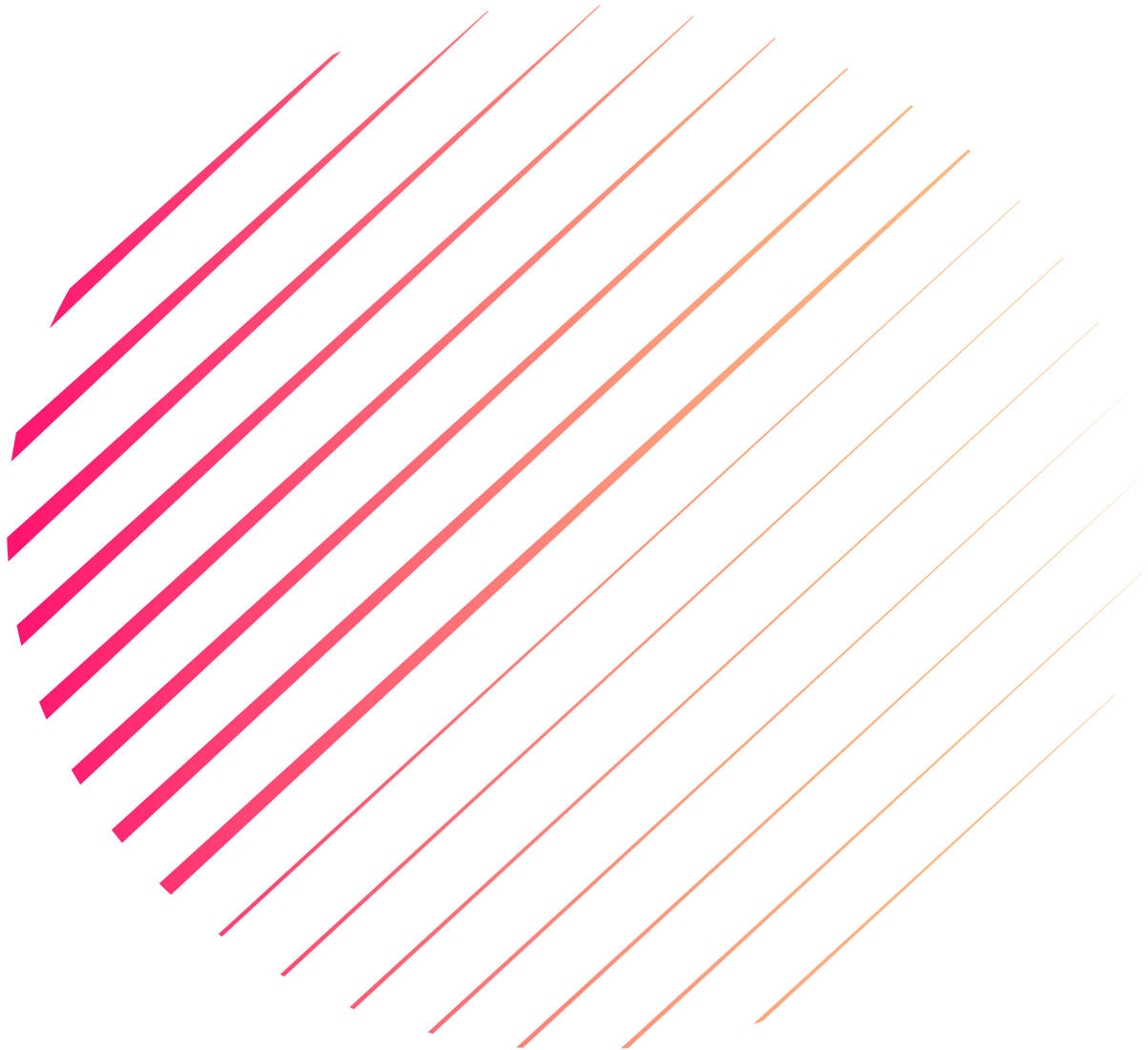


Report Sprint B




November 2024
Algoritmia Avançada

Ana Oliveira, I220954
Rafael Rocha, I201260
Beatriz Costa, I211588



User Stories

- 6.3.1 As an Admin, I want to obtain the better scheduling of a set of operations (surgeries) in a certain operation room in a specific day.
 - 6.3.2 As an Admin, I want to know till what dimension in terms of number of surgeries is possible to ask for the better solution.
 - 6.3.3 As an Admin, I want to obtain a good schedule, not necessarily the better, in useful time to be adopted.
- 

Main Predicate Analysis

```

schedule_all_surgeries(Room, Date, BestSchedule, EarliestFinishTime) :-
    get_time(StartTime), % Start the timer

    findall((OpId, OpName), operation_request(OpId, _, _, _, _, OpName),
            Operations),

    length(Operations, NumOperations),

    format_input_to_number(Date, DateNumber),

    findall((Start, End),

            dbConnection:free_slot(Room, DateNumber, Start, End),

            RoomSlots),

    findall(Permutation, permutation(Operations, Permutation), Permutations),

    find_best_schedule(Permutations, RoomSlots, Date, Room, BestSchedule,
                        EarliestFinishTime),

    preprocess_schedule(BestSchedule, PreprocessedSchedule),

    filter_best_schedule(PreprocessedSchedule, ValidBestSchedule),

    assign_staff_to_schedule(ValidBestSchedule, FinalSchedule),

    get_time(EndTime), % End the timer

    ExecutionTime is EndTime - StartTime, % Calculate total execution time

    display_schedule_and_confirm(FinalSchedule, Date, Room).

    findall(Permutation, permutation(Operations, Permutation), Permutations),

```

In this line, it will search all the permutations for the number of surgeries. Complexity: $O(n!)$.

`find_best_schedule/7(Permutations, RoomSlots, Date, Room, BestSchedule, EarliestFinishTime),`

The predicate `find_best_schedule` iterates through all the the $n!$ permutations stored in `Permutations`. For each permutation, it evaluates the schedule using the predicate `evaluate_schedule/6`.

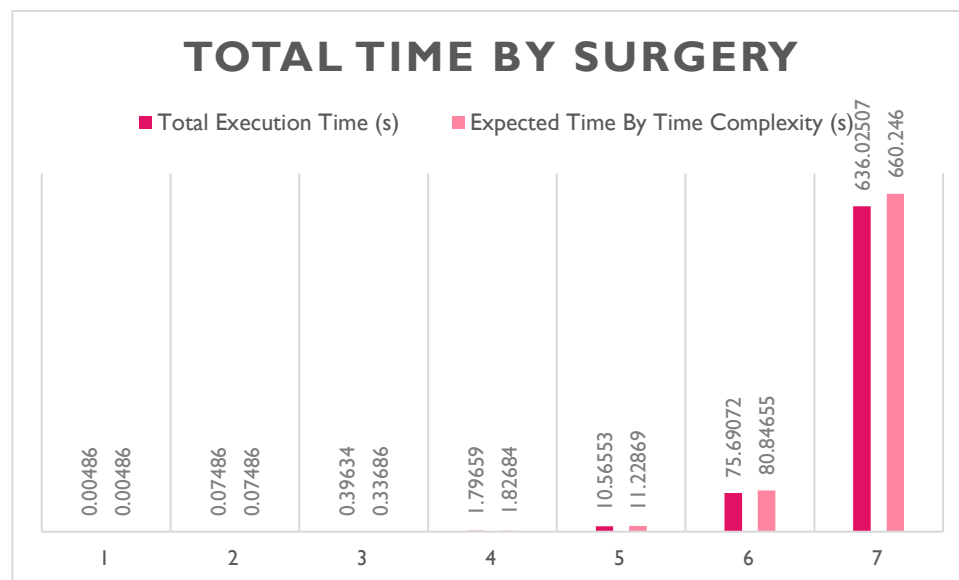
Complexity: $O(n)$.

General Complexity: $O(n * n!)$.

The complexity of $O(n!)$ comes from the utilization of the `findall(Permutation, permutation(...), Permutations)` predicate as seen before; and the complexity of $O(n)$ comes from each permutation of the `find_best_schedule/7` predicate.

Better solution

Number Of Surgeries	Total Execution Time (s)	Expected Time By Time Complexity (s)	Number of Permutations (n!)
1	0.00486	0.00486	1
2	0.07486	0.07486	2
3	0.39634	0.33686	6
4	1.79659	1.82684	24
5	10.56553	11.22869	120
6	75.69072	80.84655	720
7	636.02507	660.246	5040
8		5281.97	40.320



By analyzing the graphics, we can conclude that up to 5-6 surgeries the current algorithm is effective in finding the better solution. However, after 6 surgeries, the execution time starts to grow drastically, making it undoable for such a short time scenario. For example, for 7 surgeries, the execution time already surpasses the 10-minute mark, which is not acceptable in a context where speed is essential, like surgery scheduling in real time.

For operations that involve more than 6 surgeries, it will be used a greedy approach as the only heuristic based in the “Earliest Finish Time” (soonest finishing time). This method prioritizes the operations that can be concluded the fastest, attributing the resources and schedules available in an iterative form, to maximize efficiency and minimize the total execution time. Even if it doesn’t provide an optimal solution, this method is effective to generate a good scheduling within the reasonable time period, especially in higher priority scenarios.

