AG Tema 3

Stefan Petrovici

January 2020

1 Introduction

This report takes a closer look at what are the particularities of the genetic algorithms by comparing a genetic algorithm for solving a SAT problem against an heuristic algorithm. Multiple problem instances with various specifications were used for describing the differences between the two.

2 Problem

The problem consists in solving k-SAT problems where we have as input a number of literals and the clauses that form a boolean formula. The goal is to determine a value for each of the variables present in the problem instance such that we can (eventually) solve the assignation. All of the problems are satisfiable. For a genetic algorithm solving decidability problems is a hard task. For that we need to provide the algorithm intermediary indicators that can be used by it for evaluating the current fitness of the population. The indicator we chose was the number of satisfied clauses, considering that a higher number of satisfied clauses would lead to a solution eventually.

3 Algorithm

The used genetic algorithm consists in a mutation stage, in which we mutate our chromosomes with a 0.02 percent probability, and once in 50 generations check up if the population is stuck in a local minima and cannot exit and "bump" it a little to change its position, and a crossover stage in which chromosomes exchange genes between themselves in a random sequence such that any chromosome is paired with at most one chromosome.

3.1 The particularities of the GA algorithm

The algorithm consists in three major procedures: mutation, crossover and selection.

3.2 Mutation

The mutation procedure allows every gene in a chromosome to change, though this chance is small (1%). The mutation gene comes first in this algorithm because it is the only procedure that actually changes the information that the population carries. We can see that in a test with no mutation the variation from one population to another is slower than that in a normal run, and that could be explained like this: if we imagine the "optimal" genes that the chromosomes are carrying as units on a distribution graph, the overall "optimality" of the population does not actually change, as the number of good genes do not change in the crossover or selection procedure.

While it is true that we can talk about optimal genes only when they are in combination with the other genes, it is also a fact that the "winning" chromosome has a range of neighbors that resemble its particular "distribution" that we have talked earlier. So we can approach the optimal solution by matching its number of favorable genes.

3.3 Implementation of mutation

The standard method for mutating chromosomes is to give each individual gene a chance to change its value with a certain probability. In our implementation this probability is 0.01f, and though this is a rather small chance, it contributed to the algorithm greatly,

3.4 Crossover

The crossover procedure is also an important part of the algorithm. We have already talked about tweaking our solution chromosome to obtain a certain number of "optimal" genes. But it is more often that we will find solutions in which the genes would be more fitting in other places.

3.5 Implementation of crossover

In our implementation the population is shuffled and then each chromosome is given the chance to crossover with its (paired) neighbour. The chosen chance is 0.2f. Through the crossover process we use the standard method: we choose an index between 1 and the maximum length L and we swap the first half that splitting the candidate would produce.

3.6 Selection

The selection procedure is the part of the algorithm that actually dictates the course of the algorithm. We say this because, while the mutation and crossover power the solving mechanism for genetic algorithms, this procedure is the part that particularizes and adapts this mechanism to the given problem. This is

not only because the selection procedure offers a semi-evaluation of the current population, but also because it is the place where the programmer can use his insight to mold the finding of particular solutions.

3.7 Implementation of selection

In the implementation of this paper we have used such a selection, in which we favored the best values and overlooked the worst, and also keeping a steady value for those in the middle. Our goal is to select a chromosome with the value randomly chosen based on a distribution of values to become a member of a new population. Besides implementing a standard approach to selecting values with a Wheel of Fortune procedure, we strived to select a value that was not necessarily as much-met as it was good (in the standard algorithm if a certain value appears often it is favored). This proved to be rather good choice, because it broke one of the disadvantages of the canonical method: in the normal selection there was the possibility that similar candidates would have a combined percentage greater then those of better solutions and therefore be chosen more often. This is often happening either because of duplicates or close values. With this method the optimal values have a greater chance to be selected and the under-performing ones are rather ignored.

So the highlight of our implementation lies in the way we compute the fitness of a chromosome. The standard formula on which we based our method was

$$fitness(x) = 1.1f * max(f(population)) - f(x)$$

and we adapted it into:

$$fitness(x) = fit\_value(x) * max(f(population)) - f(x).$$

$$where \quad fit\_value(x) = \begin{cases} 1.5 + \text{gen} \ / \ 30, & \text{if the value is among the best,} \\ & \text{gen is the generation number} \\ 1.1 - 0.01 \cdot \text{gen} \ / \ 30 & \text{if the value is among the worst,} \\ & \text{and fit\_value(x)} \geq 1.01, \\ & \text{otherwise 1.01} \\ 1.1\text{f} & \text{in the other cases} \end{cases}$$

4 General particularities

For representing a chromosome we have used a bistring vector of 1s and 0es of dimension matching the number of clauses in them problem. As mentioned, the algorithm uses a standard population of 100 individuals that are initialized with a random genome. After the intialization the algorithm is run repeatedly

and we look for the best value between the runs.

The chromosome growth is done by mutating and combining them. As we mentioned, the crossover phase has a probability of 0.2 of swapping each gene of our chosen candidates. The crossing over phase has proven to be an important phase in our process, much more important than mutation, even though without mutation we could not have crossed over our candidates.

In the mutation phase of our chromosome we look for clauses (in each chromosome of the population) that are not satisfied and we change the genes such that we get a chromosome with a slightly better fitness. We do this in a Greedy manner.

4.1 Greedy

In each run, the fitness function used evaluates each chromosome in the population and retain its value in an array. The main part of the algorithm is the selection. In this experiment we used a wheel of fortune selection that privileges the best of our population but also takes into account the number of apparitions of the specific chromosomes.

4.2 Stop Condition

The stop condition of the algorithm is to reach 1000 generations. Other stop conditions were previously used as timed conditions but in the end proved to be unnecessary as the the algorithmm finished quite rapidly.

5 Results

Below we will see the results for 30 runs of the genetic algorithm

Name	Percentage	Min	Mean	Stddev	Time(s)	Clauses	Literals
GA							
RTI_100	1.00000	0.98368	0.99565	0.00094	5	429	100
uf250	0.99812	0.99061	0.99516	0.00185	5	1065	250
CBS_b90	1.00000	0.97996	0.99121	0.00435	5	449	100
frb30-15	0.99911	0.99418	0.99828	0.00684	7	19084	450
frb35-17	0.99653	0.98876	0.99447	0.00882	8	29707	595
frb40-19	0.99171	0.97318	0.98794	0.01211	10	43780	760
frb45-21	0.99228	0.95312	0.97722	0.01221	10	61855	945
frb50-23	0.99267	0.93692	0.96476	0.01232	10	84508	1150
frb53-24	0.99092	0.91724	0.95601	0.00252	10	98921	1272
frb56-25	0.99565	0.91011	0.94809	0.00179	11	114668	1400
frb59-26	0.99417	0.89898	0.93763	0.00472	12	132295	1534
Random Search							
RTI_100	0.95105	0.93706	0.94391	0.00373	7	429	100
uf250	0.92582	0.91737	0.92013	0.00247	7	1065	250
CBS_b90	0.94878	0.93541	0.94001	0.00300	7	449	100
frb30-15	0.84867	0.82981	0.83823	0.00457	7	19084	450
frb35-17	0.84327	0.82109	0.82984	0.00628	7	29707	595
frb40-19	0.82707	0.81014	0.81765	0.00509	7	43780	760
frb45-21	0.81458	0.80352	0.80980	0.00294	7	61855	945
frb50-23	0.80956	0.79616	0.80243	0.00376	7	84508	1150
frb53-24	0.81644	0.79504	0.80151	0.00542	7	98921	1272
frb56-25	0.80867	0.79125	0.79778	0.00536	7	114668	1400
frb59-26	0.79989	0.78685	0.79410	0.00298	7	132295	1534

The SAT problems had various particularities pretending to the number of variables in clauses, with a range between 15 and 24 variables in each clause. We can observe that on simpler problems - such as the uniformly distributed ones the algorithm behaves better - this may be cause by the bitstring representation - perhaps it is easier for the algorithm to work with reccuring values.

6 Comparison against a heuristic solution

The heuristic algorithm we choose was the Random Search algorithm. This method changed one gene at random and tested if the fitness of the candidate improved. Sadly, the algorithm reached a discouraging result of finding 80%. We limited the time of the random search to match the time of the genetic algorithm.

7 Conclusions

We have proposed a genetic algorithm with specific parameters to solve the SAT problem. Though our solution did not reach 100% of the clauses, we have learned important facts. Some genetic operators are more important than others and we have plenty of options to choose from for obtaining better performance. The dimension of our population can also be increased to benefit the variety of the genome we operate on. Considering all the benefits of the genetic algorithm, we conclude from this that GA is a good algorithm for solving NP-complete problems.

References

SAT publications `https://web.archive.org/web/20060219180520/http://jsat.ewi.tudelft.nl/`

3SAT
`https://en.wikipedia.org/wiki/Boolean_satisfiability_problem`

BenchMarking 1
`https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`

BenchMarking 2
`http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/benchmarks.html`

BenchMarking 2
`https://profs.info.uaic.ro/~pmihaela/GA/`