RELATÓRIO ALGAV

Sprint B

3DB Grupo 9
Vicente Cardoso (1180664)
Tiago Sousa (1191583)
Ana Beatriz Costa (1201313)
Luís Reis (1210998)
Ricardo Ribeiro (1221695)

Índice

Conteúdo

Introdução 2

Código base 3

Agendas de Staff e Salas 3

Escalonar cirurgias 5

Melhor solução 7

Estudo da complexidade 10

Heurísticas 14

Primeira disponibilidade 14

Agenda mais ocupada 18

Adaptação para considerar todo o Staff e Fases 23

Conclusões 25

Anexos 26

Segunda Função de avaliação 26

Introdução

Este relatório foi desenvolvido no contexto da Unidade Curricular (UC) de Algoritmia Avançada (ALGAV), como parte integrante da Licenciatura em Engenharia Informática no ano letivo 2024-2025, utilizando PROLOG.

No corpo deste trabalho irá ser abordado o código base disponibilizado pelos docentes da UC, bem como o predicado desenvolvido para determinar o melhor escalonamento e o estudo da sua complexidade.

Além disto, serão apresentadas duas heurísticas e a comparação com a melhor solução previamente obtida.

A adaptação necessária para considerar todo o staff e as diferentes fases será também apresentada, assim como conclusões que se podem extrair deste projeto.

Em anexo, consta uma segunda função de avaliação para a obtenção do melhor resultado, baseada na ocupação do staff.

Código base

O código que foi disponibilizado pelos docentes da UC permite encontrar a melhor solução para a sequência de escalonamento de operações, a realizar numa determinada sala e dia. Pode ser dividido em diferentes secções, como será apresentado.

Agendas de Staff e Salas

Os predicados **free-agenda** permitem converter os intervalos de tempo em que um determinado *staff* ou sala estão ocupados em intervalos em que estes se encontram disponíveis.

```
free_agenda0([],[(0,1440)]).
free_agenda0([(0,Tfin,_)|LT],LT1):-!,free_agenda1([(0,Tfin,_)|LT],LT1).
free_agenda0([(Tin,Tfin,_)|LT],[(0,T1)|LT1]):- T1 is Tin-1,
  free_agenda1([(Tin,Tfin,_)|LT],LT1).

free_agenda1([(_,Tfin,_)],[(T1,1440)]):-Tfin\==1440,!,T1 is Tfin+1.
free_agenda1([(_,_,_)],[]).
free_agenda1([(_,T,_),(T1,Tfin2,_)|LT],LT1):-Tx is T+1,T1==Tx,!,
  free_agenda1([(T1,Tfin2,_)|LT],LT1).
free_agenda1([(_,Tfin1,_),(Tin2,Tfin2,_)|LT],[(T1,T2)|LT1]):-T1 is Tfin1+1,T2 is Tin2-1,
  free_agenda1([(Tin2,Tfin2,_)|LT],LT1).
```

O predicado **adapt_timetable** recorre ao horário (*timetable*) de um *staff*, ou seja, o intervalo de tempo em que este trabalha, desde a hora de entrada até à hora de saída. De seguida, remove os intervalos anteriores à sua entrada e posteriores à sua saída dos seus intervalos de disponibilidade.

Isto garante que não serão agendadas operações para esse membro do *staff* num horário em que ele não se encontra a trabalhar.

```
adapt_timetable(D,Date,LFA,LFA2):-
timetable(D,Date,(InTime,FinTime)),treatin(InTime,LFA,LFA1),treatfin(FinTime,LFA1,LFA2
).
```

```
treatin(InTime,[(In,Fin)|LFA],[(In,Fin)|LFA]):-InTime=<In,!.
treatin(InTime,[(_,Fin)|LFA],LFA1):-InTime>Fin,!,treatin(InTime,LFA,LFA1).
treatin(InTime,[(_,Fin)|LFA],[(InTime,Fin)|LFA]).
treatin(_,[],[]).

treatfin(FinTime,[(In,Fin)|LFA],[(In,Fin)|LFA1]):-
FinTime>=Fin,!,treatfin(FinTime,LFA,LFA1).
treatfin(FinTime,[(In,_)|_],[]):-FinTime=<In,!.
treatfin(FinTime,[(In,_)|_],[(In,FinTime)]).
treatfin(_,[],[]).
```

Por sua vez, o predicado **intersect_all_agendas** permite obter os intervalos de disponibilidade comuns entre diferentes doutores. Para tal, recorre ao **intersect_2_agendas** para procurar interseções entre duas listas de disponibilidade, que por sua vez utiliza o **intersect_availability** para comparar um intervalo específico com uma lista.

```
intersect_all_agendas([Name],Date,LA):-!,availability(Name,Date,LA).
intersect_all_agendas([Name|LNames],Date,LI):-
  availability(Name,Date,LA),
  intersect_all_agendas(LNames,Date,LI1),
  intersect_2_agendas(LA,LI1,LI).

intersect_2_agendas([],_,[]).
intersect_2_agendas([D|LD],LA,LIT):-  intersect_availability(D,LA,LI,LA1),
            intersect_2_agendas(LD,LA1,LID),
            append(LI,LID,LIT).

intersect_availability((_,_),[],[],[]).

intersect_availability((_,Fim),[(Ini1,Fim1)|LD],[],[(Ini1,Fim1)|LD]):-
    Fim<Ini1,!.
```

```
intersect_availability((Ini,Fim),[(_,Fim1)|LD],LI,LA):-
    Ini>Fim1,!,
    intersect_availability((Ini,Fim),LD,LI,LA).

intersect_availability((Ini,Fim),[(Ini1,Fim1)|LD],[(Imax,Fmin)],[(Fim,Fim1)|LD]):-
    Fim1>Fim,!,
    min_max(Ini,Ini1,_,Imax),
    min_max(Fim,Fim1,Fmin,_).

intersect_availability((Ini,Fim),[(Ini1,Fim1)|LD],[(Imax,Fmin)|LI],LA):-
    Fim>=Fim1,!,
    min_max(Ini,Ini1,_,Imax),
    min_max(Fim,Fim1,Fmin,_),
    intersect_availability((Fim1,Fim),LD,LI,LA).

min_max(I,I1,I,I1):- I<I1,!.
min_max(I,I1,I1,I).
```

Escalonar cirurgias

O predicado **schedule_all_surgeries** inicialmente remove as agendas temporárias existentes dos doutores e das salas e atualiza-as, de modo a não perder as informações originais. De seguida, calcula a disponibilidade do staff e armazena-a e obtém todas as cirurgias que necessitam de agendamento.

Por fim, chama o predicado **availability_all_surgeries** para tentar agendar as cirurgias.

```
schedule_all_surgeries(Room,Day):-
  retractall(agenda_staff1(_,_,_)),
  retractall(agenda_operation_room1(_,_,_)),
  retractall(availability(_,_,_)),
  findall(_,(agenda_staff(D,Day,Agenda),assertz(agenda_staff1(D,Day,Agenda))),_),
   agenda_operation_room(Or,Date,Agenda),assert(agenda_operation_room1(Or,Date,A
genda)),
```

```
    findall(_,(agenda_staff1(D,Date,L),free_agenda0(L,LFA),adapt_timetable(D,Date,LFA,
LFA2),assertz(availability(D,Date,LFA2))),_),
   findall(OpCode,surgery_id(OpCode,_),LOpCode),
   availability_all_surgeries(LOpCode,Room,Day),!.
```

Este predicado **availability_all_surgeries** é responsável por verificar a disponibilidade e agendar cada cirurgia.

Utiliza o predicado **availability_operation** que calcula a disponibilidade dos doutores realizarem uma cirurgia. Este método obtém o tipo e duração da cirurgia e os doutores atribuídos, a disponibilidade comum entre eles e com a sala. No final, remove os intervalos nos quais a cirurgia não teria tempo para ser realizada, com o predicado **remove_unf_intervals**.

O predicado **availability_all_surgeries** recorre ainda ao **schedule_first_interval** para escolher o primeiro intervalo disponível para agendar a cirurgia e ao **insert_agenda** para atualizar as agendas.

```
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).

remove_unf_intervals(_,[],[]).
remove_unf_intervals(TSurgery,[(Tin,Tfin)|LA],[(Tin,Tfin)|LA1]):-DT is Tfin-Tin+1,TSurgery=<DT,!,
  remove_unf_intervals(TSurgery,LA,LA1).
remove_unf_intervals(TSurgery,[_|LA],LA1):- remove_unf_intervals(TSurgery,LA,LA1).

schedule_first_interval(TSurgery,[(Tin,_)|_],(Tin,TfinS)):-
  TfinS is Tin + TSurgery - 1.

insert_agenda((TinS,TfinS,OpCode),[],[(TinS,TfinS,OpCode)]).
insert_agenda((TinS,TfinS,OpCode),[(Tin,Tfin,OpCode1)|LA],[(TinS,TfinS,OpCode),(Tin,Tfin,OpCode1)|LA]):-TfinS<Tin,!.
insert_agenda((TinS,TfinS,OpCode),[(Tin,Tfin,OpCode1)|LA],[(Tin,Tfin,OpCode1)|LA1]):-
insert_agenda((TinS,TfinS,OpCode),LA,LA1).

insert_agenda_doctors(_,_,[]).
insert_agenda_doctors((TinS,TfinS,OpCode),Day,[Doctor|LDoctors]):-
  retract(agenda_staff1(Doctor,Day,Agenda)),
  insert_agenda((TinS,TfinS,OpCode),Agenda,Agenda1),
  assert(agenda_staff1(Doctor,Day,Agenda1)),
  insert_agenda_doctors((TinS,TfinS,OpCode),Day,LDoctors).
```

Melhor solução

O predicado **obtain_better_sol** tem como função encontrar a melhor solução para o agendamento de cirurgias numa sala, num determinado dia. Para tal, recorre ao predicado **obtain_better_sol1** e por fim extrai a melhor solução.

```
obtain_better_sol(Room,Day,AgOpRoomBetter,LAgDoctorsBetter,TFinOp):-
    (obtain_better_sol1(Room,Day);true),
    retract(better_sol(Day,Room,AgOpRoomBetter,LAgDoctorsBetter,TFinOp)).
```

Este predicado **obtain_better_sol1** gera e avalia diferentes permutações de cirurgias de modo a encontrar a melhor solução.

Começa por inicializar o tempo final com o valor fora dos limites para um dia, obter todas as cirurgias e limpar as agendas temporárias. De seguida, tenta agendar todas as cirurgias recorrendo ao predicado **availability_all_surgeries** e atualiza a melhor solução, através do **update_better_sol**.

Este predicado mencionado anteriormente compara o tempo final da solução atual com o armazenado e, caso o atual seja inferior, atualiza.

```
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,LF
A2),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),
          FinTime1<FinTime,
          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)).
```

Como exemplo de um predicado utilizado pelos anteriormente apresentados temos o **evaluate_final_time** calcula o tempo final de uma operação.

```
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).
```

Por sua vez, o predicado **list_doctors_agenda** obtém a lista de disponibilidades para dados doutores num dia.

```
list_doctors_agenda(_,[],[]).
list_doctors_agenda(Day,[D|LD],[(D,AgD)|LAgD]):-
agenda_staff1(D,Day,AgD),list_doctors_agenda(Day,LD,LAgD).
```

Por fim, o predicado **remove_equals** remove elementos duplicados numa lista, de forma a garantir que não existam doutores repetidos associados à mesma cirurgia.

```
remove_equals([],[]).
remove_equals([X|L],L1):-member(X,L),!,remove_equals(L,L1).
remove_equals([X|L],[X|L1]):-remove_equals(L,L1).
```

Estudo da complexidade

Para o estudo da complexidade, recorreu-se a um algoritmo de ***brute force***, cuja função de avaliação encontra a melhor solução baseada no menor tempo final da última operação agendada.

```
brute_force(Room, Day):-
    retractall(counter(_,_)),
    assertz(counter('bruto', 0)),
    (obtain_better_sol2(Room,Day);true),
    retract(better_sol(Day,Room,AgOpRoomBetter,LAgDoctorsBetter,TFinOp)).
```

O predicado **obtain_better_sol2** atualiza as agendas do staff e da sala de operação e tenta atualizar a melhor solução.

```
obtain_better_sol2(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,LF
A2),assertz(availability(D,Day,LFA2))),_),
  availability_all_surgeries1(LOpCode,Room,Day),
  agenda_operation_room1(Room,Day,AgendaR),
    update_better_sol(Day,Room,AgendaR,LOpCode),
    fail.
```

Por sua vez, o predicado **update_better_sol** avalia a nova solução e atualiza a melhor solução, caso se encontre um menor valor final.

```
update_better_sol(Day,Room,Agenda,LOpCode):-
        retract(counter(Nome,Count)),
        CountNovo is Count +1,
        asserta(counter(Nome, CountNovo)),
        better_sol(Day,Room,_,_,FinTime),
        reverse(Agenda,AgendaR),
        evaluate_final_time(AgendaR,LOpCode,FinTime1),
        FinTime1<FinTime,
        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)).
```

A função de avaliação utilizada encontra então o tempo final de uma cirurgia.

```
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).
```

Assim, aplicando este algoritmo a diferentes números de cirurgias e com base nos dados disponíveis na Base de Conhecimento fornecida pelos docentes, construi-se a seguinte tabela.

Nº de cirurgias	**Nº de soluções**	**Melhor escalonamento de atividades na OR**	**Tempo final da última operação (min)**	**Tempo para gerar a solução (seg)**
3	6	[(480,539,so100001),(540,629,so100002),(630,704,so100003),(750,780,mnt0001),(1080,1110,mnt0002)]	704	0.022
4	24	[(520,594,so100003),(595,654,so100004),(655,744,so100002),(750,780,mnt0001),(791,850,so100001),(1080,1110,mnt0002)]	850	0.036

5	120	[(500,559,so100004),(560,634,so100005),(635,724,so100002),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(1080,1110,mnt0002)]	925	0.041
6	706	[(500,559,so100004),(560,634,so100005),(635,724,so100002),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(926,985,so100006),(1080,1110,mnt0002)]	985	0.16
7	3952	[(500,559,so100004),(560,649,so100007),(650,739,so100002),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(926,985,so100006),(986,1060,so100005),(1080,1110,mnt0002)]	1060	1.052
8	5536	[(500,589,so100002),(590,649,so100004),(650,709,so100008),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(926,985,so100006),(986,1075,so100007),(1080,1110,mnt0002),(1111,1185,so100005)]	1185	7.455
9	28986	[(480,539,so100001),(540,629,so100002),(630,689,so100008),(690,749,so100009),(750,780,mnt0001),(791,880,so100007),(881,940,so100006),(961,1020,so100004),(1080,1110,mnt0002),(1111,1185,so100005),(1186,1260,so100003)]	1260	52.232
10	32828	[(480,539,so100001),(540,599,so100004),(620,679,so100008),(680,739,so100009),(750,780,mnt0001),(791,880,so100007),(881,940,so100006),(981,1070,so100010),(1080,1110,mnt0002),(1111,1185,so100003),(1186,1260,so100005),(1261,1350,so100002)]	1350	523.927
11	8256	[(480,539,so100001),(540,629,so100007),(630,689,so100008),(690,749,so100009),(750,780,mnt0001),(791,880,so100010),(881,940,so100006),(941,1015,so100011),(1016,1075,so100004),(1080,1110,mnt0002),(1111,1185,so100003),(1186,1260,so100005),(1261,1350,so100002)]	1350	5923.791

A tabela apresentada mostra o melhor escalonamento de cirurgias, obtido pelo algoritmo desenvolvido, o número total de soluções válidas, o tempo final da última cirurgia agendada e o tempo que a solução demorou para ser determinada.

Heurísticas

Primeira disponibilidade

A primeira heurística a ser apresentada permite agendar cirurgias considerando a primeira disponibilidade do staff e das salas de operação.

Inicialmente o predicado **obtain_heuristic_solution** obtém a lista de todas as cirurgias que se pretende agendar, obtém as disponibilidades do staff e da sala e chama o predicado **heuristic**.

```
obtain_heuristic_solution(Room, Day):-
  findall(OpCode,surgery_id(OpCode,_),LOC),
  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,LF
A2),assertz(availability(D,Day,LFA2))),_),
  get_time(Ti),
  (heuristic(LOC, Day,Room);true),
  agenda_operation_room1(Room,Day, AgendaRoom),
        findall((D, AgendaDoctor) ,(agenda_staff1(D, Day, AgendaDoctor)),
_LAgDoctorsBetter),
  reverse(AgendaRoom, [ (_,_TFinOp,_) | _RESTOS]),
    evaluate_average_ocupied_time(Room, Day,_),
    evaluate_final_time(Room, Day).
```

Este por sua vez encontra a cirurgia que pode começar mais cedo através do predicado **heuristicaEarly**, marca-a na agenda da sala e do staff e remove-a das cirurgias a serem agendadas.

```
heuristic([],_,_).
heuristic(LOpCodes,Day,Room):-
```

```
heuristicaEarly(LOpCodes,Day,Room,EarliestOpCode,EarliestTime),
marcar_operacao(EarliestOpCode,Day,Room,EarliestTime),
delete(LOpCodes, EarliestOpCode, LOpCodes2),!,
heuristic(LOpCodes2,Day,Room),!.
```

O predicado **heuristicaEarly** compara então o horário possível de uma cirurgia com o melhor horário encontrado até ao momento, e a que puder iniciar mais cedo é agendada.

```
heuristicaEarly([OpCode|LOpCodes],Day, Room, EarliestOpCode, (S,F)):-
    heuristicaEarly(LOpCodes,Day, Room, OpCodeComp, (Str,End)),
    availability_operation1(OpCode,Room,Day,[(StrActual,EndActual)|_],_LStaff),
    ((StrActual<Str, EarliestOpCode=OpCode,S=StrActual,F=EndActual,!);
      (EarliestOpCode=OpCodeComp,S=Str,F=End)).
```

A tabela seguinte compara esta heurística com o algoritmo *brute force* apresentado anteriormente.

Nº de cirurgias	**Solução ótima**	**Tempo final da última operação - Melhor (min)**	**Tempo final da última operação - Heurística (min)**	**Tempo para gerar a solução - Melhor (seg)**	**Tempo para gerar a solução – Heurística (seg)**	**Solução heurística**
3	[(480,539,so100001),(540,629,so100002),(630,704,so100003),(750,780,mnt0001),(1080,1110,mnt0002)]	704	704	0.022	0.006	[(480,539,so100001),(540,614,so100003),(615,704,so100002),(750,780,mnt0001),(1080,1110,mnt0002)]
4	[(520,594,so100003),(595,654,so100004),(655,744,so100002),(750,780,mnt0001),(791,850,so100001),(1080,1110,mnt0002)]	850	1050	0.036	0.007	[(480,539,so100001),(540,599,so100004),(600,674,so100003),(750,780,mnt0001),(961,1050,so100002),(1080,1110,mnt0002)]
5	[(500,559,so100004),(560,634,so100005)	925	1050	0.041	0.006	[(480,539,so100001),(540,614,so100005),(615,674,so100004),(

	,(635,724,so100002),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(1080,1110,mnt0002)]					750,780,mnt0001),(791,865,so100003),(961,1050,so100002),(1080,1110,mnt0002)]
6	[(500,559,so100004),(560,634,so100005),(635,724,so100002),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(926,985,so100006),(1080,1110,mnt0002)]	985	1050	0.16	0.017	[(480,539,so100006),(540,614,so100005),(615,674,so100004),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(961,1050,so100002),(1080,1110,mnt0002)]
7	[(500,559,so100004),(560,649,so100007),(650,739,so100002),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(926,985,so100006),(986,1060,so100005),(1080,1110,mnt0002)]	1060	1200	1.052	0.007	[(480,539,so100006),(540,629,so100007),(630,704,so100005),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(961,1020,so100004),(1080,1110,mnt0002),(1111,1200,so100002)]
8	[(500,589,so100002),(590,649,so100004),(650,709,so100008),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(926,985,so100006),(986,1075,so100007),(1080,1110,mnt0002),(1111,1185,so100005)]	1185	1275	7.455	0.007	[(480,539,so100006),(540,629,so100007),(630,689,so100008),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(961,1020,so100004),(1080,1110,mnt0002),(1111,1185,so100005),(1186,1275,so100002)]
9	[(480,539,so100001),(540,629,so100002),(630,689,so100008),(690,749,so100009),(750,780,mnt0001),(791,880,so100007),	1260	689	52.232	0.005	[(480,539,so100006),(540,629,so100007),(630,689,so100009),(750,780,mnt0001),(1080,1110,mnt0002)]

	(881,940,so100006),(961,1020,so100004),(1080,1110,mnt0002),(1111,1185,so100005),(1186,1260,so100003)]					
10	[(480,539,so100001),(540,599,so100004),(620,679,so100008),(680,739,so100009),(750,780,mnt0001),(791,880,so100007),(881,940,so100006),(981,1070,so100010),(1080,1110,mnt0002),(1111,1185,so100003),(1186,1260,so100005),(1261,1350,so100002)]	1350	689	523.927	0.005	[(480,539,so100006),(540,629,so100010),(630,689,so100009),(750,780,mnt0001),(1080,1110,mnt0002)]
11	[(480,539,so100001),(540,629,so100007),(630,689,so100008),(690,749,so100009),(750,780,mnt0001),(791,880,so100010),(881,940,so100006),(941,1015,so100011),(1016,1075,so100004),(1080,1110,mnt0002),(1111,1185,so100003),(1186,1260,so100005),(1261,1350,so100002)]	1350	704	5923.791	0.005	[(480,554,so100011),(555,644,so100010),(645,704,so100009),(750,780,mnt0001),(1080,1110,mnt0002)]

Agenda mais ocupada

A segunda heurística realiza o agendamento com base no doutor com a agenda mais ocupada.

Inicialmente o predicado **obtain_heuristic_occupied_solution** prepara as agendas do staff e da sala, obtém todas as cirurgias que necessitam de agendamento e recorre ao predicado **heuristicaOcupied**. No fim, avalia o tempo médio de ocupação.

```
obtain_heuristic_occupied_solution(Room, Day):-
  findall(OpCode,surgery_id(OpCode,_),_LOC),
  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,LF
A2),assertz(availability(D,Day,LFA2))),_),
  findall(OpCode, surgery_id(OpCode,_), LOpCode),
  heuristicaOcupied(LOpCode,[], Room, Day),
  agenda_operation_room1(Room,Day,_AgendaRoom),!,
  findall(agenda_staff1(D,Day,AG), agenda_staff1(D,Day,AG), _LAgendaDoctor),
     evaluate_average_ocupied_time(Room, Day,_AVG),
    evaluate_final_time(Room, Day),
    !.
```

O predicado **heuristicaOcupied** utiliza a ocupação do staff para definir a agenda. Começa por calcular a ocupação de cada staff e de seguida ordená-los, priorizando os mais ocupados. Por fim, escolhe o médico com agenda mais preenchida e atribui-lhe uma nova cirurgia.

```
heuristicaOcupied([],_,_,_).
heuristicaOcupied(LOpCode,Visited,Room,Day):-
  retractall(doctor_occupied_time(_,_)),
```

```
findall(
    Doctor,
    (
      surgery_id(Sur,_),
      assignment_surgery(Sur, Doctor)
    ),
    LD
  ),
sort(LD, LDoctor),
doctor_occupied_time_calc(LDoctor, Day),nl,
findall((Doc, P),doctor_occupied_time(Doc, P), LDP),
sort(LDP, Temp),
select_most_ocupied_doctor(Temp, (D,_)),
findall(OpCode, assignment_surgery(OpCode,D),LC),
select_op_code_not_visited(LC, Visited, OpCode, _RestOpCodes),!,
(availability_all_surgeries2(OpCode,Room,Day);true),!,
delete(LOpCode, OpCode, LOpCodes2),
heuristicaOcupied(LOpCodes2,[OpCode | Visited],Room, Day).
```

A tabela seguinte compara esta segunda heurística com o algoritmo *brute force* apresentado anteriormente.

Nº de cirurgias	**Solução ótima**	**Tempo final da última operação - Melhor (min)**	**Tempo final da última operação - Heurística (min)**	**Tempo para gerar a solução - Melhor (seg)**	**Tempo para gerar a solução – Heurística (seg)**	**Solução heurística**
3	[(480,539,so100001),(540,629,so100002),(630,704,so100003),(750,780,mnt0001),(1080,1110,mnt0002)]	704	850	0.022	0.028	[(500,589,so100002),(590,664,so100003),(750,780,mnt0001),(791,850,so100001),(1080,1110,mnt0002)]
4	[(520,594,so100003),(595,654,so100004),(655,744,so100002),(750,780,m	850	865	0.036	0.024	[(480,539,so100001),(540,599,so100004),(600,689,so100002),(750,780,mnt0001),(791,865,so100003),(1080,1110,mnt0002)]

	nt0001),(791,850,so100001),(1080,1110,mnt0002)]					
5	[(500,559,so100004),(560,634,so100005),(635,724,so100002),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(1080,1110,mnt0002)]	925	1055	0.041	0.021	[(500,589,so100002),(590,649,so100004),(650,709,so100001),(750,780,mnt0001),(791,865,so100003),(981,1055,so100005),(1080,1110,mnt0002)]
6	[(500,559,so100004),(560,634,so100005),(635,724,so100002),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(926,985,so100006),(1080,1110,mnt0002)]	985	1055	0.16	0.030	[(480,539,so100001),(540,599,so100004),(600,659,so100006),(660,749,so100002),(750,780,mnt0001),(791,865,so100003),(981,1055,so100005),(1080,1110,mnt0002)]
7	[(500,559,so100004),(560,649,so100007),(650,739,so100002),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(926,985,so100006),(986,1060,so100005),(1080,1110,mnt0002)]	1060	1290	1.052	0.033	[(520,594,so100003),(595,669,so100005),(750,780,mnt0001),(791,880,so100007),(881,940,so100001),(961,1020,so100004),(1080,1110,mnt0002),(1141,1200,so100006),(1201,1290,so100002)]
8	[(500,589,so100002),(590,649,so100004),(650,709,so100008),(750,780,mnt0001),(791,865,so100003),(866,925,so100001),(926,985,so100006),(986,1075,so100007),(1080,1110,mnt000	1185	-	7.455	-	-

	2),(1111,1185,so100005)]					
9	[(480,539,so100001),(540,629,so100002),(630,689,so100008),(690,749,so100009),(750,780,mnt0001),(791,880,so100007),(881,940,so100006),(961,1020,so100004),(1080,1110,mnt0002),(1111,1185,so100005),(1186,1260,so100003)]	1260	-	52.232	-	-
10	[(480,539,so100001),(540,599,so100004),(620,679,so100008),(680,739,so100009),(750,780,mnt0001),(791,880,so100007),(881,940,so100006),(981,1070,so100010),(1080,1110,mnt0002),(1111,1185,so100003),(1186,1260,so100005),(1261,1350,so100002)]	1350	-	523.927	-	-
11	[(480,539,so100001),(540,629,so100007),(630,689,so100008),(690,749,so100009),(750,780,mnt0001),(791,880,so100010),(881,940,so100006),(941,1015,so100011),(1016,1075,so100004),(1080,1110,mnt0002),(1111,1185,so100003),(1186	1350	-	5923.791	-	-

	,1260,so100005),(1261,1350,so100002)]					

Com oito cirurgias verifica-se que não é possível associá-las ao mesmo staff, de forma que seriam atribuídas a outro elemento do staff. No entanto, uma vez que a heurística apenas fornece uma solução, não é possível observar esse escalonamento a outro staff e, portanto, concluir os seus valores.

Adaptação para considerar todo o Staff e Fases

Uma cirurgia é composta por três fases: anestesia, cirurgia propriamente dita e limpeza. Cada uma destas fases requer profissionais especializados, que atuam durante a respetiva fase, com exceção da anestesia que se mantém até ao final da fase intermédia.

De forma a considerar todos os tipos de staff necessários para uma cirurgia, nomeadamente anestesistas, cirurgiões e profissionais de limpeza, elaborou-se o predicado **marcar_operacao**, para a primeira heurística desenvolvida.

Este obtém o tempo total necessário para executar a operação, ou seja, a soma do tempo de anestesia, cirurgia e limpeza e reserva-o na agenda da sala.

De seguida, para cada fase, seleciona o staff correspondente através do predicado **get_staff_with_specialty_new** e atualiza as suas agendas.

```
marcar_operacao(OpCode,Day,Room,(Str,Fin)):-
  surgery_id(OpCode,OpType),surgery(OpType,TAnesthesia,TSurgery,TCleaning),
  TTotal is TAnesthesia+TSurgery+TCleaning,
  schedule_first_interval(TTotal,[(Str,Fin)],(TinS,TfinS)),
  retract(agenda_operation_room1(Room,Day,Agenda)),
  insert_agenda((TinS,TfinS,OpCode),Agenda,Agenda1),
  assertz(agenda_operation_room1(Room,Day,Agenda1)),
  get_staff_with_specialty_new(OpCode, anesthesia,LAnesthesia),
  _TinAnesthesia is TinS,
  TfinAnesthesia is TinS + TAnesthesia+TSurgery,
  insert_agenda_doctors((TinS,TfinAnesthesia,OpCode),Day,LAnesthesia),
  get_staff_with_specialty_new(OpCode, orthopaedist,LSurgery),
  TinSurgery is TinS+TAnesthesia,
  TfinSurgery is TinSurgery + TSurgery,
  insert_agenda_doctors((TinSurgery,TfinSurgery,OpCode),Day,LSurgery),
  get_staff_with_specialty_new(OpCode, cleaning,LCleaning),
  TinCleaning is TfinSurgery,
  _TfinCleaning is TfinS,
  insert_agenda_doctors((TinCleaning,TfinS,OpCode),Day,LCleaning).
```

Relativamente à segunda heurística desenvolvida, o predicado **sum_time_ops** atua de forma semelhante ao anterior. Assim, efetua a soma dos tempos de operações dependendo da especialidade.

```
sum_time_ops([], _, 0).
%Caso especialidade orthopaedist
sum_time_ops([(_TAnesthesia,TSurgery,_TCleaning)|LOps],orthopaedist,TimeOccupied)
:-
  sum_time_ops(LOps, orthopaedist, TimeOccupied1),
  TimeOccupied is TSurgery+TimeOccupied1.

%Caso especialidade anesthesia
sum_time_ops([(TAnesthesia,TSurgery,_TCleaning) | LOps], anesthesia, TimeOccupied):-
  sum_time_ops(LOps, anesthesia, TimeOccupied1),
  TimeOccupied is TAnesthesia+TSurgery+TimeOccupied1.

%Caso especialidade cleaning
sum_time_ops([(_TAnesthesia,_TSurgery,TCleaning) | LOps], cleaning, TimeOccupied):-
  sum_time_ops(LOps, cleaning, TimeOccupied1),
  TimeOccupied is TCleaning+TimeOccupied1.

% Caso Geral
sum_time_ops([(TAnesthesia, TSurgery, TCleaning) | LOps], _, TimeOccupied) :-
  sum_time_ops(LOps, other, TimeOccupied1),
  TimeOccupied is TAnesthesia + TSurgery + TCleaning + TimeOccupied1.
```

Conclusões

A elaboração deste trabalho permitiu obter diferentes resultados para o escalonamento de cirurgias, para uma determinada sala de operações e dia. A utilização de diversos algoritmos provocou diferentes resultados, que serão analisados de forma a concluir as diferenças entre eles.

Assim, é possível comparar as funções de avaliação desenvolvidas, ou seja, a função que considera que o melhor resultado é aquele em que a última operação termina mais cedo e a função em que o melhor resultado consiste na escolha do staff com menor ocupação média, presente no anexo.

Analisando as duas tabelas apresentadas, é possível concluir que o número de soluções válidas obtidas é o mesmo e o tempo necessário para as obter é muito semelhante. No entanto, a melhor solução e, portanto, o final da última operação agendada, são diferentes, sendo que para o algoritmo que avalia a ocupação média o escalonamento das cirurgias é feito de forma que estas terminem mais tarde.

Relativamente às heurísticas, é possível afirmar que ambas têm tempos de execução muito reduzidos, uma vez que apenas uma solução é encontrada. Assim, enquanto o algoritmo de *brute force* tem um crescimento acentuado relativamente ao aumento do número de cirurgias, as heurísticas apresentam um tempo relativamente constante.

Para além disto, é possível aferir que, uma vez que os algoritmos consideram diferentes padrões para realizar o escalonamento das cirurgias, o tempo final da última cirurgia varia consideravelmente.

Desta forma, conclui-se que este trabalho permitiu aprofundar o conhecimento da equipa relativamente a diferentes algoritmos de escalonamento, recorrendo a *brute force* e a heurísticas.

Anexos

Segunda Função de avaliação

Uma vez que a nossa equipa tem cinco elementos, foi requisitada uma segunda função de avaliação. Neste sentido, desenvolveu-se um predicado que avalia a média de tempo ocupado de cada staff.

```
brute_force2(Room, Day):-
   retractall(counter(_,_)),
   asserta(counter('bruto2', 0)),
   (obtain_better_sol2_2(Room,Day);true),
   retract(better_sol(Day,Room,_AgOpRoomBetter,_LAgDoctorsBetter,_TFinOp)).
```

O predicado **obtain_better_sol2_2** tenta encontrar o melhor escalonamento para agendar operações. Para tal, atualiza as agendas do staff e da sala e recorre ao predicado **update_better_sol_occupied_average**.

```
obtain_better_sol2_2(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,LF
A2),assertz(availability(D,Day,LFA2))),_),
   availability_all_surgeries1(LOpCode,Room,Day),
   agenda_operation_room1(Room,Day,AgendaR),
      update_better_sol_occupied_average(Day,Room,AgendaR),
      fail.
```

Este predicado **update_better_sol_occupied_average** atualiza a melhor solução com base na média de tempo ocupado pelo staff. Se a nova média de ocupação for menor, atualiza a melhor solução.

```
update_better_sol_occupied_average(Day,Room,Agenda):-
        retract(counter(Nome,Count)),
        CountNovo is Count +1,
        asserta(counter(Nome, CountNovo)),
        better_sol(Day,Room,_,_,AVG_Best_Current),
        evaluate_average_ocupied_time(Room, Day, AVG),
        AVG < AVG_Best_Current,
        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)).
```

Por fim, o predicado **evaluate_average_ocupied_time** calcula a média de tempo ocupado por todos os staffs.

```
evaluate_average_ocupied_time(_Room, Day, AVG):-
  findall((Doctor,Day),agenda_staff1(Doctor,Day, _Agenda), LAgendaDoctor),
  all_doctor_ocupied_time(LAgendaDoctor, SUM, NUM),
  AVG is SUM/NUM.
```

A tabela seguinte apresenta a análise deste segundo algoritmo de *brute force*.

Nº de cirurgias	**Nº de soluções**	**Melhor escalonamento de atividades na OR**	**Tempo final da última operação (min)**	**Tempo para gerar a solução (seg)**
3	6	[(480,539,so100001),(540,629,so100002),(630,704,so100003),(750,780,mnt0001),(1080,1110,mnt0002)]	704	0.043

4	24	[(480,539,so100001),(540,629,so100002),(630,704,so100003),(750,780,mnt0001),(961,1020,so100004),(1080,1110,mnt0002)]	1020	0.051
5	120	[(480,539,so100001),(540,629,so100002),(630,704,so100003),(750,780,mnt0001),(961,1020,so100004),(1080,1110,mnt0002),(1111,1185,so100005)]	1185	0.061
6	706	[(480,539,so100001),(540,629,so100002),(630,704,so100003),(750,780,mnt0001),(791,850,so100006),(961,1020,so100004),(1080,1110,mnt0002),(1111,1185,so100005)]	1185	0.150
7	3952	[(480,539,so100001),(540,629,so100002),(630,704,so100003),(750,780,mnt0001),(791,850,so100006),(961,1020,so100004),(1080,1110,mnt0002),(1111,1185,so100005),(1186,1275,so100007)]	1275	0.831
8	5536	[(480,539,so100001),(540,629,so100002),(630,689,so100008),(750,780,mnt0001),(791,865,so100003),(866,925,so100006),(961,1020,so100004),(1080,1110,mnt0002),(1111,1185,so100005),(1186,1275,so100007)]	1275	5.825
9	28986	[(480,539,so100001),(540,629,so100002),(630,689,so100008),(690,749,so100009),(750,780,mnt0001),(791,865,so100003),(866,925,so100006),(961,1020,so100004),(1080,1110,mnt0002),(1111,1185,so100005),(1186,1275,so100007)]	1275	51.075
10	32828	[(480,539,so100001),(540,614,so100003),(620,679,so100008),(680,739,so100009),(750,780,mnt0001),(791,880,so100007),(881,940,so100006),(961,1020,so100004),(1080,1110,mnt0002),(1111,1185,so100005),(1186,1275,so100010),(1276,1365,so100002)]	1365	523.905
11	8256	[(480,539,so100001),(540,614,so100003),(620,679,so100008),(680,739,so100009),(750,780,mnt0001),(791,880,so100007),(881,940,so100006),(941,1015,so100011),(1016,1075,so100004),(1080,1110,mnt0002),(1111,1185,so100005),(1186,1275,so100010),(1276,1365,so100002)]	1365	5815.406