UNIVERSITÉ TOULOUSE III PAUL SABATIER

Advanced Algorithmic (Metaheuristics Labwork Report)

Author(s): Florian Legendre

Contents

1 Algorithms **2**
1.1 Steepest Hill with restarts 2
1.2 Tabu 3

2 Neighbours and Neighbourdhoods **4**

3 Instances and results **5**
3.1 tsp5.txt instance 5
3.1.1 Steepest Hill algorithm with restarts 5
3.1.2 Tabu algorithm 6
3.2 tsp101.txt instance 6

Algorithms

For this labwork I have implemented 2 metaheuristics algorithms. The first one being the steepest hill algorithm with restarts and the second one being the tabu algorithm.

1.1 Steepest Hill with restarts

Below is the steepest hill algorithm without restarts. We won't detail here the auxiliary functions such as the *getDistance* method from the TSP class or the *debugSteepestRun* which only allows a few debugging prints. Furthermore some methods will be detailed in the section 2.

```
1  public static int[] run(City[] cities, int[] citySequence, int maxMoves){
2      int[] tmpCitySeq, res = citySequence;
3      int nbMoves = 0; boolean stop = false;
4
5      //Not essential to the algorithm but checks coherence of the algorithm:
6      System.out.println("Random initial city sequence is: ");
7      System.out.print("[ ");
8      for(int cityID : citySequence) System.out.print(cityID + " ");
9      System.out.println("]");
10
11     //Essential to the algorithm:
12     while(!stop && nbMoves < maxMoves){
13         tmpCitySeq = bestNeighb(cities, res);
14
15         if(TSP.getDistance(cities, tmpCitySeq) < TSP.getDistance(cities,
       res))
16         res = tmpCitySeq;
17         else
18         stop = true;
19
20         nbMoves++;
21     }
22
23     //Not essential to the algorithm but checks coherence of the algorithm:
24     debugSteepestRun(nbMoves, res);
25     return res;
26 }
27
```

Listing 1: Source code of the steepest hill algorithm without restarts

The steepest hill algorithm consists in always choosing the best neighbor and always going towards a better solution. If a better solution can't be found then the algorithm stops. The number of moves acts as a fail-safe for this algorithm (eg. if it's always possible to go up). This is actually the whole meaning of the loop quoted above from line 12 to 21. The restart version just adds another while loop over this algorithm and compares the best solutions found at each iteration between themselves to select only the best solution over the best solutions. This is a strategy to get out of local optimums.

1.2 Tabu

```
1  public static int[] run(City[] cities, int[] citySequence, int maxMoves,
       int tabuSize){
2      LinkedList<int []> tabu = new LinkedList<>();
3      int[] tmpBestCitySeq = null, bestNonTabuNeighb, currentCitySeq =
       citySequence;
4      int nbMoves = 0; boolean stop = false;
5
6      while(nbMoves < maxMoves && !stop){
7        bestNonTabuNeighb = bestNonTabuNeighb(tabu, cities, currentCitySeq);
8          if(bestNonTabuNeighb == null)
9          stop = true;
10         else{
11             addTabu(currentCitySeq, tabu, tabuSize);
12             currentCitySeq = bestNonTabuNeighb;
13
14             if(isBetterNeighb(cities, currentCitySeq, tmpBestCitySeq))
15             tmpBestCitySeq = currentCitySeq;
16
17             nbMoves++;
18         }
19     }
20
21     //Not essential to the algorithm but checks coherence of the algorithm:
22     debugTabu(nbMoves, currentCitySeq, tabu);
23     return tmpBestCitySeq;
24 }
```

Listing 2: Source code of the tabu algorithm

The steepest hill algorithm uses the random restarts strategy to get out of local optimums. The tabu algorithm on the other hand uses another philosophy: it allows itself to not always pick up a better solution. But it does always pick up the best possible neighbor around (or the least worse). In order to not backtrack and get caught in an infinite loop of backtracking back and forth to the local optimum it uses a tabu list in which already explored solutions are stored so that they are not explored again (as long as the tabu list, implemented as a FIFO, doesn't overflow in which case we drop the most ancient step). The longest the tabu list, the more freedom we give to our tabu execution to roam around the local optimum hoping to find a better local optimum. Again, as with the steepest hill, the while loop reflects this metaheuristics concept (from line 6 to line 17).

NB: If no non tabu neighbour is found then in order to avoid a *null* value as the *currentCitySeq* (cf. line 12) we stop the algorithm and return the best solution found so far. This is the meaning of lines 8 to 9 above.

Neighbours and Neighbourdhoods

```
1  public static LinkedList<int[]> genNeighbourhood(int[] citySeq){
2      int nbCities = citySeq.length;
3      LinkedList<int[]> res = new LinkedList<>();
4
5      for(int i = 0; i < nbCities; i++) {
6          for (int j = i + 1; j < nbCities; j++) {
7              int[] tmp = new int[nbCities];
8              swap(citySeq, i, j);
9              System.arraycopy(citySeq, 0, tmp, 0, nbCities);
10             res.add(tmp);
11             swap(citySeq, i, j);
12         }
13     }
14     return res;
15 }
16
```

Listing 3: Source code of the neighbourhood generation in the Tabu class

While this piece of code comes from the TSP class it is actually very similar to the *bestNeighb* static method found in the SteepestHill class. This stems from the fact that the said method kills two birds in one stone by generating the whole neighborhood of a give solution while selecting the best neighbor.

What the above function shows is that in the context of the TSP problem, a neighborhood consists in the swapping of two cities from one initial sequence of cities. We can generate all the possible neighbors by generating all the possible swaps with one value then by generating all the possible swaps of the other values without including the swaps with the value already computed. Here we do this in ascending order: we generate all the possible swaps from the first city then we do this with the second city excluding the first city, etc.

Instances and results

3.1 tsp5.txt instance

3.1.1 Steepest Hill algorithm with restarts

```
chuxclub @ CRex2 ~/projects/wip/2021_m1csa_algo_metaheuristics (dev)
$ java -jar tsp.jar res/tsp5.txt steepest

Input maxMoves: 10
Input maxTrials: 3

########## TESTING STEEPEST HILL ALGORITHM (with restarts) ##########
>>> INITIAL SEQUENCE OF CITIES:
=============================================================
Vector #DataStructures.Vector@3567135c has length: 5
Vector #DataStructures.Vector@3567135c contains:
3     4     5     2     1
=============================================================
Distance: 202.72643264132455 km

>>> STEEPEST HILL ALGORITHM EXECUTION (with restarts):
Trial #1 :
Random initial city sequence is:
[ 1 2 3 5 4 ]
2 moves required to reach the following solution:
[ 1 2 4 5 3 ]

Trial #2 :
Random initial city sequence is:
[ 4 2 3 5 1 ]
3 moves required to reach the following solution:
[ 1 2 4 5 3 ]

Trial #3 :
Random initial city sequence is:
[ 2 3 1 5 4 ]
3 moves required to reach the following solution:
[ 1 3 2 4 5 ]

>>> BEST SEQUENCE OF CITIES :
=============================================================
Vector #DataStructures.Vector@54a097cc has length: 5
Vector #DataStructures.Vector@54a097cc contains:
1     2     4     5     3
=============================================================
Distance: 194.04052963659356 km
#############################################################################################
```

Listing 4: Bash output of the execution of the steepest hill algorithm on the tsp5.txt instance

3.1.2 Tabu algorithm

```
chuxclub @ CRex2 ~/projects/wip/2021_m1csa_algo_metaheuristics (dev)
$ java -jar tsp.jar res/tsp5.txt tabu

Input maxMoves: 10
Input tabuSize: 5

################################## TESTING TABU ALGORITHM
    ################################
>>> INITIAL SEQUENCE OF CITIES:
=================================================
Vector #DataStructures.Vector@3567135c has length: 5
Vector #DataStructures.Vector@3567135c contains:
4      1      5      2      3
=================================================
Distance: 258.8852276594047 km

>>> TABU ALGORITHM EXECUTION:
Number of moves used to reach last solution: 10
Last solution reached: [ 4 2 5 3 1 ]
City sequences in tabu list:
[ 3 5 4 2 1 ]
[ 3 5 2 4 1 ]
[ 3 4 2 5 1 ]
[ 5 4 2 3 1 ]
[ 2 4 5 3 1 ]

>>> BEST SEQUENCE OF CITIES:
=================================================
Vector #DataStructures.Vector@3551a94 has length: 5
Vector #DataStructures.Vector@3551a94 contains:
3      5      4      2      1
=================================================
Distance: 194.04052963659356 km
#################################################################################################################
```

Listing 5: Bash output of the execution of the tabu algorithm on the tsp5.txt instance

On the tsp5.txt we have two possible best solutions as shown above: 3 5 4 2 1 and 1 2 4 5 3 with a distance of 194.04052963659356 km. We notice that one solution is the inverted version of the other solution which trivially means that the same distance is traveled whether we do the traveling salesman circuit in one way or the other.

3.2 tsp101.txt instance

Given the amount of cities (101 of them) I couldn't find no best local optimum despite the algorithm used and the amount of moves/trials/tabu size involved. But I managed once to go down to around 994km with the tabu algorithm with 1000 moves and tabu list size of 500. Recently, the steepest hill algorithm provided a solution with 1023.92 km of traveled distance with 400 moves and 100 trials. So we can conclude that both of these algorithms find more or less the same solutions. One algorithm take more memory space (the tabu algorithm) while the other seems to take more computing time (the steepest hill algorithm) meaning that the choice of one or the other algorithm depends on the application they are used for.