Practical exercices : Metaheuristics

The following exercises can be programmed in your favourite language or in the language that seems most appropriate to the subject. The teacher's assistance will therefore privilege questions related to algorithms and data structures.

Exercice 1 ("Unconstrained Binary Quadratic Problem"). An UBQP (Unconstrained Binary Quadratic Problem) aims to **minimize** the result of a function f that can be written in the form :

$$f(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} Q_{ij}.X[i]X[j]$$

where Q is a $n \times n$ matrix of constants and X is a tuple of binary variables. One can for example consider the following matrix :

$$Q = \begin{pmatrix} -17 & 10 & 10 & 10 & 0 & 20 \\ 10 & -18 & 10 & 10 & 10 & 20 \\ 10 & 10 & -29 & 10 & 20 & 20 \\ 10 & 10 & 10 & -19 & 10 & 10 \\ 0 & 10 & 20 & 10 & -17 & 10 \\ 20 & 20 & 20 & 10 & 10 & -28 \end{pmatrix}$$

For the solution $X = \boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{0}$, $f(X) = 6$.

UBQPs are recognised for their ability to represent a wide variety of important problems, ranging from financial analysis to combinatorial problems on graphs to combinatorial problems on graphs, through task scheduling, frequency task scheduling, frequency allocation etc. [?].

We wish to find a solution X which minimises the function $f(X)$ for a matrix Q provided in a file. For this example the file `partition6.txt` contains first the dimension n (6) of the matrix then a number p (2) (we will explain what it represents later) then the 6×6 elements separated by blanks. The file `graph12345.txt` is another instance.

To test your programs you can create an array Q containing the values given in the matrix above.

Question 1.1. Write a function that returns an initial solution, that is, a random vector of n bits for this problem (the size of the vector will be read from the input file, or is simply the number of lines of the matrix).

Question 1.2. Write a function that computes the value of this solution by the function f that corresonds to the Qmatrix.

Question 1.3. Write a function `best_neighbour` that returns the best neighbour of X, where a neighbour X' of X is a sequence of bits that differs from X by only one bit. **Optimization :** If there are several best neighbours your function could choose randomly among them.

The algorithm `Steepest Hill-Climbing` is recalled here :

- `start from some initial solution` s
- nb_depl $\leftarrow 0$; $STOP \leftarrow false$

`Repeat`
- $s' \leftarrow$ best_neighbour(s)
- `if` better($f(s'), f(s)$) `then` $s \leftarrow s'$
  `else` $STOP \leftarrow$ true `/* local optimum*/`
- nb_depl++

`Until` nb_depl = `MAX_depl or` $STOP$
`Return`(s)

Question 1.4. Program the`Steepest Hill-Climbing` algorithm. Here, "better" means "strictly better".

Question 1.5. Program a variant with restarts, i.e. make an external loop around the Steepest Hill-Climbing method allowing to start from a new solution drawn at random, then to carry out a trial (that is, make at most MAX_moves moves towards better neighbours ; a trial can stop before having made the MAX_moves moves towards better neighbours), then to restart with a new solution at random. This outer loop will have to be done MAX_trials times.

Question 1.6. Run your programs on the two files `partition6.txt` and `graph12345.txt`.

We now recall the Tabu search method :

Algorithme : Tabu search

1. $s \leftarrow$ new initial solution ; Tabu $\leftarrow []$; nb_moves $\leftarrow 0$; best_sol_so_far $\leftarrow s$; STOP $\leftarrow$ False ;
2. while nb_moves $<$ MAX_moves and STOP = False do :
   (a) if s has some no non-tabu neighbour : STOP $\leftarrow$ True ;
   (b) else :
      i. Tabu $\leftarrow$ Tabu $+ s$; $s \leftarrow$ best_non_tabu_neighbour(s) ;
      ii. if better(s, best_sol_so_far) : best_sol_so_far $\leftarrow s$;
      iii. nb_moves $++$;
3. return best_sol_so_far.

where the Tabu list is implemented as a FIFO of fixed size k.

Question 1.7. Program the tabu search method, and test it on the proposed instances, trying different sizes for the Tabu list.

Question 1.8. We now impose a constraint on the solutions : we are looking for a binary sequence, the bits of which sum to at least p. Modify your best_neighborfunction so that it takes into account this constraint. For instances read from files, p is the second number given in the file (for `partition6.txt` this number is 2, for `graph12345.txt` it is 4). Test Steepest-Hill-Climbing with restart under this constraint.

Exercice 2 (Traveling salesman)**.** The famous Travelling Salesman Problem (TSP) consists in finding a tour that passes through all cities with the the shortest total distance. Some instances are on Moodle, each in a file in which we have a list of n cities (n is the first number at the beginning of the file), each on a separate line that gives its identifier and coordinates, in the form : `Id, x, y`. The cities are always given in order of increasing identifiers from 1 to n.

A solution of this problem is a sequence of n cities, it represents a tour starting from the point with coordinates $(0, 0)$ then passing through each of the cities of the sequence, then returning to the point $(0, 0)$. For simplicity, the distance between two cities will be their Euclidean distance : $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

We want to find a tour that has minimum length.

For the instance in file `tsp5.txt` , with 5 cities, the best tour visits the cities in the order

5	3	4	1	2

, it has a total length of 263.88 km.

Question 2.1. Write a function that returns a random vector of n cities, where n is the number of cities read as the first number in an instance file. For this function you can either use a constructive method (create a chained list of cities and randomly draw the index of the first city from 1 to n, delete the corresponding city from the list then randomly draw between 1 and $n - 1$ the second city etc) or an iterative method (start from the solution X where the cities are in increasing order from 1 to n and for $i = 1$ to n exchange $X[i]$ and $X[random(n)]$).

Question 2.2. Write a function that calculates the value of this solution, i.e. the distance that the

travelling salesman must travel from (0,0) through all the cities $X[1] \ldots X[n]$ and back to (0,0).

Question 2.3. Program a function `best_neighbour` that returns the best neighbour of X, where a neighbour of X is any sequence obtained by swapping two cities in the sequence X.

Question 2.4. Use the Steepest Hill-Climbing method, with restards, on both the `tsp5.txt` and `tsp101.txt`. For each trial, you should be able to display the initial solution drawn at random, the solution reached and the number of moves made from the initial solution to the solution reached.

Question 2.5. Use the tabu method to solve these instances, trying different sizes for the Tabu list and for the MAX_moves. Here again, you should be able to display the initial solution, the solution reached, the number of moves made to the solution reached, the best solution found, and the contents of the tabu list at the end of the search.