Genetic Algorithm for the optimisation of a scheduling problem

Antoine Mercier--pronchery
Université Grenoble

Abstract. Presented here is the work accomplished on a Genetic algorithm (GA) tailored for the 3-MMOPT optimisation problem. It includes a background presentation for the problem being tackled as well as what constraint are imposed. Furthermore a discussion on the different design choice is engaged and explanation regarding the aforementioned design choice will be provided. The discussion include the individual definition, the selection process and the evaluation phase. A swift analysis of obtained result is performed before concluding on what further work could be achieved given more ressources. A personal assessment is given in order to assess my performance on this project.

Keywords: Genetic Algorithm, MMOPT, optimization, Mutation, Embarked System, electronic

Style of the report to turn in: http://membres-lig.imag.fr/labbe/TER/LaTeX.zip

1 Introduction

2 Background

2.1 Context

The design of embedded visions system is a challenging task, one of the main hurdle being accessing the image memory. In order to help circuit designer meet certain goal, tools and algorithm must be devised.

2.2 Problem Presentation

2.3 Previous Work

2.4 Assignment: Objective

During this TER my main objective was to devise a genetic algorithm that would be able to optimise the 3 MMOPT problem. As discussed before, this problem has contradicting objectives, as a result my program would have to be able to present a satisfying compromise between each criteria and present a "range" of solution. These solutions would each represent a valid configuration for the MMOPT problem, however none of them would be the ideal solution as it would be left to us to decide which one is the best compromise.

Another point to consider is that genetic algorithm cannot give optimal solution, they can only approach them. In that regard, these criteria shaped the work I had to do during this month and a half.

2.5 Assignment: Method

At the start of the TER, because of the limited time allotted each week (half a day on wednesday, half a day on thursday) we opted to have a meeting each thursday.

This allowed us to discuss the different problem I had encountered the day prior as well as talk about the conception of the algorithm and what needed to be done or researched for next week meeting.

During this time, I was also tasked to writing the general structure of the genetic algorithm, and while we didn't put in place any developpement method (such as Scrum or Extreme programming), I did try to implement the principle seen in the DevOps course such as the use of a Version control system (GitHub in this case), a test architecture and a continuous integration add on (Travis-ci) in order to ensure that the code produced would be working and of quality.

As a result, after implementing these tools and just before beginning the full time period, we were able to advance to the integration of the ECM algorithm into the genetic algorithm.

2.6 Genetic algorithm: an explanation

A genetic algorithm is a type of reinforcement learning program which try to imitate biological evolution to optimise a problem.

Just like in biological reproduction, stronger individual are selected through many generation of reproduction. Random mutation in different individual may lead to new possibilities being explored while the weaker individual die off and cannot reproduce further ensuring that only the best individual can reproduce.

This biological analogy highlight the key feature of the inner working of a genetic algorithm.

First, it start with the definition of what is an "Individual". This step is a crucial one as it will make or break the result of the program. In this step we have to define the characteristic of our "generation" and in what way are they going to influence or change the execution.

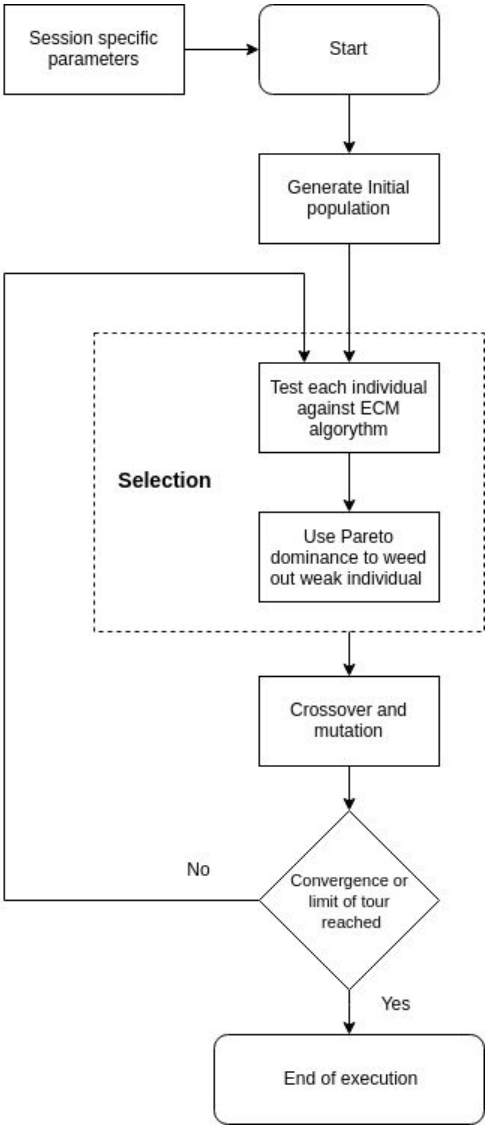
When a population of Individual is gathered the next step will be to judge them depending on different criteria depending on their use. They can be assigned grade or "fitness" for a scheduling problem where certain behavior must be encouraged (a weighed traveling salesman problem for example) or they can be compared on the result they produced when used in their context (such as in a neuroevolution application). Either way, there will be a "selection" process where good individual with either strong characteristic or interesting behavior will be kept and weaker member will be eliminated and replaced by new member.

Once this selection is done through varying method the individual that subsist are considered to be "the best one", they will now be bred to form a new supposedly "stronger" generation.

Breeding usually involve exchanging trait between individual just like how a child reflect it's parent fusion. And just like the biological example a key feature of evolution is kept: the mutation. This mutation will help the population by trying new combination that weren't possible with the current set of Individual and can lead to new discovery for trait combination for a good Individual. As such it is an important factor in preventing the program from getting stuck in local optima or converging to a solution too quickly.

With this we now have a new and improved population, ready to repeat the cycle until it converge to a good solution or an iteration limit is reached.

An important characteristic of genetic algorithm to note is that they cannot produce optimum solution and can only be used to approach them.



In our use case we ended up with this structure for our Genetic algorithm.

An easy to access file allow the user to manipulate the parameter of each execution: what is the mutation rate? (how often to mutate) How many Individual should we have per generation? Or the different range of value that each Individual can possess.

By tweaking the program we are able to fine tune these parameters to get better solutions.

The algorithm then start by generating a population according to these parameters.

In order to test the effectiveness we try each individual on the ECM algorithm in order to get their statistic during execution (Time, number of buffer needed and number of prefetch needed) which will be used to evaluate them.

We then use a pareto dominance method in order to keep the “best” individual inside the population while taking out the weaker one.

With this reduced population where only stronger or neutral subject exist we now go on to the breeding process, here a one cut reproduction is used and a mutation chance is applied after each reproduction in order to ensure diversity.

Finally after a set amount of tour in order to keep the execution time reasonable the algorithm stop itself and generate statistic for the run in order to enable the user to judge the result and correct parameters he think may improve the next run of the program.

The last generation of the program will represent a set of solution that are neither superior nor inferior to one another.
This will be considered as a pareto set and will encompass a wide range of compromise on each criteria of the selection process. In this set there will be the extreme cases where one value was never taken out because it was too overpowering, however, the interesting results will lie in the solution that are the most balanced and that are able to compromise well on the objectives.

3 Realisation and Problem Encountered (4 Pages)

For the algorithm implementation language were first discussed such as Java however due to the original ECM algorithm being in python and my tutor being mostly familiar with this language it was decided that this would be the language best suited to a fast suited project such as this one, as I was not familiar with this language a small adjustment period was necessary before developing important features

3.1 General algorithm and inspiration

The first few week were dedicated to studying the current project and its inner working in order to get a good grasp on what were the objectives and needs that the project needed to answer to.
This phase allowed me to get an understanding of the core concept of the 3 MMOPT problem, what where it's challenges, what advantage did the previous method add? Where did the previous method fall short? Or what were good path to take for the conception of the algorithm.
Along with this document, we were able to explore different paper relating to genetic algorithm design such as _____ and _____ which both presented the basics for a fundamental GA.

A noteworthy document was also _____ as it highlighted more advanced features such as the notion of elitism in the algorithm or how pareto dominance could be used to rank the solution depending on the number of solution that dominate one over another.
Both these notion were envisioned for the evaluation module and will be discussed later on.

In the meantime, while I got accustomed with the problem, and because I was wholly unfamiliar with the programming language used, we decided to put this time to use.

I was tasked with developing the basic features of the program. Before starting to develop the core functionality we decided to first build the mechanism allowing the parameters to be changed in a user friendly way through an xml file.
This proved to an invaluable task as it allowed me to finish a self-contained feature while familiarizing myself with the language and the environment.

It was also during this time that I implemented some of the principle of continuous integration with the addition of a version control system that would ensure that the source code could be traced back and the beginning of a test architecture for the project.

With a better understanding of the language requirement and of the problem specificity I was able to devise a class diagram in order to plan which module needed to be prioritised and which decision needed to be taken quickly in order to be efficient.

After this period I was able to start the development of the basic feature of the program that would not be impacted by my lack of knowledge on the 3 MMOPT problem. This included modules that were not related to the problem itself such as the Crossover module which dealt with abstract Individuals and the population generation module which also used abstract Individuals.

Finally I implemented the main frame of the program where each piece would be placed. This represented the final step before beginning the development of the class specific to this problem and adapted for the genetic algorithm : The Individual module , the Scheduling module and the Evaluation module.

3.2 Selection Method

The evaluation module of a genetic algorithm influence heavily the result obtained in the end. As this step proceed to rank or sort the different Individual and their result depending on fixed criteria, it's conception and implementation is both tricky and paramount to the success of the program.

In classic genetic algorithm problem, the standard approach is to rate each individual by examining it's "fitness" representing how good a certain solution has performed. However this method is mostly used for single objective optimisation problem and cannot be applied as is to multi objective problem.

Here the concept of a pareto set had to be taken into consideration. The goal of this program wasn't to designate a solution as "the best one" but to present a set of different solution that would represent different compromise on the criteria that we had chosen all the while remaining valid solution. This is known as a pareto set and as stated before a genetic algorithm is not able to attain the pareto optimal solution and can only approach it.

As a result, special evaluation method had to be chosen from the following

- The Scalarizing of a multi objective problem can allow it to be condensed into a single-objective optimization problem so that optimal solutions to the single-objective optimization problem are Pareto optimal solutions to the multi-objective optimization problem.

- The pareto dominance method uses the criteria established previously to select an Individual. By picking two random individual from the population it then compare their result against each other : if one Individual result id better than the other, then he will be the one surviving to the next step of the program. However if none of them is able to dominate the other, both are put back into the population.

After discussing which method we should use, it was decided that the second one would be implemented as it was tested and put into use in a problem similar to ours: In the paper pareto dominance was used to select member of the population through a tournament method.

From these two method other criteria relating to general genetic algorithm design had to be considered. As stated previously, the document _____ highlighted two interesting concept that could lead to a better efficiency when selecting members:

- Elitism is a concept where the best solution remain unaltered from one generation to the next. This could have been a great opportunity in case the program naturally or by chance stumble across a solution close to the pareto optimal and is able to retain it without altering it and protect it from mutation. It however require to define a rank in the population in order to determine who can be the current elite. This requirement meant that we had to change the evaluation method that was selected which isn't compatible with the our time constraint. It is however a great path for future improvement.

- To take care of the ranking this paper proposed that pareto dominance could be nuanced, meaning that the percentage of solution that dominate over one another (for example a given solution might dominate all other while another could dominate only half) could be used to give rank to the different individuals.

Both of these concept are condensed in the NSGA - II algorithm, a famous multi objective program that implement these two idea in order to get result for a given problem.

It's implementation was discussed but was discovered only in the middle of the project, as a result we elected not to implement it in this project. It could however be considered for future development on the algorithm as these features are tried and tested method to get better results.

3.3 Individual definition

One of the main challenge of this project regarding the genetic algorithm conception was the representation of the Individual. As stated before, this step is along with the selection method, is among the most important one in a GA.
At the beginning of the project, a first iteration was decided upon. There would be two characteristic : A random number of buffer, a random method of processing (chosen from a pool of three different method).

This first attempt at designing an Individual wasn't necessarily wrong, there was nonetheless some problem that arises from this. Firstly a genetic algorithm can only thrive when there are a wide range of value to explore, here

the space was too restrained for it to be effective. Secondly, the different algorithm (ECM, CGM and EECM) were designed with a fixed number of buffer in mind, as a result the program could not work with randomly picked number of buffer and as such could not try and explore the space unless this number was fixed.

A few path were considered for designing the Individuals. Additional characteristic such as considering how many input tile were in common or how many difference where present in each output tile were considered.
From this first test it appeared that using the scheduling method as is would not be viable for future development of the program. How we solved that problem will be discussed later on, in the end we settled on Individuals with five distinct characteristic that will be explained in the next part:

- A number of buffer picked between minimum and maximum value determined automatically depending on the Kernel being investigated
- A number representing how close a tile should be in order to be assigned higher importance
- A number representing how far the scheduling is able to see to make decision from a range defined by the user
- A float number representing a weight for grading tiles based on their overall uses
- A float number representing a weight for grading tiles based on how close is their next use

With this individual definition in place, the algorithm is able to converge to acceptable solution and seems to be working as intended. As such it is the final definition that we arrived at at the end of this TER

3.4 Evaluation method

For this program the evaluation would be the previous ECM algorithm that was devised and programed previously. As explained in the first part, this algorithm operate in two phase, the scheduling and then the computation of tiles.
The output of this code represent a triplet consisting of a number of buffer, a number of prefetch and a time, calculated based on the previously entered computation time and time needed for a prefetch.
It however, came with limitations. The program was made with specific requirement in mind :
The number of buffer is the optimal number, this means that each input tile has its own buffer which is far from being acceptable in a real world use case. This resulted in the number of prefetch being its minimum value, which is practically impossible to attain with a reasonable number of buffer.
The alternative version of the algorithm “EECM” aimed at reducing the number of buffer by influencing the scheduling so that tile not needed in the future could be replaced, was considered as it would allow a lower number of buffer. Despite this we decided to opt for an adaptation of the ECM program as it would allow a greater degree of flexibility for the genetic algorithm.

As a result we decided that modifying the scheduling process in order to accomodate a fewer number of buffer would be the better option. After discussing this matter we settled on using the optimal schedule generated by the ECM algorithm with an optimal number of buffer and change the computation step in order to accomodate fewer buffer.

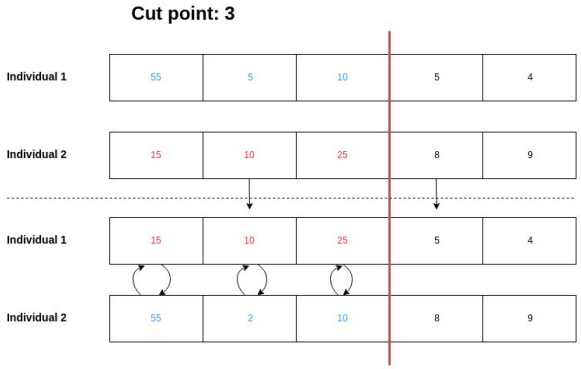
In developing a way to compute the optimal schedule into something possible for a very reduced number of buffer, we discussed different strategies that could be employed to minimise the number of prefetch and avoid overwriting existing input tiles in buffer that will soon be used :

The algorithm could try and consider how soon one tile not needed for the current computation step was going to be used again
It could try to put it into a limited context in order to improve computation time and avoid gathering unnecessary information, by limiting the effective range of the program.
It could also try to rank by importance the tiles by examining their overall usage or how soon they will be used again.

These three strategies were all put to use with new characteristic being added into the Individual definition.
While it was time consuming, this approach of customising the ECM algorithm to suit our genetic algorithm allowed us to make full use of its strong point by striking a balance between a good configurability and a wide range of possibilities.

3.5 Reproduction Method

The rest of the program follow the standard guideline of a genetic algorithm. The reproduction method chosen is a one point cut reproduction represented below with two mock Individual of length five with a cut at the third characteristic

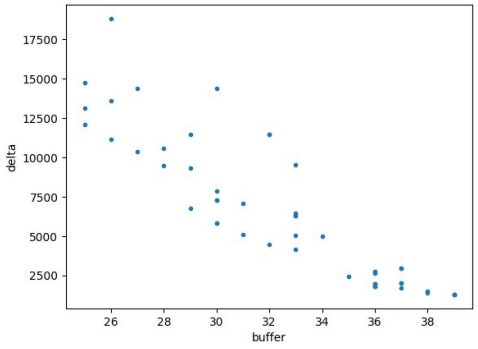


Along with this reproduction method the mutation chance is added in.

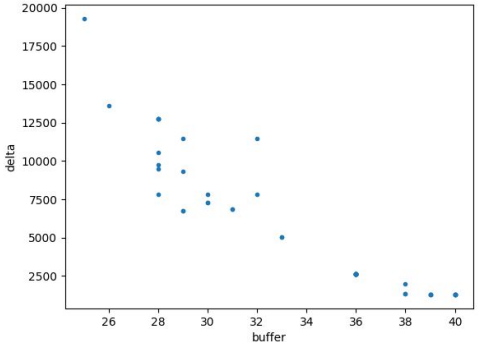
This result in an Individual being able to change in unpredictable way as one characteristic (or gene as they are commonly referred to) might be completely overwritten.

3.6 Algorithm result

After experimenting with the different parameters, I arrived a an execution that was able to produce good result for our use case. In the following graph we put in relation the number of buffer, the number of prefetch and the calculated time

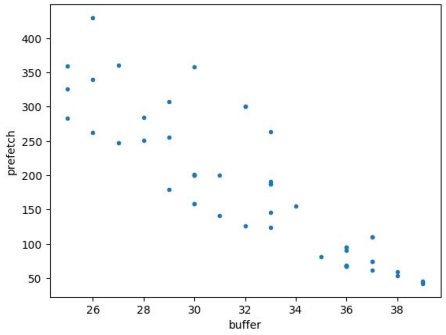


Result of the execution of generation 1
(comprised of 50 individual)
with time related to number of buffer

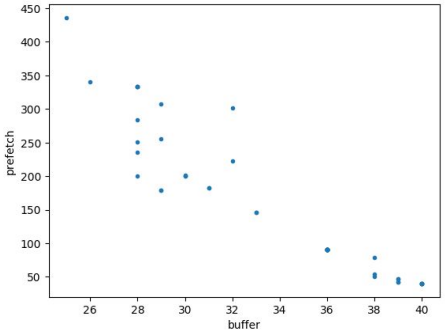


Result of the execution of generation 200
(comprised of 50 individual)
with time related to number of buffer

In these graph we can see that the first generation is quite evenly spread throughout. However by the 200th generation a front can be discerned and an area of interest where the pareto optimal solution is likely located can be seen

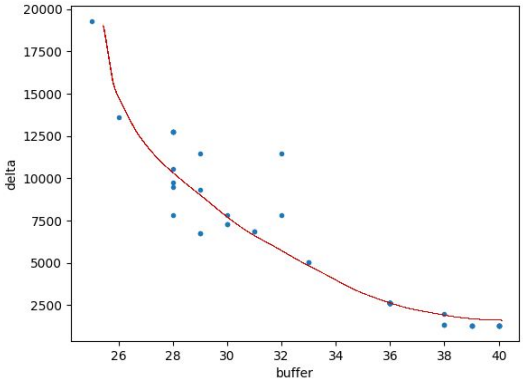


Result of the execution of generation 1
(comprised of 50 individual)
with number of prefetch related to number of buffer



Result of the execution of generation 200
(comprised of 50 individual)
with number of prefetch related to number of buffer

We can also observe that prefetch number and time calculated are closely tied. Furthermore these result are only valid for similarly constituted Kernel that are comprised of around 40 kernel.



On this graph the pareto front has been drawn. It represent a set of compromise that are all acceptable solution and present both drawback and advantages.

This is already a good result for our algorithm as it would enable the electronic designer to pick a design that is most suited for their application while being aware of each solution strong point

4 Future Work

In this report I already presented a few ways in which the algorithm could be improved. This include implementing the NSGA - II algorithm in order to get a more accurate representation of the pareto front and balancing more the repartition of solution along it.

Another opportunity would be to adapt the other algorithm devised previously (CGM and EECM) to this program in order to be able to pinpoint which method is most suited to special use case. It would also ensure that the method employed here isn't biased in any way.

More advanced behavior could be implemented: the way the program handle the reassignment of buffer could stand to be improved with other flexible option (which would lead to a further redefinition of the Individual)

There could also be another implementation of the reproduction method using a 2 point cut which can be an interesting option to explore.

Finally an important improvement could be a mechanism that is able to process a list of kernel instead of having to test the algorithm on each kernel individually (this could be handle through the use of percentage from the optimal solution to discern area where an Individual performed well). This however would lead to a heavy refactoring of the selection as previous criteria would not be applicable.

5 Conclusion

The genetic algorithm is fully functional at the end of this TER, it is able for linear kernel and most non linear kernel to find acceptable solution in a reasonable computation time. As such I would consider that I