U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Metaheuristics for Optimization/Decision Problems

IART 1st Assignment

Work specification

The selected theme of our project was Optimization Problems.

In this case, our goal was to devise a solution for the Hash Code 2017 Final Round problem *Router Placement*, using metaheuristics.

Given a building plan, decide where to put wireless routers and how to connect them to the fiber backbone to maximize coverage and minimize cost.

The score of an attempt is calculated based on the formula:

```
Score = 1000 × Target cells with coverage + Remaining budget

Remaining Budget = Total budget - (Routers × Price of Router
+ Backbone × Price of Backbone)
```

Input data

The first line contains the following numbers:

- H (1 ≤ H ≤ 1000) - number of rows of the grid
- W (1 ≤ W ≤ 1000) - number of columns of the grid
- R (1 ≤ R ≤ 10) - radius of a router range

The second line contains the following numbers:

- Pb (1 ≤ Pb ≤ 5) - price of connecting one cell to the backbone
- Pr (5 ≤ Pr ≤ 100) - price of one wireless router
- B (1 ≤ B ≤ 10^9) - maximum budget

The third line contains the following numbers:

- br, bc (0 ≤ br < H, 0 ≤ bc < W) - row and column of the initial cell that is already connected to the backbone

Example

```
8 22 3
1 100 220
2 7
----------------------
-########----########-
-#......######......#-
-#..................#-
-#..................#-
-#..................#-
-####################-
----------------------
```

1st line - H=8, W=22, R=3
2nd line - Pb=1, Pr=100, B=220
3rd line - the initial cell connected to backbone is [2, 7]

“-” - Void cells
“#” - Walls
“.” - Target Cells

Implementation work

The selected language to implement our solution was Python 3.

The development environment used by the group to to develop the project was Visual Studio Code.

The user might choose the problem building they want to solve as well as the algorithm to do it.

After calculating the final solution, information about it is printed and the building grid is also plotted.

Approach

The evaluation function we used was the score function mentioned in the Hash Code problem statement.

Each time we evaluate a solution, we make sure it doesn't exceed the budget, so we reduce the cutoff (that is, the number of routers we'll consider for our solution), but we also try to increase it (include a router that was previously discarded) if the remaining budget allows for that.

Previously, reducing or increasing the cutoff used to be an operation that was performed to generate neighbours, but this ended up allowing for invalid solutions.

So, despite calculating the necessary cutoff at each evaluation being more costly, it performs better at giving good solutions.

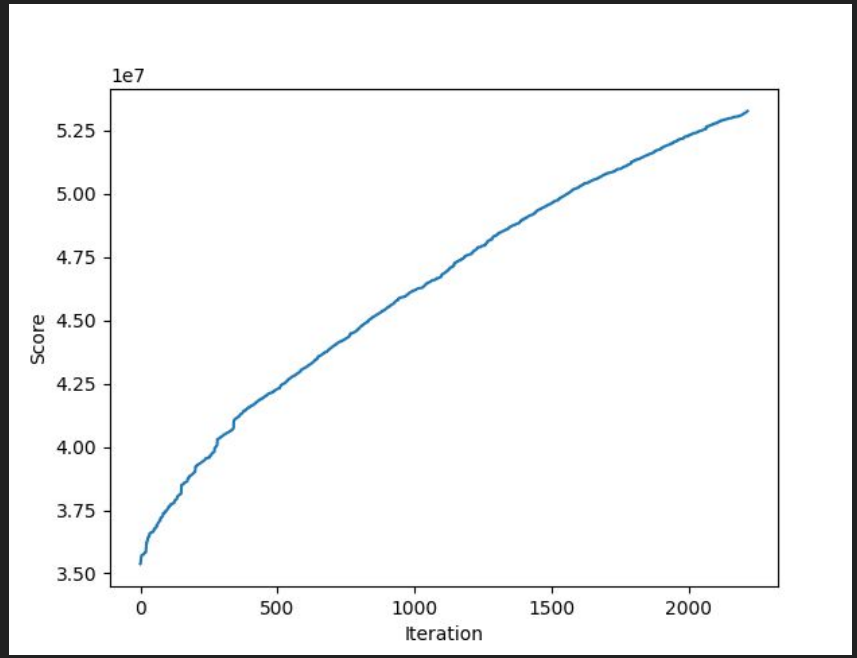
We start by generating a random starting solution for the problem, distributing the maximum possible number of connected routers by the empty cells.

Algorithms - Hill Climbing

We implemented both the random and steepest ascent versions of the hill-climbing algorithm.

Starting by a randomly generated solution, we search for a better solution by generating a given solution's neighbours and evaluating them using our heuristic so we find the best one.

The possible neighbours are generated using the operation of moving a given router to one of the 8 possible adjacent cells (or the closest possible if one of those is blocked by a wall), so, we'll have a branching factor of $8 \cdot n$ where represents the number of placed routers.



53271000 score
2000 seconds
Rue de Londres

Algorithms - Simulated Annealing

We followed the standard method for simulated annealing: starting with a randomly generated solution. Then we go through the neighbours of that solution:

- If the solution is better than the one we have we keep it
- Else we accept it with a probability of $e^{\Delta fitness/T}$

After each iteration we decrease the temperature based on a certain cooling schedule. It stops when the temperature reaches a small value or the current solution has no more neighbours.

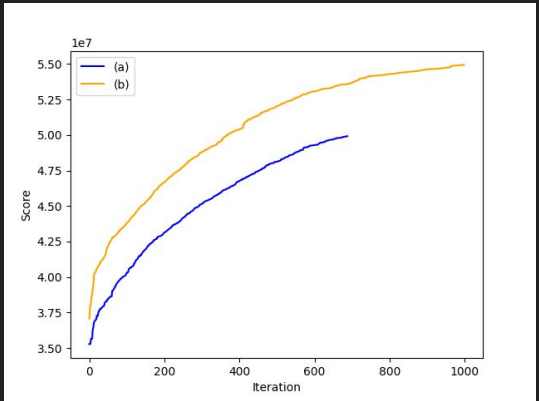
Cooling Schedules:

- $t = t_0 \cdot 0.95^{iteration}$ (Exponential multiplicative cooling)(a)
- $t = t / (1 + \beta \cdot t)$; $\beta \sim 1/(radius \cdot 1000)$(b)

Improvements:

Based on research we made the following improvements:

- The returned solution is the best found so far and not the final one.
- There is a number of iterations per temperature.



Algorithms - Genetic Algorithm

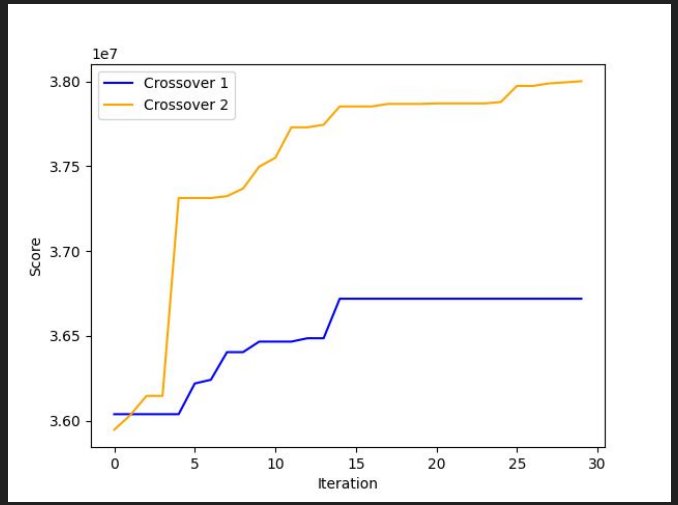
We followed a fairly standard method for the genetic algorithm, with a partial generation gap population update (we keep the 30% best members from the previous generation), with a variation on single (and multi) point crossover.

Our algorithm utilizes elitist selection, which, in our implementation, means it picks random parents to breed from the 50% best performing from the previous generation.

We tried 2 different types of crossover and mutation methods:

Crossover 1 - We split the map with a figurative horizontal line. The child gets every router above that line from one parent and every router in or under the line from the other.

Crossover 2 - Same as the first method, but with multiple lines and the parent that gives the routers to the children in each line is random.



Algorithms - Genetic Algorithm

Mutation 1 - Move a router to a tile directly adjacent or diagonal to it (selecting a possible neighbour of the current solution).

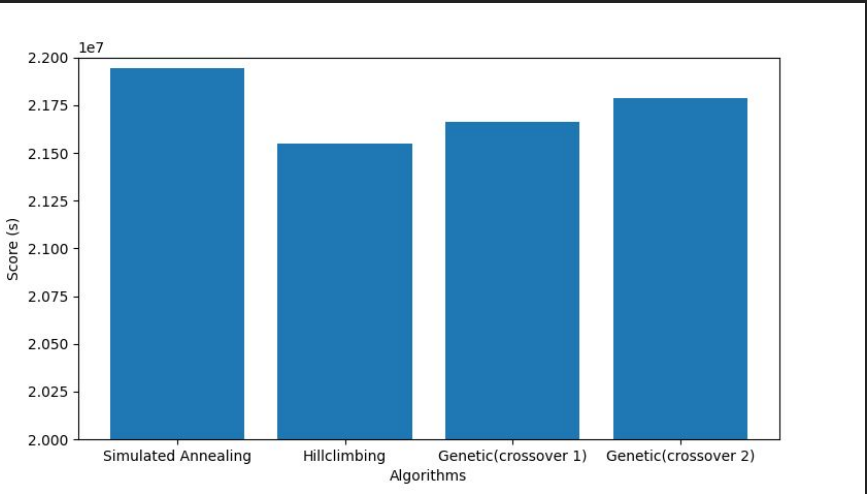
Mutation 2 - Select 10 random routers inside the cutoff, create and evaluate the coverage of solutions with each of those routers moved to the end of the list and select the solution with the most coverage (if there isn't any, it keeps the old solution)

We found that the second crossover method was slightly better overall.

We found that the second mutation performed slightly better overall, but it takes about twice as long, so the best method is subjective. We ended up using a 20% mutation rate.

In our opinion, the algorithm experienced diminishing returns when the number of iterations was greater than the population size.

Results analysis



Hillclimbing: 200 iterations

Simulated Annealing: Exponential Multiplicative Cooling (a = 0.97)

Genetic:

- Populations size: 8
- 20 iterations

These results were obtained by limiting the number of iterations in the case of the hill climbing algorithm, by limiting the temperature decrease rate in the simulated annealing one and by limiting the population amount and number of iterations on the genetic algorithm.

This was done so that each algorithm has a runtime of approximately 200 seconds.

As can be seen by the graphic all the algorithms produce similar results for this building with the difference between simulated annealing and the hillclimb algorithm with crossover 2 being 400000 points.

Conclusions

This project allowed us to gain more in-depth knowledge of the topics taught in the lectures and we think all the algorithms were correctly implemented, therefore accomplishing the objectives proposed for the assignment.

That said some areas could be improved upon:

- A different evaluation function/operation to generate neighbors that allowed for significant score improvements
- Improving the calculation of the minimum spanning tree (or even use an approximation), since this is the most costly operation we have (and it's performed many times)
- Improving upon the crossover function used on the genetic algorithm

Related work

- sbrodehl/HashCode
- admirkadriu/router_placment_ga

References

- *Genetic Algorithms for Efficient Placement of Router Nodes in Wireless Mesh Networks*
- *Handbook of Natural Computing*
- *Artificial Intelligence: A Modern Approach; Stuart Russell and Peter Norvig*
- *A Comparison of Cooling Schedules for Simulated Annealing (Artificial Intelligence)*