Algorithmic Methods for Mathematical Models

Lab Session 4 - Greedy and Local Search Heuristics

Arnau Abella
Universitat Politècnica de Catalunya

November 26, 2020

- Prepare a pseudocode for the Greedy algorithm. Specify the greedy function.

Algorithm 1: Greedy Algorithm

Input:

- A set T of tasks, each task t requires r_t resources
- A set C of computers, each computer c has a capacity of r_c resources

Output: A set w of assignments, each assignment $\langle t, c \rangle$ associates a task t with a computer c *s.t.* Each task $t \in T$ appears exactly once

$w \leftarrow \emptyset$
forall $t \in T$ **do**
  $c^{min} \leftarrow q(t, w)$
  if $q(c^{\min}) = \infty$ **then**
    return $INFEASIBLE$
  end

  $w \leftarrow w \cup \{\langle t, c^{min} \rangle\}$
end
return w

$$q(t, w) = \min\{q(\langle t, c \rangle, w) \mid c \in C\}$$

$$q(\langle t, c \rangle, w) = \begin{cases} \infty & if r_t > r_c - \sum_{t' \in w_c} r_{t'} \\ \frac{r_t + \sum_{t' \in w_c} r_{t'}}{r_c} & otherwise \end{cases}$$

where $w_c = \{t_i, \ldots, t_j\} \subseteq T$ are the tasks assigned to computer c in the partial solution w.

- Prepare a pseudocode for the Local search algorithm. What neighborhoods and exploration strategies are implemented?

Algorithm 2: Local Search: Task Swapping

Input: A set w of assignments, each assignment $\langle t, c\rangle$ associates a task t with a computer c

Output: A set w' of assignments $s.t.$ $q(w') \leq q(w)$

```
w' ← w
while execution time ≤ time limit do
    sC ← sort w' by computer's load (desc)
    moves ← ∅
    for c ∈ sC do                          /* list traversal, order matters */
        foreach t ∈ w'_c do
            for c' ∈ sC s.t. c' ≤ c do
                foreach t' ∈ w'_{c'} do
                    rc = r_c − Σ_{t''∈w_c} r_{t''}
                    rc' = r_{c'} − Σ_{t''∈w_{c'}} r_{t''}
                    if r_{t'} − r_t ≤ rc ∧ r_t − r_{t'} ≤ rc' then
                        rc_new = rc + r_t − r_{t'}
                        rc'_new = rc' + r_{t'} − r_t
                        improvement = min(rc_new, rc'_new)
                        if improvement > 0 then
                            moves ← moves ∪ {⟨t, c, t', c', improvement⟩}
                        end
                    end
                end
            end
        end
    end
    if moves = ∅ then
        return w'
    end

    ⟨t, c, t', c', _⟩ ← arg max{⟨_, _, _, _, impr⟩ ← moves}
end
w' ← w' ∪ (w'_c \ {t} ∪ {t'})
w' ← w' ∪ (w'_{c'} \ {t'} ∪ {t})
return w'
```

The *local search* uses a *task swapping* algorithm and the strategy performed is *best improvement*. The *python* code implements the same algorithm but also includes a *first improvement* strategy.

- Generate instances of increasing size. Store these instances as they will be solved in the coming lab sessions.
- Solve the instances previously generated using
    - Random only
    - Greedy function only
    - Greedy + Local search

  Plot the quality of the solutions and time to solve. Select the best combination.

  The *randomized* version can be discarded because of the quality of the solutions (figure 1). The *greedy with local search (best improvement)* can be discarded because of the execution time, see figure 1, which grows so fast that I couldn't even manage to run the smallest instances without hitting the soft time limit of the program. Between the *greedy without local search* algorithm and the *greedy with local search (first improvement)* it is more disputed. The version with local search always returns better solutions but the execution time grows some order of magnitudes faster.

  So, from the collected data, I would recommend using the *greedy with local search (first improvement)* algorithm for small instances where the execution time is feasible and the *greedy without local search* algorithm for big instances.

(a) Solve the instances previously generated using the ILP from lab session 2. Configure CPLEX to stop after 30min or GAP $\leq 1\%$.

- Solve the instances previously generated using the ILP from lab session 2. Configure CPLEX to stop after 30min or GAP $\leq 1\%$.
- Plot the best combination for the Greedy and the ILP of quality of the solutions and time to solve.

  The *ILP* version performs better in both speed and solution's quality, see figure 2. Although, this comparison is not fair at all, the greedy algorithm is executing non-optimized *python* code which is per se 10-100x orders of magnitude slower than *C* while the ILP version is executing highly-optimized *C* code.

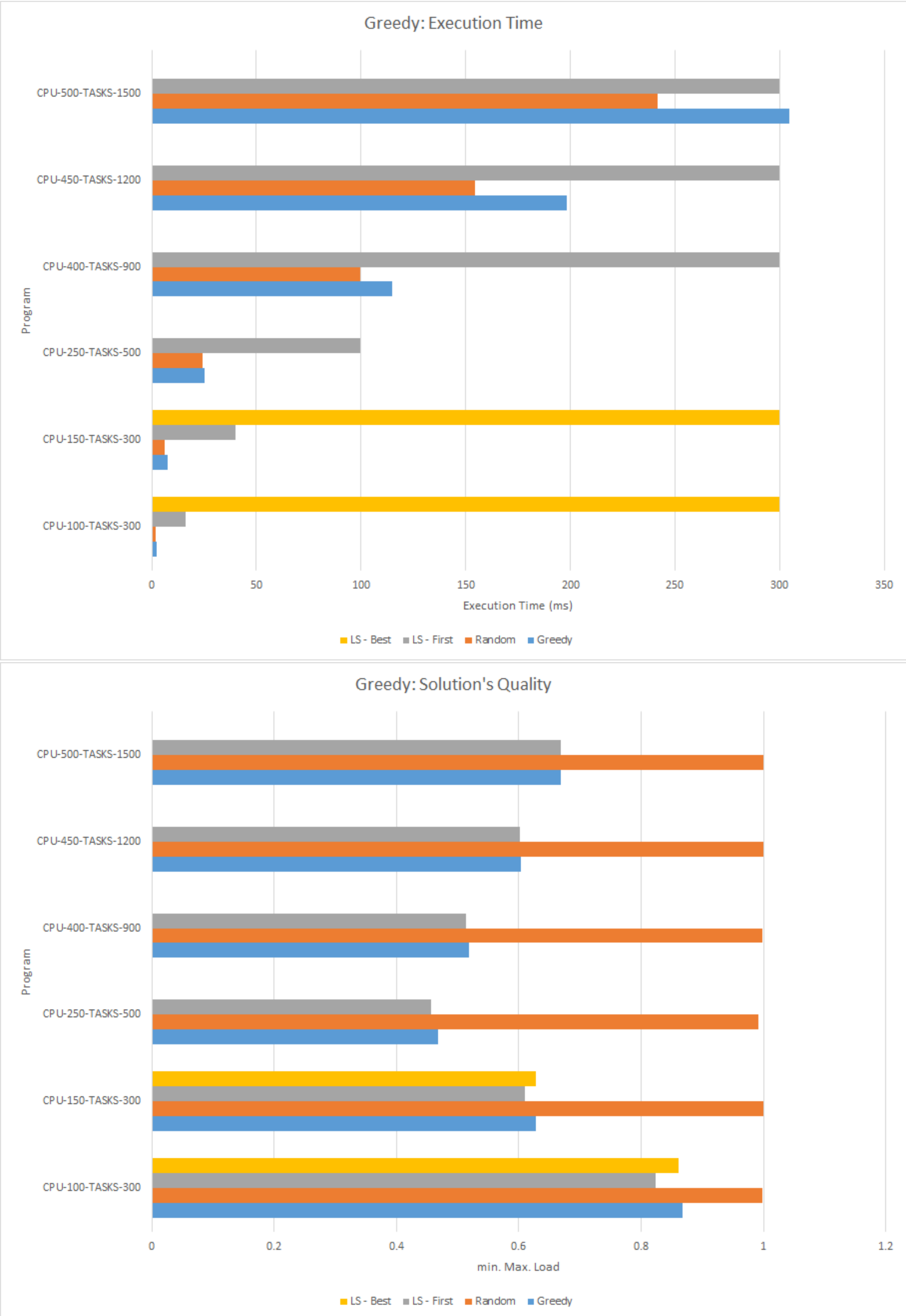
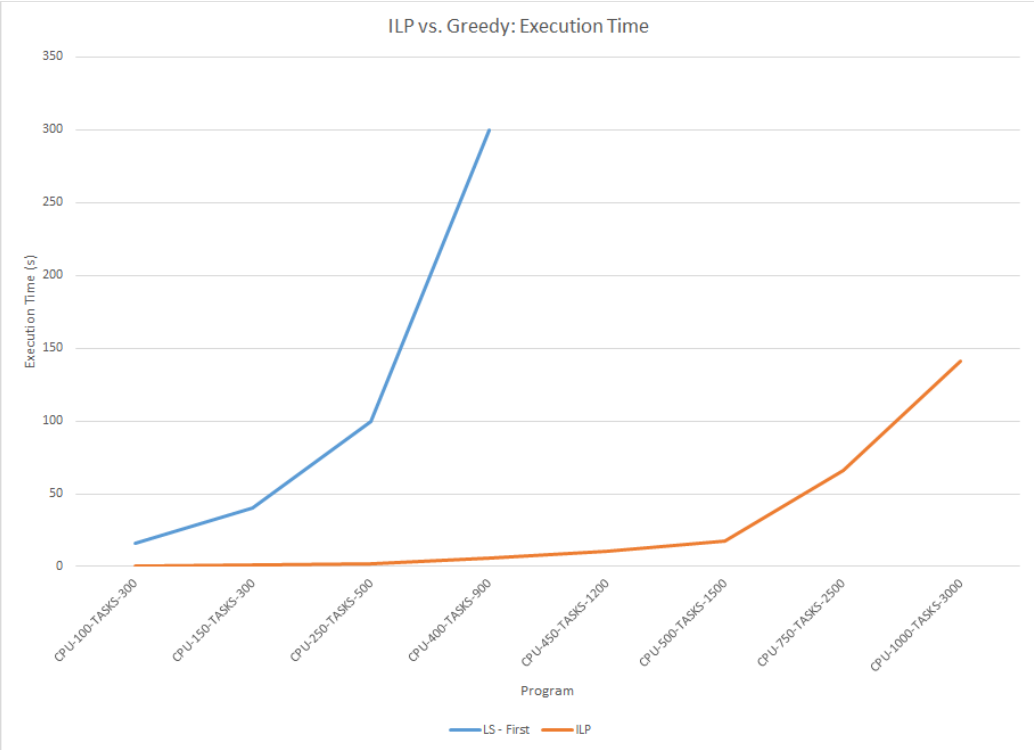


Figure 1: Comparison between the *greedy algorithms*



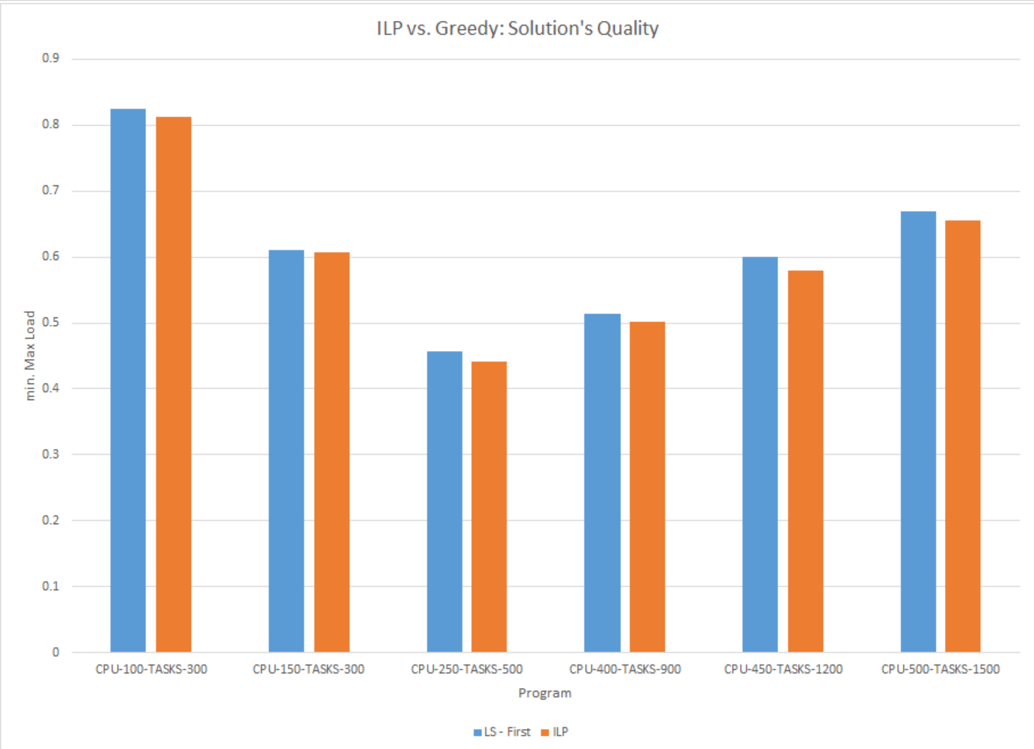


Figure 2: Comparison between the *Greedy* and the *ILP* algorithm