Operations Research. Laboratory Session

Introduction to Heuristic Optimization with `metaheuR`

(TSP problem)

The aim of this laboratory session is to learn how to implement heuristic algorithms for solving combinatorial optimization problems using the `metaheuR` package in R. Be careful, in R there exists a package called `metaheur`, but that's not the one we'll use.

1. Installing `metaheuR`

You can install it directly from RStudio. Download the file `metaheuR_0.3.tar.gz` from the eGela platform to a working directory in your computer. I saved it here:

/Users/JosuC/Desktop

To install the package, write the path that corresponds to the working directory where you saved the file `metaheuR_0.3.tar.gz` and execute the following commands:

```
setwd("/Users/JosuC/Desktop")  # write yours
install.packages("ggplot2", dependencies=TRUE)
install.packages("metaheuR_0.3.tar.gz", repos = NULL, type="source")
library(metaheuR)
```

Once the package is installed and loaded, you can go to RStudio "Packages" and click on the name of the package to see the help pages of all the functions defined in it.

For more extensive documentation, the book "*Bilaketa Heuristikoak: Teoria eta Adibideak R lengoaian*" published by the UPV/EHU is suggested. It is written in Basque and freely accesible in:

https://addi.ehu.es/handle/10810/25757

2. The Travelling Salesman Problem (TSP)

Let us illustrate this introductory session to `metaheuR` with the Travelling Salesman Problem. The TSP problem can be stated as follows: there are n cities and the cost of travelling between any pair of cities is known; values $d_{i,j}$ represent the cost of travelling from city i to city j. It is assumed that all the transitions between cities are possible. So, given matrix of distances $D = [d_{i,j}]_{n \times n}$, the goal is to find the minimum cost tour that passes through every single city once and only once. An interesting reference to the "Travelling salesman problem" can be found in Wikipedia (english version).

Formulating the problem in RStudio

First of all, we need to define the problem. We can do it in two different ways: introduce the cost matrix that contains the distances $D = [d_{i,j}]_{n \times n}$ directly in RStudio or read it from a file. In both cases, we'll have to create the appropriate R object: a matrix. If the problem is small, we can introduce the cost matrix $D = [d_{i,j}]_{n \times n}$ directly in RStudio. Remember how to use the function `matrix` to create a matrix from the given set of values. You can give names to the cities using the functions `colnames` and `rownames`.

```
# Introduce the data and create the matrix object with it.
# WRITE HERE
```

To read the cost matrix from a file, we'll use the particular instance of a TSP problem downloaded from the eGela platform: *ins20a.tsp*. It is a text file, you can open it using any text editor. You'll see that there are 20 lines that contain 20 integer numbers: the distances between the 20 cities of this particular instance of TSP problem.

Some very useful functions to read it from RStudio are the following: `scan` to read data into a vector or list from a file, `matrix` to create a matrix from the given set of values. Take into account that you need to compute the number of rows (columns) for the matrix. Think about the necessary functions for that.

```
# Read the data from ins20a.tsp file and create the matrix object with it.
# WRITE HERE
```

3. Definition of the elements for the Heuristic Optimization

In order to formulate the optimization problem and solve it using heuristic methods, we need to select a codification to represent the solutions, and define the necessary elements to do the searching process and find an aproximate solution.

- Codification. The standard codification used to represent the solutions for the TSP problem is the permutation one. In fact, a solution to the problem is a tour that passes through all the cities, and the most natural and intuitive representation of a tour is a path representation, where the n cities that should be visited are put in order. For example, for a TSP problem with 8 cities, $(3, 2, 4, 1, 7, 5, 8, 6)$ represents the tour that starts at city 3, and goes to 2, then to 4 and so on, to end back at the initial city 3.
- Viability of solutions. Using permutations to represent the solutions makes it very easy to control the viability of them. In fact, the tour has to visit every single city (every city has to be present in each solution) once and only once (cities cannot appear repeated in the representation). Permutations make implicitly the two conditions to be satisfied.
- Operators. Having a solution, we need to operate on it to generate new solutions for the searching process. "Swap" operators are typically used to compute new solutions. A "swap" operation interchanges selected positions in a permutation. For example, having the permutation $(3, 2, 4, 1, 7, 5, 8, 6)$, a "swap" move between positions 3 and 5 will give as a result the new solution $(3, 2, 7, 1, 4, 5, 8, 6)$.
- Objective function. It represents the goal of the problem; in this case, to find the minimum cost tour. So, given a solution σ, we need to evaluate it; compute the cost associated to the tour it represents. Let us denote by $\sigma(i)$ the position i in permutation σ. For example, for $\sigma = (3, 2, 4, 1, 7, 5, 8, 6)$, we need to compute the sum of the distances of the tour: $d_{3,2} + d_{2,4} + \cdots + d_{6,3}$. In general terms, the objective function can be formulated like this:

$$\min f(\sigma) = \min \sum_{i=1}^{n-1} d_{\sigma(i),\sigma(i+1)} + d_{\sigma(n),\sigma(1)}$$

Questions:

- Does the "swap" operator guarantee integrity? And, connectivity?
- How many solutions are there in the search space for a TSP problem with 20 cities?
- Are neighboring solutions computed by the swap operator similar?
- How many neighboring solutions has a solution under the "swap" operator?

4. Solving the problem with `metaheuR`

Once the cost matrix for the TSP problem has been defined, the function `tspProblem` implemented in `metaheuR` can be used. It returns a list with three elements: a function to evaluate solutions, the number of cities in the problem and a function to plot a solution for the problem. Have a look at the `tspProblem` function in the help pages in RStudio.

In the following, you are asked to use the function `tspProblem` to create the object associated to the cost matrix defined at the beginning of this lab session. After that, generate two solutions (objects of class permutation) at random and compute the corresponding objective value. Consider also the solution $(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20)$. Some very useful functions you may use in RStudio are the following: `randomPermutation` and `permutation`.

```
# Create the TSP object
# WRITE HERE



# Generate two solutions at random
# WRITE HERE




# Generate the permutation (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)
# WRITE HERE




# Compute the objective value for the three solutions
# WRITE HERE
```

Questions:

- Which are the two solutions computed at random? What is the cost of the three solutions considered? Which one is the best?
- Is it possible to solve the TSP problem using a deterministic method like the simplex algorithm or the branch and bound algorithm? Have a look at the article "A Survey on Travelling Salesman Problem" (2010) at https://www.semanticscholar.org/ or "An Application of the Hungarian Algorithm to Solve Traveling Salesman Problem" at https://www.scirp.org/ or any other source to find the answer.

5. Constructive (Greedy) algorithms

Deterministic algorithms are not feasible for solving the TSP since they take exponential time to return the optimal solution. As a result, algorithms such as the heuristic and metaheuristic methods can be used to approximate it in a reasonable time span. As seen in the first lecture, the most standard heuristic algorithms are those called "constructive heuristics" which, using the intuition, build the solution step by step, normally choosing the best option ("closest city?") at each step. Have a look at the "Heuristic and approximation

algorithms" section in the "Travelling salesman problem" article in Wikipedia. There is a subsection that talks about "Constructive heuristics".

For the TSP problem, let us assume that we are given the city where to start the tour. We will build the tour step by step, selecting always the nearest city to the current one. In `metaheuR` there is a function that implements such constructive algorithm for the TSP problem. Try to guess which one it is; it is not so difficult, its name starts with `tsp`, use the help pages in RStudio.

In the following, you are asked to use the cited function to solve the TSP problem at hand.

```
# Solve the TSP problem using a greedy algorithm
# Show the approximate solution obtained and compute its cost
# WRITE HERE
```

Questions:

- What is the approximate solution computed by the constructive algorithm? Which is the initial city for the tour?
- Execute the constructive algorithm again. Do you obtain the same approximate solution? Why? Have a look at the code of the function to try to understand it.
- Is the approximate solution computed by the constructive algorithm better than the ones considered in the previous section? Can we say it is optimal?

6. Local search

Local search is a heuristic method that is based on the intuition about the searching process: given a solution, the idea is to find better solutions in its neighborhood. At each iteration, the algorithm keeps a current solution and substitutes it with another one in the neighborhood.

The concept of "neighborhood" is very important for the local search: it is a subset of solutions formed by neighboring solutions to the current one. In theoretical terms, to be neighbors means to be similar, according to a predefined concept of similarity or distance. In practice, neighbors are computed by the operators defined for the problem, which are supposed to compute similar solutions. However, it depends on the codification and operator defined. The "swap" operator (to interchange selected positions in a permutation) is typically used to compute the neighbors for the TSP problem.

In `metaheuR` there is a function that implements such a local search. It is called `basicLocalSearch`. Have a look at the help pages to see how to use it. It requires quite a lot of parameters. There is also an interesting function related to the "swap" operator: `swapNeighborhood`. Have a look at the help pages.

In the following, you are asked to use the cited function `basicLocalSearch` to solve the TSP problem at hand. Suggestions:

1. Use the function 'randomPermutation' to select the initial solution to start the local search.
2. Use the function 'greedySelector' to select a solution from the neighborhood. The parameter "selector" in function 'basicLocalSearch' controls it.
3. Select the option "discard" for the parameter "non.valid", not to consider non valid solutions.
4. There is a function called 'cResource' that can be used to define the resource limit we want to define to finish the searching process: a limit in time, in the number of evaluations performed or in the number of iterations to be carried out. Let's say, for example, that we want to limit the search to 10000 evaluations. The parameter "resources" in function 'basicLocalSearch' controls it.
5. The parameter "neighborhood" in function 'basicLocalSearch' controls the type of neighborhood to be used. Let's say, for example, that we want to consider the one generated by function 'swapNeighborhood'.

You can either create a list with all the parameters and use the function `do.call` to call `basicLocalSearch` with the list of parameters, or simply call it. The function returns an object with all the information about the search. You can use the functions `getSolution`, `getEvaluation`, `getConsumedEvaluations` and `getResources` to extract the solution obtained, show the objective value, etc.

```
# WRITE HERE









# Extract the approximated solution found and the objective value
# WRITE HERE
```

Questions:

- What are "non valid" solutions for the TSP problem?
- Is the approximate solution computed using the local search algorithm better than the ones considered in the previous sections?
- Can we still improve it and go on with the search? Or, is it a global optimum?
- You can try to change the initial solution and do the local search again. Use one of the solutions obtained in the previous sections, for example. Do you arrive at the same approximated solution? Why?
- If you reached to the this final question... try now different neighborhood operators for the local search algorithm, and compare the obtained results with the previous local search algorithm implemented.

7. Final suggestion: "Travelling Salesman (2012 film)"

Did you know that there is a film about the TSP problem and its complexity? "Travelling Salesman (2012)". It is an intellectual thriller about four mathematicians hired by the U.S. government to solve the most elusive problem in computer science history: P vs. NP.

http://www.travellingsalesmanmovie.com/

https://www.wired.co.uk/article/travelling-salesman

Enjoy it!