SOLVING POWER DISTRIBUTION USING AN EVOLUTIONARY ALGORITHM

Brent Verpaalen s2158124

Brentverpaalen95@hotmail.com

UNIVERSITY LEIDEN

Athanasios Agrafiotis s2029413

Athaagrak@gmail.com

Index

INTRODUCTION

When Darwin introduced evolution as a concept which suggest that species change over time, people though he was crazy. Time showed us that Darwin was not crazy, he even became the founding founder of the evolution theory creating a whole new field of biology. Darwin's impact on the world was and still is huge. Evolution in biology explains how and why species differ. But evolution can also be used in other scientific field such as computer science, where algorithms can evolve giving better solutions for current hard to solve problems. We call this field evolutionary algorithms.
Evolutionary algorithms heavily relay on the evolution theory. Where algorithms for each new generation try to adapt creating mutated crossover offspring, just as in nature [i]. Where each generation the algorithm tries to optimize the solution even more.
Our problems consist of optimizing electric power distribution networks. Current society uses huge amount of electricity. Generated in huge amounts by different sources the electricity needs to be distributed between users. In our network we have a network of houses, our goal is to create these networks in such a way that we minimal power loss. Because the down side of these huge and complex networks is that during transmission electrics has the tendency to lose power. For huge distances we can decrease this by increasing the voltage, for the distribution between substations and houses this isn't an option. Therefore, we need to create efficient networks that create an efficient solution reducing power loss resulting in decreasing power usage reducing costs while helping the environment. In this paper we will ask the question: "How can we use an evolutionary algorithm to create an optimal electric power distribution network? ".

PROBLEM DESCRIPTION

This paper focusses on an optimal electric power distribution network. As already talked about in the introduction current society produces huge amount of electricity. When electricity gets transmitted it losses power which can be calculated with the following formula:

$$P_{loss} = I^2R$$

Where P_{loss} stands for the power in watts lost in the transmission, I stands for the number of amperes and R is the resistance of the material over the given length. For long distance transmission we decrease I to nearly zero to reduce the power los. This can be done thanks to the following formula which calculates the power:

$$P = U * I$$

Where P stands for power in watts, U for the amount of volt and I the amount of ampere. If we increase the amount of ampere we decrease the amount of volts.
When using cables high above the ground (or deep in the earth) these huge amounts of amperes do not mater as they are not used but only transmitted. But our electric devices can not process these huge number of volts and will break. This is why we decrease the number of volts, when we look at the formula to calculate power we can see that this will result in an increasement of amperes. This will also mean that the power loss will also increase. This where our problem starts. We want to decrease this power loss as much as possible.
The problem starts after conversion from high to lower voltage, this happens at so called substations. After these substations we create a network of houses combined by wires and switches. One major constraint that we have is that we can not create a loop, this will short circuit the substation creating a power outage. Therefore, we have open switches; these switches prevent the substation to short circuit bus can also be used as a back up when a network partly breaks. In our problem we want to place the open switches in the power distribution network at optimal points in the network where the powerless is minimal while avoiding short circuit and does not separate houses from the network.

Implementation

We created two algorithm: a Monte Carlo (MC) algorithm and an genetic algorithm (GA). Our main focus was on EA where MC was used as a reference algorithm using random solutions. Both algorithms were created in matlab using an edited matpower4.1 toolbox provided by Dr. K. Yang and two matlab functions where the first one checked if a solution was valid and the second function calculated the power loss of a valid solution.
 The MC algorithm creates a pre-defined random number of solutions and picks the best solution with the lower power loss.
The GA algorithm has a more complex approach. First, we create a set of random valid solutions, these are the parents of our first generation. When we start the first generation we create new solutions by first creating a uniform crossover of each parent combination creating children. After the crossover we mutate the children, check if they are valid an select the best set of solutions from the parents and children. This set is the new parent set for the next generation.

Data representation

The original dataset consists of graph where each node represents a house and an edge could represent an open or a closed switch[iii]. Before we could implement any algorithm we needed to parse this graph into a data format we could easily use as a genotype. The final data structure we used was that of a vector made of integers. Each element in the vector represents a set of edges of the graph that should have exactly one open switch, if we added more we would separate houses from the network and if we added less we would cause a short circuit. The dataset used in our experiments had a vector length of 15. Before starting the experiments we used the lower and upper bound to constrain each random solution generation and individual mutation.

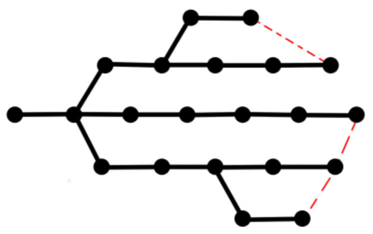


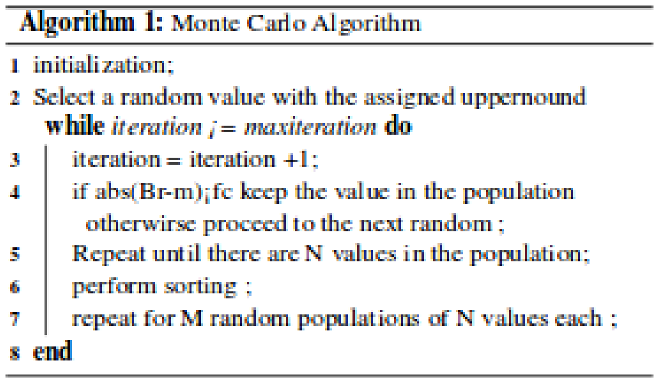
Figure 1 Example of electric network distribution

[1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15]

Figure 2 Example of possible solution

Monte Carlo

The Monte Carlo (MC) algorithm is a very basic brute force method of finding a solution. The basics behind this algorithm is that we create random solutions, check if they are feasible and take the optimal scoring solution from our results. In theory given infinity time this method will find the global optimum of a problem. In practice this method will almost never find a global solution or even a good solution. This is because of the random generation. The resulting individual could come from the whole search space, which consist of all the solution between the lower and upper bound, the MC algorithm does not follow any direction.
We implemented the MC algorithm based on the following pseudo code:



```
Algorithm 1: Monte Carlo Algorithm
initialization;
Select a random value with the assigned uppernound
 while iteration ¡= maxiteration do
    iteration = iteration +1;
    if abs(Br-m)¡fc keep the value in the population
      otherwirse proceed to the next random ;
    Repeat until there are N values in the population;
    perform sorting ;
    repeat for M random populations of N values each ;
end
```

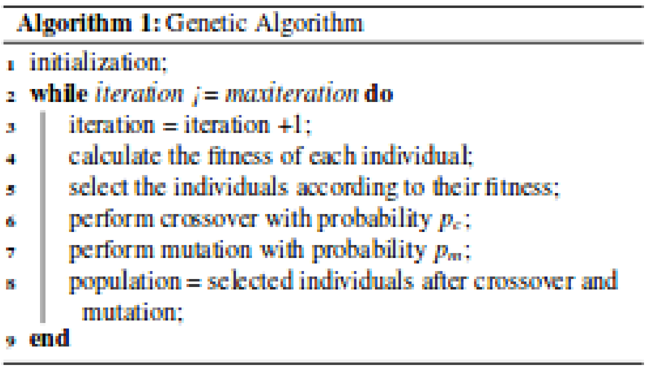
Code block 1Monte Carlo Algorithm which creates random answer and takes the best one as a solution.

Genetic Algorithm

The Genetic Algorithm (GA) tries to find a more specific direction towards the optimal solution. We have different parameters: mutation rate, number of parents, terminator and depending on the crossover method we need the indexes of the crossover.
At the start of the algorithm we create a set of parents. This first set of parents consists of random solutions. After each generation we pick a new set of parents from the current population containing the best solutions. When the parents initialization is done we start with the first generation. We create sets of partners, in our algorithm we combined each parent with each other parent. As an example: if you have 10 staring partings this creates 45 combinations, if you have 4 parents this creates 6 combinations. The formula to calculate the number of combinations is as following:

$$combinations = \sum_{i=1}^{number\ of\ parents\ -\ 1} i$$

After creating combinations we create one crossover for each combination, the resulting set are the children of the current generation. Before comparing the fitness of each individual in the population we mutated all the children with the given mutation rate. This increases the diversity of our data set and reduces the chance of a local optima. When the children are mutated we create big array adding the children and parents together to create the population. For the next generation we want the same number of parents as before, we do this by selection the best solutions from the population. After selection we start again at the combination step until the end criteria is met. Afterwards we return the best individual of the current population. We implemented the GA based on the following pseudo code:



```
Algorithm 1: Genetic Algorithm
initialization;
while iteration ¡= maxiteration do
    iteration = iteration +1;
    calculate the fitness of each individual;
    select the individuals according to their fitness;
    perform crossover with probability $p_c$;
    perform mutation with probability $p_m$;
    population = selected individuals after crossover and
      mutation;
end
```

Code block 2 Generic genetic algorithm

Crossover

To create diversion in our population we use crossover. With crossover we merge different solutions to create a new solution having elements from both parents. The goal of crossover is to create an offspring that has better elements from both parents. If an offspring is indeed better it will have a better score which means it will be very likely that this offspring will be one of the parents in the next generation keeping the good scoring genes in the gene pool and removing the worse scoring genes by replacing a worse scoring parent.

The crossover technique used in our paper is uniform crossover. When creating an offspring from a combination of parents we take for each index an gene from a random parent at the same index position from the chosen parent. Normally we have two parents which means that at each index we have a 50 percent change of having the gene of the first parent and a 50 percent change of having the gene of the second parent.

When creating the GA algorithm we also tried k-point crossover but we saw that with uniform crossover we had better results faster. Which was back up by [ii] which prefers uniform crossover compared to k-point crossover.

Mutation

One of the major keystones of evolution is DNA mutation. When a DNA gets replicated there is a chance that some points of the DNA will change which could result in a change of function. We use this same practice in genetic algorithms. When we start the algorithm we request a parameter mutation rate, this is number between 0 and 100 which indicates the change that a single gene (value) in our current individual will chance. When running the mutation we create a random number between 0 and 100, if this value is below or equal to the mutation rate we change the gene random to a number between its lower and upper bound. We use mutations to widen our research space each generation, if we would solely use crossover we are restricted to values in our initial parents as we can only use their genes.

Selection

The last phase of the algorithm selects the individuals of the population that will be the parents in the next generation. Our algorithm uses a very basic but affective way of selecting parents. We take the whole population and take the optimal scoring ones. In total we take the same number of parents from the population which means that most individuals will be removed (depending on your number of starting parents). The resulting parents will have a optimal score compared to the whole population, one of the dangers using this selection method is local optima. When a lot of individuals of the population are similar and high scoring we will have a high concentration of genes reducing diversity.

Experiments

Our experiments where run on a computer with a quad core 2.4GHz cpu and 8GB RAM. Our program developed and tested using the open-source program GNU Octave, version 4.4.1. In the end we created two algorithms: Monte Carlo (MC) which is based on randomization and a genetic algorithm. During our experiments we looked at the average and best performance of a given algorithm in a reasonable time. We also created figures showing the optimization process of each algorithm while running.

At the end we also looked at different configuration for the genetic algorithm. Changing mutation rate, parent size and even the crossover method.

Monte Carlo

Average	Standard deviation	Best power loss
1535	148	1198

To test the MC algorithm we ran it 20 times where the stop condition for each run a maximum evaluation usage of one thousand was. We wanted to give MC a maximum evaluation usage of ten thousand but could not do this due to time constrains. At table 1 you can see the result of the 20 runs. We took the average, standard deviation and the best individual. We can see an average of 1535 watt with a standard deviation of 148. With the given population that means that 97 percent of the solutions is between 1239 and 1831. The best solution of this population is at least two standard deviations away from the average with a value of 1198.
At figure x you can see a typical MC learning curve. The algorithm shows little to no improvement per evaluation but large improvements for specific evaluations. When there is improvement it is very unpredictable, we can not know hum much it will improve.

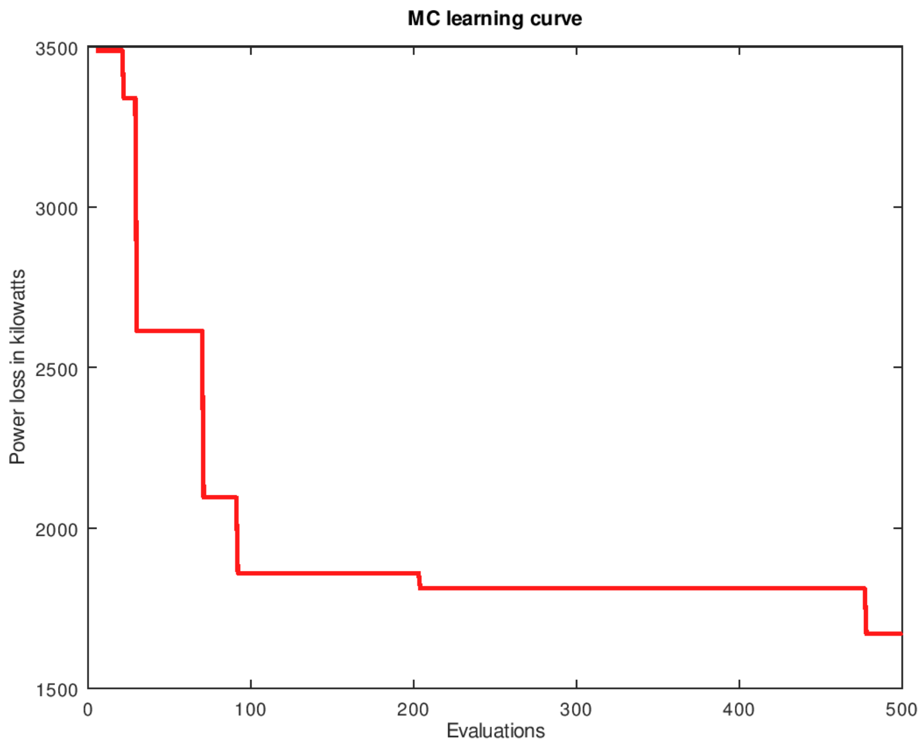


Figure 3 Monte Carlo power loss plotted against the evaluations. Using a maximal of 500 evaluations.

Genetic Algorithm

Average	Standard deviation	Best power loss
869	28.5	792

To test the GA we ran it 20 times where the stop condition for each run was a maximum evaluation usage of ten thousand, one generation contains one or multiple evaluations. During our experiment we used the following parameter settings: 10 parents, 25 percent mutation rate and uniform crossover. The results of our experiment can be found at table 2. We took the average, standard

deviation and the lowest individual from the 20 experiment results. Our lowest individual can be found at figure x in an integer array wise data structure with a power loss of only 792.
Figure x shows an the learning curve of a GA using a mutation rate of 25 and a total of 10 parents for each generation while using a maximum of one thousand evaluations. The first 100 generations we see a fast decline in powerloss which

$$[21,16,1,18,18,10,10,10,10,2,18,17,1,1,19]$$

Optimal solution found by our genetic algorithm

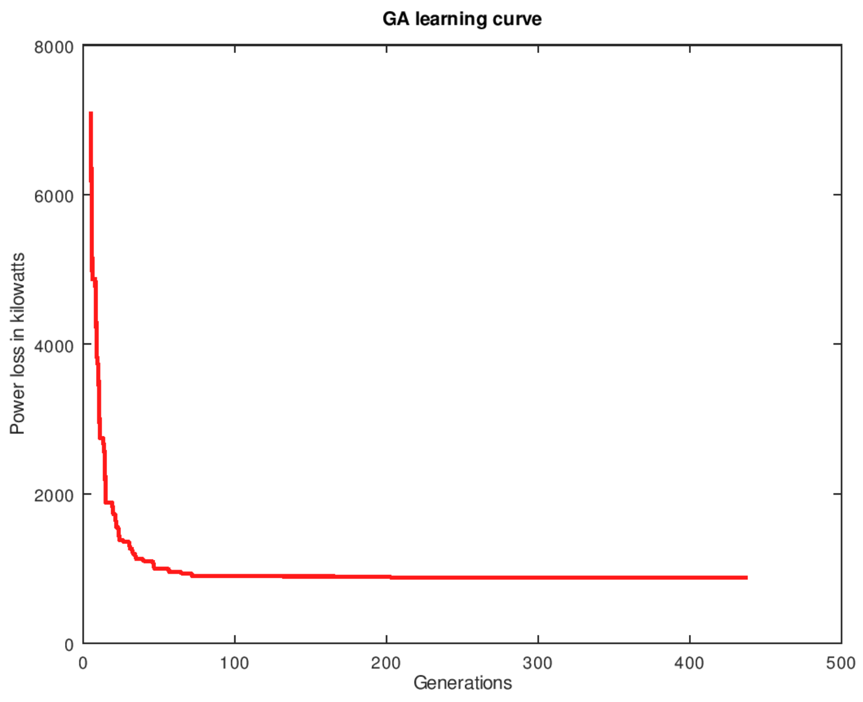


Figure 4 Genetic Algorithm power loss plotted against the generations, Using a maximal of 1000 evaluations.

Discussion and Conclusion

In our paper we wanted to create an optimal electric power distribution network with the help of an evolutionary algorithm. To have something to compare we also created a Monte Carlo algorithm solving this problem randomly. We saw that Monte Carlo can find a result averaging around 1500 power loss with a best scoring one of 1198. This was very time consuming as Monte Carlo creates a lot of faulty solutions increasing the computing time. We even had to lower our expectations because Monte Carlo took too long for a decent run. This is very unpractical as we often need a solution relatively fast. The genetic algorithm showed huge improvements, having an average power loss solution of 869 with a best result of 792.
We are very happy with these results even surpassing current know evolutionary algorithms by finding a solution which decreases the power loss almost 9 percent (where before the best solution was 869).
We can also find some improvements in our algorithm. The first improvement is optimizing the parameters. During the creation of this paper we did not create a huge variety of parameter experiments. We could maybe even decrease the average found power loss by using hyperparameter optimization. The second improvement we would like to propose is the changing the selection method. The current selection method only takes the best scoring individuals of a population. Our idea is to use tournament selection increasing the chance of diversity in the parent set of the new generation.

REFERENCES

i) Smith, G.P.. (1976). Smith GP. Evolution of repeated DNA sequences by unequal crossover. Science 191: 528-535. Science (New York, N.Y.). 191. 528-35. 10.1126/science.1251186.

ii) Jośe Fernando Gonc¸alves, Jorge Jośe de Magalh̃aesMendes, and Maurıcio G.C. Resende. “A hybrid geneticalgorithm for the job shop scheduling problem”. In:Eu-ropean Journal of Operational Research167.1 (2005),pp. 77–95.ISSN: 0377-221

iii) Zhu, J. (2002). Optimal reconfiguration of electrical distribution network using the refined genetic algorithm. *Electric Power Systems Research*, *62*(1), 37-42. doi: 10.1016/s0378-7796(02)00041-x