group03

# ODBDP REPORT

## HIGH LEVEL VIEW

The main algorithm is made up by a hybrid meta-heuristic (*Simulated Annealing with a TabuList*) that uses as initial solution the one produced by some *greedy heuristics*.

In order to exploit, on average, the best results, it has been created a pool of 4 threads that will run the same meta-heuristic with different initial solutions:

- Two threads will use *the best greedy heuristic* (the one that produces the best initial solution).
- One with the *second best of the greedy*.
- One with the *initial 0-solution*.

On average, the best greedy produces best results, but the same algorithm with the same inputs varies due to random choices; that's why there are two identical threads with identical inputs. According to numerous tests, depending on the instance of the problem, it could be that the best result is exploited using the second best of the greedy or the initial 0-solution. In order to increase the output of the algorithm and to generalize it as much as possible, two additional threads are used: one with the second best greedy and one with the 0-solution as initial ones.

The stopping criterion is *time-driven*. When the timer ends the available seconds, every thread is stopped and every local best of the 4 threads is compared due to save the best of the bests on the non-volatile memory.
Furthermore, to avoid losses in case of crashes or other problems, one of the two threads that uses the best greedy heuristic has an additional task that consists of intermediate ".sol" file saving, overwriting only if the local best is updated with a better local best. Only one thread can save the file because it is a CPU+I/O task and can be heavy. It has been adopted that solution to give the more Core/CPU time at the algorithm per se.

## GREEDY HEURISTICS

Sometimes, a good initial solution is the key to exploit better and better final ones. With the purpose of producing a good feasible initial solution, they have been created two greedy heuristics:

- **GAIN-COST** per CONFIGURATION
- **MEMORY/GAIN** per CONFIGURATION

The first greedy approach, for each configuration, calculates its total **GAIN-COST**, (where **GAIN** is the sum of all queries gain greater than the one already selected for that query and **COST** is the sum of all indexes cost that are not already selected).

Finally, the algorithm sorts the configurations in descending order and it will use the first configuration that is not selected as final output.
The second greedy approach, for each configuration, calculates its total **MEMORY/GAIN** (e.g. Memory required for each unit of gain; where **MEMORY** is the memory of all indexes memory that are not already selected and **GAIN** is the sum of all queries gain greater than the one already selected for that query).
Then the algorithm sorts the configurations in ascending order and it will use the first configuration that is not selected as final output.

After a bunch of tests, it has been proved that most of the time the MEMORY/GAIN greedy is better than the GAIN-COST one. The second one has an average gap of 43.63% with a 16.76% best result. On the other hand, the first one has an average gap of 16.76% with a 4.77% best result. Moreover, it is necessary to underline the fact that those greedy heuristics need 2/3 seconds of CPU time to produce their output, so they are *very fast*.

## SIMULATED ANNEALING WITH TABU LIST

It's the meta-heuristic core of the entire algorithm. The implemented Simulated Annealing has some differences with respect to the original one, but in general we can summarize as follows:

- The initial **temperature** is *very high*, chosen by multiple running on different PCs with faster/slower CPU.
- The cooling parameter **α** depends on the time passed by the user through the creation of 5 possible intervals with the corresponding parameter. This choice is because, depending on the *time*, the temperature must decrease and ideally become very low, on average, on all PCs with any time (to give the possibility to accept best solutions too) and the only way is to increase or decrease alpha accordingly.
- The probability **p** that, based on the difference between *new* and *old gain* and on the *temperature*.
- The **TabuList**, a hybridization for the Simulated Annealing that uses the concept of tabu moves from the *Tabu Search* algorithm.

At each iteration, the temperature is scaled by the cooling parameter that has a value between $\approx 9 \times 10^{-6}$ and $\approx 9 \times 10^{-8}$. While the given time is not ended, the algorithm choses a *random query* and so a random configuration. Once it has been checked that the query is not in the tabu list (otherwise it will be chosen another one) it is created a configuration *backup* and the array of chosen indexes is updated. After that the elaboration of gain, cost and memory has finished, the algorithm must decide to accept or not the solution. On one hand, if the solution is unfeasible it is obvious that it is dropped, and it will be restored the previous backup. On the other hand, the choice is delegated to the probability **p** that depends on the new gain, the old gain and

the temperature. If that probability is greater than a random one, the solution is accepted even if it is a *worsening* solution in order to escape from the local minimums, otherwise it will be accepted only if it is a better one.

This mechanism, as it is, on the behavioral level, allows at higher temperatures to accept worsening solutions and at lower temperatures to accept only better ones.

Temperature, alpha and the length of the TabuList are chosen before the while cycle. The TabuList length depends on the number of the queries, it varies from 1 to 10 and once it is full, the elements are removed from the older one at runtime.

During the cycle the algorithm uses some variables to take the best found so far that will be given in output at the end of the time.

## FINAL CONSIDERATIONS

Even though the entire program uses multithread and randomization to archive on average best results, it has been proved that the implemented *greedy heuristic* and the *hybrid metaheuristic* reaches quite always the **optimality on most of the instances**. Furthermore, in some cases it overcomes the best solution found so far on the "not optimal" instances. To give some numbers, the average gap is 0.93%, with a -3.12% of best result.

## THE TEAM

Arakelyan Alisa

Anjum Muhammad Ali

Bhandari Hema

Gravile Andrea

Ferrettino Luigi

Sommese Giuseppe

Valente Francesco