Relatório ALGAV

Sprint B

3DJ G059

Feito por:

André Ferreira, 1190378

David Marques, 1221276

Diogo Cunha, 1221071

João Monteiro, 1221023

Tomás Peixoto, 1221948

Professor:

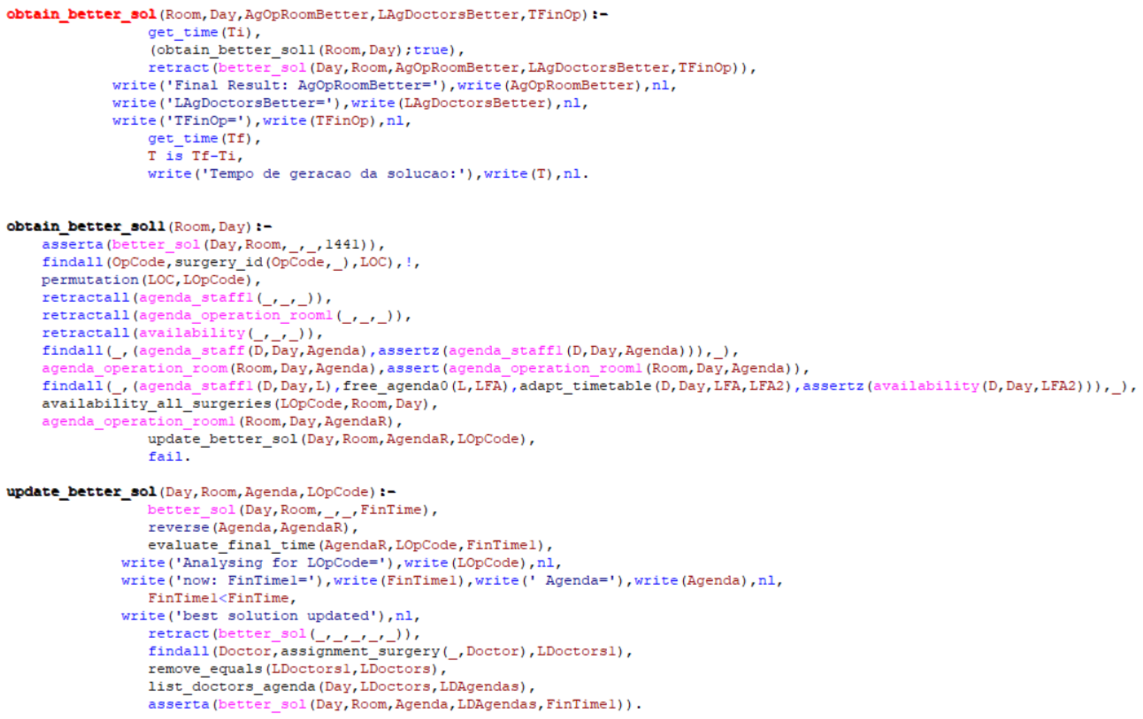
Luis Manuel Silva Conceição (MSC)

19/11/2024

Index

1. Brief explanation of the basic code

The code presented to us in the first 3 TP classes contains the methods needed to obtain the best schedule from a set of operations

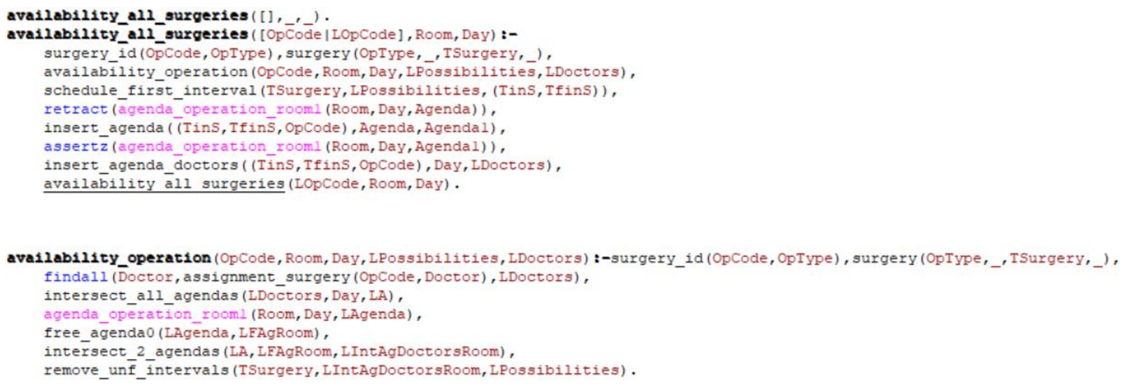


```
obtain_better_sol(Room,Day,AgOpRoomBetter,LAgDoctorsBetter,TFinOp):-
                get_time(Ti),
                (obtain_better_sol1(Room,Day);true),
                retract(better_sol(Day,Room,AgOpRoomBetter,LAgDoctorsBetter,TFinOp)),
            write('Final Result: AgOpRoomBetter='),write(AgOpRoomBetter),nl,
            write('LAgDoctorsBetter='),write(LAgDoctorsBetter),nl,
            write('TFinOp='),write(TFinOp),nl,
                get_time(Tf),
                T is Tf-Ti,
                write('Tempo de geracao da solucao:'),write(T),nl.


obtain_better_sol1(Room,Day):-
    asserta(better_sol(Day,Room,_,_,1441)),
    findall(OpCode,surgery_id(OpCode,_),LOC),!,
    permutation(LOC,LOpCode),
    retractall(agenda_staff1(_,_,_)),
    retractall(agenda_operation_room1(_,_,_)),
    retractall(availability(_,_,_)),
    findall(_,(agenda_staff(D,Day,Agenda),assertz(agenda_staff1(D,Day,Agenda))),_),
    agenda_operation_room(Room,Day,Agenda),assert(agenda_operation_room1(Room,Day,Agenda)),
    findall(_,(agenda_staff1(D,Day,L),free_agenda0(L,LFA),adapt_timetable(D,Day,LFA,LFA2),assertz(availability(D,Day,LFA2))),_),
    availability_all_surgeries(LOpCode,Room,Day),
    agenda_operation_room1(Room,Day,AgendaR),
                update_better_sol(Day,Room,AgendaR,LOpCode),
                fail.

update_better_sol(Day,Room,Agenda,LOpCode):-
                better_sol(Day,Room,_,_,FinTime),
                reverse(Agenda,AgendaR),
                evaluate_final_time(AgendaR,LOpCode,FinTime1),
             write('Analysing for LOpCode='),write(LOpCode),nl,
             write('now: FinTime1='),write(FinTime1),write(' Agenda='),write(Agenda),nl,
                FinTime1<FinTime,
             write('best solution updated'),nl,
                retract(better_sol(_,_,_,_,_)),
                findall(Doctor,assignment_surgery(_,Doctor),LDoctors1),
                remove_equals(LDoctors1,LDoctors),
                list_doctors_agenda(Day,LDoctors,LDAgendas),
                asserta(better_sol(Day,Room,Agenda,LDAgendas,FinTime1)).
```

Figure 1. ***obtain_better_sol/5*** *method*

This method runs every possible combination of surgeries, seeing if each one is the best one. When the program ends, it prints:

- Execution time (in seconds).
- End time of final operation (in minutes).
- Room's agenda (including any pre-processed operations).
- Each doctor's agenda.

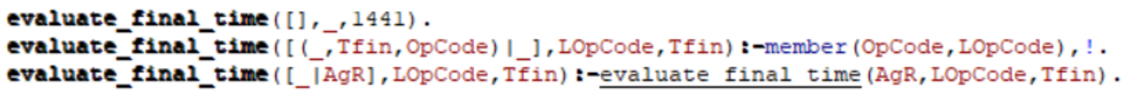


```
availability_all_surgeries([],_,_).
availability_all_surgeries([OpCode|LOpCode],Room,Day):-
    surgery_id(OpCode,OpType),surgery(OpType,_,TSurgery,_),
    availability_operation(OpCode,Room,Day,LPossibilities,LDoctors),
    schedule_first_interval(TSurgery,LPossibilities,(TinS,TfinS)),
    retract(agenda_operation_room1(Room,Day,Agenda)),
    insert_agenda((TinS,TfinS,OpCode),Agenda,Agenda1),
    assertz(agenda_operation_room1(Room,Day,Agenda1)),
    insert_agenda_doctors((TinS,TfinS,OpCode),Day,LDoctors),
    availability_all_surgeries(LOpCode,Room,Day).


availability_operation(OpCode,Room,Day,LPossibilities,LDoctors):-surgery_id(OpCode,OpType),surgery(OpType,_,TSurgery,_),
    findall(Doctor,assignment_surgery(OpCode,Doctor),LDoctors),
    intersect_all_agendas(LDoctors,Day,LA),
    agenda_operation_room1(Room,Day,LAgenda),
    free_agenda0(LAgenda,LFAgRoom),
    intersect_2_agendas(LA,LFAgRoom,LIntAgDoctorsRoom),
    remove_unf_intervals(TSurgery,LIntAgDoctorsRoom,LPossibilities).
```

*Figure 2.**availability_all_surgeries/3** method*

This method is called by the previous ones to schedule the operations. To do this, it checks the room's agenda and the doctor's agenda and inserts this new plan to their agendas. Since this is done with every surgery, the next one will have to account for how the previous one filled the various agendas. This way, we can ensure that no two operations use the same room or the same staff at the same time.

Note: For now, this is only considering **one** doctor per surgery. It's also just considering the **Surgery** time of an operation, ignoring the **Anesthesia** time (before **Surgery**) and the **Cleaning** time (after **Surgery**).



```
evaluate_final_time([],_,1441).
evaluate_final_time([(_,Tfin,OpCode)|_],LOpCode,Tfin):-member(OpCode,LOpCode),!.
evaluate_final_time([_|AgR],LOpCode,Tfin):-evaluate_final_time(AgR,LOpCode,Tfin).
```

*Figure 3. **evaluate_final_time/3** method*

This method represents the current evaluation function, the criteria that decides if a schedule is considered "better" than another. It aims to minimize the final time of the last surgery of the sequence. For this reason, it receives the scheduled agenda and returns its final time.

2. Complexity Study (6.3.2)

Brute Force Approach - Enumerate all possible schedules and evaluate them.

The number of solutions grows factorially with ***N!***

Importance of the Complexity Study:

- **An estimate of the time required** to generate all possible solutions for a given number of surgeries and select the optimal one.
- **Feasibility limits**, identifying the maximum problem size (number of surgeries) that can be solved within a given time frame.

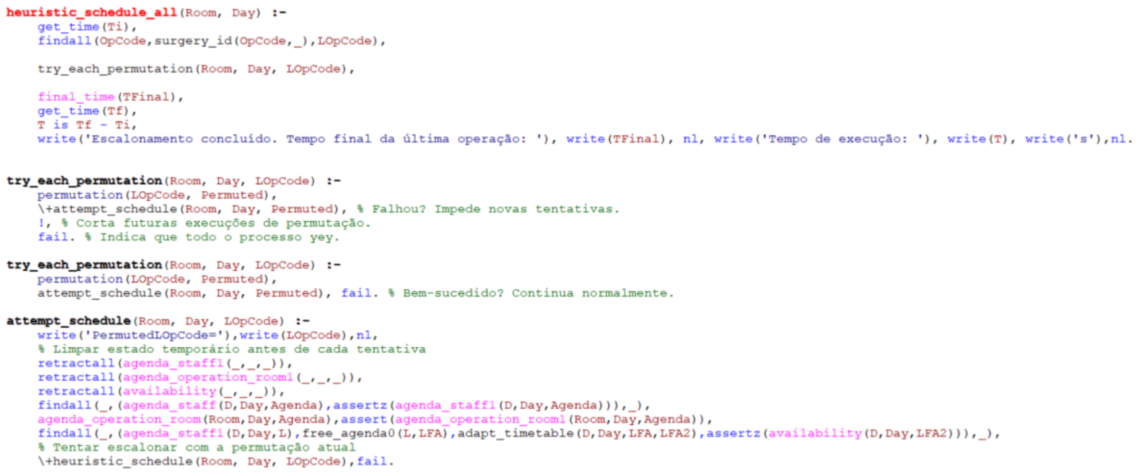
N. of Surgeries	N. of solutions	Best Schedule of activities (including surgeries) of the operation room	Final Time for the last Surgery (min)	Time to generate the solution (s)
3	6	[(520, 579, so100000), (580, 639, so100001), (640, 714, so100003), (715, 804, so100002), (1000, 1059, so099999)]	804	0.028
4	24	[(520, 579, so100000), (580, 654, so100003), (655, 714, so100004), (715, 804, so100002), (805, 864, so100001), (1000, 1059, so099999)]	864	0.27
5	120	[(520, 579, so100000), (580, 639, so100004), (640, 714, so100005), (715, 804, so100002), (805, 879, so100003), (880, 939, so100001), (1000, 1059, so099999)]	939	1.17
6	720	[(520, 579, so100000), (580, 639, so100004), (640, 714, so100005), (715, 804, so100002), (805, 879, so100003), (880, 939, so100001), (940, 999, so100006), (1000, ..., ...)]	999	5.04
7	5040	[(520, 579, so100000), (580, 639, so100004), (640, 714, so100005), (715, 804, so100002), (805, 879, so100003), (880, 939, so100001), (940, 999, so100006), (1000, ..., ...), (..., ...)]	1149	23.11
8	40320	[(520, 579, so100000), (580, 639, so100004), (640, 699, so100008), (700, 789, so100002), (791, 865, so100003), (866, 925, so100001), (926, 985, so100006), (1000, ..., ...), (..., ...)|...]	1224	29.48
9	362880	[(520, 579, so100000), (580, 639, so100004), (640, 699, so100008), (700, 789, so100002), (790, 849, so100009), (850, 909, so100001), (910, 969, so100006), (1000, ..., ...), (..., ...)|...]	1299	72.23
10	3628800	[(520, 579, so100000), (580, 639, so100004), (640, 699, so100008), (700, 759, so100009), (791, 865, so100003), (866, 925, so100001), (926, 985, so100006), (1000, ..., ...), (..., ...)|...]	1374	876.69
11	39916800			
12	479001600			
13	6227020800			

Analyzing this, we can conclude that it stops being reasonable to ask for the best possible solution when you must schedule 10 or more surgeries.

3. Heuristics (6.3.3)

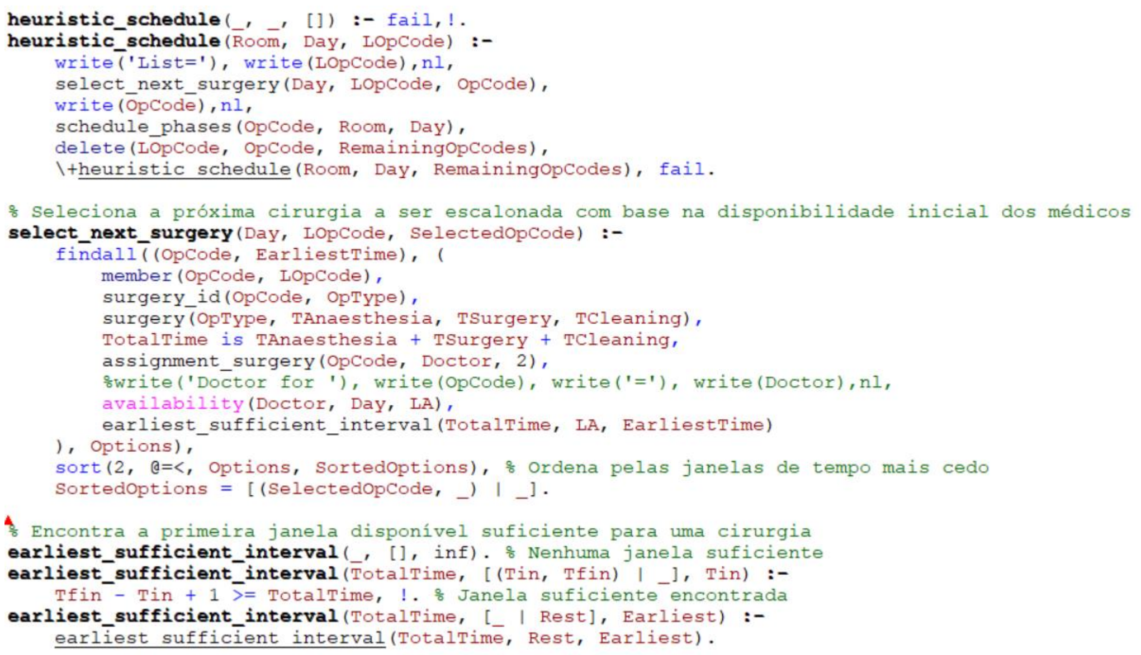
Heuristic 1

In this heuristic scheduling system, the concept of "being available early" refers to the ability to identify and select a doctor who has a sufficient time window available **early in their schedule** to accommodate the total duration of a given surgery.



```
heuristic_schedule_all(Room, Day) :-
    get_time(Ti),
    findall(OpCode,surgery_id(OpCode,_),LOpCode),

    try_each_permutation(Room, Day, LOpCode),

    final_time(TFinal),
    get_time(Tf),
    T is Tf - Ti,
    write('Escalonamento concluído. Tempo final da última operação: '), write(TFinal), nl, write('Tempo de execução: '), write(T), write('s'),nl.


try_each_permutation(Room, Day, LOpCode) :-
    permutation(LOpCode, Permuted),
    \+attempt_schedule(Room, Day, Permuted), % Falhou? Impede novas tentativas.
    !, % Corta futuras execuções de permutação.
    fail. % Indica que todo o processo yey.

try_each_permutation(Room, Day, LOpCode) :-
    permutation(LOpCode, Permuted),
    attempt_schedule(Room, Day, Permuted), fail. % Bem-sucedido? Continua normalmente.

attempt_schedule(Room, Day, LOpCode) :-
    write('PermutedLOpCode='),write(LOpCode),nl,
    % Limpar estado temporário antes de cada tentativa
    retractall(agenda_staff1(_,_,_)),
    retractall(agenda_operation_room1(_,_,_)),
    retractall(availability(_,_,_)),
    findall(_,(agenda_staff(D,Day,Agenda),assertz(agenda_staff1(D,Day,Agenda))),_),
    agenda_operation_room(Room,Day,Agenda),assert(agenda_operation_room1(Room,Day,Agenda)),
    findall(_,(agenda_staff1(D,Day,L),free_agenda0(L,LFA),adapt_timetable(D,Day,LFA,LFA2),assertz(availability(D,Day,LFA2))),_),
    % Tentar escalonar com a permutação atual
    \+heuristic_schedule(Room, Day, LOpCode),fail.
```

Figure 4. Heuristic part 1



```
heuristic_schedule(_, _, []) :- fail,!.
heuristic_schedule(Room, Day, LOpCode) :-
    write('List='), write(LOpCode),nl,
    select_next_surgery(Day, LOpCode, OpCode),
    write(OpCode),nl,
    schedule_phases(OpCode, Room, Day),
    delete(LOpCode, OpCode, RemainingOpCodes),
    \+heuristic_schedule(Room, Day, RemainingOpCodes), fail.

% Seleciona a próxima cirurgia a ser escalonada com base na disponibilidade inicial dos médicos
select_next_surgery(Day, LOpCode, SelectedOpCode) :-
    findall((OpCode, EarliestTime), (
        member(OpCode, LOpCode),
        surgery_id(OpCode, OpType),
        surgery(OpType, TAnaesthesia, TSurgery, TCleaning),
        TotalTime is TAnaesthesia + TSurgery + TCleaning,
        assignment_surgery(OpCode, Doctor, 2),
        %write('Doctor for '), write(OpCode), write('='), write(Doctor),nl,
        availability(Doctor, Day, LA),
        earliest_sufficient_interval(TotalTime, LA, EarliestTime)
    ), Options),
    sort(2, @=<, Options, SortedOptions), % Ordena pelas janelas de tempo mais cedo
    SortedOptions = [(SelectedOpCode, _) | _].

% Encontra a primeira janela disponível suficiente para uma cirurgia
earliest_sufficient_interval(_, [], inf). % Nenhuma janela suficiente
earliest_sufficient_interval(TotalTime, [(Tin, Tfin) | _], Tin) :-
    Tfin - Tin + 1 >= TotalTime, !. % Janela suficiente encontrada
earliest_sufficient_interval(TotalTime, [_ | Rest], Earliest) :-
    earliest_sufficient_interval(TotalTime, Rest, Earliest).
```

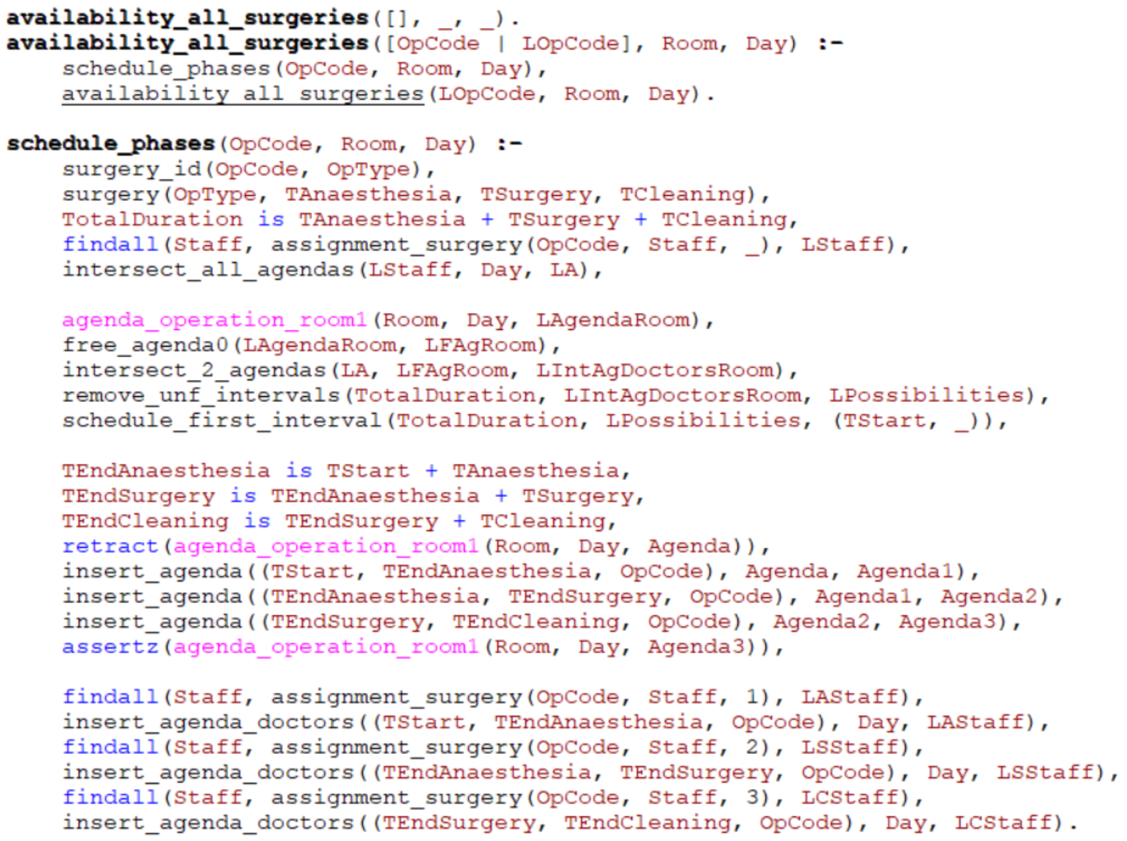
Figure 5. Heuristic part 2

Here’s how this is evaluated step-by-step in the predicate:

1. **Calculate Total Surgery Time,** the total time required is determined by**:**
    a. **Anaesthesia time**
    b. **Surgery time**
    c. **Cleaning time**
2. **Check Doctor's Availability**: Each surgery is assigned to a specific doctor using the predicate.
3. **Find the Earliest Sufficient Time Window**: For each surgery and its assigned doctor
4. **Sort and Prioritize**: All surgeries that can be scheduled (based on available intervals) are sorted by their earliest possible start times. The surgery with the earliest feasible start time is selected for scheduling.
5. **Example Scenario**:
    a. **Doctor's availability**: [(8:00, 8:30), (9:00, 11:00)]
    b. **Surgery type**: SO2 (requires 60 minutes)
    c. The first interval, (8:00, 8:30), is insufficient because it only allows for 30 minutes.
    d. The second interval, (9:00, 11:00), is sufficient. The surgery can start at 9:00.
    e. Thus, this doctor would not be "available early" for this surgery at 8:00, but would be available starting at 9:00.

4. Involve other staff (6.3.1)

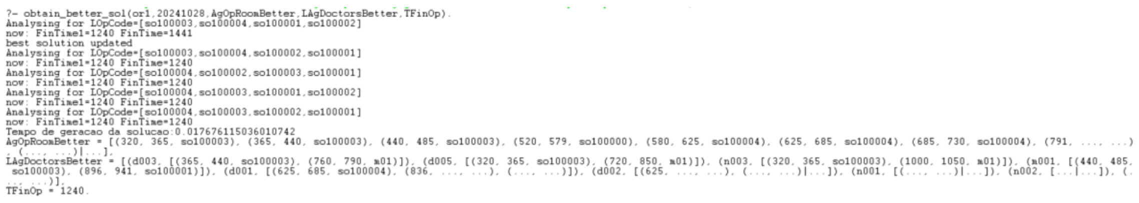
To improve the given code, involving the other staff of an operation and its phases (**Anesthesia**, **Surgery** (already scheduled) and **Cleaning**), the code was changed dramatically.



```
availability_all_surgeries([], _, _).
availability_all_surgeries([OpCode | LOpCode], Room, Day) :-
    schedule_phases(OpCode, Room, Day),
    availability_all_surgeries(LOpCode, Room, Day).

schedule_phases(OpCode, Room, Day) :-
    surgery_id(OpCode, OpType),
    surgery(OpType, TAnaesthesia, TSurgery, TCleaning),
    TotalDuration is TAnaesthesia + TSurgery + TCleaning,
    findall(Staff, assignment_surgery(OpCode, Staff, _), LStaff),
    intersect_all_agendas(LStaff, Day, LA),

    agenda_operation_room1(Room, Day, LAgendaRoom),
    free_agenda0(LAgendaRoom, LFAgRoom),
    intersect_2_agendas(LA, LFAgRoom, LIntAgDoctorsRoom),
    remove_unf_intervals(TotalDuration, LIntAgDoctorsRoom, LPossibilities),
    schedule_first_interval(TotalDuration, LPossibilities, (TStart, _)),

    TEndAnaesthesia is TStart + TAnaesthesia,
    TEndSurgery is TEndAnaesthesia + TSurgery,
    TEndCleaning is TEndSurgery + TCleaning,
    retract(agenda_operation_room1(Room, Day, Agenda)),
    insert_agenda((TStart, TEndAnaesthesia, OpCode), Agenda, Agenda1),
    insert_agenda((TEndAnaesthesia, TEndSurgery, OpCode), Agenda1, Agenda2),
    insert_agenda((TEndSurgery, TEndCleaning, OpCode), Agenda2, Agenda3),
    assertz(agenda_operation_room1(Room, Day, Agenda3)),

    findall(Staff, assignment_surgery(OpCode, Staff, 1), LAStaff),
    insert_agenda_doctors((TStart, TEndAnaesthesia, OpCode), Day, LAStaff),
    findall(Staff, assignment_surgery(OpCode, Staff, 2), LSStaff),
    insert_agenda_doctors((TEndAnaesthesia, TEndSurgery, OpCode), Day, LSStaff),
    findall(Staff, assignment_surgery(OpCode, Staff, 3), LCStaff),
    insert_agenda_doctors((TEndSurgery, TEndCleaning, OpCode), Day, LCStaff).
```

*Figure 6. **schedule_phases** method*

For each operation, we arrange a time slot that could contain the entire operation and intercept the agendas of all the staff assigned to it. Then, the code creates a slot for each phase and inserts them into the respective staff (staff only in the cleaning phase of an operation only receive that time slot).



```
?- obtain_better_sol(or1,20241028,AgOpRoomBetter,LAgDoctorsBetter,TFinOp).
Analysing for LOpCode=[so100003,so100004,so100001,so100002]
now: FinTime1=1240 FinTime=1441
best solution updated
Analysing for LOpCode=[so100003,so100004,so100002,so100001]
now: FinTime1=1240 FinTime=1240
Analysing for LOpCode=[so100004,so100002,so100003,so100001]
now: FinTime1=1240 FinTime=1240
Analysing for LOpCode=[so100004,so100003,so100001,so100002]
now: FinTime1=1240 FinTime=1240
Analysing for LOpCode=[so100004,so100003,so100002,so100001]
now: FinTime1=1240 FinTime=1240
Tempo de geracao da solucao:0.017676115036010742
AgOpRoomBetter = [(320, 365, so100003), (365, 440, so100003), (440, 485, so100003), (520, 579, so100000), (580, 625, so100004), (625, 685, so100004), (685, 730, so100004), (791, ..., ...)
, (..., ...)|...],
LAgDoctorsBetter = [(d003, [(365, 440, so100003), (760, 790, m01)]), (d005, [(320, 365, so100003), (720, 850, m01)]), (n003, [(320, 365, so100003), (1000, 1050, m01)]), (m001, [(440, 485,
 so100003), (896, 941, so100001)]), (d001, [(625, 685, so100004), (836, ..., ...), (..., ...)]), (d002, [(625, ..., ...), (..., ...)|...]), (n001, [(..., ...)|...]), (n002, [...|...]), (
..., ...)],
TFinOp = 1240.
```

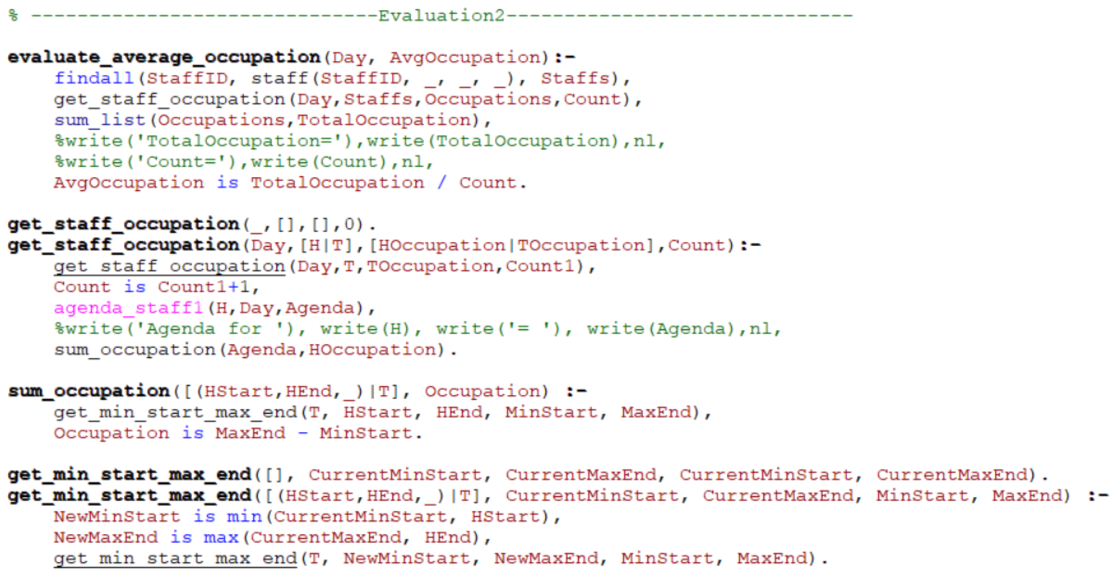
Figure 7. Output example

This is what the output looks like. As you can see, the phases of an operation are consecutive, and a staff's agenda is only filled with the phase they're involved in.

5. Second evaluation function (Appendix)

New evaluation function:

- Focused on minimizing the medium occupied time of staff members.

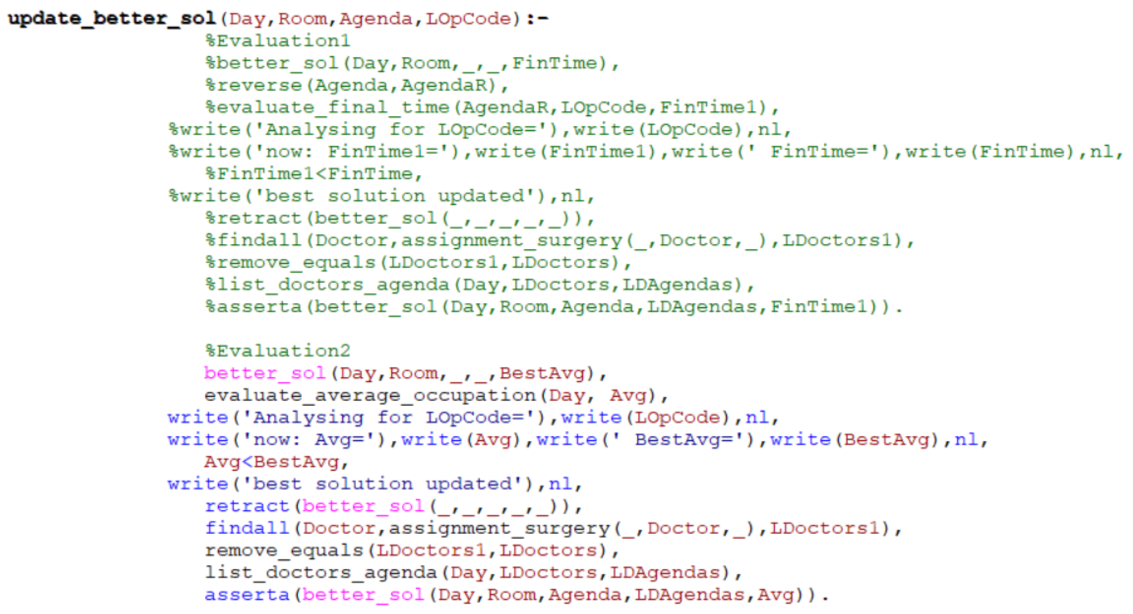


```
% ------------------------------Evaluation2------------------------------

evaluate_average_occupation(Day, AvgOccupation):-
    findall(StaffID, staff(StaffID, _, _, _), Staffs),
    get_staff_occupation(Day,Staffs,Occupations,Count),
    sum_list(Occupations,TotalOccupation),
    %write('TotalOccupation='),write(TotalOccupation),nl,
    %write('Count='),write(Count),nl,
    AvgOccupation is TotalOccupation / Count.

get_staff_occupation(_,[],[],0).
get_staff_occupation(Day,[H|T],[HOccupation|TOccupation],Count):-
    get_staff_occupation(Day,T,TOccupation,Count1),
    Count is Count1+1,
    agenda_staff1(H,Day,Agenda),
    %write('Agenda for '), write(H), write('= '), write(Agenda),nl,
    sum_occupation(Agenda,HOccupation).

sum_occupation([(HStart,HEnd,_)|T], Occupation) :-
    get_min_start_max_end(T, HStart, HEnd, MinStart, MaxEnd),
    Occupation is MaxEnd - MinStart.

get_min_start_max_end([], CurrentMinStart, CurrentMaxEnd, CurrentMinStart, CurrentMaxEnd).
get_min_start_max_end([(HStart,HEnd,_)|T], CurrentMinStart, CurrentMaxEnd, MinStart, MaxEnd) :-
    NewMinStart is min(CurrentMinStart, HStart),
    NewMaxEnd is max(CurrentMaxEnd, HEnd),
    get_min_start_max_end(T, NewMinStart, NewMaxEnd, MinStart, MaxEnd).
```

Figure 8. ***evaluate_average_occupation/2*** *method*

This function, instead of deciding that a schedule is better because it ends earlier, decides that a schedule is better if **the medium of occupied time of staff members** (ignoring any gaps in their agendas, just the final time – initial time) is smaller. It checks the agenda of every staff member (***get_staff_ocupation/4***), sums all of their occupied times together and returns the average (***AvgOccupation***).

Just like the previously mentioned ***evaluate_final_time*** method, this one is invoked by ***update_better_sol/4*** in order to analyze if one schedule is better than another while generating various solutions:



```
update_better_sol(Day,Room,Agenda,LOpCode):-
                %Evaluation1
                %better_sol(Day,Room,_,_,FinTime),
                %reverse(Agenda,AgendaR),
                %evaluate_final_time(AgendaR,LOpCode,FinTime1),
             %write('Analysing for LOpCode='),write(LOpCode),nl,
             %write('now: FinTime1='),write(FinTime1),write(' FinTime='),write(FinTime),nl,
                %FinTime1<FinTime,
             %write('best solution updated'),nl,
                %retract(better_sol(_,_,_,_,_)),
                %findall(Doctor,assignment_surgery(_,Doctor,_),LDoctors1),
                %remove_equals(LDoctors1,LDoctors),
                %list_doctors_agenda(Day,LDoctors,LDAgendas),
                %asserta(better_sol(Day,Room,Agenda,LDAgendas,FinTime1)).

                %Evaluation2
                better_sol(Day,Room,_,_,BestAvg),
                evaluate_average_occupation(Day, Avg),
             write('Analysing for LOpCode='),write(LOpCode),nl,
             write('now: Avg='),write(Avg),write(' BestAvg='),write(BestAvg),nl,
                Avg<BestAvg,
             write('best solution updated'),nl,
                retract(better_sol(_,_,_,_,_)),
                findall(Doctor,assignment_surgery(_,Doctor,_),LDoctors1),
                remove_equals(LDoctors1,LDoctors),
                list_doctors_agenda(Day,LDoctors,LDAgendas),
                asserta(better_sol(Day,Room,Agenda,LDAgendas,Avg)).
```

Figure 9. Difference with the two solutions in ***update_better_sol/4***

We can see that there aren’t many differences between the way the two evaluation methods are called and used.

6. Conclusion

The report thoroughly examines the essential aspects of the project, from analyzing the basic code to implementing improvements, such as introducing heuristics and considering different phases and team members for each operation.

Initially, the study highlighted the impracticality of a brute-force approach for more complex schedules due to combinatorial explosion. To address this limitation, heuristics were implemented, prioritizing criteria such as early resource availability and minimizing the average occupation time of staff members. The introduction of a new evaluation function was pivotal in balancing the staff's and the rooms' schedules in a relatively good time frame.

The changes made to the original code, including the refinement of schedules and the adaptation to consider all phases of each operation, improved the functionality, with the potential for future optimizations based on more in-depth performance analyses.