Solving *p-Hub* problem with a Steady State Genetic Algorithm

Esteve Soria Fabián
essofa@alumni.upv.es

July 2022

1 Introduction

In this work I develop an algorithm aiming to solve the single hub location problem. Due to the nature of the hub location problem, the use of meta-heuristics can simplify the process of finding an optimal solution.

In this work the optimal solution for the dataset provided in the original paper[1] is found. The solution for 2, 3 or 4 hubs is studied but this can this system can be expanded for more.

There is, also, a study of hyperparameters selection to achieve the solution in the least resource intensive way.

The code used in this work can be found here `https://github.com/sorny92/genetic_algorithm`.

2 *p-Hub* problem

A common problem to deal with in logistics is the hub location problem. A hub is a location that serves as a point of connections for different locations. If a person needs to go from A to B by plane is probable that there's no direct connection between this points so the user needs to traver to a hub that will connect them to the point B or a hub that is connected to the point B. Hubs have the purpose of connecting non-hub locations. Normally these hubs come with a cost, but they also bring economy of scales so transporting between hubs can be beneficial.

Knowing where to create a hub is important because it will help to decrease the cost of transportation optimizing the flows and inherit costs.

To define the problem we need to consider a series of nodes and each one of them has a flow that needs to go from the node i to the node j. This travel from node to node has a cost. So the goal to solve the problem is to minimize the cost of the whole network.

One of the key elements of this problem is that is not possible to have an optimal solution in an acceptable time due to the high combinatorial characteristics of it.

There are several variants of this problem. For example, the single hub problem which considers there is only one hub to allocate. Then you have the p-hub which you have a p number of hubs to allocate where every non-hub is connected to only one hub. These connections could be set through an optimal policy, so it's another part of the problem to optimize or could be allocated by distance/cost. There are more variants with multiple connections from non-hubs to different hubs, variants with limitations in the hub flows, variants with costs on setting up links, etc...

In this essay we will focus on the p-hub problem where the assignation of non-hubs to hubs is done through cost minimization. The code in this system allows to modify an α value to proportionally decrease the cost of hub to hub connection but because we use the minimum cost allocation this value does not change the result.

3 Method

3.1 Steady stage genetic algorithm

The implementation of this solution is based on the code developed by E. Alba here `https://neo.lcc.uma.es/software/ssga/index.php`. This software package is developed with Java but the implementation used in this work is reimplemented in C++ to know more in deep how to develop this kind of systems.

To solve the hub location problem I implemented a steady state genetic algorithm. This type of genetic algorithms work with a defined sized population of individuals.

In every iteration the fitness of each individual is calculated, then we apply selection of genetic operators to the population.

In this case, two individuals are selected through binary tournament, which means that for every tournament there are two randomly selected individuals and the one with the best fitness is returned.

Then a single point crossover operator is applied. SPC operator takes two individuals then based on a given probability this operator will be applied, in case of it being applied an individual will be returned that will be a crossover of the genomes of both individuals. This crossover point is selected randomly. In the contrary case where the crossover is not going to be done, one of the input individuals is selected randomly as the returned.

Once we have an individual from SPC the mutation operator is applied, this will mutate each gene of the genome based on a given probability.

This individual that is received from the mutation operator will substitute the worst performer of the whole population.

3.2 Genome design

In the previous section the ssGA is explained, in this section the genome encoding is explained.

There are several ways to encode the genome. Originally in this work it was implemented as a binary vector where there were p number of bits as

total number of nodes in the problem. This approach allows to fit the GA to the problem but it makes limiting the number of hubs problematic.

The second version uses an integer encoded vector where there are p genes as the number of hubs to be allocated in the graph. This version is much better because it limits the amount of hubs that can be assigned in a natural way. There's only a need to check that the values are not repeated in each gene. Doing it this way all the information to solve the problem is encoded because the value of the integer will indicate the index of the hubs and then the assignment of non-hubs happens thought nearest cost allocation.

Another potential version is a two-allele integer gene encoding. [2] This could be really useful to solve a single-HLP without nearest hub assignment because every gene will set in the first allele if it is a hub and the second to which hub is linked.

4 Experiments

In this work the number of iterations, individuals, length of the genome, crossover probability and the probability of mutation can be configured. So the experiment has been set to do 500 iterations in each run and 30 runs by set of parameters, then there's a grid generation for the next values:

- Number of individuals: [5, 10, 25]
- Genome length (number of hubs): [2, 3, 4]
- Probability of crossover: [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
- Probability of mutation: [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]

The parameter selections is inspired from the work at [3].

A script is written to run all this experiments and every run will have the best performer, worst performer and average fitness. Also, the genome of the best performer, all of this per each iteration.

In the next section there's an analysis of the results

5 Results

In this section I show the results of the experiments. The analysis takes into account the parameters selected and which ones would be ideal for this use case. Then analyze how the influence to achieve an optimal solutions. Lastly, how this parameters affect the time to converge to an optimal solution.

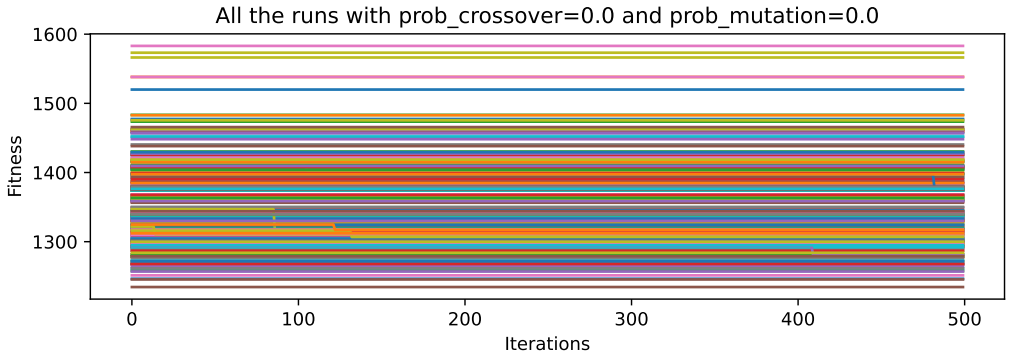

Figure 1: When the probability of crossover or mutation is 0 there's no possibility of improvement.

Depending on the probability values selected for crossover or mutation the speed to achieve an optimal solution will be different. As it can be seen in Figure 1, due to not being able to generate new different genomes there's no improvement in the fitness metric.

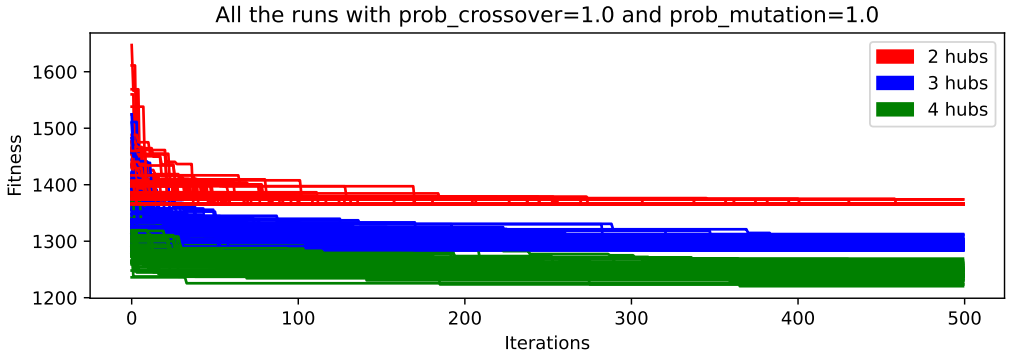

Figure 2: When the probability of crossover or mutation is 1 the variability of final results is big due to the big pool of different genomes.

In the opposite direction of Figure 1, we can observe in Figure 2 when

the crossover and mutation probability is at the maximum the variability increases fast so the system is able to achieve a bigger variability of results.

Another interesting observation in Figure 2 is the different fitness values that are achieved based on the number of hubs. As it should be obvious the more hubs you have the more interconnected is the network so the lower the cost is. This is why having hubs of zero cost does not make sense because your ideal fitness would be to make a hub in every node.

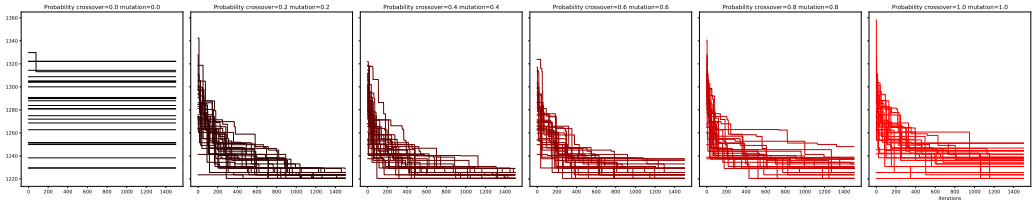

Figure 3: In this figure a sweep from 0 to 1 of the two variables can be seen.

In Figure 3 it can be seen that with a bigger value of probability the spread in the final results is bigger. This is due to the big changes that happen to the genome. Every iteration a new individual with a lot of mutations is added, and it has been created from random individuals with ver diverse genome keeping the population with a wide spread of genomes.

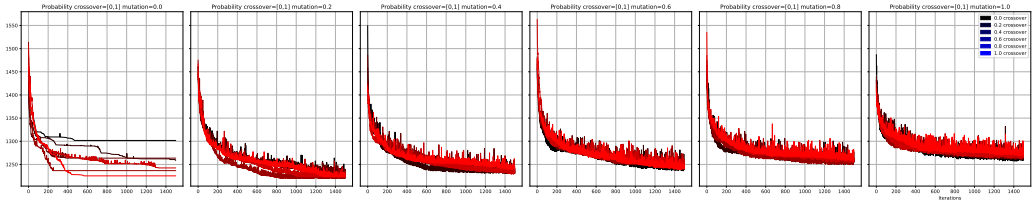

Figure 4: Similar graph as Figure 3 but in this case the average fitness of the best run for that set of values is plotted. The intensity of colour indicates a higher probability of crossover. Left to right an increased probability of mutation.

In Figure 4 We can observe that high probability values increase the variability and so it does the average value. This means achieving an optimal result takes much longer.

Based on the data, a probability of mutation around 0.2 is the best to achieve a fast optimal result. Then applying crossover with a probability between 0.4 and 0.6 looks like best setting in this case.

On Figure 5 it can be seen at high probabilities of mutation it usually takes less time to achieve an optimal solution. It can also be observed that the higher ratio of success happens in the range of [0, 0.2] in probability.

It can also be observed that with higher crossover probability there is higher chances of finding an optimal solution although this influence is not as strong as with setting a proper value of mutation.

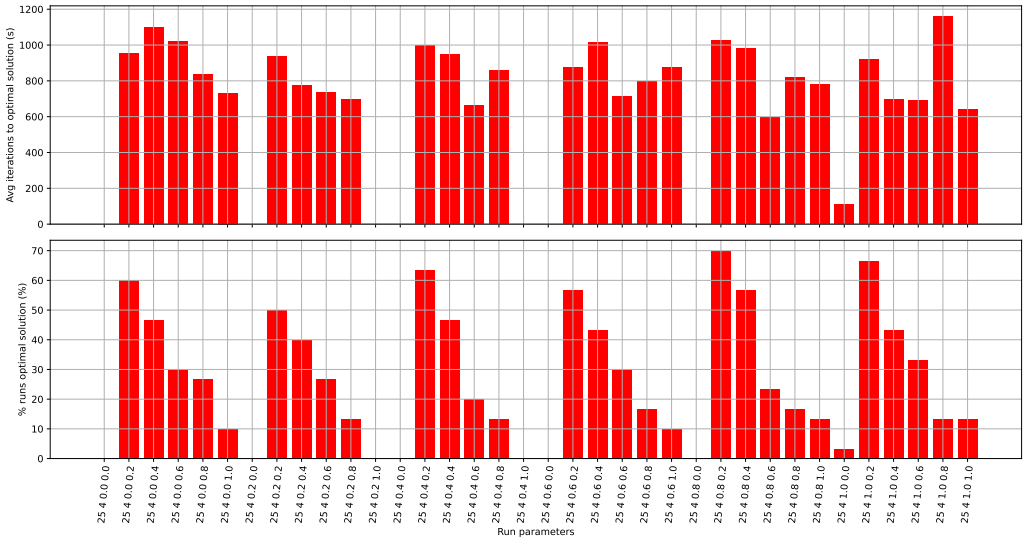

Figure 5: Top graph shows the average number of iterations to achieve the optimal solution in each set of hyper-parameters. Bottom graph shows what percentage of run with the same set of hyper-parameters achieved in less than 1500 iterations a solution

Lastly in Figure 6 it can be seen than a mutation probability close to $1/n\_genes$ is the ideal value to achieve an optimal solution fast. Also the increase of crossover probability helps to have higher success but not as important as the right value for mutation probability.

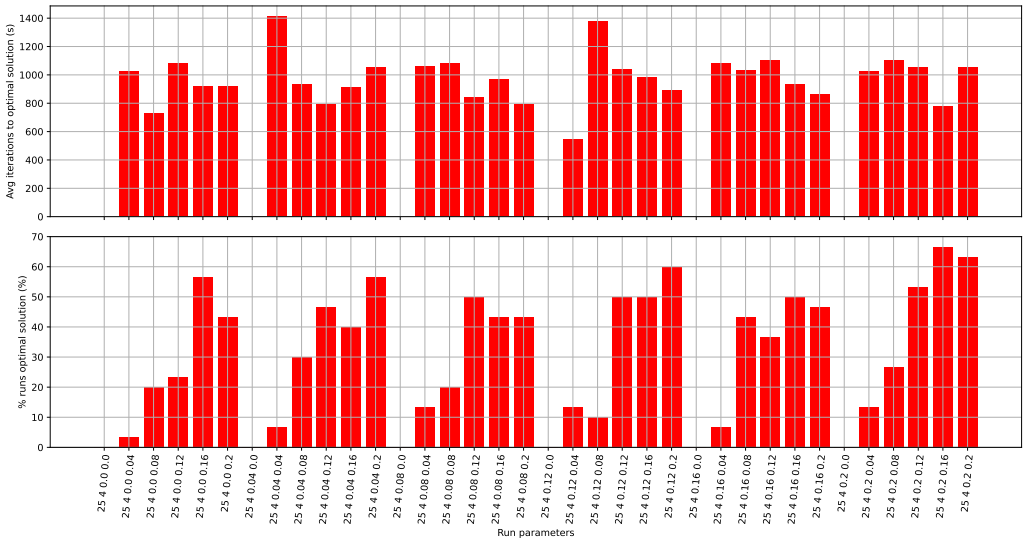

Figure 6: Same graph as Figure 5 but with a probability range of [0, 0.2] for mutation and crossover

6 Conclusion

Implementing the algorithm from scratch allowed to understand better how genetic algorithms work. Also implementing it in C++ brings an edge of performance to be able to run plenty of experiments. This implementation has parallel run for the fitness calculation which in the machine used for this work provided around x10 more performance compared to the single threaded version.

Having a right set of hyper-parameters in the genetic algorithm can help to accelerate finding an optimal solution. In this case a probability of $1/n\_genes$ for mutation operator and a probability close to 1 for the crossover operator shows the best chances of finding an optimal solution in the fastest time.

References

[1] Morton E. O'kelly. A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, 32(3):393–404, 1987.

[2] Zorica Stanimirović. Solving the capacitated single allocation hub location problem using genetic algorithm. 11 2007.

[3] YiYe Zhou, DengKai Yao, QianRui Sun, and QiKe Wu. Application of genetic algorithm in p-hub airline network design problem. In *Proceedings of the 2nd International Conference on Electronics, Network and Computer Engineering (ICENCE 2016)*, pages 298–303. Atlantis Press, 2016/09.