Heuristics for task assignment

Roberto Meroni* Hugo Sanz González†

November 2023

Consider the problem of assigning the workload required to complete a set of task T to a set of computers C. Each task $t \in T$ requires $r_t \geq 0$ units of work to complete, while each computer $c \in C$ has $r_c \geq 0$ compute units available. Define the load of a computer $c \in C$ as the total number of compute units handled divided by its capacity r_c, and let P_2 be the problem of minimizing the load of the highest-loaded computer. We evaluate the performance and efficiency of different search strategies—some of them based on heuristics—on instances of P_2 with different characteristics.

1. *Prepare a pseudocode for the greedy algorithm. Specify the greedy function.*

Without loss of generality, let $C = \{c_1, \ldots, c_m\}$ be the set of computers. The following algorithm produces an assignment of computers to a subset of the given tasks that meets all restrictions on computer capacities while trying to minimize the load of the highest-loaded computer.

1. [Initialize.] Set $S \leftarrow \varnothing$, $i \leftarrow 1$; sort the tasks in T by their required compute units in descending order, and let the resulting sequence be $t_1, \ldots, t_n$. Then set $l_j \leftarrow 0$ for all $1 \leq j \leq m$. (Throughout the algorithm, S is the partial solution built so far, i is an index into the sequence of sorted tasks, and l_j is the number of compute units assigned to computer c_j.)

2. [Start the search.] If $i = n$, go to step 6. Otherwise set $F \leftarrow \varnothing$, $j \leftarrow 1$. (Here $F \subseteq C$ is the set of computers to which task t_i may be assigned while keeping the load of all computers under their respective capacities, and j is an index into the set of computers.)

3. [Obtain feasible assignments.] If $j = m$, jump to step 4. Otherwise if $l_j + r_{t_i} \leq r_{c_j}$, set $F \leftarrow F \cup \{c_j\}$. Next, set $j \leftarrow j + 1$ and repeat this step.

4. [Infeasible?] The set F is empty whenever no more tasks can be assigned; if so, go to step 6.

5. [Apply fittest assignment.] Let $c^\star$ be the computer $c' \in F$ that minimizes $\max_{c \in C}(l_c + \mathbf{1}_{c=c'} r_{t_i})/r_c$. Then set $l_{c^\star} \leftarrow l_{c^\star} + r_{t_i}$, $S \leftarrow S \cup \{(t_i, c^\star)\}$, increase i by one and return to step 2.

6. [Done.] Terminate with S as the answer.

The quality function $q(t, c') = \max_{c \in C}(l_c + \mathbf{1}_{c=c'} r_t)/r_c$ in step 5, where $\mathbf{1}_{\cdot}$ is the indicator function, represents the load of the highest loaded computer if task t were assigned to computer c' on top of the partial solution S.

*roberto.meroni@estudiantat.upc.edu
†hugo.sanz@estudiantat.upc.edu

#	Computers	Tasks
1	100	500
2	120	500
3	100	550

Table 1: Some characteristics of the generated instances.

2. *Prepare a pseudocode for the local search algorithm. What neighborhoods and exploration strategies are implemented?*

Given a partial solution S, the program considers two kinds of neighborhoods:

- For each assignment $(t, c) \in S$, construct a new solution with exactly the same assignments as S but with task t assigned to a computer in C different from c with enough free resources.
- For each pair of assignments $(t_1, c_1), (t_2, c_2) \in S$, create a new solution with the same assignments as S except that (t_1, c_1) is replaced by (t_1, c_2) and (t_2, c_2) is replaced by (t_2, c_1) provided that the new assignments do not exceed the capacity of c_1 nor c_2.

Also, the program can be configured to select an element from the neighborhood of S in two ways: *a*) explore all near solutions until one with a better objective is found, or *b*) calculate the load of the highest-loaded computer for all solutions in the neighborhood and pick one that minimizes this quantity.

If $i_{\max}$ denotes the maximum number of iterations, then the local search scheme proceeds as follows:

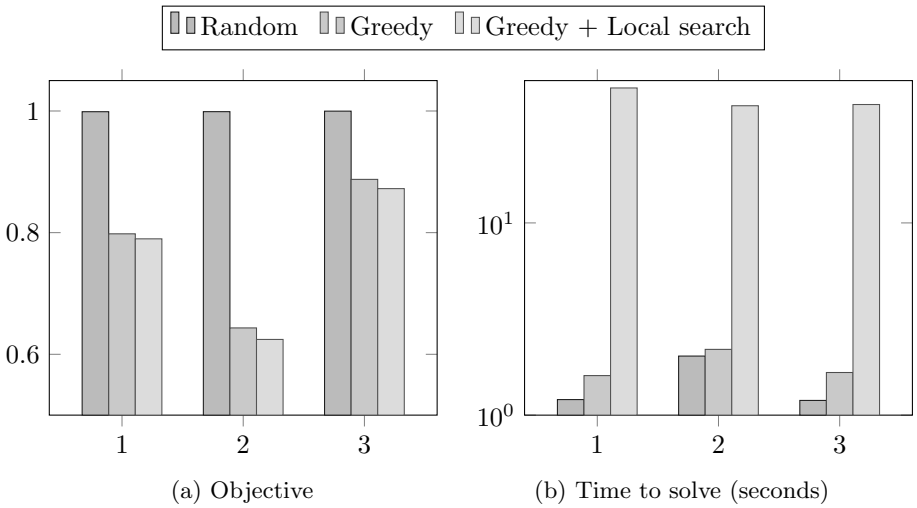
1. [Construct initial solution.] Let S be the partial solution returned by the greedy search algorithm in exercise 1, let z be the load of the highest-loaded computer in S, and set $i \leftarrow 1$.
2. [Explore.] Let S' be a solution in the neighborhood of S such that the load z' of the highest-loaded computer in S' is less than z. If such an S' does not exist, go to step 3. Otherwise set $S \leftarrow S'$ and $z \leftarrow z'$.
3. [Done?] Increase i by 1; if $i \leq i_{\max}$ return to step 2. Otherwise terminate with S as the answer.

In step 2 the solution S' is picked according to one of the exploration strategies described above from one of the two kinds of neighborhoods.

3. *Generate instances of increasing size.*

We generate three instances of P_2 with varying numbers of computers and tasks; the specific values appear in Table 1. The difference in $|C|$ and $|T|$ between these instances is moderate in order to facilitate comparisons in the upcoming figures, but nonetheless they let us see how the performance of the algorithms changes as the problem size increases. The capacity of each computer is drawn uniformly from the interval $[15\,000, 20\,000]$, whereas the compute units required to complete each task is a random number uniformly distributed in $[500, 5000]$.

4. *Solve the instances previously generated using random search only, greedy function only, and greedy + local search.*



(a) Objective (b) Time to solve (seconds)

Figure 1: Solutions to the minimization problems generated in Exercise 3 using different search strategies. The solution times appear in a logarithmic scale.

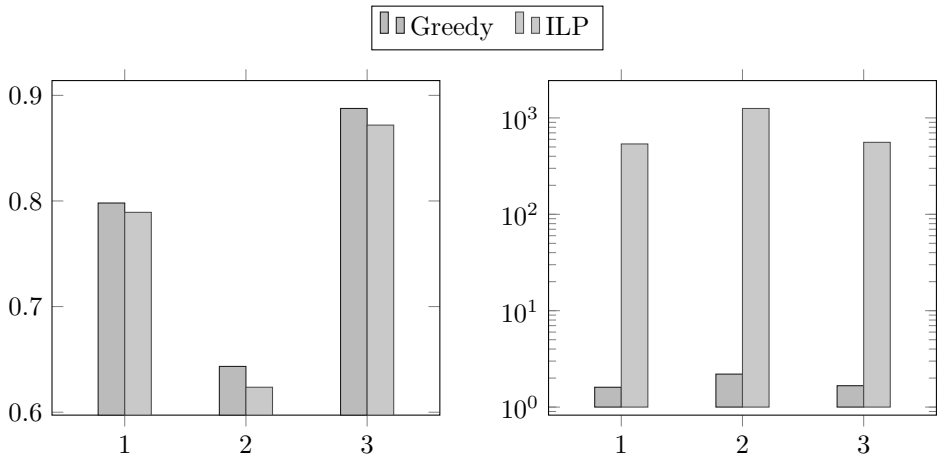
Figure 1 shows the objective of the best solution found and the solution time obtained using the different search strategies. Note that random search yields a quite suboptimal solution in which a computer is close to being overloaded and some tasks are left unassigned, but it does so extremely fast. In contrast, greedy search obtains much better solutions and the scheme is only slightly slower. When local search is performed to explore the solution space around the partial solution found by the greedy algorithm, the load of the highest-loaded computer is decreased by 1–2% through a sequence of task–computer assignment exchanges (as explained in the first paragraph of exercise 2 and specified via the `TaskExchange` option) but the cost of the algorithm increases by a factor of ≈ 25.

Thus, the greedy approach to solving instances of P_2 similar to those generated in the previous exercise offers a good balance between the quality of the solutions obtained and running time. Moreover, the contrasting behavior between the search schemes does not seem to vary much when either the number of computer or tasks changes.

Also observe that the algorithms take longer to find a solution to the second instance than to the other two. The introduction of 20 additional computers means that there are more possibilities for the greedy algorithm to consider for each task, and so the innermost computation in step 5 takes longer to run. The effect when the number of tasks increases is not as significant.

5. *Solve the instances previously generated using the ILP from lab session 2. Configure CPLEX to stop after 30 minutes or GAP* $\leq 1\%$.

We modify the main model file `main.opl` in order to define a stopping criterion that checks if the difference between the objectives of the best feasible solution found so far by CPLEX and the best potential one is below a set tolerance. Experiments suggest that GAP = 1.5% is enough to increase the running time of CPLEX to more than 30 minutes when given an instance generated in



(a) Objective (b) Time to solve (seconds)

Figure 2: Solutions to the minimization problems generated in Exercise 3 using the Greedy and ILP approaches. The solution times appear in a logarithmic scale.

exercise 3; hence we write `cplex.epgap = 0.015;`

6. *Plot the best combination for the Greedy and the ILP in terms of quality of the solutions and time to solve.*

Figure 2 shows the running time and the objective of the best solution found by the greedy algorithm defined in exercise 1 and by CPLEX when given the three instances generated in exercise 3. Note that the ILP model is much harder to solve; for example, the running time of the greedy solver when finding a solution to the second instance is 570 times smaller than that achieved by CPLEX. However, the ILP approach yields solutions that are closer to global minima: in the solution to the third instance, the load of the highest loaded computer is 8.25% lower than that in the greedy solution.

Another advantage of the ILP formulation is ease of programming: designing a greedy search algorithm (or a local search procedure) for the task assignment problem requires evaluating the performance of different quality functions. In contrast, someone with little knowledge of computers is probably capable of describing the problem in the OPL language and defering the search task to solvers like CPLEX.