance, based on our results, the parallelization of the merge-sort is relevant as no other parallel work occurs at the same time, and the complexity $O(Nlog(N))$ can have a large impact when the number of individuals is high.

6 Related Work

Several implementations of the genetic algorithm were made throughout the years. We will shortly talk about two implementations and how they fit in the scope of our work.

Dominic and Willis [1] developed a MATLAB toolbox, GPTIPS, which is able to perform regression through genetic programming. The main difference

between their approach and ours is that they do not explore the parallelism of the algorithm, focusing on the usability of the toolbox. Moreover, they chose to include nonlinear operators, that we decided to leave out. Our work is, therefore, important to whom intends to develop a fast approach of the genetic classifier. Additionally, GPTIPS requires the purchase of a payed software (MATLAB), available to a less broader population.

Jenetics [13] is another genetic programming implementation. It is a Java library designed to abstract different concepts within the genetic programming panorama, such as Gene, Genotype and Chromosome, allowing it to serve a vast spectrum of domains. This library implements the Java Stream Interface and provides ForkJoin Parallelization. This is, therefore, a generic purpose implementation for genetic algorithm. On the other hand, in our work we provided a study of genetic programming specific for regression, where other parallelization techniques were able to achieve better results than ForkJoin, like the Island Models.

7 Conclusions

Here you should resume the major conclusions taken from discussion. Ideally, these should align with the objectives introduced in the introduction.

You should also list the future work, i. e., tasks and challenges that were outside your scope, but are relevant.

Acknowledgements

First Author wrote the part of the program implemented the phasers. Second Author implemented the MergeSort in parallel.

Both authors wrote this paper, with First Author focusing on the introduction, related work and conclusions while the Second Author focused on approach and evaluation.

Each author spent around 30 hours on this project.

References

1. Dominic, P., Leahy, D., Willis, M.: Gptips:an open source genetic programming toolbox for multigene symbolic regression. Lecture Notes in Engineering and Computer Science **2180** (12 2010)
2. Espejo, P.G., Ventura, S., Herrera, F.: A survey on the application of genetic programming to classification. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **40**(2), 121–144 (March 2010). https://doi.org/10.1109/TSMCC.2009.2033566
3. Etemadi, H., Rostamy, A.A.A., Dehkordi, H.F.: A genetic programming model for bankruptcy prediction: Empirical evidence from iran. Expert Systems with Applications **36**(2), 3199–3207 (2009)

4. George, A., Rajakumar, B.R., Binu, D.: Genetic algorithm based airlines booking terminal open/close decision system. In: Proceedings of the International Conference on Advances in Computing, Communications and Informatics. pp. 174–179. ICACCI '12, ACM, New York, NY, USA (2012). https://doi.org/10.1145/2345396.2345426, http://doi.acm.org/10.1145/2345396.2345426
5. Grefenstette, J., Gopal, R., Rosmaita, B., Van Gucht, D.: Genetic algorithms for the traveling salesman problem. In: Proceedings of the first International Conference on Genetic Algorithms and their Applications. pp. 160–168 (1985)
6. Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A., Hu, B.: Genetic Programming: 16th European Conference, EuroGP 2013, Vienna, Austria, April 3-5, 2013, Proceedings. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2013), https://books.google.pt/books?id=6Y-5BQAAQBAJ
7. Langdon, W.B., Qureshi, A.: Genetic programming-computers using" natural selection" to generate programs. In: WC1E 6BT. Citeseer (1995)
8. Lin, W.Y., Lee, W.Y., Hong, T.P.: Adapting crossover and mutation rates in genetic algorithms. J. Inf. Sci. Eng. **19**, 889–903 (09 2003)
9. Manwade, K.: Analysis of parallel merge sort algorithm. International Journal of Computer Applications **1** (02 2010). https://doi.org/10.5120/401-597
10. Rawlings, J.O., Pantula, S.G., Dickey, D.A.: Applied regression analysis: a research tool. Springer Science & Business Media (2001)
11. Sivanandam, S., Deepa, S.: Genetic algorithm optimization problems. In: Introduction to Genetic Algorithms, pp. 165–209. Springer (2008)
12. Whitley, D., Rana, S., Heckendorn, R.: The island model genetic algorithm: On separability, population size and convergence. Journal of Computing and Information Technology **7** (12 1998)
13. Wilhelmstötter, F.: Jenetics, Library user's manual 4.3 (2018), https://books.google.pt/books?id=6Y-5BQAAQBAJ