Router Placement

Artificial Intelligence

Project 1 - Final Delivery

Group 131

Bruno Oliveira	202208700
Henrique Fernandes	202204988
Rodrigo Silva	202205188

FEUP - L.EIC029

Problem Description

Topic 3a: Optimization Problems - Router Placement

Given a `m×n` grid of cells (void, walls or target) in a certain initial position, obtain maximum coverage of target cells covered by routers.

Routers have a range of `R`, covering every surrounding cell where there are no walls blocking the signal. Every router has to be connected to the backbone and they cannot be placed in wall cells. Initially, only one cell is connected to the backbone. Consider connections in all 8 adjacent cells.

Placing routers has a price `Pr` and connecting a cell to the backbone has a price `Pb`. In the final solution, the sum of these costs cannot be higher than the initial budget `B`.

Link to router placement problem description

Related Work

Team *Gyrating Flibbittygibbitts*' Solution

This solution serves as an example of how to approach the problem, providing a baseline implementation. It is important to us as it helps us understand the overall strategy, though it does not use the approaches we are exploring in our project.

Steiner trees

The goal of a Steiner Tree Problem is to find the minimum spanning tree in a graph which includes a set of terminal nodes (not necessarily every node). This is important to our problem because it helps us find the best way to connect routers efficiently, minimizing overall costs while maintaining optimal network coverage.

Paper on Genetic Algorithms for Router Placement

Sylejmani K, Kadriu A, Ilazi E, Krasniqi B. Genetic algorithms and greedy-randomized adaptive search procedure for router placement problem in wireless networks. *Journal of High Speed Networks*.

https://doi.org/10.3233/JHS-190616

Problem Formulation

- **Solution Representation:** `S = (R, C)`, `R` = list of coordinates of routers, `C` = list of coordinates of connected cells
- **Neighborhood/Mutation Function:**
  - Place router and connect to backbone
  - Remove router and reconnect nodes
- **Crossover Function:**
  - Extract routers in a random rectangular area, disconnect backbone along the border, exchange routers in the areas of both solutions and reconnect them
- **Hard Constraints (final solution):**
  - All routers have to be connected to the backbone, cannot be placed on walls and the solution must respect:

$$(\text{NumberRouters} \times \text{RouterPrice}) + (\text{NumberBackboneCells} \times \text{BackbonePrice}) \leq \text{Budget}$$

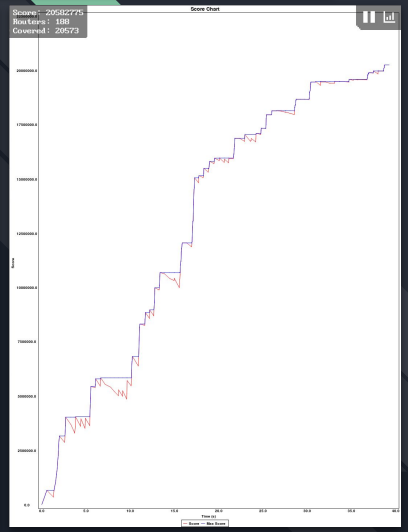
- **Evaluation Function:**

$$1000 \times \text{ConnectedCells} + (\text{Budget} - (\text{NumberRouters} \times \text{RouterPrice} + \text{NumberBackboneCells} \times \text{BackbonePrice}))$$

Implementation Work

- Development Environment
  - Python -> Programming Language
  - NumPy -> Matrix Structure
  - pygame -> Graphical Interface
  - pygame-chart -> Time/Score Graph
- Data Structures
  - NumPy Array -> Map Representation
  - Union-Find Data Structure -> Reconnection Algorithm
- Implemented Features
  - Input File and Solution Parsing
  - Interactive Command-Line Interface w/ Selection of Different Algorithms and Parameters
  - Problem Map and Optimization Display
  - Time/Score Graph Visualization
  - Solution Scoring and File Dumping

```
❯ ./main.py
[router-solver]# help
Commands:
    load <file>             Load a problem from a file.
    load-solution <file>    Load and show a solution from a file.
    show                    Show the current problem.
    solve                   Solve the current problem.
    exit                    Exit the program.
    quit                    Exit the program.
    help                    Show this help.
[router-solver]#
```



Approach

Heuristics

Steiner Trees Approximation:

We used a (2 - 2/t) cost ratio heuristic for approximating the repositioning backbone cells after a router is disconnected. This algorithm performs a BFS from every unconnected router and merges paths when they meet. A Union-Find Disjoint Set is used to track router connection.

Evaluation Functions

The evaluation functions used are the ones specified in the Hash Code 2017 problem description and mentioned previously:

```
Routers = RouterPrice x NumRouters
Cells = BackbonePrice x NumCells
Price = Routers + Cells
Money = Budget - Price
Connections = 1000 x ConnectedCells
Score = Connections + Money
```

Operators

- **Neighborhood/Mutation Function:**
  - Place router and connect to backbone
  - Remove router and reconnect nodes
- **Crossover Function:**
  - Extract routers in a random rectangular area, disconnect backbone along the border, exchange routers in the areas of both solutions and reconnect them

Note: Operators were given some sense of optimality to improve performance, otherwise local search algorithms (like random descent) would stop immediately and the solutions would take much longer to compute for simpler operators, especially with bigger inputs

Implemented Algorithms

Due to the large branching factor of our approach, neighbors are lazily chosen at random based on operations.

- **Random Walk:** at each step, an operation is used to generate a neighbor which is then explored. The only parameter is the maximum number of iterations (neighbors explored).
- **Random Descent:** at each step, an operation is chosen and if it improves the current score, that neighbor is then explored. The parameters are the maximum number of iterations and neighborhood size.
- **Simulated Annealing:** at each step, an operation is chosen and if it improves the current score, then that neighbor is immediately explored. If it doesn't improve the score, it can still be accepted based on a probability given by $e^{\frac{\Delta s}{t}}$ The parameters are the initial temperature, the cooling schedule and the maximum number of iterations.
- **Tabu Search:** at each step, a subset of the neighbors is chosen and the best one that is not tabu is chosen as the next one (even if it doesn't improve on the best solution found until now). Members of the tabu set are neighbors whose operators include adding or removing routers close to a previous position. The parameters are the tabu tenure, the neighborhood size and the maximum number of iterations.
- **Genetic Algorithm:** at each step, a new generation is created based on the current population. Pairs of the current population are chosen to perform crossover and mutate. Members of the current population who are better fitted have higher chances of being chosen. The parameters are the population size, maximum number of routers in initial solution, mutation probability, maximum number of generations, neighborhood size and if there should be a final memetic improvement.

Experimental Results

Mega Cactus Farm (≅ 5 minutes)

Algorithm	Parameters	Score
Random Walk	100.000 iterations	399050
Random Descent	Best Accept	432054
Simulated Annealing	Cooling Schedule: 0.999 Temperature: 10000, 2000 iterations	432043
Tabu Search	Best Accept, Default tenure: $\frac{\sqrt{targets}}{range^2}$ 4000 iterations	462053
Genetic Algorithm I	Population 50, 750 generations	454046
Genetic Algorithm II	Population 100, 400 generations, max neighborhood 100	462050

Charleston Road (≅ 30 minutes)

Algorithm	Parameters	Score
Random Walk	16000 iterations	21781123
Random Descent	Max neighborhood 20	21958642
Simulated Annealing	Cooling Schedule: 0.999, 3000 iterations	21960082
Tabu Search	Max neighborhood 100, 750 iterations	21960371
Genetic Algorithm	Population 30, 200 generations, max neighborhood 50, memetic improvement	21960016

Conclusions

In this project, we explored the application of metaheuristic algorithms to tackle a complex real-world optimization problem—the Google Hash Code 2017 Finals problem on Router Placement.

We developed a problem formulation and implemented multiple algorithms and metaheuristics: Random Walk, Random Descent, Simulated Annealing, Tabu Search and Genetic Algorithms. Through testing on multiple problem instances, we observed that the different approaches can be considered adequate for different situations.

On small instances, Tabu Search and Genetic Algorithms thrive due to the way they are able to explore the solution space. However, on larger inputs, these approaches end up taking too much time, and, although they would eventually converge to better solutions, that would take much longer than it would take to obtain a “decent” solution with other approaches.

The large inputs that were given in the Google Hash Code 2017 were also part of the reason why we formulated the problem and implemented the solution using some sense of optimality, as, otherwise, the programs would take much longer to generate good solutions.

Overall, the project demonstrated the effectiveness of metaheuristics for solving hard optimization problems where exact methods are impractical.

References

Google. (2017). Hash Code 2017 final round problem statement. Retrieved from https://storage.googleapis.com/coding-competitions.appspot.com/HC/2017/hashcode2017_final_task.pdf

Brodehl, S. (n.d.). HashCode Final Round Solution [GitHub repository]. Retrieved from https://github.com/sbrodehl/HashCode/tree/master/Final%20Round

Wikipedia contributors. (n.d.). Steiner tree problem. Wikipedia. Retrieved April 4, 2025, from https://en.wikipedia.org/wiki/Steiner_tree_problem

Wu, B. Y., Widmayer, P., & Wong, C. K. (1986). A new approach to the Steiner tree problem. Acta Informatica, 23(3), 315–323. https://doi.org/10.1007/BF00289500

A genetic algorithm approach for Google Hash Code problem. (2019). Journal of High Speed Networks, 25(3), 247–255. https://doi.org/10.3233/JHS-190616