Router Placement Optimization Problem

IART - Checkpoint 1, Group 44

Afonso Caiado, up201806789
Diogo Nunes, up201808546
João Pinto, up201806667

Router Placement

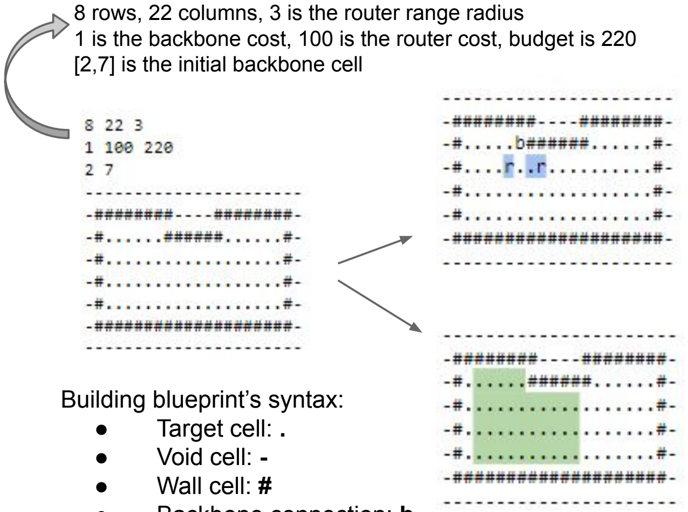
Objective: Given a building plan, decide where to put wireless routers and how to connect them to the fiber backbone to maximize coverage and minimize cost

The program receives a text file as input, that contains the blueprint of the building and the area that should be wirelessly connected to the internet. In the blueprint there is a socket that represents the fiber backbone, which provides the routers with access to the internet.

The input file specifies the building's blueprint, the range of a router, the cost of a tile of a backbone connection, the cost of a router, the budget and the cell where the fiber backbone is connected.

Note that the wireless connections cannot pass through walls. Void cells ("-") don't need wireless internet coverage. A router covers a square of side 2*R+1, being R its radius, centered in the router's position.

The cost of a router's connection to the backbone fiber is calculated by N*Pb, being N the number of cells of the connection to the backbone, and Pb the cost of a connection that is provided by the input file. The total cost of a solution is the sum of all the routers' costs and the costs of all cells connected to the backbone. The total cost cannot exceed the budget.



Building blueprint's syntax:

- Target cell: **.**
- Void cell: **-**
- Wall cell: **#**
- Backbone connection: **b**
- Router cell: **r**

Related Work & References

https://core.ac.uk/download/pdf/159237783.pdf

https://www.cse.ust.hk/~gchan/papers/ICC14_PRACA.pdf

https://stackoverflow.com/questions/50452893/finding-the-optimal-location-for-router-placement

https://github.com/sbrodehl/HashCode

https://onlinelibrary.wiley.com/doi/full/10.1002/itl2.35

---

https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)

https://storage.googleapis.com/coding-competitions.appspot.com/HC/2017/hashcode2017_final_task.pdf

Problem formulation

Solution representation (in Python):

Note: Identifiers starting with x and y are integers

```
[[xRouter1, yRouter1], [xRouter2, yRouter2], ...]
[[Router1Backbone1, Router1Backbone2, ...], [Router2Backbone1, Router2Backbone2, ...], ...]

RouterPBackboneQ: (xBackCellP, yBackCellQ)
```

Our representation consists on **2 lists of length N**, where N is the number of routers to place. The first list consists of **router coordinates** and the second represents **paths from the backbone to the routers**. Note that the same index in each list corresponds to information regarding the same router.
The path is represented by a **list of tuples** where each tuple contains the **coordinates** of a backbone connected cell. We use **tuples** because the path will determined by the **A* algorithm**.

Neighborhood/Mutation Functions: Only the router coordinates will be mutated.

```
def mutate (xRouter, yRouter) :
    randomX = rand(-1,1)
    randomY = rand(-1,1)
    return xRouter + randomX, yRouter + randomY
```

Crossover Functions:

Our chromosomes will be the router's coordinates. We are still in doubt about what crossover algorithm to use (Single-point crossover, Multi-point crossover, and Uniform crossover).

```
def crossover (sol1, sol2) :     # sol1 and sol2 are lists containing routers' coordinates
    chooseMultiCrossPoint() / chooseSingleCrossPoint()
    cross(sol1, sol2)
    return sol1, sol2
```

Problem formulation

Strong constraints: these constraints can never be broken, as they automatically invalidate a solution.

- Two routers cannot be placed in the same cell
- A router cannot be placed in a **wall** cell
- Every router must be connected to the **backbone**
- The cost cannot exceed the available budget: N*Pb + M*Pr <= B

Weak constraints: these constraints, if broken, will not make the solution invalid, they will probably just make it a weaker solution performance wise.

- A router's provided wireless connection cannot go through **walls**
- A **void** cell does not need to have wireless connection

Evaluation Functions: each submission has a score, and the higher the **score**, the better the solution is.

score = 1000*t + (B - (N*Pb + M*Pr))

- 1000 points for each **target** cell covered with Internet access;
- 1 point for each unit of **remaining budget**

Current work progress

Programming language: We are using Python as a programming language.

Development environment: We are using Spyder as our development environment.

Current progress: At this point, we have developed functions in charge of reading the given problem input files.

Data Structures: Still not implemented, however we intend to use classes to store the inputs and to build modular code.

File Structure: For now, we have an *inputs/* folder (with all the input files we are testing) and a *src/* folder, where we pretend to structure our work on different independent modules, where each module has different files, mainly composed by classes.

Algorithms to implement:

- Simulated Annealing
- Hill-Climbing
- Tabu Search
- Genetic Algorithms
- A* Algorithm