



Instituto Superior de  
**Engenharia** do Porto

## **Relatório ALGAV Sprint 2&3**

Turma 3DD - Grupo 23

**Aluno:**

Rafael Brandao, 1220879

**Professor:**

Francisco da Silva, FPS

**Unidade Curricular:**

Algoritmia Avançada

## Índice

Sprint 2.....	2
US 6.3.1 .....	2
US 6.3.2 .....	7
Sprint 3.....	9
<b>Introdução .....</b>	<b>9</b>
<b>Aleatoriedade no cruzamento entre indivíduos da população.....</b>	<b>9</b>
<b>Seleção da nova geração da população garantindo que pelo menos o melhor indivíduo entre a população atual e os seus descendentes passe para a próxima geração .....</b>	<b>10</b>
<b>Parametrização da condição de término .....</b>	<b>12</b>
<b>Adaptação do Algoritmo Genético para o problema do Escalonamento de Cirurgias a Blocos de Operação de Hospitais .....</b>	<b>13</b>
<b>Consideração de vários blocos de operação, com um método de atribuição das operações às salas .....</b>	<b>14</b>
<b>Conclusão.....</b>	<b>16</b>

## Sprint 2

### US 6.3.1

**Requisitos:** *“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.”*

O código fornecido implementa um sistema robusto para o escalonamento de cirurgias em salas de operações, considerando as agendas dos médicos e a disponibilidade das salas de cirurgia.

Sendo assim, o funcionamento deste sistema pode ser descrito da seguinte forma:

#### Definição dos Dados Base:

##### 1. Agendas dos Médicos:

- O predicado `agenda_staff/3` especifica os horários ocupados para cada médico em um determinado dia. Cada entrada indica intervalos de tempo já reservados para atividades específicas.
- O predicado `timetable/3` define o horário de trabalho permitido para cada médico em um determinado dia.
- O predicado `staff/3` representa as informações específicas dos trabalhadores do hospital, como a função (doutor, enfermeiro, técnico) e as suas especialidades

```
agenda_staff(d001,20241028,[(720,790,m01),(1080,1140,c01)]).
agenda_staff(d002,20241028,[(850,900,m02),(901,960,m02),(1380,1440,c02)]).
agenda_staff(d003,20241028,[(720,790,m01),(910,980,m02)]).
agenda_staff(d004,20241028,[(850,900,m02),(940,980,c04)]).

timetable(d001,20241028,(480,1200)).
timetable(d002,20241028,(500,1440)).
timetable(d003,20241028,(520,1320)).
timetable(d004,20241028,(620,1020)).

staff(d001,doctor,orthopaedist,[so2,so3,so4]).
staff(d002,doctor,orthopaedist,[so2,so3,so4]).
staff(d003,doctor,orthopaedist,[so2,so3,so4]).
staff(d004,doctor,orthopaedist,[so2,so3,so4]).

%doctor,nurse, tectinician
```

##### 2. Cirurgias e Associações:

- O predicado `surgery/4` descreve os tempos necessários para anestesia, cirurgia e limpeza para cada tipo de cirurgia.
- A relação entre tipos de cirurgia e cirurgias específicas é definida pelo predicado `surgery_id/2`.

- O predicado `assignment_surgery/2` associa médicos às cirurgias que podem realizar.

```
%surgery(SurgeryType,TAnesthesia,TSurgery,TCleaning).

surgery(so2,45,60,45).
surgery(so3,45,90,45).
surgery(so4,45,75,45).

surgery_id(so100001,so2).
surgery_id(so100002,so3).
surgery_id(so100003,so4).
surgery_id(so100004,so2).
surgery_id(so100005,so4).

%surgery_id(so100006,so2).
%surgery_id(so100007,so3).
%surgery_id(so100008,so2).
%surgery_id(so100009,so2).
%surgery_id(so100010,so2).
%surgery_id(so100011,so4).
%surgery_id(so100012,so2).
%surgery_id(so100013,so2).

assignment_surgery(so100001,d001).
assignment_surgery(so100002,d002).
assignment_surgery(so100003,d003).
assignment_surgery(so100004,d001).
assignment_surgery(so100004,d002).
assignment_surgery(so100005,d002).

%assignment_surgery(so100005,d003).
%assignment_surgery(so100006,d001).
%assignment_surgery(so100007,d003).
%assignment_surgery(so100008,d004).
%assignment_surgery(so100008,d003).
%assignment_surgery(so100009,d002).
%assignment_surgery(so100009,d004).
%assignment_surgery(so100010,d003).
%assignment_surgery(so100011,d001).
%assignment_surgery(so100012,d001).
%assignment_surgery(so100013,d004).
```

### 3. Ocupação das Salas de Operação:

- O predicado `agenda_operation_room/3` lista os horários em que uma sala de operação já está ocupada num dia específico.

```
agenda_operation_room(or1,20241028,[ (520,579,so100000),(1000,1059,so099999) ]).
```

## Gestão das Agendas:

### 1. Identificação de Intervalos Livres:

- `free_agenda0/2` processa as agendas ocupadas para determinar os intervalos de tempo disponíveis.

- Caso as agendas não possuam horários é assumido que o horário livre é o dia todo e caso contrário usa free\_agenda1/2 para calcular os horários livres a partir do primeiro horário ocupado.
- adapt\_timetable/4 ajusta os intervalos livres para coincidir com as agendas de cada médico e operações sem sobreposição. Adicionalmente, usamos treatin/3 e treatfin/3 de modo a verificar se o horário de entrada ou de fim de uma operação também ser colocado na agenda.

```

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

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(_,[],[]).

```

## 2. Interseção de Agendas:

- intersect\_all\_agendas/3 combina as disponibilidades de diferentes médicos para encontrar horários em comum.
- intersect\_2\_agendas/3 calcula a interseção entre duas agendas específicas, refinando os horários disponíveis.

```

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,LIT,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).

```

## Agendamento de Cirurgias:

### 1. Marcação de Cirurgias:

- O predicado principal, `schedule_all_surgeries/2`, realiza o escalonamento de todas as cirurgias para uma sala de operação num dia específico.
- Este predicado copia as agendas iniciais, calcula os intervalos livres para médicos e salas (`availability_all_surgeries/3`), e insere as cirurgias em horários válidos.

```
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,Agenda)),
    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),!.

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).
```

### 2. Determinação de Horários Disponíveis:

- `availability_operation/5` avalia os horários possíveis para uma cirurgia, considerando a disponibilidade de médicos/salas.

```
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).
```

### 3. Atualização das Agendas:

- `insert_agenda/3` insere uma cirurgia nas agendas dos médicos e/ou salas.
- `insert_agenda_doctors/3` insere o horário da mesma cirurgia nas agendas dos médicos lhe designados.

```
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)|LA1]):-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).
```

## Busca pela Melhor Solução:

### 1. Minimização do Tempo Total:

- O predicado `obtain_better_sol/5` encontra a melhor sequência de cirurgias de modo a minimizar o tempo total necessário para as realizar.
- `permutation/2` é usado para gerar todas as possíveis ordens de execução das cirurgias.
- `evaluate_final_time/3` calcula o final da última cirurgia em cada sequência, o que permite a comparação de soluções.

```
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.
```

### 2. Atualização da Melhor Solução:

- O predicado `update_better_sol/5` armazena a solução atual caso seja melhor do que a anterior, tendo em consideração o menor tempo total para as operações.

```
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)).

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).

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

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

## US 6.3.2

**Requisitos:** “As an Admin, I want to know till what dimension in terms of number of surgeries is possible to ask for the better solution.”

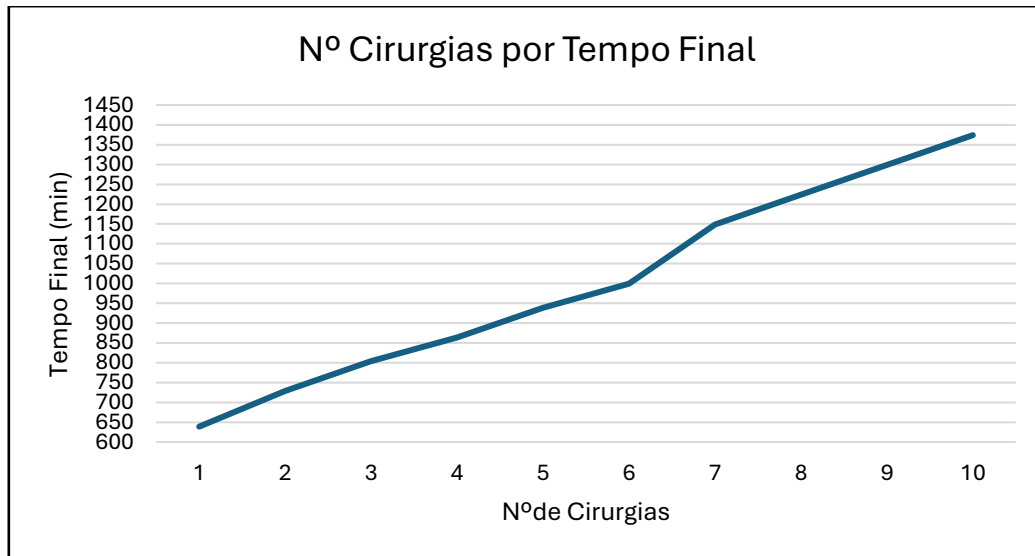
Resultados de uma análise de complexidade do código base:

Nº de cirurgias	Melhor Resultado	Tempo Final (min)	Tempo de geração
1	[(520, 579, so100000), (580, 639, so100001), (1000, 1059, so0999999)]	639	0.012403964
2	[(520, 579, so100000), (580, 639, so100001), (640, 729, so100002), (1000, 1059, so0999999)]	729	0.020236968
3	[(520, 579, so100000), (580, 639, so100001), (640, 714, so100003), (715, 804, so100002), (1000, 1059, so0999999)]	804	0.044281005
4	[(520, 579, so100000), (580, 654, so100003), (655, 714, so100004), (715, 804, so100002), (805, 864, so100001), (1000, 1059, so0999999)]	864	0.290410995
5	[(520, 579, so100000), (580, 639, so100004), (640, 714, so100005), (715, 804, so100002), (805, 879, so100003), (880, 939, so100001), (1000, 1059, so0999999)]	939	0.349401951
6	[(520, 579, so100000), (580, 639, so100004), (640, 714, so100005), (715, 804, so100002), (805, 879, so100003), (880, 939, so100001), (940, 999, so100006), (1000, ..., ...)]	999	2.22795701
7	[(520, 579, so100000), (580, 639, so100004), (640, 714, so100005), (715, 804, so100002), (805, 879, so100003), (880, 939, so100001), (940, 999, so100006), (1000, ..., ...), (... , ...)]	1149	6.533479915
8	[(520, 579, so100000), (580, 639, so100004), (640, 699, so100008), (700, 789, so100002), (791, 865, so100003), (866, 925, so100001), (926, 985, so100006), (1000, ..., ...), (... , ...)]...	1224	13.66150604
9	[(520, 579, so100000), (580, 639, so100004), (640, 699, so100008), (700, 789, so100002), (790, 849, so100009), (850, 909, so100001), (910, 969, so100006), (1000, ..., ...), (... , ...)]...	1299	48.07397785
10	[(520, 579, so100000), (580, 639, so100004), (640, 699, so100008), (700, 759, so100009), (791, 865, so100003), (866, 925, so100001), (926, 985, so100006), (1000, ..., ...), (... , ...)]...	1374	388.55271291



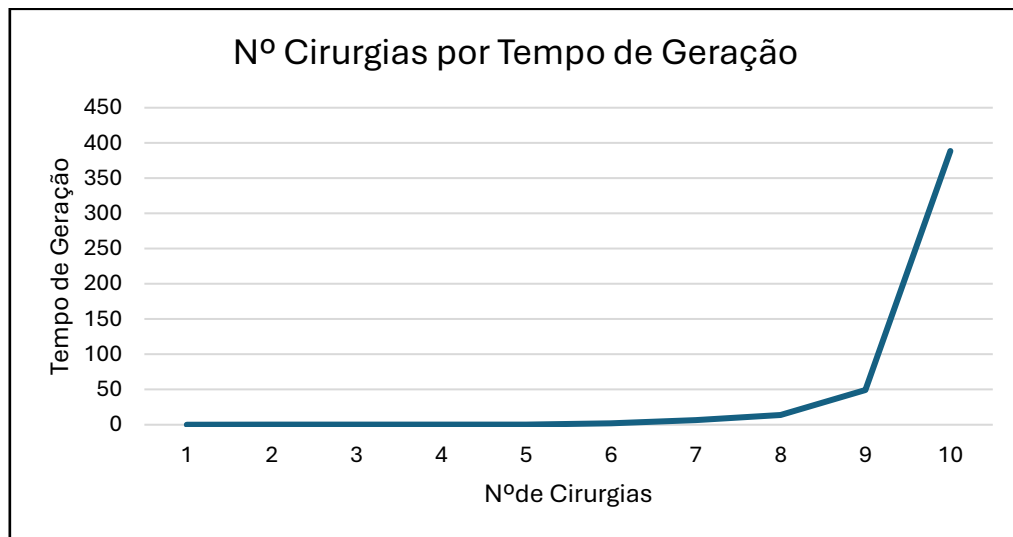
## Análise da Complexidade

### 1. Tempo Final:



- O **tempo final** representa a eficiência do algoritmo a ordenar as cirurgias de forma a minimizar o tempo total necessário para as completar.
- Observa-se um **crescimento linear** do tempo final, ou seja, o algoritmo consegue gerar soluções eficientes mesmo com o aumento do número de cirurgias.

### 2. Tempo de Geração da Solução:



- O **tempo de geração** dita o tempo necessário para encontrar a melhor solução.
- Aqui, o crescimento é **exponencial**, o que é consistente com a abordagem do algoritmo, pois utiliza permutações para avaliar todas as possíveis ordens das cirurgias.

- Podemos concluir que a complexidade do algoritmo é  $O(N! \times N)$ , onde  $N!$  é o número de permutações para  $N$  cirurgias  $N$ , onde se calcula o o tempo total de cada cirurgia.

Assim sendo, conseguimos perceber que a partir de 10 cirurgias o tempo de processamento torna-se excessivamente grande, o que remove a viabilidade deste algoritmo.

## Sprint 3

### Introdução

O código apresentado implementa um **Algoritmo Genético (AG)** aplicado ao problema do escalonamento de cirurgias em blocos operatórios de hospitais. O objetivo é otimizar a alocação de cirurgias a recursos hospitalares (como salas de operações e equipas médicas), respeitando restrições de tempo, penalizações associadas às cirurgias e agendas de disponibilidade.

O AG simula processos biológicos, como cruzamento, mutação e seleção natural, para explorar o espaço de soluções e encontrar resultados viáveis e potencialmente ótimos para o problema.

### Aleatoriedade no cruzamento entre indivíduos da população

O cruzamento é realizado de forma aleatória, conforme implementado na função `crossover/3`. Assim sendo, podemos analisar o código da seguinte forma:

- **Randomização no comportamento de cruzamento:** Inicialmente, é gerado um número aleatório de modo a decidir ocorre cruzamento ( $P_c \leq P_{cruz}$ , onde  $P_c$  é a probabilidade gerada aleatoriamente, e  $P_{cruz}$  é a taxa de cruzamento definida pelo utilizador). Caso contrário, os indivíduos permanecem inalterados.
- **Preservação da integridade dos indivíduos:** Após o cruzamento, elementos duplicados são removidos e as listas resultantes são ajustadas, garantindo que os indivíduos gerados sejam válidos para o problema.
- **Seleção de pontos de cruzamento aleatórios:** A função `generate_crossover_points/2` define dois pontos de corte aleatórios dentro do intervalo permitido (número de cirurgias). Estes pontos determinam as secções dos indivíduos que serão trocadas.

```

crossover([Ind1*_ , Ind2*_|Rest], [NInd1, NInd2|Rest1]) :-
    random(0, 2, Rand),

    length(Rest, Len),
    (Len > 0 ->
        random(0, Len, RandomIndex),
        nth0(RandomIndex, Rest, NextInd*_ )
        ; NextInd = Ind2
    ),

    (Rand == 0 ->
        generate_crossover_points(P1, P2),
        prob_crossover(Pcruz),
        random(0.0, 1.0, Pc),
        (Pc <= Pcruz ->
            cross(Ind1, NextInd, P1, P2, NInd1),
            cross(NextInd, Ind1, P1, P2, NInd2)
        ;
            NInd1 = Ind1, NInd2 = NextInd
        )
    ;
        NInd1 = Ind1, NInd2 = Ind2
    ),
    crossover(Rest, Rest1).

```

## Seleção da nova geração da população garantindo que pelo menos o melhor indivíduo entre a população atual e os seus descendentes passe para a próxima geração

A seleção para a próxima geração combina aspetos elitistas e não elitistas:

- **Seleção de método:** Inicialmente, no predicado `initialize/0` o usuário é questionado quanto ao método de seleção que irá querer utilizar (puramente elitista ou não elitista).

```

write('Selection method (elitist/non-elitist):'), read(SM), (SM = elitist ; SM = non-elitist),
(retract(selection_method(_));true), asserta(selection_method(SM)),

```

- **Preservação do melhor indivíduo:** Na função `generate_generation/5` o melhor valor da população atual é identificado e comparado ao melhor valor global e, caso seja menor são chamados os métodos `elitist_method/3` ou `non_elitist_method/3` para o adicionar à população.

```

% Current and New population
NPopOrd = [_NewBest*NewBestValue|_],
(NewBestValue @=< BestValue
-> FinalPop = NPopOrd,
    (BestValue @=< ValidValue -> N1 is N + 1 ; N1 is 0),
    nl, write('NewBest value:'), write(NewBestValue), nl
;
    selection_method(Method),
    (Method = elitist ->
        elitist_method(Best*BestValue, NPopOrd, FinalPop)
    ;
        non_elitist_method(Best*BestValue, NPopOrd, FinalPop)
    ),
    N1 is 0,
    nl, write('NewBest value:'), write(NewBestValue), nl
),

```

- **Método puramente elitista:** Quando queremos um método elitista simplesmente adicionamos o melhor valor anterior à população, efetuado a partir do método `add_best/3`.

```
% Elitist selection method
elitist_method(Best*BestValue, Population, FinalPopulation) :-
    add_best(Best*BestValue, Population, FinalPopulation).
```

```
add_best(Best*BestValue, Population, FinalPopulation) :-
    append(Front, [_|Rest], Population),
    append(Front, Rest, TempPopulation),
    append([Best*BestValue], TempPopulation, FinalPopulation).
```

- **Método não puramente elitista:** Quando o método não é elitista, randomizamos um valor entre 0 e 1, de modo a escolher entre um método similar ao elitista (a partir de `add_best/3`) ou um procedimento de seleção por torneio. Nesse caso:
  - São escolhidos dos valores ao acaso da população (`tournament_selection/2`).
  - Randomizamos de novo um número de 0 a 1 de modo a avaliar se iremos queremos o pior dos valores (20% de chance) ou o melhor e adicionamos o escolhido na próxima geração.
  - Procedemos com este processo até se formar uma nova população.

```
% Non-elitist selection method
non_elitist_method(Best*BestValue, Population, FinalPopulation) :-
    random(0.0, 1.0, R), % Generate random number between 0 and 1
    (R < 0.5 ->
        % Keep the best individual
        add_best(Best*BestValue, Population, FinalPopulation)
    );
    % Tournament selection
    tournament_selection(Population, BestSelected, WorstSelected),
    random(0.0, 1.0, R2),
    % 20% chance to keep the worst individual
    (R2 < 0.2 ->
        Selected = WorstSelected % Select worst from tournament
    );
    Selected = BestSelected % Select best from tournament
    ),
    add_best(Selected, Population, FinalPopulation)
).

% Tournament selection for non-elitist approach
tournament_selection(Population, BestSelected, WorstSelected) :-
    % Get random individuals for comparison
    length(Population, PopLen),
    random(0, PopLen, Indx1),
    random(0, PopLen, Indx2),
    nth0(Indx1, Population, Ind1*Val1),
    nth0(Indx2, Population, Ind2*Val2),
    % Select the better and worse individual
    (Val1 =< Val2 ->
        (BestSelected = Ind1*Val1, WorstSelected = Ind2*Val2)
    );
    (BestSelected = Ind2*Val2, WorstSelected = Ind1*Val1)
).
```

## Parametrização da condição de término

Para além do número máximo de gerações que já tinha sido dado, o código permite outras condições de término configuráveis, sendo estes inicialmente pedidos ao usuário no predicado initialize/0.

```
initialize:-
write('Number of generations: '),read(NG),
(retract(generations(_));true), asserta(generations(NG)),

write('Population size: '),read(PS),
(retract(population(_));true), asserta(population(PS)),

write('Probability of crossover (%):'), read(P1), PC is P1/100,
(retract(prob_crossover(_));true), asserta(prob_crossover(PC)),

write('Probability of mutation (%):'), read(P2), PM is P2/100,
(retract(prob_mutation(_));true), asserta(prob_mutation(PM)),

write('Selection method (elitist/non-elitist):'), read(SM), (SM = elitist ; SM = non-elitist),
(retract(selection_method(_));true), asserta(selection_method(SM)),

% Stop conditions
write('Maximum runtime (seconds): '),read(Time),
(retract(max_time(_));true), asserta(max_time(Time)),

write('Best value to stop:'), read(ValidValue),
(retract(value_limit(_));true), asserta(value_limit(ValidValue)),

write('Number of generations for pop stability:'), read(SG),
(retract(stability_generations(_));true), asserta(stability_generations(SG)).
```

Após a decisão, voltamos a verificar as condições de término no predicado generate\_generation/6, onde se verificam as seguintes condições:

- **Tempo máximo de execução:** A condição `ElapsedTime > MaxTime` termina a execução caso o tempo total de execução exceda o limite especificado pelo utilizador. Isto evita execuções excessivamente longas.

```
generate_generation(_, _, _, Pop, Counter, StartTime, MaxTime) :-
get_time(CurrentTime),
ElapsedTime is CurrentTime - StartTime,
ElapsedTime > MaxTime, !, % Stop condition if elapsed time exceeds MaxTime
write('Runtime limit met.'), nl,
write('Final Generation '), write(Counter), write(':'), nl, write(Pop), nl.
```

- **Número de Gerações:** Esta condição termina a execução do sistema quando se verifica que a geração atual ultrapassa o número de gerações total introduzido.

```
generate_generation(G, G, _, Pop, Counter, _, _) :- !,
nl, write('Final Generation '), write(Counter), write(':'), nl, write(Pop), nl.
```

- **Valor-alvo:** Uma condição adicional termina o sistema se o valor da melhor solução encontrada for menor ou igual a um limite pré-definido (`BestValue =< ValidValue`).

```
; (BestValue =< ValidValue ->
write('Value limit met at value'), write(BestValue), nl
```

- **Estabilidade da população:** Finalmente, também paramos a execução do programa se a população estabilizar/manter-se por um número consecutivo de gerações escolhidas (StabilizationCounter >= MaxStabilizationGenerations).

```
; (N1 >= MaxStabilizationGenerations ->
write('Population stabilized at '), write(MaxStabilizationGenerations), write(' generations'),
nl, write('Final Population: '), write(FinalPop), nl
```

Estas condições fornecem maior flexibilidade e permitem ajustar o algoritmo aos seus problemas específicos.

## Adaptação do Algoritmo Genético para o problema do Escalonamento de Cirurgias a Blocos de Operação de Hospitais

O código está adaptado ao problema do escalonamento de cirurgias a partir das seguintes características:

- **Estrutura hospitalar:** Modificamos o algoritmo de modo a considerar a estrutura hospitalar (de cirurgias) fornecida em vez de usar tasks.

```
agenda_staff(d001,20241028,[(720,790,m01),(1080,1140,c01)]).
agenda_staff(d002,20241028,[(850,900,m02),(901,960,m02),(1380,1440,c02)]).
agenda_staff(d003,20241028,[(720,790,m01),(910,980,m02)]).
agenda_staff(d004,20241028,[(850,900,m02),(940,980,c04)]).

timetable(d001,20241028,(480,1200)).
timetable(d002,20241028,(500,1440)).
timetable(d003,20241028,(520,1320)).
timetable(d004,20241028,(620,1020)).

staff(d001,doctor,orthopaedist,[so2,so3,so4]).
staff(d002,doctor,orthopaedist,[so2,so3,so4]).
staff(d003,doctor,orthopaedist,[so2,so3,so4]).
staff(d004,doctor,orthopaedist,[so2,so3,so4]).

%surgery(SurgeryType,TAnesthesia,TSurgery,TCleaning).
surgery(so2,45,60,45).
surgery(so3,45,90,45).
surgery(so4,45,75,45).

surgery_id(so100001,so2).
surgery_id(so100002,so3).
surgery_id(so100003,so4).
surgery_id(so100004,so2).
surgery_id(so100005,so4).

assignment_surgery(so100001,d001).
assignment_surgery(so100002,d002).
assignment_surgery(so100003,d003).
assignment_surgery(so100004,d001).
assignment_surgery(so100004,d002).
assignment_surgery(so100005,d002).

agenda_operation_room(or1,20241028,[(520,579,so100000),(1000,1059,so099999)]).
agenda_operation_room(or2,20241028,[]).
agenda_operation_room(or3,20241028,[]).
```

- **Função de avaliação:** A qualidade de cada indivíduo é medida pela função `evaluate/3`, que considera:
  - O tempo total das cirurgias agendadas (`surgery_time/2`).
  - Penalizações associadas a cirurgias específicas (`surgery_penalty/3`).
  - Penalizações adicionais caso o limite de tempo seja excedido.

```
evaluate([Surgery|Rest], TotalTime, V) :-
    value_limit(ValidValue),
    surgery_penalty(Surgery, SurgeryName, Penalty),
    surgery_time(SurgeryName, SurgeryTime),

    NewTotalTime is TotalTime + SurgeryTime,

    ( NewTotalTime =< ValidValue ->
        evaluate(Rest, NewTotalTime, VRest),
        V is NewTotalTime + VRest + Penalty
    );
    ExceedPenalty is Penalty + (NewTotalTime - ValidValue),
    evaluate(Rest, NewTotalTime, VRest),
    V is NewTotalTime + VRest + ExceedPenalty
).

% Calculate the total time of a surgery
surgery_time(SurgeryName, TotalTime) :-
    surgery(SurgeryName, Time1, Time2, Time3),
    TotalTime is Time1 + Time2 + Time3.
```

## Consideração de vários blocos de operação, com um método de atribuição das operações às salas

Neste caso foi utilizado um diferente código prolog, com base no desenvolvido no sprint anterior, em que foram acrescentados métodos que se focam no agendamento de cirurgias e numa distribuição eficiente destas por várias salas de operação (blocos operatórios) a partir de um sistema de escalonamento

Assim sendo, o método principal de atribuição das operações às salas encontra-se no predicado `schedule_surgeries_by_room/1`, onde se efetua a preparação para agendamento:

- Através de `retracts` e `asserts`, o sistema limpa e prepara as agendas existentes para médicos (`agenda_staff1`) e salas de operação (`agenda_operation_room1`).
- As agendas dos médicos e salas são transformadas em intervalos de disponibilidade através de funções como `free_agenda0` e `adapt_timetable`.

```

schedule_surgeries_by_room(Day) :-
    % initialization of memory
    retractall(agenda_staff1(_,_,_)),
    retractall(agenda_operation_room1(_,_,_)),
    retractall(availability(_,_,_)),

    % Reinitialize staff and operation room agendas
    findall(_, (agenda_staff(D,Day,Agenda), assertz(agenda_staff1(D,Day,Agenda))), _),
    findall(_, (agenda_operation_room(Room,Day,Agenda), assert(agenda_operation_room1(Room,Day,Agenda))), _),

    % Prepare staff availability
    findall(_, (agenda_staff1(D,Day,L), free_agenda0(L,LFA), adapt_timetable(D,Day,LFA,LFA2), assertz(availability(D,Day,LFA2))), _),

    findall(Room, agenda_operation_room(Room, Day, _), LRooms),
    write('Rooms: '), write(LRooms), nl,
    findall(SurgeryId, surgery_id(SurgeryId, _), Surgeries),
    write('Surgeries: '), write(Surgeries), nl,
    room_distribution(Surgeries, LRooms, Day).

```

De seguida, é efetuada uma chamada à lógica principal para a atribuição às salas (room\_distribution/3), onde:

- As cirurgias são listadas e atribuídas sequencialmente às salas, percorrendo-as de forma circular.
- Para cada cirurgia, verifica-se a sala corrente em LRooms (usando o índice do round-robin).
- A cirurgia é programada na sala correspondente, desde que existam intervalos disponíveis compatíveis.

```

% Distribute surgeries to rooms
room_distribution([], _, _).
room_distribution([SurgeryId|Rest], LRooms, Day, RoomIndex) :-
    length(LRooms, NumRooms),
    (NumRooms > 0 ->
        CurrentRoomIndex is RoomIndex mod NumRooms,
        nth0(CurrentRoomIndex, LRooms, CurrentRoom),

        % Find surgery details
        surgery_id(SurgeryId, OpType),
        surgery(OpType, _, TSurgery, _),

        % Find possible doctors for the surgery
        findall(Doctor, assignment_surgery(SurgeryId, Doctor), LDoctors),

        % Try to schedule surgery in current room
        (surgery_by_room(SurgeryId, CurrentRoom, TSurgery, Day, LDoctors) -> true ; true),

        % Continue with next surgery and next room
        room_distribution(Rest, LRooms, Day, RoomIndex + 1)
    );
    write('Error: No rooms available'), nl,
    fail
).

```

Finalmente, é chamado o predicado surgery\_by\_room/5, onde se realiza o agendamento de uma só cirurgia em uma sala específica, ou seja, ele:

- Identifica as disponibilidades comuns entre os médicos necessários (intersect\_all\_agendas) e a sala atual (free\_agenda0).
- Calcula os intervalos disponíveis através da interseção entre as disponibilidades dos médicos e da sala.



- Remove os intervalos incompatíveis (remove\_unf\_intervals) e seleciona o primeiro intervalo viável para a cirurgia.
- Atualiza as agendas da sala e dos médicos envolvidos e mostra o resultado.

```
% Schedule a single surgery
surgery_by_room(SurgeryId, Room, Duration, Day, Doctors) :-
    write('Trying to schedule surgery '), write(SurgeryId), write(' in room '), write(Room), nl,

    % Find availability for doctors and room
    intersect_all_agendas(Doctors, Day, LA),
    agenda_operation_room1(Room, Day, LAgenda),
    free_agenda0(LAgenda, LFAgRoom),
    intersect_2_agendas(LA, LFAgRoom, LIntAgDoctorsRoom),
    remove_unf_intervals(Duration, LIntAgDoctorsRoom, LPossibilities),

    % If possible intervals exist, schedule the surgery
    (LPossibilities = [(Start, _)]_ ->
        End is Start + Duration - 1,

        % Select first possible interval
        schedule_first_interval(Duration, LPossibilities, (Start, End)),

        % Update room agenda
        retract(agenda_operation_room1(Room, Day, Agenda)),
        insert_agenda((Start, End, SurgeryId), Agenda, Agenda1),
        assertz(agenda_operation_room1(Room, Day, Agenda1)),

        % Update agendas
        insert_agenda_doctors((Start, End, SurgeryId), Day, Doctors),

        write('Surgery '), write(SurgeryId), write(' scheduled in room '), write(Room),
        write(' at '), write(Start), write(' to '), write(End), write(' for a total time of '), write(Duration), nl
    ).
```

## Conclusão

Em suma, podemos afirmar que, após trabalharmos com um Algoritmo Genético, este é eficaz e bem estruturado para o problema do escalonamento de cirurgias.

Combina aleatoriedade, diversidade e estratégias de seleção eficientes, além de oferecer parametrização para diferentes condições de término.

Caso esteja bem utilizado, podemos concluir que este é método adequado para lidar com as complexidades do ambiente hospitalar e pode ser ajustado para resolver problemas semelhantes noutros contextos.