Uma imagem com Tipo de letra, Gráficos, design gráfico, captura de ecrã

Descrição gerada automaticamente

**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 3](#_Toc186751418)

[US 6.3.1 3](#_Toc186751419)

[US 6.3.2 3](#_Toc186751421)

[Sprint 3 3](#_Toc186751423)

[US 7.3.1 3](#_Toc186751424)

[US 7.3.2 3](#_Toc186751426)

# Sprint 2

## US 6.3.1

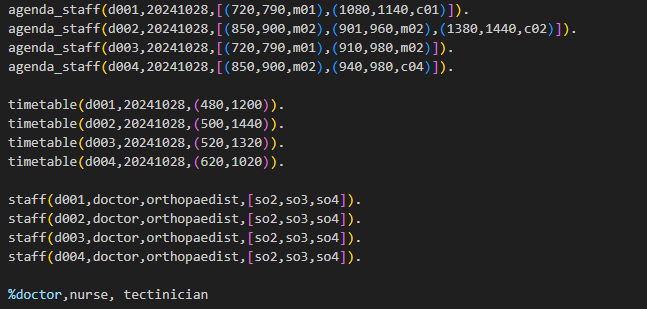
Requisitos: “*As an Admin, I want to obtain the better scheduling of a set of operations (surgeries) in a certain operation room in a specific day.”*

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.

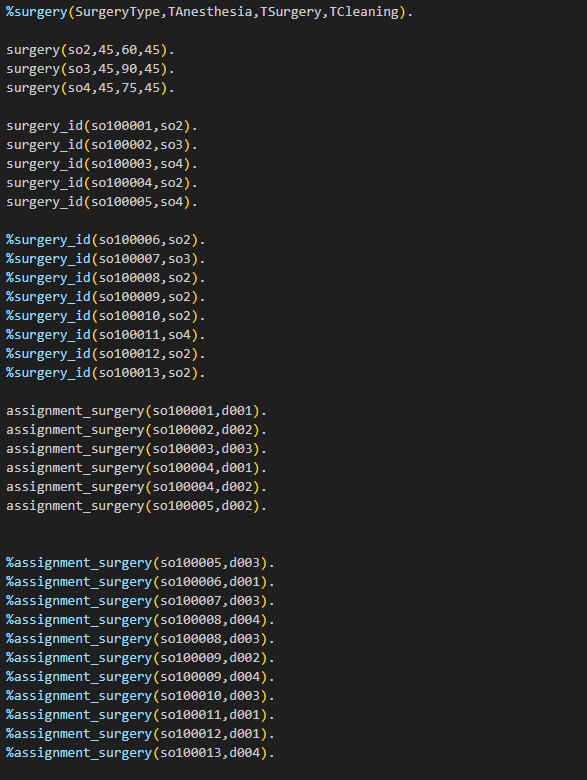
Sendo assim, o funcionamento deste sistema pode ser descrito da seguinte forma:

**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 especificas dos trabalhadores do hospital, como a função (doutor, enfermeiro, técnico) e as suas especialidades



1. **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.

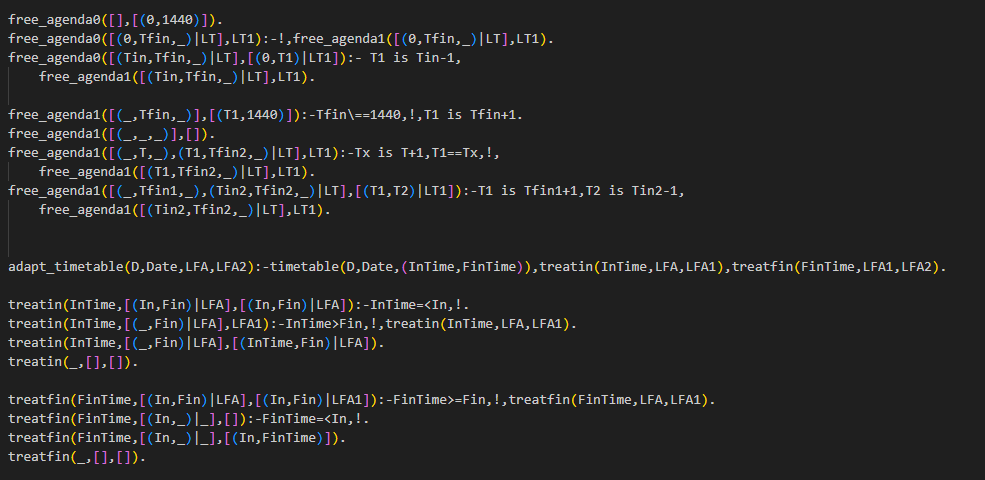


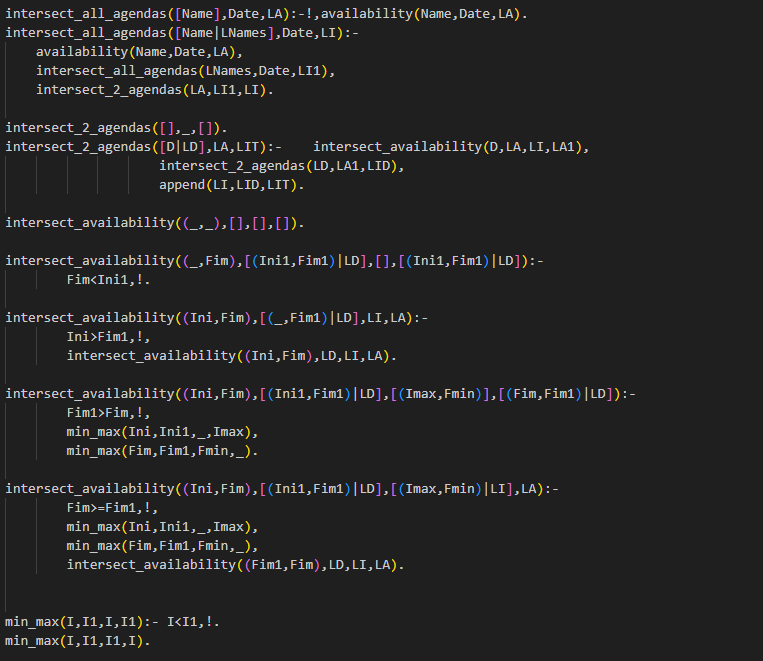
1. **Ocupação das Salas de Operação**:
   * O predicado agenda\_operation\_room/3 lista os horários em que uma sala de operação já está ocupada num dia específico.



**Gestão das Agendas:**

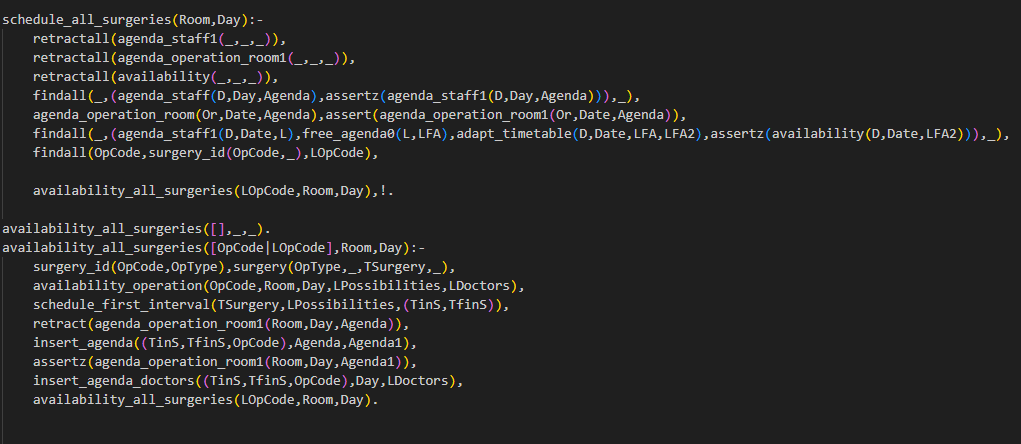
1. **Identificação de Intervalos Livres**:
   * free\_agenda0/2 processa as agendas ocupadas para determinar os intervalos de tempo disponíveis.
   * Caso as agendas não possuam horários é assumido que o horário livre é o dia todo e caso contrário usa free\_agenda1/2 para calcular os horários livres 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.



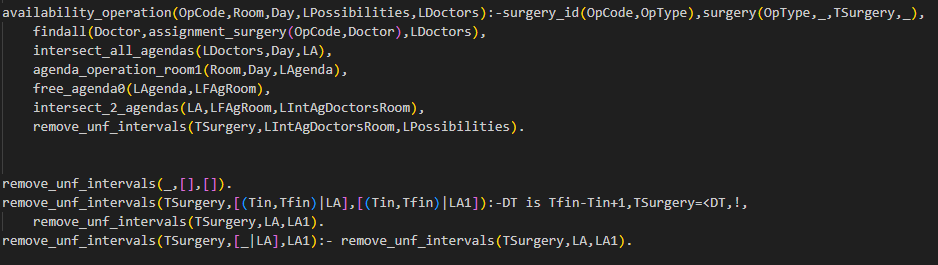
1. **Interseção de Agendas**:
   * intersect\_all\_agendas/3 combina as disponibilidades de diferentes médicos para encontrar horários em comum.
   * intersect\_2\_agendas/3 calcula a interseção entre duas agendas específicas, refinando os horários disponíveis.

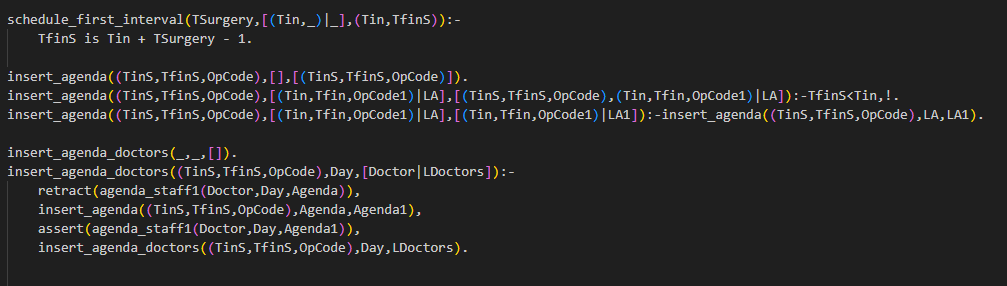
**Agendamento de Cirurgias:**

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.



1. **Determinação de Horários Disponíveis**:
   * availability\_operation/5 avalia os horários possíveis para uma cirurgia, considerando a disponibilidade de médicos/salas.



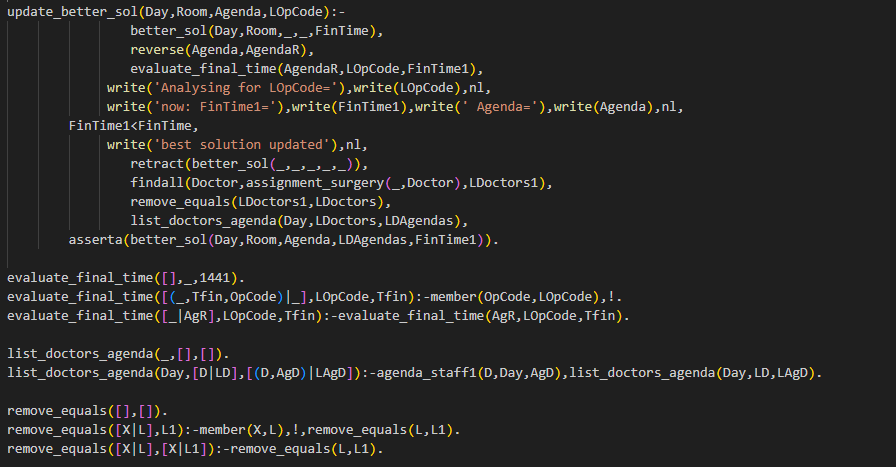
1. **Atualização das Agendas**:
   * insert\_agenda/3 insere uma cirurgia nas agendas dos médicos e/ou salas.
   * insert\_agenda\_doctors/3 insere o horário da mesma cirurgia nas agendas dos médicos lhe designados.

**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.



1. **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.



## 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, so099999)] | 639 | 0.012403964 |
| 2 | [(520, 579, so100000), (580, 639, so100001), (640, 729, so100002), (1000, 1059, so099999)] | 729 | 0.020236968 |
| 3 | [(520, 579, so100000), (580, 639, so100001), (640, 714, so100003), (715, 804, so100002), (1000, 1059, so099999)] | 804 | 0.044281005 |
| 4 | [(520, 579, so100000), (580, 654, so100003), (655, 714, so100004), (715, 804, so100002), (805, 864, so100001), (1000, 1059, so099999)] | 864 | 0.290410995 |
| 5 | [(520, 579, so100000), (580, 639, so100004), (640, 714, so100005), (715, 804, so100002), (805, 879, so100003), (880, 939, so100001), (1000, 1059, so099999)] | 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!×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**

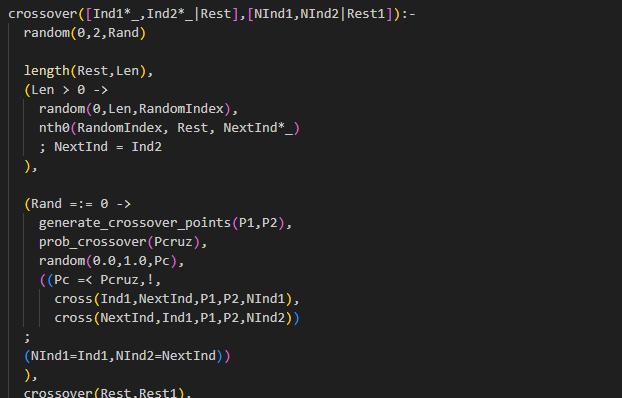
O código apresentado implementa um **Algoritmo Genético (AG)** aplicado ao problema do escalonamento de cirurgias em blocos operatórios de hospitais. O objetivo é otimizar a alocação de cirurgias a recursos hospitalares (como salas de operações e equipas médicas), respeitando restrições de tempo, penalizações 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 (Pc <= Pcruz, onde Pc é a probabilidade gerada aleatoriamente, e Pcruz 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.

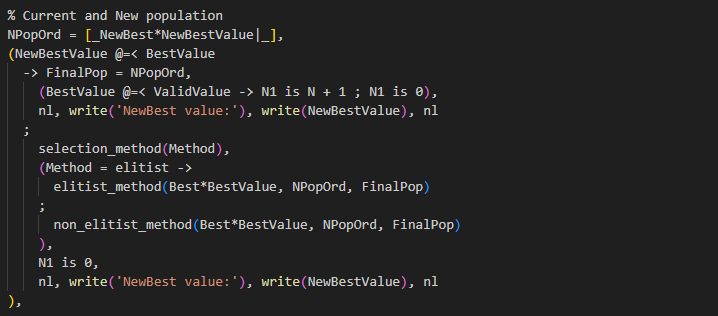


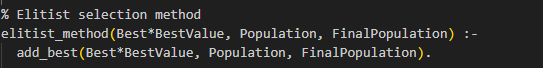
## **Seleção da nova geração da população garantindo que pelo menos o melhor indivíduo entre a população atual e os seus descendentes passe para a próxima geração**

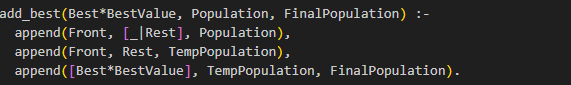
A seleção para a próxima geração combina aspetos elitistas e não elitistas:

* **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 eltitista).

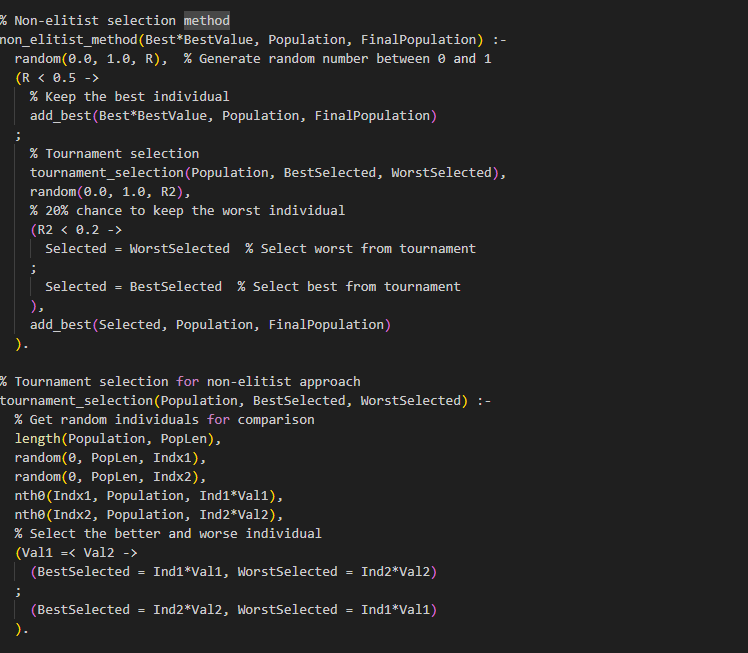


* **Preservação do melhor indivíduo**: Na função generate\_generation/5 o melhor valor da população atual é identificado e comparado ao melhor valor global e, caso seja menor são chamados os métodos elitist\_method/3 ou non\_elitist\_method/3 para o adicionar à população.
* **Método puramente elitista**: Quando queremos um método elitista simplesmente adicionamos o melhor valor anterior à população, efetuado a partir do método add\_best/3.

****



* **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.



## **Parametrização da condição de término**

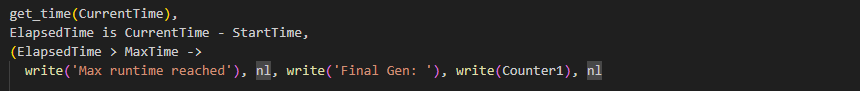
Para além do número máximo de gerações que já tinha sido dado, o código permite outras condições de término configuráveis, sendo estes inicialmente pedidos ao usuário no predicado initialize/0.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

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.

****

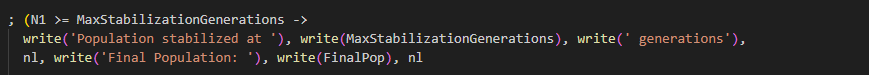
* **Número de Gerações**: Esta condição termina a execução do sistema quando se verifica que a geração atual ultrapassa o número de gerações total introduzido.



* **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).



* **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).



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:

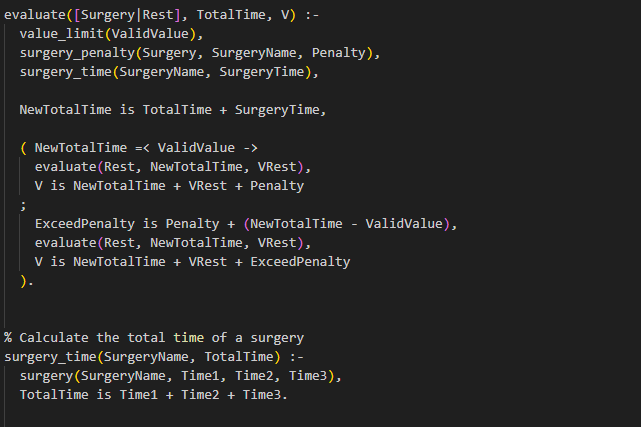
* Uma imagem com texto, captura de ecrã

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

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

* **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.

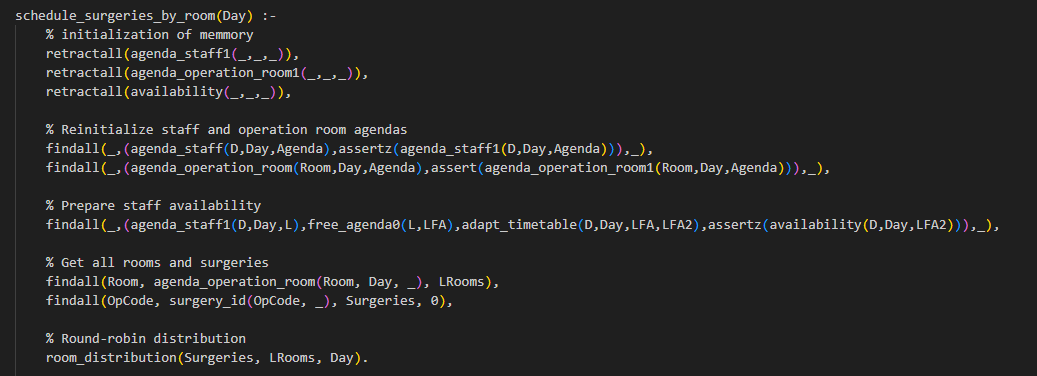


## **Consideração de vários blocos de operação, com um método de atribuição das operações às salas**

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

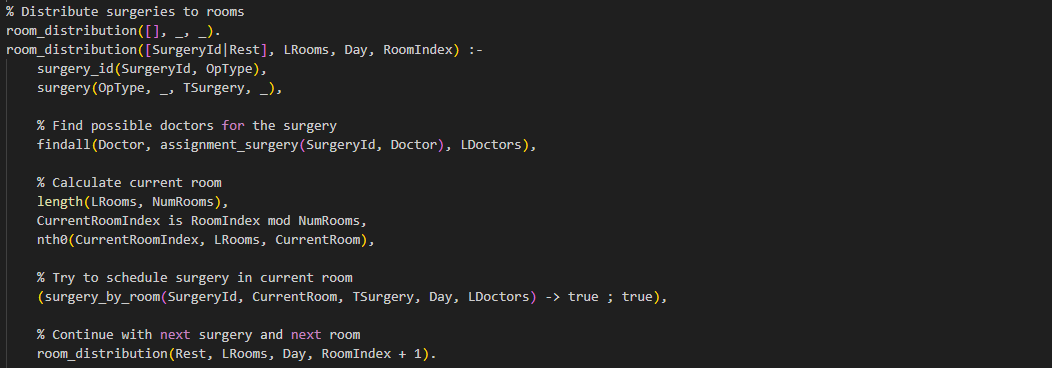
Assim sendo, o método principal de atribuição das operações às salas encontra-se no predicado schedule\_surgeries\_by\_room/1, onde se efetua a preparação para agendamento:

* 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.



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.



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.

