Assignment: ColoTe

Optimization methods and algorithms – A.Y. 2016-2017

Massimo Tumolo (236037)

Enrico Balsamo (222340)

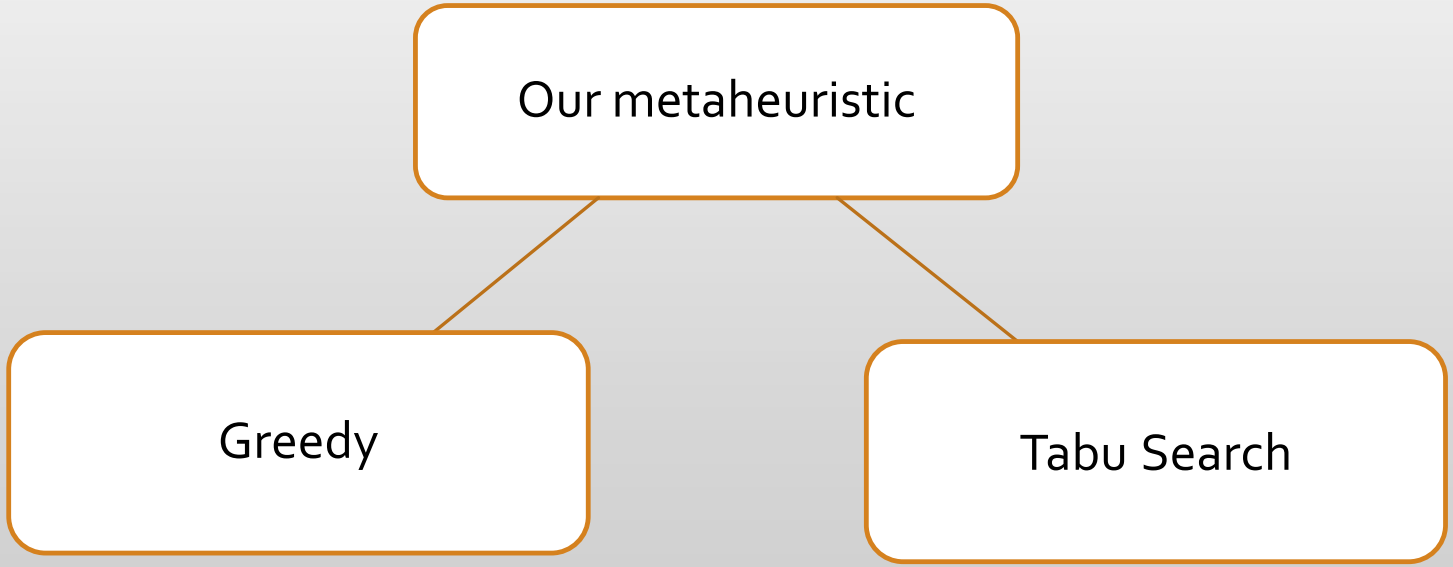
Angelo Filannino (242140)

Davide Gallitelli (251421)

Daniele Montisci (230073)

The Metaheuristic I

- Our metaheuristic is a combination of a Greedy heuristic and of the Tabu Search metaheuristic.



The Metaheuristic II

Greedy

- We iterate on the set of operational/destination cells, satisfying their requests of tasks one cell at a time.
- In order to assign a user to a cell, we select one among those who offer the lowest cost per task ratio.

Tabu Search

- A Solution of the problem consists in an array of indexes whose size is equal to the number of cells (called "indexes mask") that defines the order in which we satisfy the request of tasks of each cell.
- A Move consists in the swapping of two of these indexes of the array.
- Therefore, a Neighbourhood is the set of solutions in which only two indexes have been swapped with respect to the current solution.

The Metaheuristic: Greedy

- Our greedy algorithm iterates on the set of operational/destination cells. Given a sorted array of those cells' indexes, the algorithm flows through it, satisfying all the requests in a particular order.
- To solve a request, the algorithm follows a best-first logic: it chooses the user that currently offers the best cost per task ratio. If more then one share the same ratio, the one who can perform less tasks will be chosen first (in order not to waste tasks).
- All of the user tasks' will be assigned to the requesting cell and thus the user will become unavailable to satisfy other requests.
- It is glaring that what makes the algorithm offer different results is how the cells requesting tasks are sorted.

The Metaheuristic: Tabu Search

- The greedy algorithm being the internal algorithm used in the MetaHeuristic, we need the external Tabu Search layer to provide it a solution to evaluate.
  - Our solution is an array of indexes, called ***indexes mask***, sized as the number of cells, used as a mask to apply to the iterating index to access to destination/operational cells.
  - Thus the cells will not be accessed in a static order, but in a pseudo-random fashion, characterizing each solution.
- A move consists in the scrambling of the ***indexes mask*** by swapping two of them.
- Therefore, a Neighbourhood is the set of solutions in which only two indexes have been swapped starting from the ***indexes mask*** of the current solution.

Metaheuristic tuning I

- For each operational/destination cell we have to look multiple times (depending on its number of required tasks) for the non-assigned user that can offer the lowest cost per task ratio. Performing a minimum search on the set of all available users (a three-dimensional matrix), is very time consuming.
- Therefore, before starting the algorithm, we pre-compute for each cell an array of all ratios and we sort it incrementally. By doing so, the minimum search consists in selecting the first non-assigned user from this array, with a much faster execution.

Metaheuristic tuning II

- In order to evaluate a move, we must unassign all users from the two involved cells and redo the assignment in the opposite order. To do the unassignments we should iterate on the set of all assigned users (a four-dimensional matrix) to put them back in the right position in the set of all available users, which is time consuming (quadratic on the number of cells).
- To avoid this, for each solution we keep an array of "modifications" that led to it, which are the movements from the set of all available users to the one of all assigned users. Reverting these modifications proved to be much faster.

Metaheuristic tuning III

- For each logical core of the CPU we run an instance of the Tabu Search, and each instance starts with an indexes mask randomly initialized. In this way we implement a random multi-start, which proved to be effective since in many cases each instance comes up with a different solution, among which we choose the best.
- Since each user can perform a different number of tasks and according to their type and they are payed for the tasks they can do, it is possible to have an over-assignment of tasks to certain cells which results in an increased total cost. To avoid this as much as possible, in case we can choose among users that offer the same cost per task ratio we select the one that allows us to reduce as much as possible the waste.

Results I

- The metaheuristic is capable of solving all base instances, with an acceptable optimality gap for each of them.
- The metaheuristic performs better on the instances with 20 time steps rather than on the ones with only 1: we only get optimal solutions with the first ones, as well as lower gaps on average.
- With all the instances it is possible to remain below the 5 seconds time constraint, because the high number of iterations (in the order of thousands) allows to easily control this stopping condition (no iteration needs to be interrupted).

Results II

Some statistics (may slightly change for different executions):

- Optimal solutions: **26.7%**
- Solutions with a gap < 1%: **95%**
- Maximum gap: **1.35%**
- Average gap: **0.25%**

Considerations

- The metaheuristic is able to solve also the "TT" and "TL" hard instances, but for 3 of them the optimality gap is higher than 2% (on average 1,50%).
- It is not able to solve the "ST" hard instances, because of the over-assignment of tasks: the difference between the available tasks and the request of tasks of all cells is inferior to the lowest number of tasks that a user can perform, which results in some of his tasks to be "wasted" (paid but not performed because not necessary), causing the whole problem to have no solution.
- Applying a genetic algorithm instead of the Tabu Search (or together with it) provided no real benefits.