**Introduction to Evolutive Algorithms (Theory and Practice) (From Scratch)**

I have been at University at some time now and working with genetic algorithms for more than 2 years straight. I’ve never found a good tutorial/guide to it, I think genetic/evolutive algorithms are not as established outside universities as other techniques, so this can be the result of not having that many good content about it.

Also, much of the content talks about existing frameworks or do approach extremely simple problems, although they can be used as a learning experience, evolutive algorithms are more powerful than that.

Other times, the language used is not… a very good choice… I find a lot of examples in Java and even Matlab, the performance in an Evolutive Algorithm is critical, the algorithm itself is heavily CPU bound so using those kind of programming languages may not be the best decision.

And finally, also it is hard to find complete code guide on it, reading about theory is easy but coding is a different problem. Maybe this happens because, again, EA are used the most in Universities/Academic Context, and usually they are not really concerned on implementation aspects of the problem, and probably they shouldn’t.

**WHAT PEOPLE WON’T SAY TO YOU ABOUT EVOLUTIVE ALGORITHMS**

This could be a whole separated text called “What people don’t tell you about Evolutive Algorithms”, but I’m just going to write it here. There are chances that you won’t understand a lot of things written in this section, but at least take a quick read, come back at this part when you have a better understanding of Evolutive Algorithms.

Evolutive Algorithms have so much variables. So. Much. Variables. Usually these guides will just tell you: “Oh, use this population size, these factors, these operators...”, but not really tell you why, and you, a beginner, doesn’t really imagine that there is more to it, but there is, there is always more to it.

I decided to talk here about the not so glamourous things about Evolutive Algorithms.

**First Thing: Testing.** It is not a secret that testing is a really important part of Software Development, they assure you that your code works correctly. But for Evolutive Algorithms random number generation plays a big role and its quite the essence of the technique, so… How exactly are you going to test something that relies heavily on rng?

There are some ways, they are hard, but there are ways, we will talk about that. Hint: it is way more painful than regular software testing.

**What is the population size?** 10? 20? 50? 100? 200? What is your population size? The only way to answer this is implementing the algorithm, testing all the sizes and seeing which one suits better your prerequisites. *Why not just a big population?* Although changing population size from 10 to 50 or from 50 to 100 can make a big change and improve your results, changing it from 50 to 100 or from 100 to 200 maybe won’t do that much of change and the execution will take almost twice the time. What is going to determine it is the problem, there are problems that 10 is enough, there are problems that 100 is ok, there problems that you will need 200. Keep that in mind.

**Reproduction Rate?** You have your population size, based on that, how many new individuals are you going to generate? 80%? 90%? 50% - Variability on the population plays a HUGE role on not getting stuck on local optimums. But again, what will tell you the best is testing.

**Mutation Rate?** What is the probability of a certain candidate to mutate? 1%? 2%? 5%? 10%? – Testing.

**Crossover Operators?** Operators can also play a big role, which one is better to your problem? What if you have two? – Testing.

**Mutation Operators?** Same as above.

**Ordenation?** Ordenation is important, it can be even more important if your selection or survival operators take in consideration the fitness. From my experience, ordination is sometimes neglected, but it can mess up your population due to implementation issues and can slow down your algorithm if the right technique is not chosen (I’m looking at you Quicksort)

**Repeated Individuals?** Individuals with good fitness have the tendency to take over the whole population if you don’t act cautiously with them, you don’t want the same individual on the entire population, do you?

**How Many Generations?** Be ready for statistics!

**Population Initialization?** Random? Tournament? Biased?

**Selection Operators?** Random? Roulette? Wheel?

**Survival Operators?** Elitism? Reinsertion?

**What we are going to do?** A simple evolutive algorithm in C++.

**What we are going to need?** C++. Text Editor.

Introduction – Part II

I wanted to write this guide for a long, long time. I like writing, I like the possibility of teaching people new things and I like Evolutive Algorithms. Why? Nondeterminism. Getting the same results over and over again is booooooooring.

At first I was going to write this guide approaching the travel salesman problem. I’m not going to do it anymore. This semester I’m taking the subject Cryptography/Security and my second essay is to optimize the size of three sets in a way to maximize triple hash collision. This is practically a way to forge false signatures over the internet and scam people, but hey, we’re going to use old hashing functions and just believe the ones people use nowadays are safe from this kind of thing (probably they are).

I will divide this in 2 Parts: Theory and Practice. On the theory, I will talk briefly about genetic algorithms, evolutive algorithms, this won’t be different than reading an article about it, so you might as well do it because it will explain better than me. I will try to give a simplified overview and focus on the aspects that used to trick me the most (such as the difference of evolutive algorithms, genetic algorithms, genetic programming, …). I will also try to structure it in a way that, at least for me, is of better understanding and more relatable to coding.

On the practice I will explain the problem we are going to talk about and we will write the code of the evolutive algorithm to solve the problem. I will briefly talk about design decisions and those kind of things, this doesn’t mean at all that this is the uttermost correct way to do it, probably there are better design options, but working it through the time I came up with the one I will talk about and it feels natural for me, probably it can be improved over, only the time will tell.

OBS: The Implementation part of this guide I will write while doing the essay, for example, when I code the data structure I will write about the data structure part and so on… So I will also talk about problems I will be facing and common problems that might happen. Or it can turn into a chaos.

All in all, you can find the code at my github. (link will be here)

**Introduction to Evolutive Algorithms**

1. What is it?
2. Difference of Evolutive Algorithms and Genetic Algorithms
3. Flow of an Evolutive Algorithm
4. Individual (Data Structure)
5. Population
6. Random Population Generation
7. Parent Selection
8. Crossover
9. Mutation
10. Survival of the Fittest
11. Fitness Function

**Implementation**

**What are we going to do?**

**The problem is:** Given an integer *n,* where *n* is the amount of hashing work you can do, find the size of three sets, M1, M2 and M3, where |M1| + |M2| + |M3| = *n* that maximizes the probability of having a triple hash collision*.* That means, maximize the possibility of, given a m1 in M1, a m2 in M2 and a m3 in M3, H(m1) == H(m2) == H(m3) where H(x) is a hashing function.

I’m going to use two hash functions: SHA1 and MD5, both of them are a bit outdated but they probably won’t give us any memory management problem.

Why two hash functions?

1. MD5 will be used to generate the messages we want to hash; these messages will be generated by hashing with MD5 the string equivalent to 0… 1… 2… 3…. On increasing order. So we won’t have to store it in memory, we can just calculate the message with the MD5 (which is fast).
2. SHA1 will be used to properly hash the messages and will be our hash function on this problem.

To solve this problem we need an algorithm that, given the size of M1, M2 and M3, generate the messages and search for triple collisions within the sets. we can do that with just one function.

**Setting MD5 and SHA1.**

Chances are that you don’t want to write your own hashing functions, so we are going to use one that already exists, I’ve chosen to go with hashlib++ (<http://hashlib2plus.sourceforge.net)> and you can see how to set it up on the example after I talk how to solve everything with just one function.

We need a simple algorithm to find triple collisions, the one I will be using to start with is the one suggested in class, probably there are better ways but later we work on that.

The algorithm is a function that receives the size of three sets and:

1. Hashes M1 and store the values.
2. Check for collisions on the hashed M1 and the hashed M2, to obtain a new set.
3. Check for collision on this new set and on hashed M3 – If there is a match, we count for a collision.
4. At the end of the algorithm, it just prints, for now, M1, M2, M3 and the amount of collisions.

This is not really a difficult task, the problem is: how to find good values for the size of each set? We will try to find them using evolutive algorithms.

If you want to see how I did this function that solves everything you can check it here:

<https://github.com/guilhermepo2/triple-collision-ea/blob/8aa2dbaaedc694abe54615992353b7868521bce8/src/searchForHash.cpp>

**Thinking on Evolutive Algorithms…**

To correctly code an evolutive algorithm, we need a population, which is formed by individuals. The algorithm works with operations such as initializing the population, selecting parents, crossover, mutation, survival of the fittest and sorting. The algorithm also haves some attributes such as population size, crossover rate, mutation rate, and others…

How to organize all of that?

The way I like to do it is:

1. Creating a class for the individual with all its data.
2. Create a class for the population, which has a vector of individuals and ensure any rules about the evolutive algorithms (in this case, it will have a max size and ensure that the size of the sets do not goes beyond n)
3. Create a class for the evolutive algorithm, it will hold a population and the operations, which I like to write on their own files in a way that is easier to write new methods.

That being said, from now on I’m going to start building this model of evolutive algorithm, this is by no means the best one and feel free to criticize it and say how it could improve

**The Individual**

The individual is pretty simple, to the point it made me think if I really need a class for it. The individual is just 3 integers, size of M1, M2 and M3.

A simpler and maybe faster approach would be to just have a vector where 0 == size of M1, 1 == M2 and 2 == M3, and code it on the population, so the population would be a matrix on 3 columns and N rows where N is the size of the population.

But for better readability and thinking on the future where the individual class can get or be complex, I will code the individual in its own class. (simple)

**The Population**

Show and explain the source code…

**The Evolutive Algorithm Class**

I like to have a class to cluster the specifics of the evolutive algorithm, and one thing that I like is having easily extendable operators, so I have pointers to each operator and class for each one of them, new operators can be implemented by extending the virtual class and can be set on the application program.

1. **The Operators**
   1. **Random Initialization**
   2. **Parent Selection**
   3. **Crossover**
   4. **Mutation**
   5. **Survival of the Fittest**
2. **Ordenation, Testing, Parameters, …**
3. **Finishing it.**

**Possible Issues:**

1. There isn’t a random number generation class, so it would require a big refactor to have one and change how the numbers and generated (Yes, there are many ways to generate random numbers)