ColoTe Project

Group 34

- Moriello Paolo
- Coccia Giuseppe
- Guerriero Mario
- Marallo Graziano

OUTLINE

- The problem
- Introduction to the proposed solution
- Detailed description of the solution
- Results presentation

Variable:

x is the number of customers of type *m* that are asked to do *n* tasks in cell *j*, starting from *i* at time *t*

Parameters:

- *c* is the cost of the reward for a customer of type *m* in cell *i* at time *t* that goes in cell *j*
- *N* is the number of tasks that must be done in the operational cell *i* during the time
- *n* is the number of tasks that a customer of type *m* can do
- Θ is the number of customer of type *m* in cell *i* during time step *t*

Objective Function:

$$\text{minimize} \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{t=1}^{T} \sum_{m=1}^{M} c_{ij}^{tm} x_{ij}^{tm}$$

Constraints:

$$\sum_{t=1}^{T} \sum_{m=1}^{M} \sum_{i=1}^{I} n_m x_{ij}^{tm} \geq N_j \quad \forall\, j \ \in \mathcal{I}$$

$$\sum_{j=1}^{J} x_{ij}^{tm} \leq \theta_i^{tm} \quad \forall\ i \ \in \mathcal{I}\ t\ \in \mathcal{T}\ m\ \in \mathcal{M}$$

$$x_{ij}^{tm} \in \mathbb{N} \quad \forall\ i\ \in \mathcal{I}\ j\ \in \mathcal{J}\ t\ \in \mathcal{T}$$

Many experiments…

- Genetic
- Greedy
- Taboo Search
- Simulated Annealing
- Memetics (Simulated Annealing + Greedy, Genetic + Greedy…)

...no acceptable results

- A main greedy based heuristic supported by three simpler ones
- Exploitation of the knowledge gained after our several experiments
- Distributed computation among a configurable number of parallel threads
- Good results both in terms of quality and time

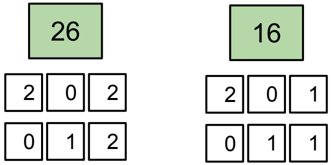
1.

THE MAIN ALGORITHM

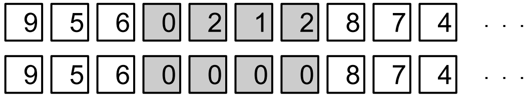
Combined search

STEPS

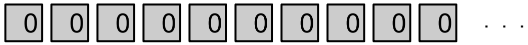
0) Generate **combinations**



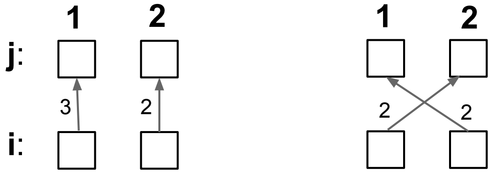
1) Manage **certain moves**

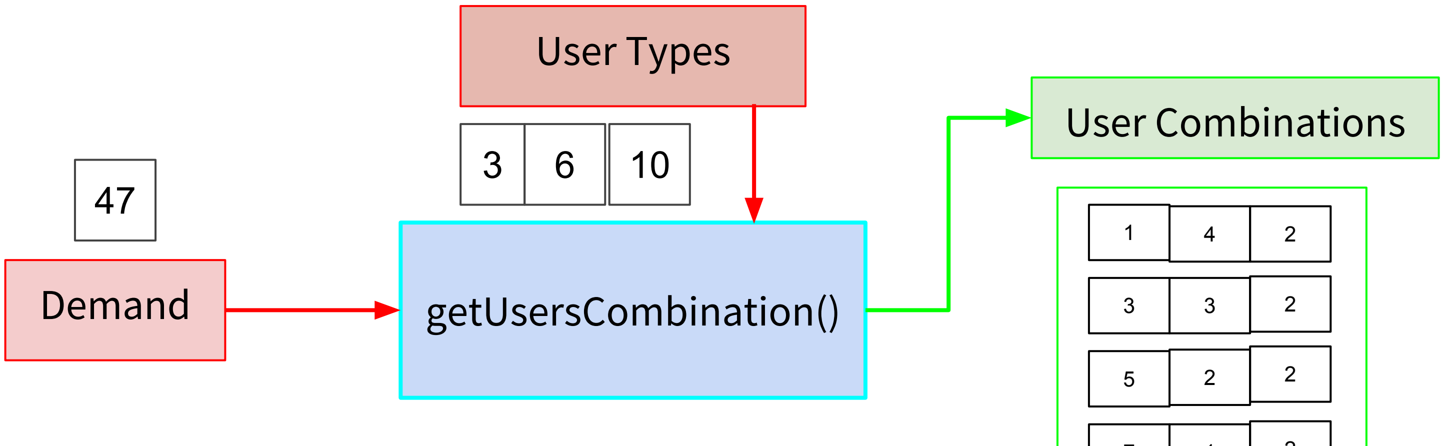


2) **Solve** the problem



3) **Improve** solution



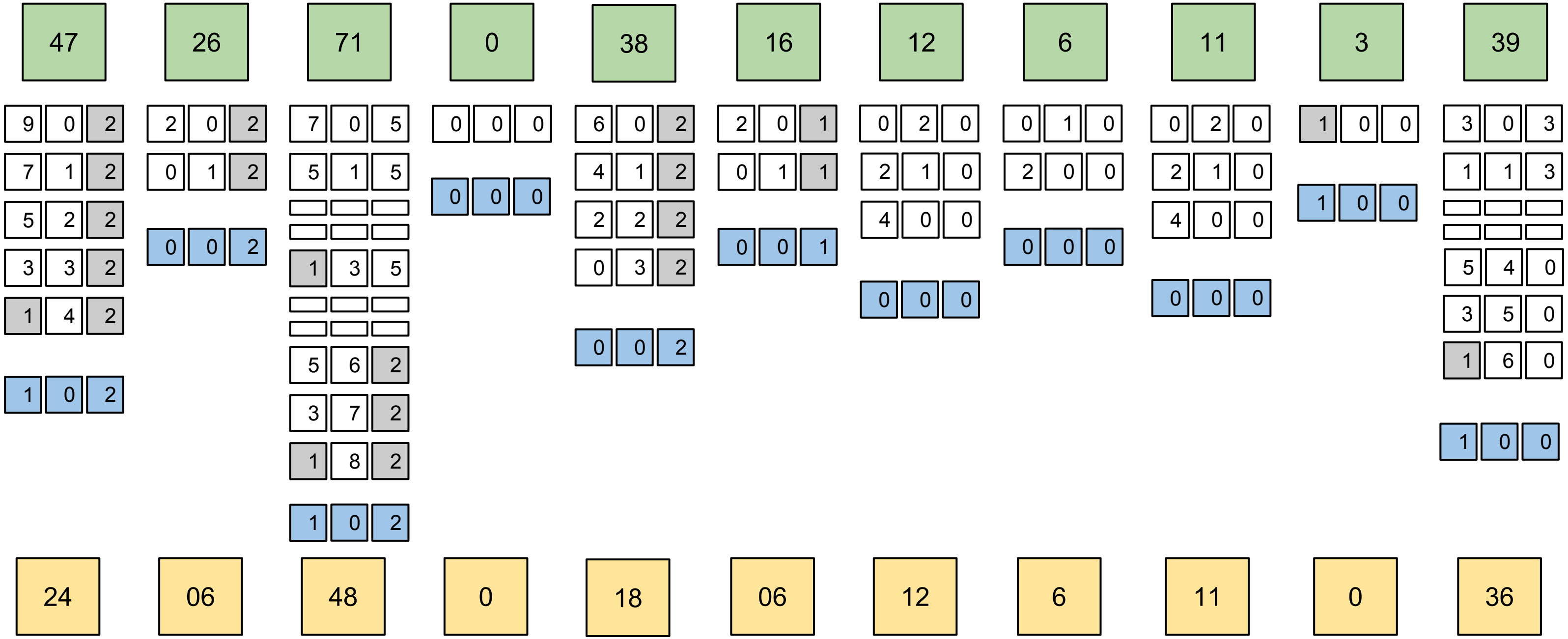


- It generates combinations that satisfy exactly the demand

Loop until there are certain moves to do:

- Search for the lowest cost (combined) move among all the certain moves found and among all the cells - every i, j and t is reassessed every time.
- Apply min-cost move by moving the right amount of users from the earlier found cell(s).

COMBINED SEARCH: CERTAIN MOVES

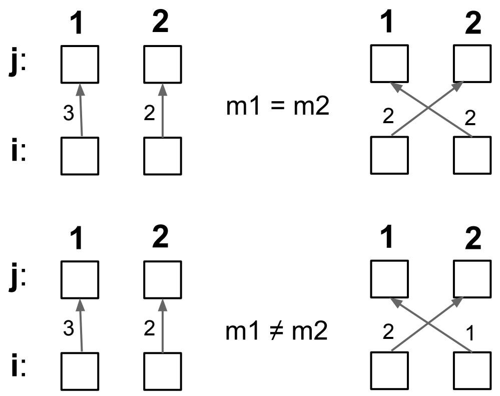


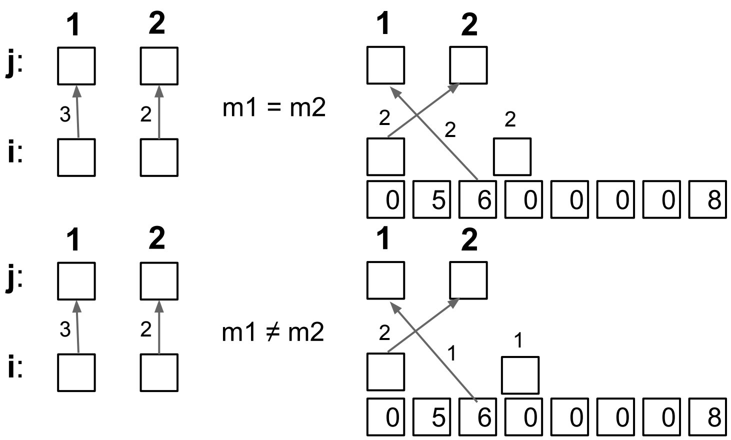
Loop until a feasible solution has not been found:

- Search for the lowest cost (combined) move among all the combinations and all the cells - every i, j and t is reassessed every time.
- Apply min-cost move by moving the right amount of users from the earlier found cell(s).

IMPROVE SOLUTION: FOUR TYPES OF SWAP

- Same type of users
- Different type of users
- Swap between solutions
- Swap between solutions and cells





2.

THE 'SUPPORT' ALGORITHMS

Greedy search

Support Algorithm 1 :

- Search for the lowest unit cost task move among all combinations (user cost/tasks that user could do).
- IF currentCost == minCost THEN choose user that could do more tasks.
- IF choosen user could do more tasks than those required in cell j THEN do again the research for the lowest cost move (IF currentCost == minCost THEN choose user that could do more tasks).

Support Algorithm 2 :

- Search for the lowest unit cost task move (considering also required tasks) among all combinations ((user cost/tasks that user could do) * required tasks in cell j).

Support Algorithm 3 :

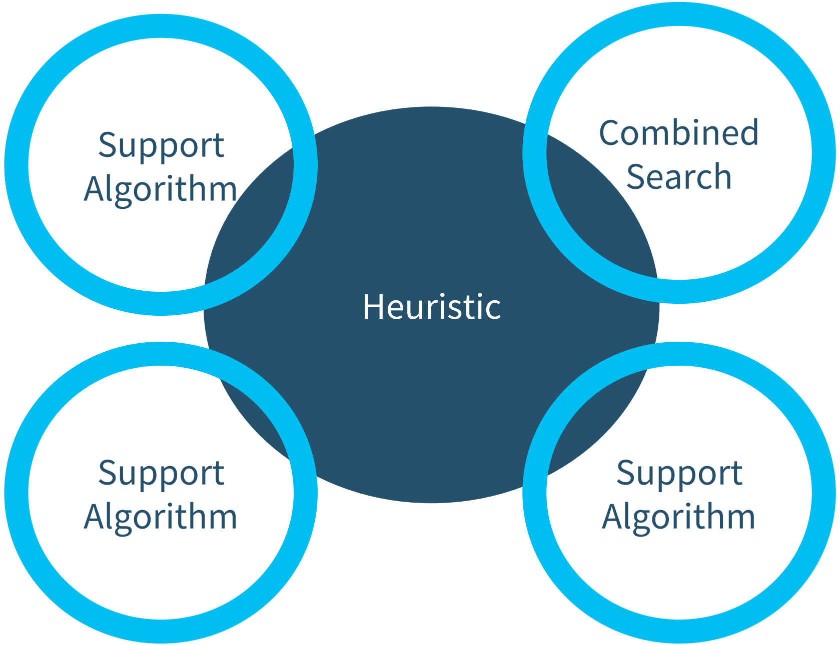
- Search for the lowest unit cost task move among all combinations (user cost/tasks that user could do).

- IF currentCost == minCost THEN choose user that could do more tasks.
- Swap solutions.

3.

THE RESULTS

COMBINED SEARCH - GREEDY ALGORITHMS



Point of strength:

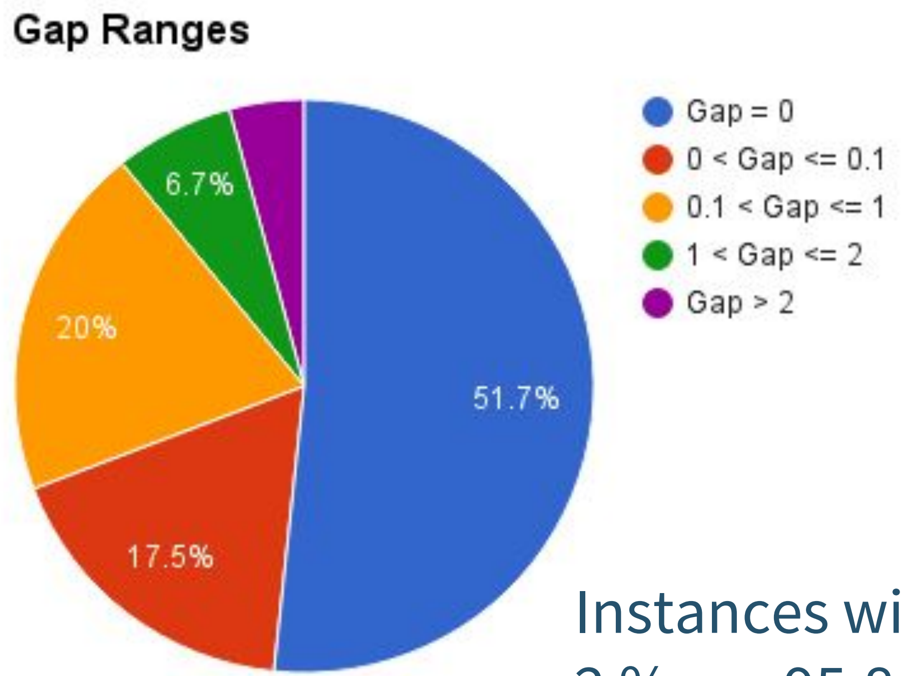
- Good on problem instances
- Capable of solving all instances

Main algorithm solves the problem in a very good way, but is slow in terms of performances.

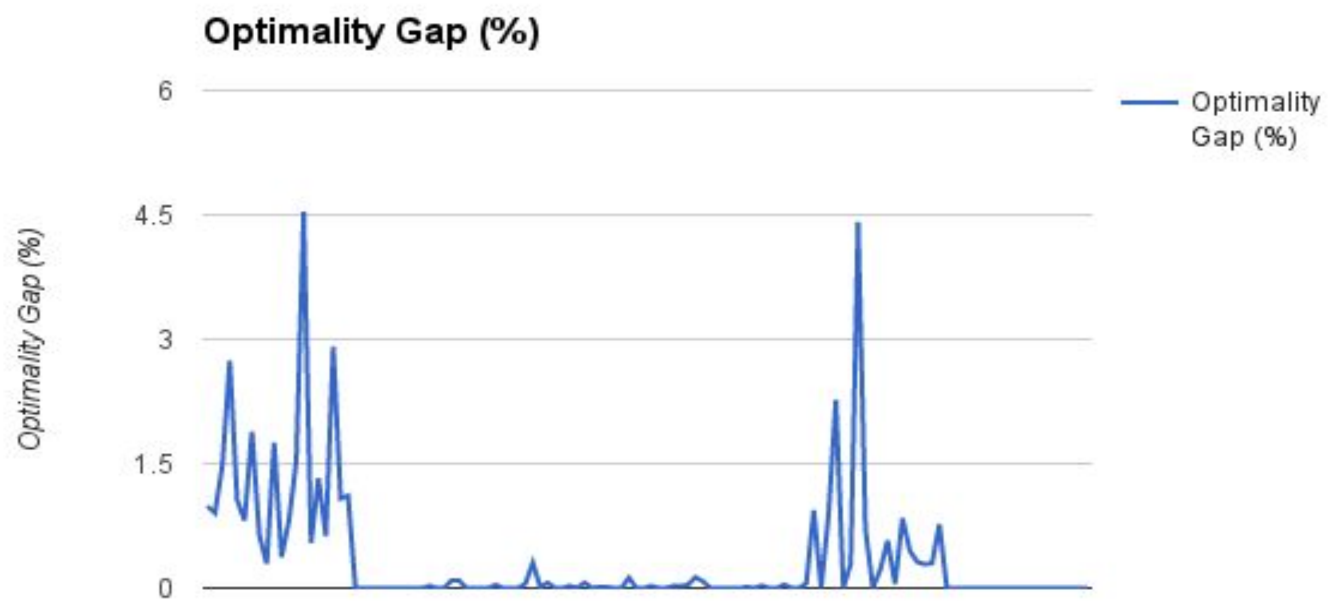
In order to solve biggest instances we combined it with support algorithms (at least one).

The execution has been performed with three different configurations.

- Increasing the number of threads (min 2, max 4) is possibile to improve the performance.
- All the algorithms are executed in parallel and the best result is taken at the end.



Instances with gap below 2 % are 95.8 % of the total


Optimality Gap (%)
6
4.5
3
1.5
0
Optimality Gap (%)
Optimality
Gap (%)

PERFORMANCE ON PROBLEM INSTANCES

	Average Optimality Gap (%)	Objective Function	Optimal value of Objective Function
4 THREADS	0,34	500961	499257
3 THREADS	0,36	501065	499257
2 THREADS	0,37	501144	499257

HARD INSTANCES

The algorithm has shown good results even on hard instances.

- Indeed, the algorithm is capable of solving all of them
- All solutions we have obtained are feasible

The execution of the algorithm has presented this results:

Average Optimality Gap (%)	Objective Function	Optimal value of Objective Function
3,29	116902	107391

Thanks for listening

Group 34

- Moriello Paolo
- Coccia Giuseppe
- Guerriero Mario
- Marallo Graziano