

**Leonardo Luiz Andrade Rocha**

**INTERFACE PARA OTIMIZAÇÃO DE RECURSOS DE PROCEDIMENTOS CIRÚRGICOS ELETIVOS UTILIZANDO ALGORITMOS GENÉTICOS COM INSPIRAÇÃO QUÂNTICA**

Trabalho de Conclusão de Curso apresentado ao curso de Ciências Biológicas da Pontifícia Universidade Católica do Rio de Janeiro como requisito parcial para a obtenção do título de Bacharel em Ciências Biológicas.

Rio de Janeiro, Dezembro de 2020.

# 

**RESUMO**

Esse trabalho trata da criação de uma interface gráfica multiplataforma que permita a utilização do modelo AEIQ-AS [2] de otimização de filas de hospital em cirurgias eletivas utilizando algoritmos genéticos inspirados em computação quântica. Tendo sido disponibilizada a implementação do modelo na linguagem de programação MATLAB, foi feita a tradução do mesmo para que não fosse necessária uma licença pelo usuário da interface. Para a tradução do modelo e implementação da interface foram utilizados principalmente a ferramenta MATLAB Coder e as bibliotecas Python CTypes e Flask.

**ABSTRACT**

This project deals in the creation of a multiplatform graphic interface that enables the use of the model AEIQ-AS [2] of hospital queue optimization on elective surgeries using genetic algorithms inspired by quantic computing. Having been made available a MATLAB implementation of the model, a translation was made in the interest of not requiring the user to have a MATLAB license. For the model translation and for the interface creation the main tools used were the MATLAB Coder application and the Python libraries CTypes and Flask.

**SUMÁRIO**

[1 Introdução 9](#_Toc65521100)

[1.1 Objetivo 10](#_Toc65521101)

[1.2 Descrição do Trabalho 10](#_Toc65521102)

[1.3 Estrutura do Projeto 11](#_Toc65521103)

[2 AEIQ-AS 11](#_Toc65521104)

[2.1 Algoritmos Genéticos 11](#_Toc65521105)

[2.2 Computação Quântica 14](#_Toc65521106)

[2.3 Algoritmos Genéticos com Inspiração Quântica 15](#_Toc65521107)

[2.4 Algoritmo Evolutivo com Inspiração Quântica para a Área de Saúde (AEIQ-AS) 18](#_Toc65521108)

[3 Tecnologias Utilizadas 22](#_Toc65521109)

[3.1 MATLAB Coder 22](#_Toc65521110)

[3.2 Compilador GCC 24](#_Toc65521111)

[3.3 Python 24](#_Toc65521112)

[3.3.1 CTypes 24](#_Toc65521113)

[3.3.1 Flask 25](#_Toc65521114)

[4 Projeto e Especificação 25](#_Toc65521115)

[4.1 Etapas do Projeto e Problemas Encontrados 35](#_Toc65521116)

[4.1.1 Tradução do Código MATLAB para a Linguagem Python 36](#_Toc65521117)

[4.1.2 Tradução do Código MATLAB para a Linguagem C 38](#_Toc65521118)

[4.1.3 Conexão do Modelo em C com a Linguagem Python 43](#_Toc65521119)

[4.1.4 Desenvolvimento da Interface 45](#_Toc65521120)

[5 Conclusão 45](#_Toc65521121)

[6 Referências Bibliográficas 45](#_Toc65521121)

**LISTA DE FIGURAS**

|  |  |  |
| --- | --- | --- |
| Figura 1. | Fluxo base de um algoritmo genético. | 12 |
| Figura 2. | Representação gráfica de uma seleção por torneio. | 13 |
| Figura 3. | Cruzamento Uniforme Baseado em Ordem. | 14 |
| Figura 4. | Diagrama do processo cirúrgico. | 10 |
| Figura 5. | Árvore de decisão para prioridade do paciente. | 20 |
| Figura 6. | Classificação dos recursos. | 21 |
| Figura 7. | Fluxo de geração da interface. | 22 |
| Figura 8. | Tabela de dependência dos *templates* HTML. | 25 |
| Figura 9. | Página “Fila” sem o modelo rodar e no modo “*month”*. | 26 |
| Figura 10. | Página “Fila” com modelo rodando e no modo “*month*”. | 27 |
| Figura 11. | Página “Fila” sem rodar o modelo e no modo “*day*”. | 28 |
| Figura 12. | Página “*Details*” com a Operação 3 selecionada. | 20 |
| Figura 13. | Página “Teste” sem alteração escolhida. | 30 |
| Figura 14. | Página “Teste” com remoção de um médico assistente. | 31 |
| Figura 15. | Página “Teste” com adição de uma sala de operação. | 31 |
| Figura 16. | Página “Teste” com adição de um cirurgião (1ª página). | 32 |
| Figura 17. | Página “Teste” com adição de um cirurgião (2ª página). | 32 |
| Figura 18. | Página “Hospital” sem nenhuma alteração. | 33 |
| Figura 19. | Página “Comparar Alterações” sem nenhuma pasta escolhida. | 34 |
| Figura 20. | Página “Comparar Alterações” com mensagem de erro. | 34 |
| Figura 21. | Página de comparação com “hosp” e “test” selecionados. | 35 |
| Figura 22. | Cabeçalho de uma função MATLAB. | 38 |
| Figura 23. | Cabeçalho de uma função C. | 38 |
| Figura 24. | Tela do MATLAB Coder de definição de tipo dos parâmetros. | 39 |
| Figura 25. | Inicialização de variáveis no MATLAB. | 40 |
| Figura 26. | Definição da função de teste. | 42 |
| Figura 27. | Execução de 25 gerações no MATLAB em 20h16min. | 45 |
| Figura 28. | Execução de 25 gerações no Python em 5h. | 45 |

# 1 **Introdução**

No Brasil, assim como em diversas outras partes do mundo, a demanda da população por Unidades de Saúde é maior do que a capacidade das mesmas de atender aos pacientes [1], tornando-se comum o surgimento de filas de espera. Entretanto, planeja-las é um grande desafio devido à grande quantidade de fatores e recursos que devem ser considerados. Os procedimentos cirúrgicos são um caso especialmente crítico, devido a sua complexidade, necessitando tanto de recursos humanos, cirurgiões, anestesistas, enfermeiros e instrumentadores, quanto de recursos materiais variados, como os diferentes tipos de leitos e salas cirúrgicas, cada qual com os seus equipamentos dependendo do tipo de procedimento para o qual eles foram desenvolvidos.

O objetivo desse trabalho é desenvolver uma ferramenta computacional amigável com interface gráfica que permita a utilização de um modelo de otimização de filas de hospital. Isso é necessário para permitir que um usuário que não seja da área de computação possa utilizar o modelo em um ambiente real. A necessidade desta ferramenta se dá pela grande complexidade do problema de otimização de filas de hospital, pois apesar de existir uma solução trivial, gerar a fila a partir simplesmente do horário de cadastro da operação, um atraso gerado pela mesma pode provocar uma deterioração no estado do paciente, podendo inclusive levar à morte do mesmo.

O modelo utilizado para a construção desta ferramenta precisa de algumas características importantes. Primeiramente, é essencial que ele leve em consideração a gravidade da operação e o estado do paciente para executar o agendamento, pois apesar desses dados serem muito importantes na definição do prazo de uma operação eles não são considerados na solução trivial. Em seguida, é necessário que o modelo tenha a capacidade de explorar o extenso universo de soluções trazido por um problema de ordenação na escala necessária para atender às necessidades de uma unidade de saúde. Esta é uma das principais características dos algoritmos genéticos, que conseguem, através de uma grande quantidade de indivíduos, explorar um vasto espaço de soluções. Ademais, é necessário também que o algoritmo utilizado tenha um tempo de convergência razoável para poder ser utilizado de forma frequente mesmo com um grande volume de dados. Para lidar com esse requisito foi utilizado um modelo com inspiração quântica o que também diminuí a probabilidade de um indivíduo ficar preso em um mínimo local.

## 1.1 Objetivo

O objetivo principal desse projeto é a criação de um protótipo de interface para o modelo AEIQ-AS, desenvolvido em [2], baseado em algoritmos genéticos com inspiração quântica, para a automatização e otimização do planejamento de procedimentos cirúrgicos eletivos. O protótipo permitirá a utilização do modelo por um público-alvo leigo, fornecendo de forma amigável o planejamento dos procedimentos cirúrgicos.

Sobre a ferramenta desenvolvida neste trabalho, além de sua facilidade de uso, também é necessário prezar as seguintes funcionalidades:

* Registrar os pacientes e os recursos existentes e necessários;
* Utilizar um modelo de otimização para a automatização da alocação destes;
* Permitir a visualização e controle intuitivos dos resultados na forma de um calendário de cirurgias que informe a data e horário de cada cirurgia e quais os equipamentos e pessoal necessário para as mesmas; e
* Permitir a comparação entre diferentes configurações hipotéticas de hospitais, permitindo simular e indicar melhores direções de investimento que o hospital deve seguir.

## 1.2 Descrição do Trabalho

A implementação do modelo AEIQ-AS, disponibilizada pelo seu criador, foi feita em linguagem de programação MATLAB, cujo ponto forte é a execução de modelos matemáticos e que necessita de um software proprietário para ser utilizada. Dessa forma, se tornou necessária à tradução do modelo para uma linguagem na qual fosse possível tanto a execução do mesmo quanto a criação da interface.

Dada à necessidade de uma aplicação multiplataforma, ou seja, que possa ser utilizada nas Unidades de Saúde que não possuem uma padronização de máquinas previamente definida, a interface foi desenvolvida como uma aplicação web. Para isso, foi necessário o desenvolvimento de três etapas distintas na implementação do trabalho:

* Tradução da implementação no modelo para outra linguagem;
* Desenvolvimento da interface como uma aplicação web;
* Geração da infraestrutura necessária para que o modelo possa passar informações para a interface.

## 1.3 Estrutura do Projeto

Este trabalho está dividido em cinco capítulos, sendo este o primeiro e os seguintes de acordo com o descrito a seguir:

O modelo utilizado é apresentado no segundo capítulo que contém as definições dos conceitos necessários para a compreensão do modelo, assim como explicação do mesmo. Dentre os conceitos apresentados encontram-se os Algoritmos Genéticos, a Computação Quântica, e os Algoritmos Genéticos com Inspiração Quântica.

O terceiro capítulo aborda as ferramentas básicas utilizadas ao longo do projeto para cumprir os passos definidos anteriormente, na descrição do trabalho. Nesse caso incluiu-se a ferramenta MATLAB Coder, o Compilador GCC e algumas bibliotecas da linguagem Python que foram essenciais para o projeto.

Já o quarto capítulo apresenta a interface criada para a interação com o modelo de otimização de filas, assim como detalha cada uma das fases do desenvolvimento do projeto.

Por fim, no último capítulo encontram-se as conclusões e os trabalhos futuros.

# 2 AEIQ-AS

Como o projeto foi feito com base no modelo AEIQ-AS, nesse capítulo serão descritas as tecnologias necessárias para compreender este modelo, além de detalhar o modelo em si.

## 2.1 Algoritmos Genéticos

Os algoritmos genéticos são um paradigma da área denominada Computação Evolucionária. Neles as soluções geradas são chamadas de indivíduos ou cromossomos, enquanto o conjunto de cromossomos é denominado de população. Os indivíduos são modificados, a cada geração, por operadores genéticos que tentam simular os processos naturais de combinação e mutação genética. Além destes operadores, existem os conceitos de seleção e elitismo, os quais definem os indivíduos que devem pertencer à geração seguinte, simulando o princípio de seleção natural. Dado que os indivíduos com melhores avaliações têm mais possibilidades de sobreviver, a população deve evoluir gradualmente. Estes algoritmos são chamados de Algoritmos Genéticos “clássicos” por serem os mais utilizados na literatura [17].

Para utilizar um algoritmo genético, primeiro é necessário definir a codificação de uma solução em um cromossomo, e definir a função de avaliação do indivíduo. Posteriormente, o algoritmo segue o fluxo definido na figura 1 aonde a inicialização da população é seguida por um ciclo de gerações.

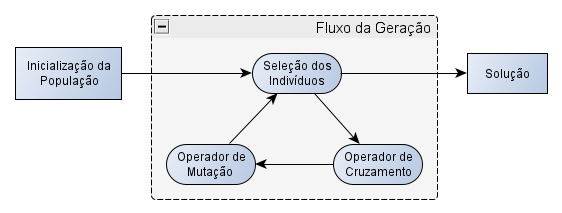


Figura 1. Fluxo base de um algoritmo genético.

Antes de iniciar o fluxo é necessário definir a codificação do cromossomo, a codificação representa a definição o espaço de busca do problema e por isso pode variar bastante de acordo com o formato das soluções possíveis ao problema analisado. Em um problema de ordem, por exemplo, no qual a solução é uma lista ordenada de elementos imutáveis, a solução é melhor codificada por um vetor de identificadores dos elementos a serem ordenados. A maior restrição é de que a representação que contenha informações suficientes para reconstituir a solução relacionada à mesma, mas uma escolha inadequada pode aumentar desnecessariamente a complexidade da busca.

Da mesma forma, a função de avaliação dos indivíduos também pode apresentar bastante variabilidade de acordo com as características do problema. Com o seu principal objetivo sendo avaliar uma solução, a partir do seu cromossomo, gerando uma pontuação (aptidão), que possa ser utilizada para comparar a eficiência de duas ou mais soluções. Dependendo do problema, esta solução pode ser tão simples ou tão complexa quanto o necessário, por exemplo, como será explorado no capítulo 2.4, no modelo utilizado, a função de avaliação necessita primeiramente alocar as cirurgias de acordo com a disponibilidade dos recursos do hospital, utilizando a ordem de priorização do cromossomo, para posteriormente avaliar de quatro formas diferentes o cronograma e finalmente gerar a aptidão a partir da soma ponderada dos valores encontrados.

Definidas a codificação e a função de avaliação, a inicialização da população tende a ser um dos passos mais simples nos algoritmos clássicos, simplesmente gerando uma população com tamanho pré-determinado de indivíduos aleatórios, dentro do espaço de busca. Em seguida poderá ser iniciada a primeira geração.

Iniciando a geração, a seleção dos indivíduos decide algoritmicamente quais os indivíduos que poderão participar do ciclo, descartando os outros. Com o objetivo de simular a seleção natural, a seleção utilizada no algoritmo dá preferência aos cromossomos mais bem avaliados, dando probabilidades menores deles serem descartados. Entretanto, classicamente, a seleção é feita de forma a possibilitar a sobrevivência de um indivíduo menos apto para evitar uma convergência prematura do modelo. A forma de seleção utilizada nesse trabalho foi a seleção por torneio, onde os indivíduos da geração anterior são pareados e é mantido somente o mais apto de cada um dos pares, como mostrado na figura seguinte.

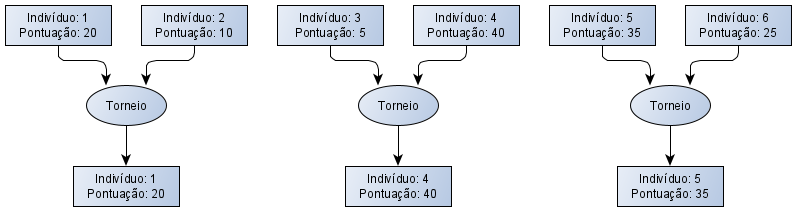


Figura 2. Representação gráfica de uma seleção por torneio

Dentro de uma geração são utilizados dois operadores diferentes para gerar os novos indivíduos. O primeiro, operador de cruzamento, tenta simular a reprodução sexual dos seres vivos, onde todos os indivíduos de uma geração são pareados e têm suas características combinadas com a de seus pares de forma a gerar os indivíduos da nova geração. Um exemplo o mesmo para problemas de ordem é o cruzamento uniforme baseado em ordem [21] no qual certos elementos de cada um dos pais são fixados e os outros são reorganizados de acordo com a ordem do outro como explicado na figura 3.

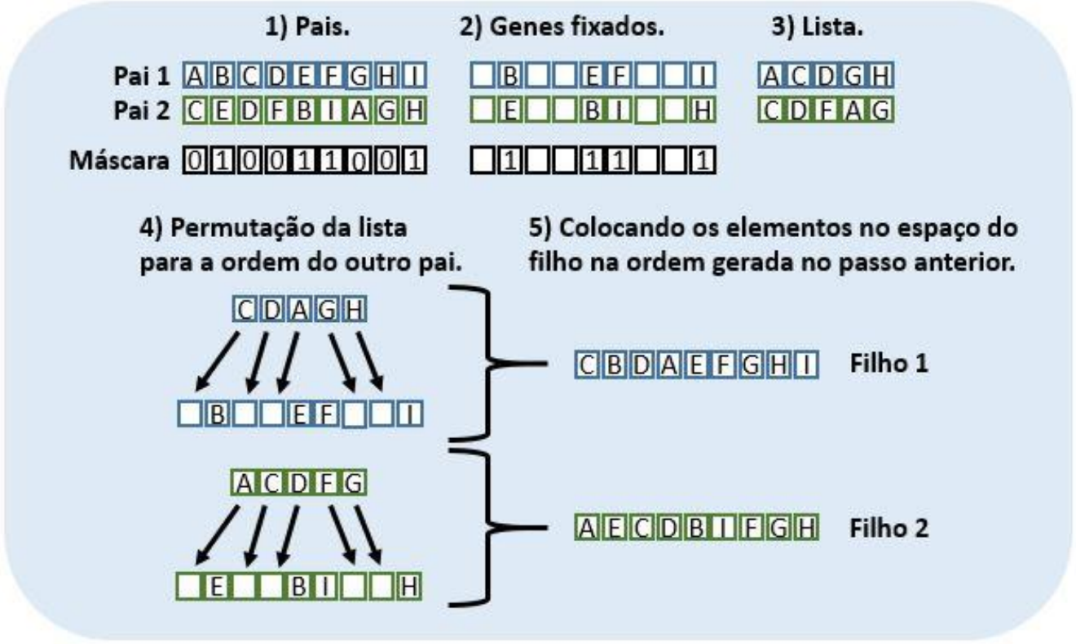


Figura 3. Cruzamento Uniforme Baseado em Ordem [2].

Já o operador de mutação, diferentemente do operador de cruzamento, que gera indivíduos novos a partir da geração anterior, tem como objetivo aumentar a diversidade da população ao introduzir alterações aleatórias nos cromossomos. Para exemplificar, uma mutação possível seria a troca de lugar de dois elementos do cromossomo entre si.

Outro mecanismo dos algoritmos genéticos que pode ser aplicado é o elitismo. Nem sempre aplicado nos algoritmos clássicos, o elitismo força a sobrevivência de um número fixo de indivíduos da geração anterior que são copiados sem alteração para a nova geração. Isso é feito de forma a garantir que a melhor solução de uma nova geração sempre seja ao menos tão boa quanto à da geração anterior.

## 2.2 Computação Quântica

A computação quântica é um paradigma de computação que teve sua inspiração na mecânica quântica. Em mecânica quântica, é possível que uma partícula esteja em dois ou mais estados ao mesmo tempo [18]. A propriedade de estar simultaneamente em vários estados é nomeada superposição. Os dispositivos quanticamente inspirados são exponencialmente mais eficientes que os dispositivos clássicos já que, graças à superposição, todo o espaço de busca de um problema é considerado simultaneamente. Alguns trabalhos demonstram que o uso de sistemas clássicos inspirados neste fenômeno pode produzir um ganho substancial em relação ao tempo de processamento e à qualidade da solução [19,18].

Enquanto na programação clássica a menor unidade de armazenamento é o bit, podendo ter o valor “0” ou “1”, na computação quântica o Qbit (bit quântico) se encontra em uma superposição entre os estados “0” e “1”; ao observá-lo, tem-se certa probabilidade de encontrá-lo no estado “0” e outra de encontrá-lo no estado “1”. No caso em que temos n Qbit, cada um dos 2n estados terá certa probabilidade de ser observado, sendo o somatório das probabilidades sempre igual a 1. Além disso, após observar um conjunto de Qbit no estado Ψ, a sua superposição muda de forma que a probabilidade de encontrar os Qbit no estado Ψ se torna 100% e a de observá-los em qualquer outro estado se torna 0%.

Da mesma forma que a computação quântica introduz a sua contraparte do bit, ela faz o mesmo com as portas lógicas, criando os Q-Gates, que consistem em operações matriciais de rotação que atuam nas probabilidades de uma superposição, permitindo assim a atualização de um Qbit sem observá-lo.

## 2.3 Algoritmos Genéticos com Inspiração Quântica

Um algoritmo genético com inspiração quântica deve, por definição, ser um algoritmo evolucionário com indivíduos representados por bits quânticos e atualizados por portas quânticas [18].

Como nos algoritmos genéticos clássicos, a representação do indivíduo e dos operadores utilizados dependerá sempre do problema a ser resolvido. O indivíduo quântico proposto por [22], por exemplo, desenvolvido com enfoque em problemas de ordem, é modelado por um vetor de tamanho n composto por mQbits, cada qual podendo assumir qualquer elemento do universo a ser ordenado com certa probabilidade, e o somatório de cada probabilidade sendo 1 para cada mQbit. De forma equivalente, cada indivíduo quântico pode ser representado por uma matriz n x m em que cada linha representaria um Qbit, com as suas colunas sendo a probabilidade de o Qbit ser observado como o elemento específico, ou seja:

Na inicialização dos indivíduos quânticos o tratamento é um pouco diferente do que nos modelos clássicos. Enquanto nos modelos clássicos o objetivo de inicializar os indivíduos aleatoriamente é permitir a maior capacidade de exploração do universo de soluções, os modelos quânticos definem explicitamente a cobertura de uma solução específica. Isso acontece porque, como citado anteriormente, cada indivíduo quântico representa ao mesmo tempo diversas soluções clássicas possíveis, escolhendo uma somente no momento de observação. Dessa forma, o indivíduo quântico com maior cobertura do universo de soluções e menos tendencioso possível (no caso de não se ter informações a priori para direcionar a busca), é aquele cuja probabilidade para a observação de qualquer solução é igual, ou seja:

Um passo necessário adicionado para lidar com a natureza desse tipo de modelo é a observação dos indivíduos quânticos. Essa observação deve conseguir gerar um indivíduo clássico simplesmente com a escolha de valores para cada um dos Qbit do indivíduo quântico correspondente, e de forma que a solução gerada seja válida. Em um problema de ordem, no qual não é permitida a repetição de elementos, isso pode ser feito observando os Qbits sequencialmente e ao observar um deles atualizar os coeficientes de todos os outros de forma a anular a probabilidade do elemento observado se repetir. A atualização pode ser feita então com:

Onde é a probabilidade que tinha o elemento selecionado na linha r. Para selecionar a observação de um Qbit específico, dada a matriz atualizada, podemos sortear um número e selecionar o k-ésimo elemento tal que:

Exemplificando, o processo de observação de um indivíduo quântico com dimensão igual a 3 x 3 poderia se dar da seguinte forma:

* Indivíduo inicial:
* Primeiro elemento selecionado (x=0.45) e atualização:
* Matriz atualizada:
* Segundo elemento selecionado (x=0.12) e atualização:

* Observação do indivíduo:
* Conversão em indivíduo clássico

Conseguindo gerar e avaliar um indivíduo quântico é necessário também poder atualizar os indivíduos quânticos. Não podemos utilizar os indivíduos avaliados diretamente, dado que a observação de um indivíduo quântico que pode ser avaliada é equivalente a um indivíduo clássico, mas é possível utilizarmos as observações mais bem avaliadas de um indivíduo quântico para atualizá-lo. Isso pode ser feito utilizando a equação:

Onde é o indivíduo quântico original, o indivíduo quântico atualizado, a observação selecionada e 𝜀 a taxa de atualização. Por exemplo:

Dado: e

Temos:

Ou seja:

Sendo essa atualização feita para cada uma das observações escolhidas. Como é possível ver, essa atualização mantém as exigências sobre um indivíduo quântico enquanto aumenta a probabilidade de ser observado a partir de podendo chegar a 100% no caso em que . A escolha de deve ser feita então de acordo com as necessidades do problema, com valores pequenos aumentando o tempo de convergência e valores grandes saturando rapidamente os indivíduos quânticos.

Isso significa que além de gerar uma população e avaliá-la de acordo com as diferentes gerações, atualizando-a entre as mesmas, como é feito na abordagem clássica, é necessário, antes de avaliar os indivíduos, gerar uma população clássica a partir de n observações de cada indivíduo da população quântica. A atualização da população quântica é feita utilizando portas quânticas de acordo com os melhores indivíduos da população clássica.

## 2.4 Algoritmo Evolutivo com Inspiração Quântica para a Área de Saúde (AEIQ-AS)

O AEIQ-AS foi criado com base no algoritmo desenvolvido por Silveira [19]. Entretanto, como o custo computacional da observação dos indivíduos quânticos cresce rapidamente com a dimensão das matrizes, no AEIQ-AS os indivíduos clássicos gerados pela observação são evoluídos por algumas gerações antes de serem utilizados na atualização dos indivíduos quânticos. Essa estratégia permite o aproveitamento do potencial de busca do espaço de soluções da representação com inspiração quântica sem comprometer o tempo de processamento.

O objetivo principal do modelo é ordenar as cirurgias eletivas dos pacientes de uma unidade de saúde da melhor forma possível de acordo com a função de avaliação escolhida. Para tal, é necessário um entendimento das características específicas do problema que queremos resolver, nominalmente:

* O processo cirúrgico
* A prioridade dos pacientes
* Os recursos necessários

Para melhor representar o processo cirúrgico de um paciente ele primeiramente foi dividido em quatro partes diferentes, cada qual requisitando os seus recursos específicos, tanto humanos quanto materiais, e tendo os seus tempos determinados pelas características específicas de cada indivíduo. Como explicitado na figura 4, cada cirurgia será dividida em um evento pré-operatório, um evento cirúrgico, um evento pós-operatório e um evento de recuperação, eles devem ser alocados no calendário de cirurgias em ordem e cumprindo o tempo mínimo para cada um, ditado pelo tipo de cirurgia e estado do paciente.

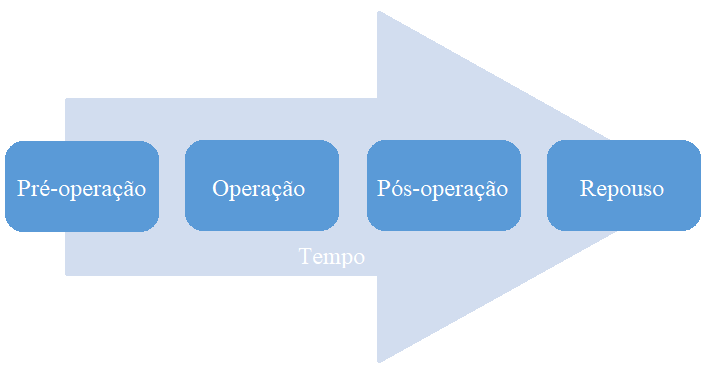


Figura 4. Diagrama do processo cirúrgico. Adaptada de [2]

Para determinar o nível de prioridade de um paciente, ele é categorizado em relação à gravidade da sua situação, ou seja, qual o efeito do atraso da operação sobre o progresso da enfermidade e o nível de incapacitação do indivíduo devido à falta da cirurgia. Uma ilustração do processo de definição se encontra na figura a seguir.

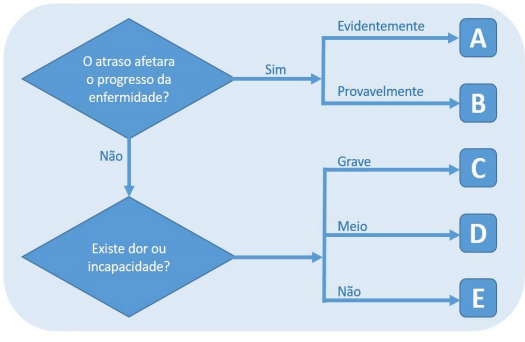


Figura 5. Árvore de decisão para prioridade do paciente [2].

Como mencionado anteriormente, também é necessário considerar os recursos necessários para a execução da cirurgia. Entre os recursos materiais necessários se encontram os diferentes leitos, cada qual com os seus equipamentos e localização em áreas específicas do hospital dependendo de para quais procedimentos ele foi instalado. Por outro lado, temos os recursos humanos, que incluem basicamente os médicos do hospital, incluindo os anestesistas, os cirurgiões e os auxiliares que, apesar de serem necessários principalmente na etapa cirúrgica do processo, podem ser requisitados em diversas combinações de acordo com o tipo de cirurgia. Outra característica importante dos recursos humanos é que as habilidades deles com cada tipo de cirurgia podem variar de acordo com um espectro, dessa forma, foi utilizada uma nota de 0 a 3 para cada relação possível de médico com especialidade.

Idealmente, todas as cirurgias seriam executadas somente pelo médico mais qualificado de cada área para cada função, entretanto, isso pode ser uma impossibilidade quando se tem como objetivo realizar todas as cirurgias com o menor atraso possível, dessa forma, o modelo possibilita a alocação de médicos em cirurgias de especialidades cuja sua nota é 1 ou 2, mas coloca isso como um dos parâmetros a serem minimizados, cabendo então ao usuário decidir o quanto que isso será penalizado em detrimento da velocidade de atendimento. Na figura 6 é possível visualizar todos os tipos de recursos necessários considerados pelo modelo.

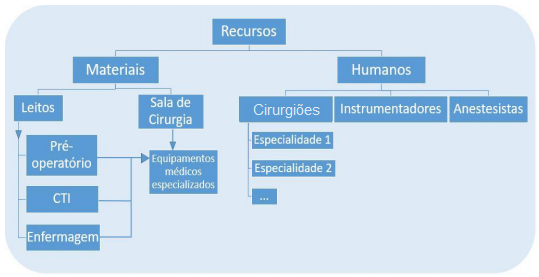


Figura 6. Classificação dos recursos [2].

Todas as particularidades do problema terão que ser utilizadas por fim na avaliação das soluções dadas pelo modelo e para realizar essa avaliação são necessárias duas fases distintas. Primeiramente, é necessário alocar as cirurgias no calendário, isso é feito simplesmente alocando as cirurgias na ordem da fila da solução no primeiro horário cujos recursos necessários para a mesma estão disponíveis. Ao escolher o horário para uma cirurgia, os recursos necessários para a mesma também são alocados no mesmo horário e não serão considerados disponíveis naquele horário para as cirurgias seguintes. Note que uma cirurgia pode ser alocada em um horário posterior ao de outra que estava após ela na fila, dependendo da disponibilidade dos recursos necessários para cada uma das cirurgias.

Posteriormente, é necessário avaliar o calendário utilizando a função de avaliação. Por esse ser um problema multiobjetivo, a função de avaliação é uma combinação linear de outras funções, cada qual representando um dos objetivos possíveis da otimização, e onde os coeficientes de cada uma podem ser escolhidos pelo usuário. Os objetivos a serem minimizadas são:

* O tempo total decorrido entre a primeira e a última cirurgia;
* O tempo médio que um paciente aguardará entre o vencimento do seu prazo e o seu atendimento, caso exista algum caso de paciente sendo atendido fora do seu prazo; e
* A quantidade de cirurgias realizadas por cirurgiões com especialidades adjacentes à da cirurgia, sendo esta separada entre a quantidade de atendimentos sendo feitos por médicos de segundo grau de especialidade e aqueles de terceiro grau.

# 3 Tecnologias Utilizadas

O fluxo de geração do código da interface utilizou diversas ferramentas, começando pelo MATLAB Coder que permite a tradução automática de um código MATLAB para C, seguido do compilador GCC para poder gerar uma biblioteca compartilhável a partir do código C em questão, e, finalmente, o Python, cuja biblioteca CTypes permite a leitura da biblioteca exportada e o Flask que permite a criação da interface em si utilizando o código Python. Na figura 7 esse fluxo é representado de forma gráfica para facilitar o entendimento.

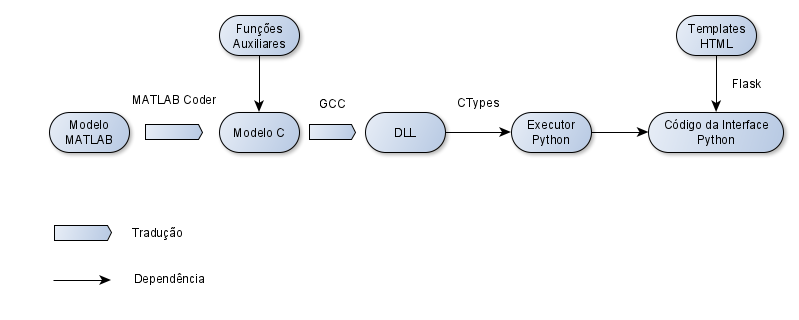


Figura 7 Fluxo de geração da interface.

## 3.1 MATLAB Coder

O MATLAB Coder [3] é uma ferramenta comercial da empresa MathWorks que permite a tradução automática de um código MATLAB para um código C ou C++, possibilitando a utilização do código em outros projetos e sem a necessidade de uma licença MATLAB para executar o código. Esta ferramenta permite também a geração de um código MEX pré-compilado com um tempo de execução menor do que o código original, faceta da aplicação que não será explorada ao longo do texto. A utilização dessa ferramenta se dá em diversos passos, conforme listado a seguir:

* + - Definir quais serão as possíveis funções de entrada do código.

Por ser uma linguagem interpretada, multiparadigma [4] o MATLAB permite a execução das funções em ordem qualquer, permitindo que o usuário chame as funções no momento que achar necessário, sem uma ordem pré-estabelecida descrita em código. Sendo assim, é necessário realizar certas inicializações nas funções de entrada que não serão necessárias nas outras quando o fluxo de execução for descrito na linguagem C.

* + - Determinar o tamanho de cada uma das variáveis a serem passadas como parâmetro.

O MATLAB, por padrão, considera todas as variáveis como matrizes n-dimensionais com dimensão variável. Entretanto, pela linguagem C tratar de forma altamente diferente variáveis de dimensão fixa e variável no cabeçalho de suas funções, é necessária, antes de qualquer passagem pelo código, a determinação de quais serão os tamanhos de cada uma das variáveis utilizadas como parâmetro, incluindo o número de dimensões e o tamanho de cada dimensão no caso dela ter um tamanho fixo.

A ferramenta MATLAB Coder provê certa ajuda nesse aspecto, com sugestões de tamanho para os parâmetros de acordo com uma execução do modelo. Essas sugestões, entretanto, tendem a não identificar quando o tamanho de uma variável é variável, utilizando simplesmente um dos tamanhos possíveis, dado o arquivo de teste.

* + - Verificar e consertar problemas de execução

Dadas as diversas facilidades proporcionadas pelo MATLAB como uma linguagem de mais alto nível, é impossível a tradução direta da maior parte dos códigos feitos nela para o C. Por exemplo, um trecho de código do MATLAB pode ser interpretado de forma diferente dependendo de quais são os tipos dos parâmetros utilizados. Dependendo do caso, isso pode ser resolvido gerando mais de uma função C, cada uma para um caso específico, mas o programa alerta tanto os casos que podem ser resolvidos dessa forma quanto os que não podem, para que o programador possa editar o código original tornando-o mais compatível com a linguagem final. Os problemas específicos encontrados e resolvidos ao longo do projeto serão detalhados no capítulo 6.

* + - Gerar arquivo fonte

A aplicação permite a visualização do código gerado e acesso a algumas configurações de mais baixo nível da tradução, para que o programador possa verificar quais foram as traduções de cada módulo e alterar as configurações como for necessário.

## 3.2 Compilador GCC

O GCC (*GNU Compiler Collection*) foi criado originalmente como o compilador do sistema operacional criado pelo Projeto GNU [6], e hoje em dia é o compilador padrão da maioria dos projetos que envolvem ou são desenvolvidos no GNU ou no Linux, inclusive o próprio kernel do Linux.

A versão utilizada neste projeto foi a distribuída pelo MinGW (*Minimalist GNU for Windows*) do GCC para Windows.

## 3.3 Python

Como MATLAB, Python também é uma linguagem interpretada, multiparadigma, e de alto nível, mas diferentemente do primeiro Python é uma linguagem para propósitos diversos [7]. O que significa que o objetivo da linguagem é poder escrever códigos para as mais variadas aplicações. A linguagem é código aberto e possui mais de 275.000 bibliotecas disponibilizadas pelos seus usuários além das várias bibliotecas padrão que a linguagem já oferece. [8]

### 3.3.1 CTypes

A biblioteca CTypes é uma das bibliotecas incluídas padrão do Python desde sua versão 2,5. Ela é utilizada para fazer a intermediação entre códigos C e Python, [9] permitindo principalmente chamar funções definidas em DLLs e bibliotecas compartilhadas escritas em C, mas exportando também formas de criar, modificar e pegar valores de variáveis C a partir do código Python, permitindo inclusive que seja criado um wrap puramente Python para as funções da DLL específica.

### 3.3.1 Flask

Flask é um dos mais populares *frameworks* de aplicação web para Python [10]. Um framework de aplicação web é um *software* que permite a automatização das atividades comuns do desenvolvimento web, por exemplo, o acesso às bases de dados, a edição dos arquivos HTML de forma mais fácil, e controle de sessões [11].

O Flask foi criado com o objetivo de ser altamente configurável, permitindo um início rápido utilizando as configurações padrão, mas mantendo a escalabilidade ao permitir que qualquer uma das configurações padrão seja alterada de forma fácil para uma maior compatibilidade com o projeto específico. Ele possibilita a utilização de *templates* HTML criados pelo próprio desenvolvedor para simplificar a reutilização de código e permite a utilização de códigos Python dentro do HTML com o objetivo de facilitar a interação do mesmo com o código.

### 4 Projeto e Especificação

Apesar do número reduzido de páginas visíveis na interface certas telas podem mostrar conteúdos diferentes dependendo das interações do usuário. Dessa forma na figura 8 é mostrado um gráfico de dependência dos *templates* HTML e as figuras 9 a 21 exemplificam as possíveis visualizações das telas.

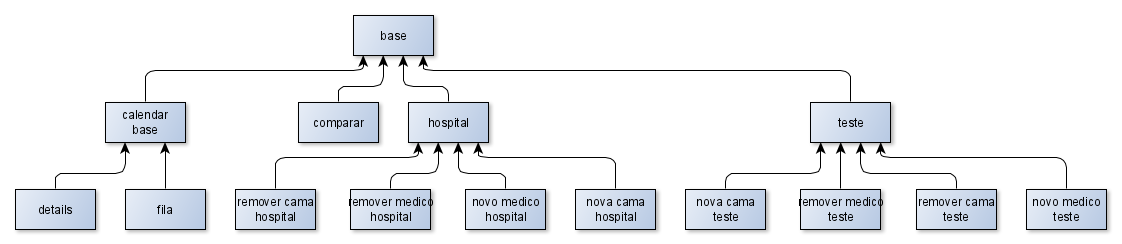


Figura 8. Tabela de dependência dos *templates* HTML.

A figura 9 mostra página inicial da interface que permite a visualização do calendário de cirurgias, quando o modelo já está sendo executado, uma mensagem de texto substitui o botão “Rodar Modelo” como mostrado na figura 10. É possível filtrar para ver somente um dia, como mostrado na figura 11 ou clicar em uma das operações para ver os requisitos necessários, como na figura 12.

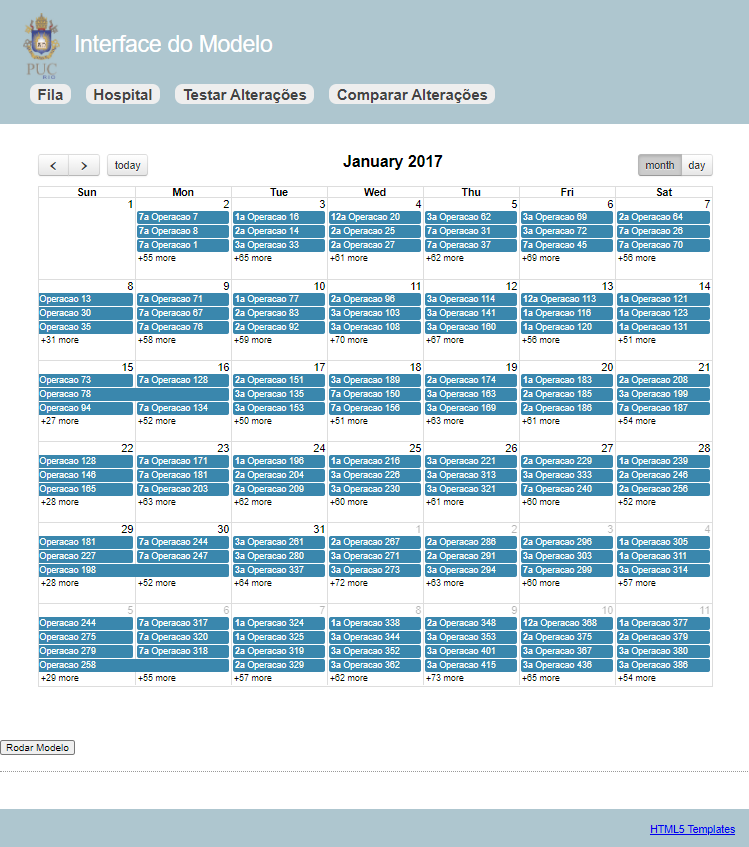


Figura 9. Página “Fila” com o modelo não executando e no modo “*month”*.

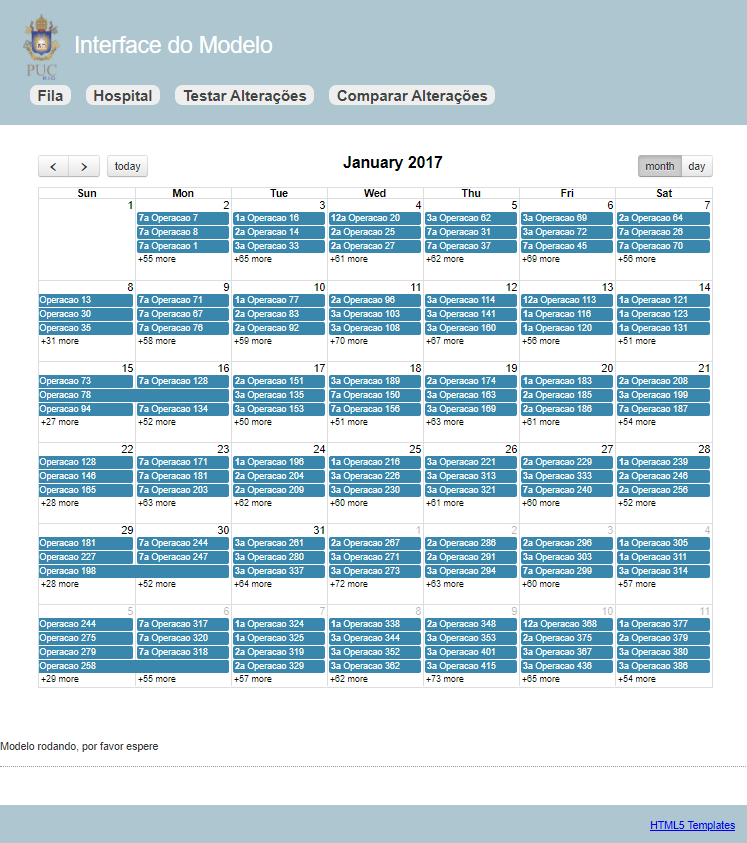


Figura 10. Página “Fila” com modelo em execução e no modo “*month*”.

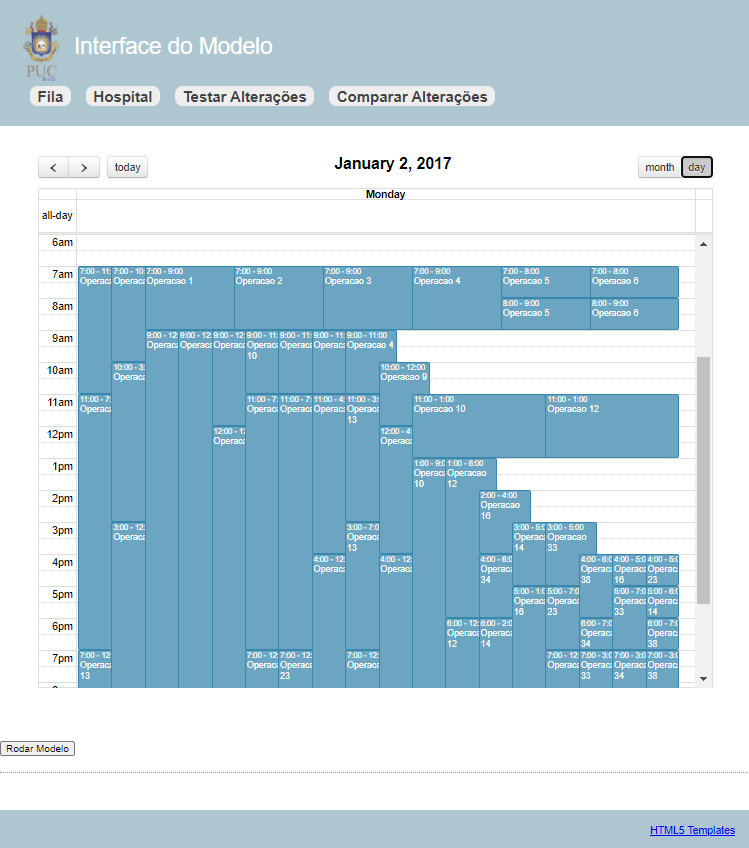


Figura 11. Página “Fila” sem o modelo estar sendo executado e no modo “*day*”.

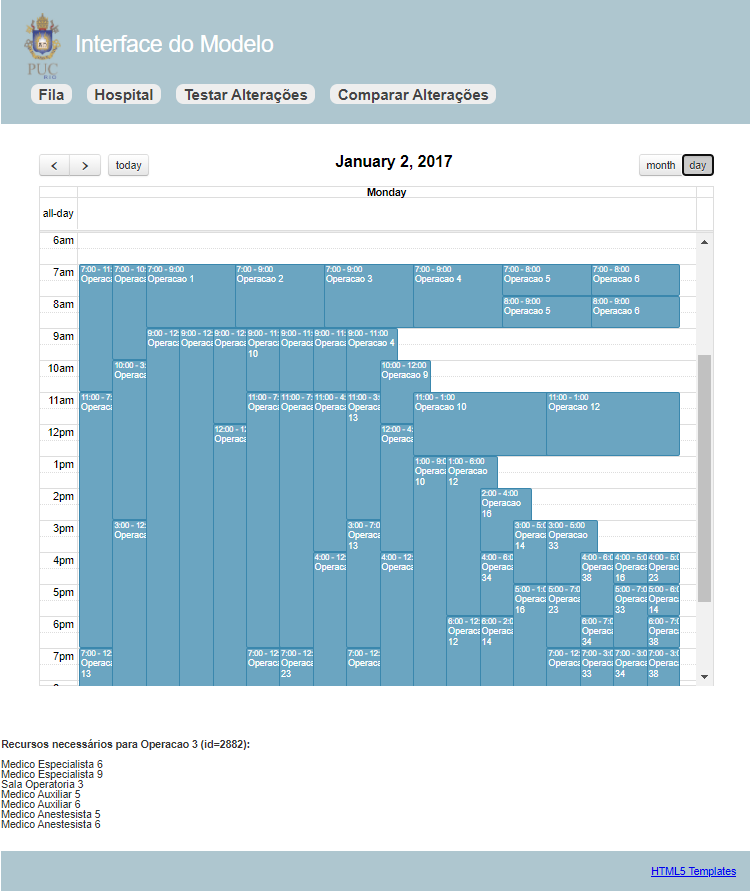


Figura 12. Página “*Details*” com a Operação 3 selecionada.

A página “Teste”, mostrada na figura 13, permite que o usuário faça a configuração de um hospital hipotético para facilitar a alocação de verbas na aquisição de recursos ao saber o impacto gerado por cada alteração no cronograma gerado. Ela possui dois campos de texto para que o usuário consiga carregar um dos testes feitos anteriormente e para escolher qual será o identificador do teste a ser rodado. Ao fazer alguma edição, uma barra lateral aparece para detalhamento da opção, isso é implementado com o usuário sendo redirecionado para uma página diferente que estende o *template* da original.

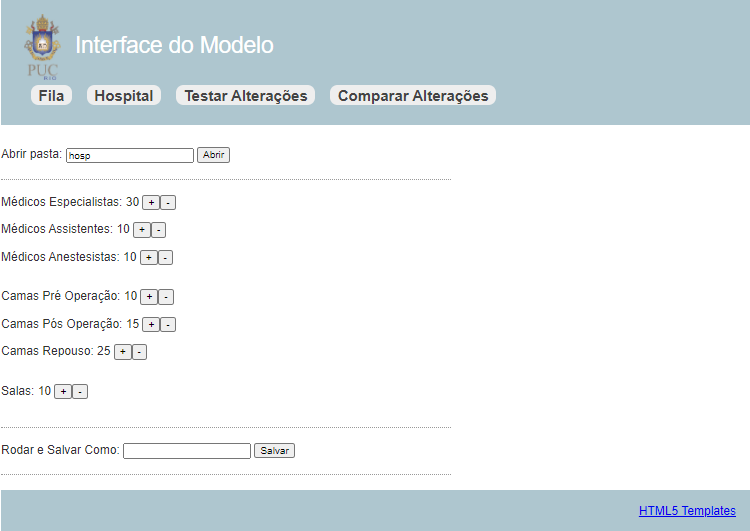


Figura 13. Página “Teste” sem alteração escolhida.

No caso de retirada de qualquer recurso a barra lateral pedirá que o recurso em si seja especificado a partir de um drop-down, como mostrado na figura 14. No caso de adição de uma cama ou sala é perguntado para quais especialidades do hospital aquele recurso está habilitado como mostrado na figura 15. Caso o recurso seja um médico, será pedido para categorizar numa escala de 1 a 3 o quão hábil o profissional é naquela área (podendo ser 0 no caso do cirurgião) como mostrado na figura 16. Adicionalmente, no caso do cirurgião são requisitados também quais os dias de trabalho do mesmo, como mostrado na figura 17.

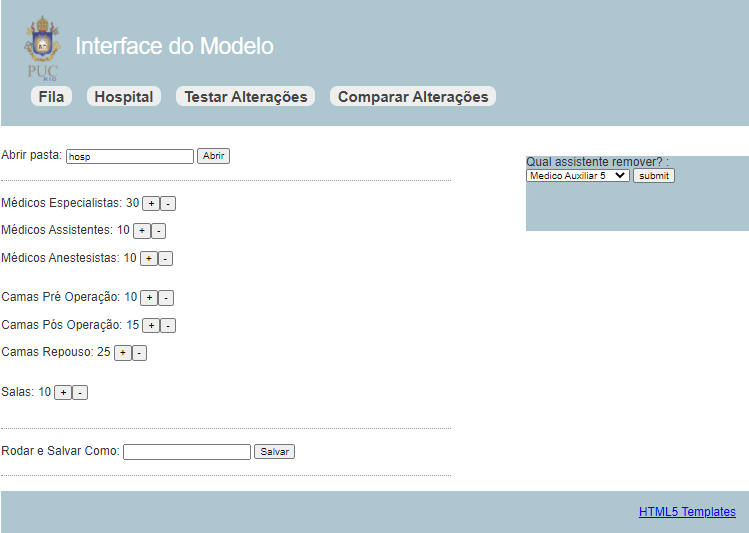


Figura 14. Página “Teste” com remoção de um médico assistente.

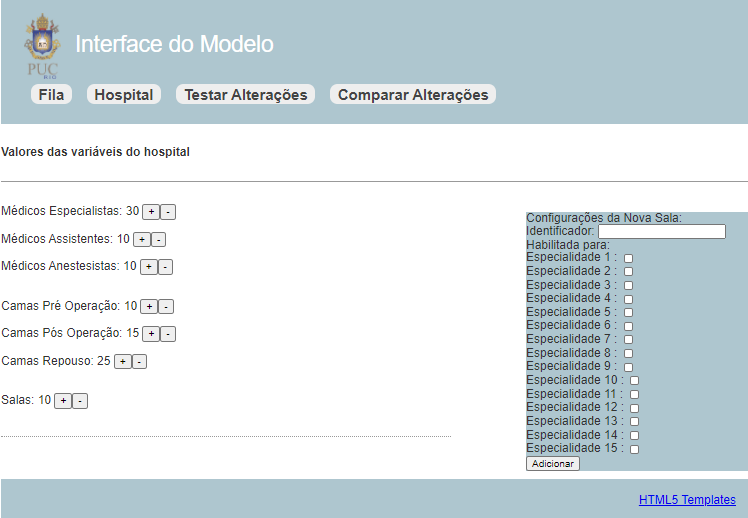


Figura 15. Página “Teste” com adição de uma sala de operação.

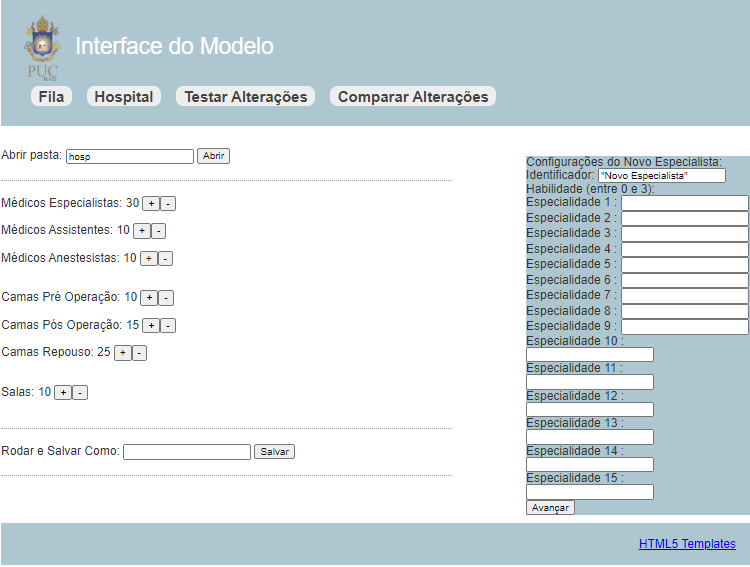


Figura 16. Página “Teste” com adição de um cirurgião (1ª página).

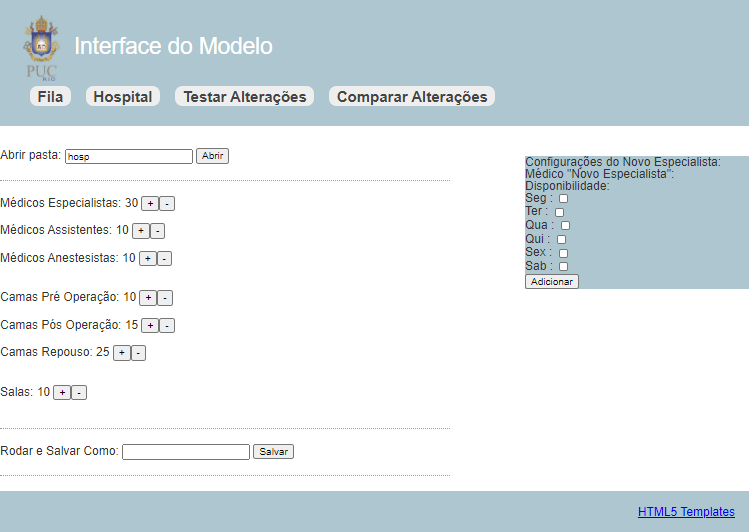


Figura 17. Página “Teste” com adição de um cirurgião (2ª página).

De forma semelhante à página de testes, a página “Hospital” permite o usuário fazer as mesmas configurações, mas diretamente no hospital para o qual é gerada a fila, como exemplificado pela figura 18.



Figura 18. Página “Hospital” sem nenhuma alteração.

Finalmente, a página “Comparar Alterações” permite que sejam comparados dois hospitais diferentes para os quais o modelo foi executado. Ela inicialmente solicita apenas o nome da pasta no qual cada uma das versões foi salva (nome esse que foi colocado como identificador na página de teste), como mostrado na figura 19. No caso de ser colocado o nome de uma pasta inválida ou para qual o modelo está sendo rodado no momento ela apresenta uma mensagem de erro acima da seleção, como mostrado na figura 20. Após escolher duas pastas para fazer a comparação, a página vai passar a mostrar a avaliação do calendário feito para aquele hospital, com o peso de cada um dos critérios utilizados em parênteses e o resultado final para a ordenação em questão, como mostrado na figura 21.

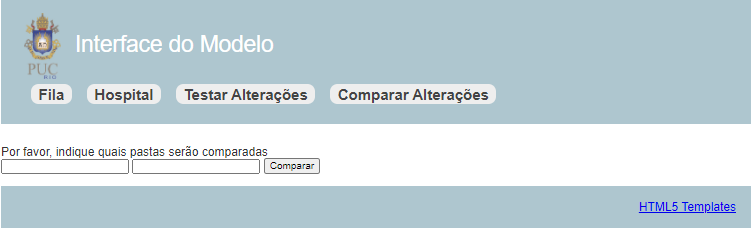


Figura 19. Página “Comparar Alterações” sem nenhuma pasta escolhida.



Figura 20. Página “Comparar Alterações” com mensagem de erro.

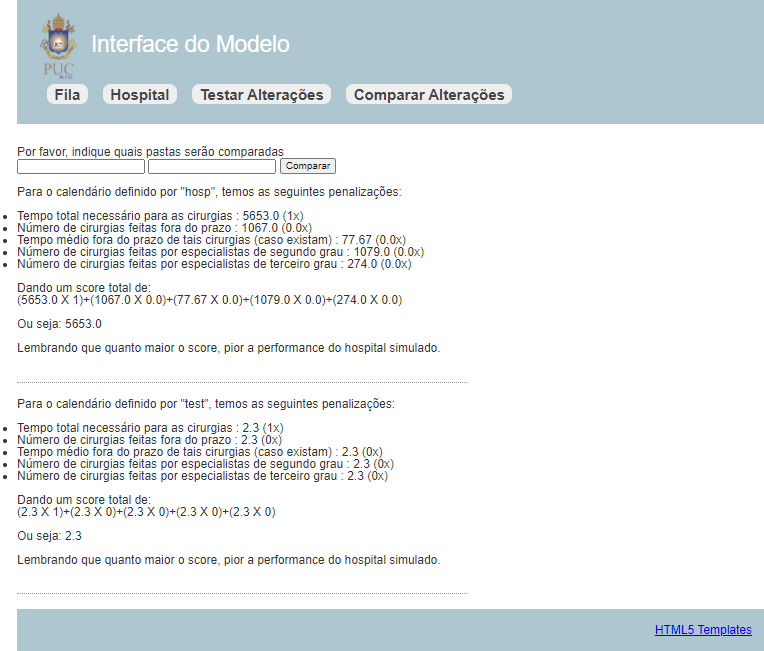


Figura 21. Página de comparação com “hosp” e “test” selecionados.

# 4.1 Etapas do Projeto e Problemas Encontrados

A dissertação na qual o modelo AEIQ-AS foi desenvolvido incluiu uma implementação base deste na linguagem MATLAB. Assim, o modelo não fica acessível por um público não familiarizado com a ferramenta e/ou que não possua a licença do programa para poder executá-lo.

Foi necessário então, primeiramente, fazer um trabalho de tradução do AEIQ-AS para que ele pudesse ser executado por uma linguagem de alto nível. Uma tradução direta, entretanto, não traria os melhores benefícios, pois linguagens que conseguem executar uma gama maior de projetos tendem a ter um desempenho mais lento, pois não são otimizadas para o problema específico.

Assim, optou-se por realizar a tradução para a Linguagem Python, essa escolha se deu pela velocidade de desenvolvimento e facilidade de manutenção de um projeto na linguagem. Como dito anteriormente, o Python é uma linguagem cujo ponto forte está na sua velocidade de criação de projetos, disponibilizando ferramentas nativas e em bibliotecas que permitem ao desenvolvedor se abstrair de partes não relevantes ao projeto. Isso permite não somente concluir a parte de implementação de forma mais rápida, mas também facilita na manutenção e edição do código depois de gerada uma versão inicial do produto.

## 4.1.1 Tradução do Código MATLAB para a Linguagem Python

Inicialmente, para permitir a utilização do código MATLAB dentro do Python, sem a necessidade de uma licença MATLAB, foi tentado fazer uma tradução literal do código para a linguagem. Apesar do conhecimento de que a tradução literal significaria renunciar às qualidades específicas de otimização da linguagem fonte, esse passo foi necessário para podermos quantificar a perda de desempenho, e permitir uma análise melhor dos benefícios e problemas de fazer um código diretamente na linguagem destino que copiasse os passos feitos na original.

Naturalmente, existem certos pontos no modelo que são melhores para fazer a comparação entre as diferentes implementações, por exemplo, enquanto uma linha específica do código pode ser executada mais rapidamente em uma linguagem, a diferença no *overhead* necessário para executar essa mesma linha pode tornar a execução mais lenta nessa mesma linguagem. Ao traduzir aproximadamente 24% do código (339 linhas de 1425 totais), um bom momento para fazer a comparação foi encontrado, onde um passo específico do modelo tinha sido concluído, permitindo que a eficiência das implementações pudesse ser verificada.

No código original o trecho de código era executado em aproximadamente três minutos, onde na tradução literal isso resultou em uma execução de duas horas. Ao fazer essa comparação ficou claro que a tradução literal do código não seria um caminho viável. Supondo uma eficiência semelhante para o restante do código, isso faria com que a execução do código original, que pode demorar dois dias, passasse a ser executado em mais de dois meses, atrasando demasiadamente qualquer teste da ferramenta e impossibilitando o seu uso.

Devido a essa situação, foi descartada a possibilidade de fazer a tradução literal da implementação original. Isso deixou disponível algumas possibilidades:

* + Chamar as funções do MATLAB a partir do Python
  + Reimplementar o código do modelo em Python
  + Utilizar uma ferramenta de tradução automática
* MATLAB => Python
* MATLAB => outra linguagem

Existem diversas bibliotecas Python que utilizam a API disponibilizada pela MathWorks [12] para permitir a chamada de funções MATLAB. Consequentemente, essa opção é a mais recomendada pela MathWorks para aplicações semelhantes à encontrada. Entretanto essa estratégia mantém a dependência da licença MATLAB no código final limitando o uso da ferramenta.

Seria possível também reimplementar o modelo diretamente na linguagem Python, aproveitando todas as vantagens da linguagem que uma tradução literal ignora. O problema com essa estratégia, no entanto, seria a falta de garantias sobre o código gerado, pois além da dificuldade e do grande tempo necessário para recriar um modelo do início em uma nova linguagem, seriam necessários extensivos testes para confirmar que a nova implementação gera os mesmos resultados que a original. Ademais, não existem garantias sobre o tempo de execução do novo código, podendo apresentar tempo de execução muito extenso. Isso se torna ainda mais provável dado que a linguagem original é mais otimizada para resolução de problemas matemáticos enquanto a linguagem destino tem como maior enfoque facilitar a criação de qualquer tipo de aplicação, ao custo da velocidade de execução e utilização de memória.

De modo alternativo, seria possível utilizar uma ferramenta de tradução entre as duas linguagens. Isso poderia de certa forma se enquadrar na possibilidade anterior, mas com grande melhora na velocidade de tradução e maiores garantias sobre o código gerado. Entretanto dentre todas as ferramentas encontradas [13][14][15] nenhuma conseguiu fazer a tradução de forma aceitável, cada uma delas teve uma combinação específica de três possíveis problemas: realizar a tradução da sintaxe sem se preocupar com a viabilidade de rodar o código; gerar código Python de uma versão já depreciada; e não conseguir reconhecer diversas linhas do código MATLAB original.

Finalmente, a possibilidade restante seria utilizar uma ferramenta de tradução que pudesse converter o código do modelo para uma linguagem intermediária que não precisasse de licença para o seu uso e conseguisse se comunicar com o Python. A linguagem escolhida para esse papel foi a linguagem C, pois além de existirem ferramentas nativas da MathWorks [3] que fazem essa tradução existe também uma biblioteca padrão do Python [9] que permite a interação entre as duas linguagens.

## 4.1.2 Tradução do Código MATLAB para a Linguagem C

Como explicado na Seção 3.1, a empresa MathWorks disponibiliza uma aplicação para o MATLAB, chamada MATLAB Coder, que permite a tradução automática de um código MATLAB para C ou C++. Essa ferramenta, entretanto, possui algumas limitações devido à grande quantidade de diferença entre os paradigmas de cada uma das linguagens. O primeiro problema encontrado foi a necessidade de definir o tamanho e tipo das variáveis utilizadas como parâmetro das funções MATLAB, dada a natureza da linguagem C como fortemente tipada, e sua necessidade de determinar no cabeçalho de cada função exatamente os tipos dos parâmetros. Nas imagens 22 e 23 é possível ver as diferenças de tipagem nos cabeçalhos das duas linguagens, e na figura 24 é mostrada a tela do MATLAB Coder na qual é possível identificar os tipos dos parâmetros.

Header%20MATLAB

Figura 22 – Cabeçalho de uma função MATLAB.

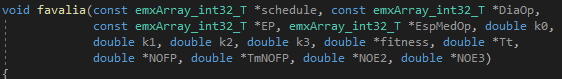


Figura 23. Cabeçalho de uma função C.

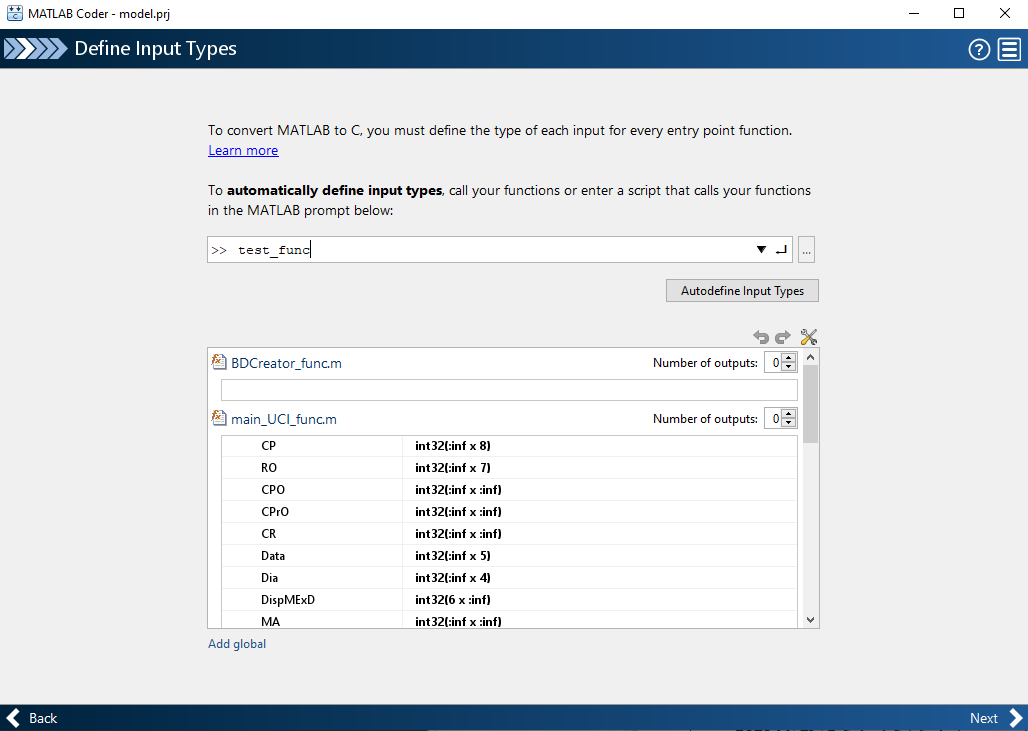


Figura 24. Tela do MATLAB Coder de definição de tipo dos parâmetros.

Após determinar os tipos das variáveis de parâmetros, outra necessidade foi fazer a declaração e inicialização explicitamente das variáveis no início do código, como é possível ver na figura 25, outro dos pré-requisitos da linguagem C na utilização de variáveis.

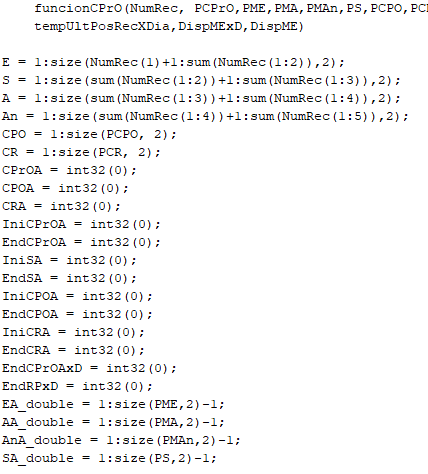


Figura 25. Inicialização de variáveis no MATLAB.

A incompatibilidade encontrada a seguir se deu com as linhas de código na implementação original do modelo que alterava de forma dinâmica o tamanho de uma variável ao longo do código. Esse problema englobou instâncias fáceis e difíceis de resolver, enquanto algumas foram resolvidas simplesmente trocando um *append* por uma inicialização prévia e alteração por indexação, em alguns casos foi necessário rever a lógica utilizada para que o código MATLAB não necessitasse de variáveis que modificassem o seu tamanho dentro de uma mesma função.

Alguns dos problemas encontrados também foram mais específicos da forma que o modelo foi implementado e das concessões que o MATLAB faz quando a programação é feita diretamente nele. Por exemplo, em um dos casos encontrados o código original incluía a multiplicação de uma matriz por um número (escalar), mas o código que fazia essa multiplicação especificava que o tipo de multiplicação a ser feito não seria possível fazer entre uma matriz e um escalar, mas seria entre uma matriz e um vetor. Ao executar esse código dentro do MATLAB o programa implicitamente transformava o segundo parâmetro da multiplicação em um vetor (replicando o escalar até o tamanho compatível com a matriz), entretanto, quando o MATLAB Coder fazia a leitura da expressão ele apontava a inconsistência no tamanho dos mesmos. Na implementação original também, os dados dos pacientes e dos hospitais foram salvos simplesmente como matrizes dentro de um arquivo “. mat”, uma extensão específica do MATLAB que identifica um arquivo binário guardando uma ou mais variáveis, mas que não é facilmente lido por outras linguagens (análogo ao uso base da biblioteca *pickle* do Python).

Foi necessário então transformar o arquivo de *input* em diversos arquivos *csv* (*comma separated variable* – variáveis separadas por vírgula) que pudesse descrever as matrizes utilizadas de uma forma legível por qualquer programa. Após isso, foi criada também uma função simples de leitura dos *csv* em C, pois enquanto na arquitetura completa esse trabalho seria feito pelo código em python utilizando uma biblioteca especializada, seria necessário que o código C pudesse ser testado com os arquivos csv para que a tradução pudesse ser testada.

Depois da resolução dos problemas identificados anteriormente o MATLAB Coder passou a conseguir gerar um código C a partir do modelo. Este código não funcionou corretamente em um primeiro momento, sendo necessárias algumas alterações no código MATLAB para que o código gerado retornasse as respostas esperadas. Entretanto, a partir desse momento os problemas poderiam aparecer não só nas mensagens de erro do MATLAB Coder, mas também na observação e execução do código em C.

Um problema inesperado encontrado na tradução feita foi a otimização excessiva de algumas partes do código. Inicialmente a tradução estava sendo feita em uma versão do código com o menor número possível de gerações para o modelo (uma geração), permitindo que a conversão fosse feita da forma mais rápida possível para identificar os problemas a serem resolvidos rapidamente, entretanto o código gerado a partir dessa tradução passou a considerar que o modelo sempre teria somente uma geração retirando diversas iterações que deveriam ser feitas e, além disso, retirando a menção de qualquer uma das variáveis que tinham o seu valor definido na função *main* do modelo, considerando que a definição delas estava dentro do código, o seu valor era para ser uma constante.

Em seguida, foi necessário aumentar o número de gerações utilizado no código a ser traduzido e criar uma nova função *main* que simplesmente decidisse o valor dos parâmetros do modelo e chamasse a função *main* original. Esse novo código não seria colocado como parte do código a ser traduzido, e sim como uma função de teste que chama o código a ser traduzido, essa funcionalidade do MATLAB Coder pode ser vista na figura a seguir.

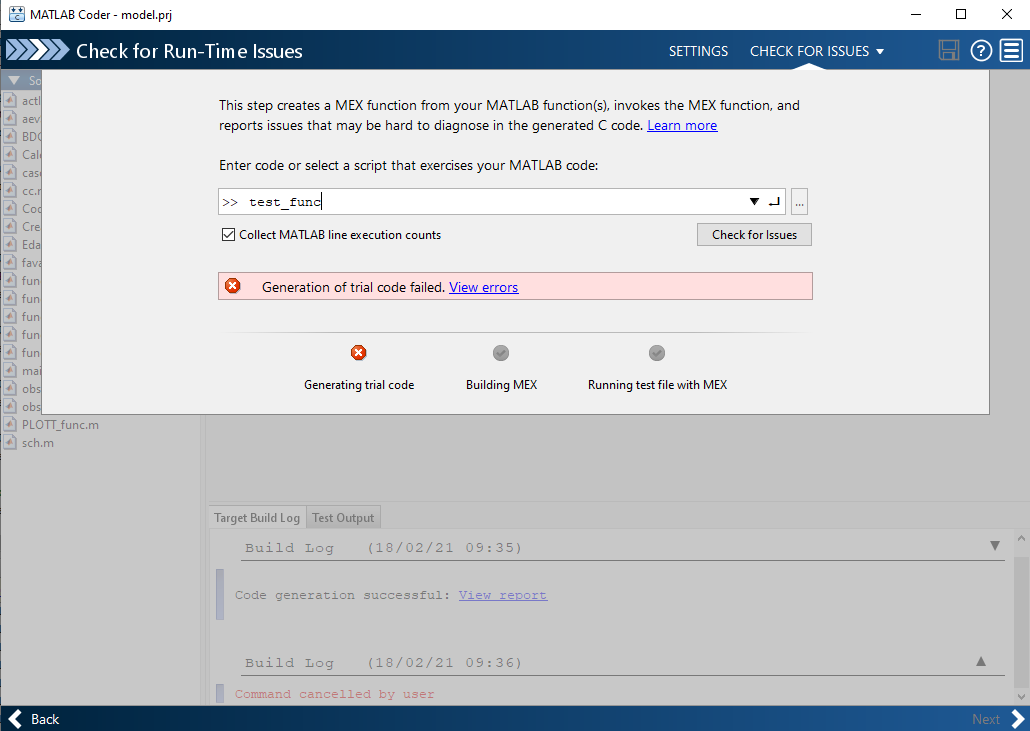


Figura 26. Definição da função de teste.

Outra necessidade foi alterar todas as partes em que no código original foi utilizado uma das funções *sum*, *find* e *max* em uma matriz, para dizer explicitamente em qual a dimensão na qual fazer a aplicação da função, dado que apesar de isso poder ser determinado implicitamente dentro do MATLAB pelo contexto de uso, a seleção automática feita pela tradução pode não funcionar adequadamente, causando erros no código final. Apesar de não ser absolutamente essencial, essa é uma boa prática ao utilizar o Coder que pode poupar erros futuros.

Após este passo, apesar do MATLAB Coder não apontar mais nenhum problema com o código original, a versão gerada ainda não estava funcionando como esperado. Sendo assim foram necessárias alterações subsequentes nas configurações do tradutor, tanto com o objetivo de esclarecer a tradução e facilitar a resolução de bugs quanto para corrigir os problemas encontrados. As alterações foram a ativação das opções “*Report differences from MATLAB*” e “*MATLAB source code as comments*”, a desativação da “*Saturate on integer overflow*”, e a configuração das opções: “*Build Configuration*”, “*Dynamic memory allocation*”, “*Preserve variable names*”, e “*Preserve array dimensions*” para “*Faster runs*”, “*For all variable sized arrays*”, “*User names*”, e “*Yes - Column major*” respectivamente.

A última dessas alterações foi feita com o objetivo de acelerar e corrigir a execução dos *loops* no código C, além de torná-lo mais legível. Esta alteração tem como função alterar o comportamento do tradutor que, por padrão, transforma uma matriz n dimensional em um vetor C (equivalente a uma matriz unidimensional) com o tamanho suficiente para guardar os dados da matriz inteira. Entretanto, ela não pareceu ter o efeito esperado.

Para tentar corrigir este problema foi instalado o Embeded Coder, uma extensão do MATLAB Coder feita para otimizar os códigos gerados pelo mesmo para maior eficiência e facilitar integração com outros códigos [16]. A instalação alterou os comentários gerados automaticamente e melhorou a tabulação, entretanto os dados continuaram sendo representados no código como vetores e os problemas anteriores se mantiveram. Sendo assim, em uma segunda tentativa, a opção foi alterada para “*Yes – row major*” (a diferença entre essa e a anterior se dá na ordem de armazenamento em memória das dimensões de uma matriz), com isso, apesar da representação dos dados no código C ainda ser a partir de vetores, o código se tornou muito mais rápido e rodou perfeitamente.

Além do resultado de ambos os códigos serem semelhantes, após a alteração a tradução passou a ser mais rápida do que o original. Por exemplo, enquanto o modelo no MATLAB, ordenando um total de 2000 (duas mil) operações roda um total de 50 (cinquenta) gerações em aproximadamente 42,5h (quarenta e duas horas e meia), o modelo consegue rodar a mesma quantidade de gerações em 10h (dez horas) utilizando a implementação em C.

## 4.1.3 Conexão do Modelo em C com a Linguagem Python

Tendo o modelo pronto em C, o passo seguinte passou a ser a utilização das funções do modelo a partir do Python. Para esse tipo de atividade a linguagem disponibiliza o CTypes, uma biblioteca padrão que implementa funções e classes que permitem a interação com código em C desde que compilado em uma biblioteca compartilhada.

Entretanto, a compilação de uma biblioteca compartilhada, geralmente “.DLL” no Windows e “.so” em sistemas UNIX é dependente do sistema operacional e acabou gerando dificuldades. Inicialmente foi utilizado o compilador padrão do Visual Studio, mas o Python não conseguiu reconhecer o arquivo gerado como um DLL válido, apesar de diversos testes com diferentes classes de DLL do CTypes e diversas configurações do Visual Studio. Após certo tempo foi levantada a possibilidade de utilizar o GCC para compilar o código como “.so”, mesmo que no Windows, dada a maior familiaridade com a ferramenta e a disponibilização pelo CTypes de classes que poderiam lidar com isso. A partir disso, foi utilizada a ferramenta “Mingw para Windows” que permitiu a utilização do GCC para fazer a compilação.

Com isso foi possível gerar um arquivo “.so” legível pela biblioteca, mas, interessantemente, a classe gerada a partir da leitura do mesmo não possuía nenhuma das funções definidas no código C como atributo. Apesar disso, após explorar diversos outros métodos de compilar o modelo foi descoberto que os atributos necessários são “mágicos”, esse é um paradigma de programação utilizado ocasionalmente no Python que significa que o atributo não é visível, mas que ao ser chamado ele aparece e funciona “magicamente”. Ao descobrir isso, passou a ser possível chamar as funções do modelo em C pelo Python. Então, com um estudo maior da biblioteca CTypes, e a criação de funções auxiliares em C para facilitar a inicialização de Structs a partir de dados do Python, foi possível reescrever a função *“main”* a partir do Python chamando as funções do modelo em C.

Outra alteração que pode ser feita foi o a leitura dos arquivos csv pelo Python utilizando a biblioteca Pandas, o que acelerou bastante o modelo, dado que a função de leitura escrita em C foi implementada da forma mais simples possível, para permitir a execução de testes do código sem gastar muito tempo em uma função temporária. Com isso, o modelo passou a rodar as 50 gerações de antes em um total de 8h30min, 5x mais rápido do que a implementação original. Para exemplificar, as figuras 4.2.3-1 e 4.2.3-2 mostram os tempos de execução de 25 gerações pelo MATLAB e pelo Python, no caso de menos gerações a diferença fica um pouco menor (4x mais rápido), pois o MATLAB demora mais nas gerações de maior número, provavelmente dado algum acúmulo de informação armazenada em memória.

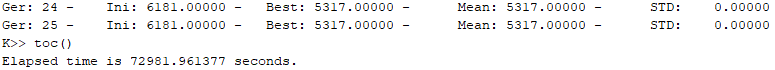


Figura 27. Execução de 25 gerações no MATLAB em 20h16min.

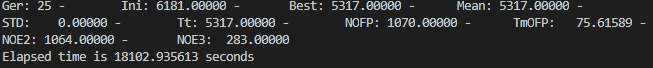


Figura 28. Execução de 25 gerações no Python em 5h.

## 4.1.4 Desenvolvimento da Interface

Após conseguir chamar o modelo a partir do Python foi possível fazer a programação da interface em si. Para auxiliar nessa tarefa foi utilizada a biblioteca Flask do Python que permite a definição de templates HTML com certas funcionalidades do Python. As maiores dificuldades nesse estágio foram principalmente no planejamento das telas, por vezes com a mudança de quais informações precisavam ser mostradas eram necessárias alterações em todos os módulos diferentes do projeto.

# 5 Conclusão

O principal objetivo desse trabalho foi a criação de uma interface que pudesse permitir a utilização do modelo de otimização de fila de hospital AEIQ-AS, permitindo a organização da fila de cirurgias e o planejamento de verbas do mesmo sem necessidade do conhecimento de computação. Para alcançar o objetivo proposto foi feita a tradução da implementação original do modelo para C, de forma que ele poderia ser utilizado sem a licença do MATLAB, a alteração do código gerado para permitir acesso às informações pelo código da interface e por fim a criação da interface em si. Isto foi feito utilizando diversas ferramentas que requisitaram estudo do funcionamento e por vezes a busca de outras que poderiam realizar o trabalho de forma mais eficiente. Com a realização desse projeto foi possível realizar um estudo dos modelos mais recentes de algoritmos genéticos e de diversas ferramentas diferentes. Além disso, nele foi criado um projeto com diversas linguagens e ferramentas trabalhando em sintonia.

Durante a realização do projeto, foram encontradas diversas dificuldades não somente na parte prática, como também no planejamento da ferramenta. A busca pela melhor solução para conseguir utilizar o modelo a partir da linguagem Python, por exemplo, sofreu de diversas iterações por ser um problema que não possuía uma resposta única e correta, sendo necessário investigar diversas possíveis soluções para poder descobrir os lados positivos e negativos de cada uma delas. Outra dificuldade foi o fato do projeto ser composto de diversos módulos trabalhando de forma quase independente, pois quando ficava claro que era necessária uma informação nova para o HTML da interface, por exemplo, era necessário rastrear as alterações em diversos módulos para que a informação fosse coletada e passada adiante pelos caminhos corretos. Uma parte positiva e inesperada no começo do trabalho foi a grande diminuição no tempo total de execução do modelo utilizado, que ao final desse projeto passou a rodar aproximadamente 5x mais rápido.

Para trabalhos futuros seria interessante incluir no modelo a funcionalidade de descrever mais detalhadamente os horários dos diferentes médicos. Permitindo que os horários de chegada e saída de um médico sejam determinados para cada dia, e que o cronograma de um mesmo médico possa variar ao longo do tempo, dessa forma, além do modelo se tornar mais realista, ele também conseguiria lidar com situações fora do padrão, como contratações, demissões e férias de pessoal. No entanto, ainda mais interessante seria a aplicação do modelo e da interface em uma unidade de saúde real não só melhorando o projeto a partir de testes em condições reais, mas também levando o projeto adiante e com isso ajudando o sistema de saúde para que ele possa atender cada vez melhor os seus pacientes.

# 6 Referências Bibliográficas

[1] ESPERANÇA, A. C.; CAVALCANTE, R. B.; MARCOLINO, C. **Estudo da demanda espontânea em uma unidade de saúde da família de uma cidade de médio porte do interior de Minas Gerais, Brasil**. REME – Rev. Min. Enf; 10(1) p. 30-36, jan./mar., 2006.

[2] GONZÁLEZ, R.; VELLASCO, M.; FIGUEIREDO, K. **Otimização de Recursos para Procedimentos Cirúrgicos Eletivos Utilizando Algoritmos Genéticos com Inspiração Quântica**. 78p. Dissertação (Mestrado em Engenharia Elétrica) – Pontifícia Universidade Católica do Rio de Janeiro. 2018.

[3] MathWorks. **MATLAB Coder**. Disponível em: <https://www.mathworks.com/products/matlab-coder.html> Acesso em: 10 dez. 2020.

[4] MathWorks. **MATLAB**. Disponível em: <https://www.mathworks.com/products/matlab.html> Acesso em: 10 dez. 2020.

[5] [ISO/IEC 9899:1999 specification](http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf) (PDF). [S.l.: s.n.] p. 313, § 7.20.3 "Memory management functions"

[6] **GCC, the GNU Compiler Collection**. Disponível em: <<https://gcc.gnu.org/>> Acesso em: 01 dez. 2020.

[7] Kuhlman, D. [**A Python Book: Beginning Python, Advanced Python, and Python Exercises**](https://web.archive.org/web/20120623165941/http:/cutter.rexx.com/~dkuhlman/python_book_01.html). Open Source Initiative, 2013.

[8] **Python Package Index (PyPI)**. Disponível em: <<https://pypi.org/>> Acesso em:01 dez. 2020.

[9] **Ctypes, A foreign function library for Python**¶. Disponível em: <<https://docs.Python.org/3/library/ctypes.html>> Acesso em: 02 dez. 2020.

[10] The Pallets Project. **FLASK**. Disponível em: <<https://palletsprojects.com/p/flask>> Acesso em 10 dez. 2020.

[11] Docforge. [**Web application framework**](https://web.archive.org/web/20150723163302/http:/docforge.com/wiki/Web_application_framework). Disponível em: <https://web.archive.org/web/20150723163302/http:/docforge.com/wiki/Web\_application\_framework> Acesso em: 16 dez. 2020

[12] MathWorks. **Calling MATLAB from Python.** Disponível em: <<https://www.mathworks.com/help/matlab/matlab-engine-for-python.html>> Acesso em 10 dez. 2020.

[13] **LiberMate**. Disponível em: <https://github.com/awesomebytes/libermate> Acesso em: 11 dez. 2020.

[14] **Smop**. Disponível em: <<https://github.com/victorlei/smop>> Acesso em: 11 dez. 2020.

[15] Open-Source MATLAB®-to-Python® Compiler (**OMPC**) Disponível em: <<http://ompc.juricap.com/>> Acesso em: 03 dez. 2020.

[16] MathWorks. **Embedded Coder**. Disponível em: <<https://www.mathworks.com/products/embedded-coder.html>> Acesso em: 05 dez. 2020.

[17] REJER, I. **Genetic algorithm with aggressive mutation for feature selection in BCI feature space**. Pattern Anal. Appl., v. 18, n. 3, p. 485–492, aug. 2015.

[18] ZHANG, G. & GEXIANG. **Quantum-inspired evolutionary algorithms: a survey and empirical study**. J. Heuristics, v. 17, n. 3, p. 303–351, jun. 2011.

[19] TESTI, A. *et al*. **Prioritizing surgical waiting lists**. J. Eval. Clin. Pract., v. 14, n. 1, p. 59–64, jan. 2008.

[20] SILVEIRA, L. R.; TANSCHEIT, R.; VELLASCO, M. **Quantum inspired evolutionary algorithm for ordering problems**. Expert Syst. Appl., v. 67, p. 71–83, jan. 2017.

[21] CHAURASIA, S. N.; SUNDAR, S.; SINGH, A. **Hybrid metaheuristic approaches for the single machine total stepwise tardiness problem with release dates**. Oper. Res., v. 17, n. 1, p. 275–295, apr. 2017.

[22] SILVEIRA, L. R.; TANSCHEIT, R.; VELLASCO, M. **Algoritmo Genético de Ordem com Inspiração Quântica**. 141p. Dissertação (Mestrado em Engenharia Elétrica) – Pontifícia Universidade Católica do Rio de Janeiro. 2014