



LEONARDO LUIZ ANDRADE ROCHA

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

Projeto Final de Projeto de Conclusão de Curso,
apresentado ao curso de **Engenharia de Computação** da PUC-
Rio como requisito parcial para a obtenção do título de
Engenheiro de Computação.

Orientadoras: Marley Vellasco
Karla Figueiredo

Rio de Janeiro
Dezembro de 2020.

Sumário

1. Introdução
 - 1.1. Objetivo
2. Algoritmo Evolutivo com Inspiração Quântica para a Área de Saúde AEIQ-AS
 - 2.1. Algoritmos Genéticos
 - 2.2. Computação Quântica
 - 2.3. Algoritmos Genéticos com Inspiração Quântica
 - 2.4. AEIQ-AS
3. Tecnologias Utilizadas
 - 3.1. MATLAB Coder
 - 3.2. Compilador GCC
 - 3.3. Python
 - 3.3.1. CTypes
 - 3.3.2. Flask
4. Projeto e Especificação
 - 4.1. Etapas do Projeto e Problemas Encontrados
 - 4.1.1. Tradução do Código MATLAB para a Linguagem Python
 - 4.1.2. Tradução do Código MATLAB para a Linguagem C
 - 4.1.3. Conexão do Modelo C com a Linguagem Python
 - 4.1.4. Desenvolvimento da Interface
5. Conclusão
6. Referências Bibliográficas

1 Introdução

No Brasil e em diversas partes do mundo a demanda da população por Unidades de Saúde é maior do que a capacidade das mesmas de atender aos pacientes, conforme explicitado em [1], tornando-se comum o surgimento de filas de espera. Entretanto, planejar as mesmas é um grande desafio devido à grande quantidade de recursos que devem ser considerados. Os procedimentos cirúrgicos, particularmente, são um caso especialmente crítico, pois necessitam de diversos recursos diferentes para sua realização, tanto de recursos humanos, médicos especialistas, anestesistas, enfermeiros e instrumentadores, quanto de recursos materiais, como os diversos 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 de tal 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, qualquer atraso gerado pela mesma pode provocar uma deterioração grave no estado do paciente e possivelmente levar inclusive à morte do mesmo.

O modelo utilizado para esse processo 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 no atendimento 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, uma das características principais 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 volume de dados grande. Para isso foram considerados os modelos com inspiração quântica que simultaneamente diminuem 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 dos mesmos
- 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;
- 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 que foi 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 a tradução do modelo para uma linguagem na qual fosse possível tanto a execução do mesmo quanto a criação da interface.

Dada à necessidade de ser uma aplicação multiplataforma, 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.

Com isso, foram necessárias 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 mais quatro capítulos, como descrito a seguir:

No Capítulo 2 é apresentado o modelo utilizado. Nele são definidos os conceitos necessários para o entendimento do mesmo, além da explicação do modelo em si. 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.

O Capítulo 4 apresenta a interface criada para a interação com o modelo de otimização de filas. Além disso, o capítulo detalha cada uma das fases do desenvolvimento do projeto.

Finalmente, 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. 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, o qual define quais indivíduos devem sobreviver e pertencer à próxima geração, 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 2.1-1 aonde a inicialização da população é seguida por um ciclo de gerações.

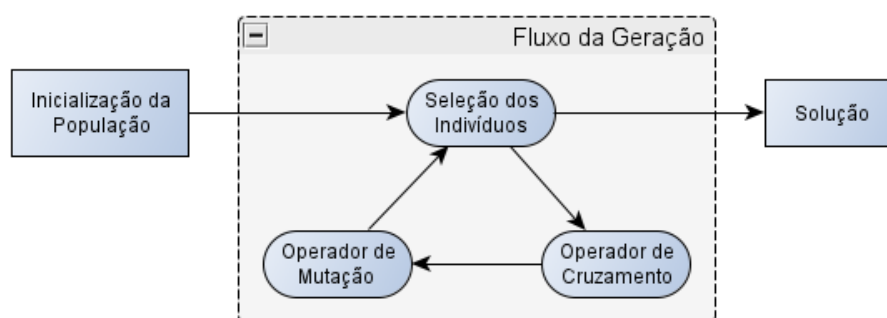


Figura 2.1-1 – Fluxo base de um algoritmo genético

O primeiro passo necessário, a codificação do cromossomo, 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 mais bem codificada por um vetor de identificadores dos elementos a serem

ordenados. A única restrição é de que a representação que contenha informações suficientes para reconstituir a solução relacionada à mesma, mas como será visto adiante cada operador espera certos tipos de codificação, incentivando o uso de representações mais tradicionais.

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, a mesma 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.

Tendo essas características decididas, 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, após isso poderá ser iniciada a primeira geração.

Iniciando a geração, a seleção dos indivíduos decide algoritmicamente quais serão os indivíduos que poderão participar do ciclo, descartando os outros. Com o objetivo de simular a seleção natural, a mesma dá preferência aos cromossomos mais bem avaliados, dando probabilidades menores dos mesmos 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.

Dentro de uma geração são utilizados dois operadores diferentes para gerar os novos indivíduos. O 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 crossover uniforme baseado em ordem [21] aonde 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 2.1-2.

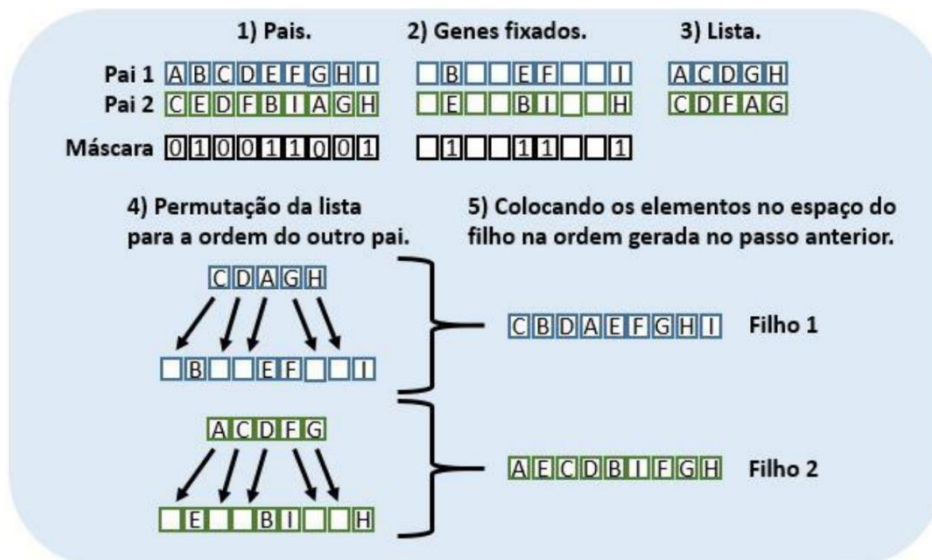


Figura 2.1-2 – Crossover Uniforme Baseado em Ordem

Diferentemente do operador de cruzamento, que gera indivíduos novos a partir da geração anterior, o operador de mutação tem como objetivo aumentar a variabilidade nova 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 aspecto dos algoritmos genéticos que pode ser aplicado é o conceito do 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 [19,18] 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.

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 de se ter n Qbit, cada um dos 2^n 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 também faz o mesmo com as portas lógicas, criando os Q-Gates. Basicamente, os Q-Gates são 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 onde os indivíduos quânticos são representados por bits quânticos e devem ser 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 m Qbits, cada qual podendo assumir qualquer elemento do universo a ser ordenado com certa probabilidade, e o somatório de cada probabilidade sendo 1 para cada m Qbit. De forma equivalente, cada indivíduo quântico pode ser representado por uma matriz $n \times m$ onde cada linha representaria um Qbit, com as suas colunas sendo a probabilidade do Qbit ser observado como o elemento específico, ou seja:

$$Q = \begin{bmatrix} q_{11} & \cdots & q_{1m} \\ \vdots & \ddots & \vdots \\ q_{n1} & \cdots & q_{nm} \end{bmatrix} \text{ onde } q_{ij} \in [0,1] \text{ e } \sum_{j=1}^m q_{ij} = 1, \forall i \in \{1, \dots, n\}$$

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 visto 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:

$$q_{ij} = \frac{1}{m}, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, m\}$$

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 preferencialmente 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:

$$q_{ij}' = \frac{q_{ij}}{1 - q_{ip}}, \forall i \in \{r + 1, \dots, n\}$$

Onde q_{ip} 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 $x \in (0,1]$ e selecionar o k -ésimo elemento tal que:

$$\begin{aligned} q_{i1} &> x && \text{Para } k = 1 \\ \sum_{j=1}^k q_{ij} &> x \text{ e } \sum_{j=1}^{k-1} q_{ij} &\leq x & \text{Para } k > 1 \end{aligned}$$

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

- Indivíduo inicial:

$$Q = \begin{bmatrix} 0.\bar{3} & 0.\bar{3} & 0.\bar{3} \\ 0.\bar{3} & 0.\bar{3} & 0.\bar{3} \\ 0.\bar{3} & 0.\bar{3} & 0.\bar{3} \end{bmatrix}$$

- Primeiro elemento selecionado ($x=0.45$) e atualização:

$$Q = \begin{bmatrix} 0 & 1 & 0 \\ 0.\bar{3}/1-0.\bar{3} & 0 & 0.\bar{3}/1-0.\bar{3} \\ 0.\bar{3}/1-0.\bar{3} & 0 & 0.\bar{3}/1-0.\bar{3} \end{bmatrix}$$

- Matriz atualizada:

$$Q = \begin{bmatrix} 0 & 1 & 0 \\ 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 \end{bmatrix}$$

- Segundo elemento selecionado ($x=0.12$) e atualização:

$$Q = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0.5/1-0.5 \end{bmatrix}$$

- Observação do indivíduo:

$$Q = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Conversão em indivíduo clássico

$$C = [2 \quad 1 \quad 3]$$

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:

$$Q^{tr} = (1 - \varepsilon) * Q^t + \varepsilon * E^t, \quad \varepsilon \in (0,1]$$

onde Q^t é o indivíduo quântico original, Q^{tr} o indivíduo quântico atualizado, E^t a observação selecionada e ε a taxa de atualização. Por exemplo:

$$\text{Dado: } Q^t = \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix} \text{ e } E^t = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\text{Temos: } Q^t = (1 - \varepsilon) * \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix} + \varepsilon * \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\text{Ou seja: } Q^t = \begin{bmatrix} (1 - \varepsilon)/3 & (1 - \varepsilon)/3 & (1 + 2\varepsilon)/3 \\ (1 + 2\varepsilon)/3 & (1 - \varepsilon)/3 & (1 - \varepsilon)/3 \\ (1 - \varepsilon)/3 & (1 + 2\varepsilon)/3 & (1 - \varepsilon)/3 \end{bmatrix}$$

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 E^t a partir de Q^t podendo chegar a 100% no caso em que $\varepsilon = 1$. A escolha de ε 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 o algoritmo escolhido. 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 2.4-1, 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 2.4-1 – Diagrama do processo cirúrgico

Para determinar o nível de prioridade de um paciente, o mesmo 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 2.4-2.

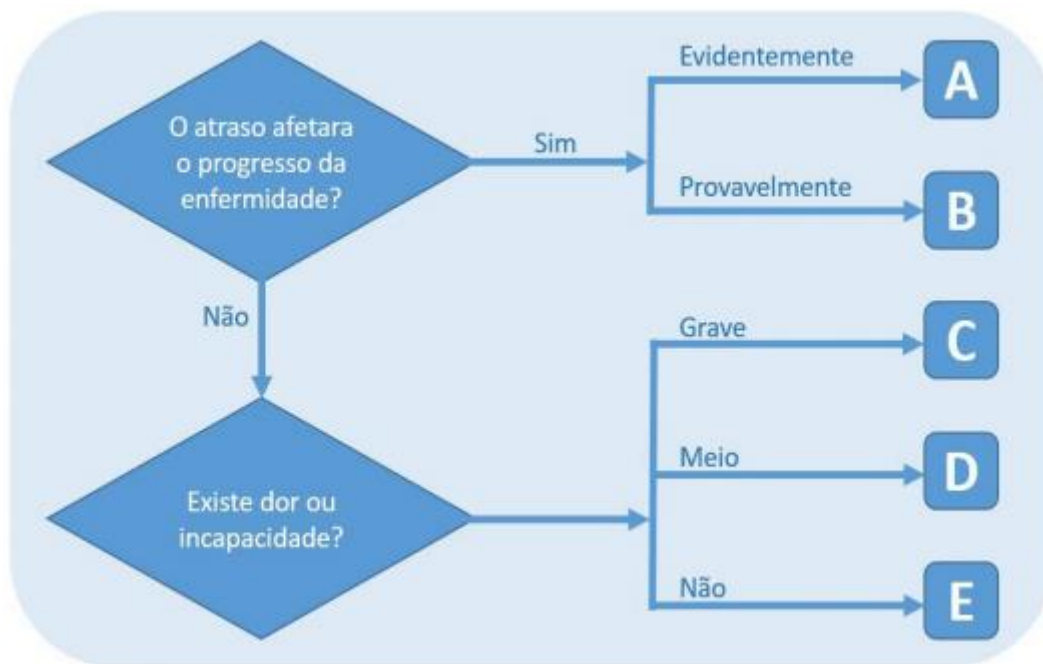


Figura 2.4-2 – Árvore de decisão para prioridade do paciente

Como mencionado anteriormente, também é necessário considerar os recursos necessários para a execução da cirurgia, dentre eles os recursos materiais e os recursos humanos. Entre os recursos materiais 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, tanto os anestesiistas, quanto os especialistas, quanto 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 2.4-3 é possível visualizar todos os tipos de recursos necessários considerados pelo modelo.



Figura 2.4-3 – Classificação dos recursos

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. As funções 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;
 - A quantidade de cirurgias realizadas por médicos não especialistas, 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, onde a biblioteca CTypes permite a leitura da biblioteca exportada e o Flask permite a criação da interface em si utilizando o código Python. Na Figura 3-1 esse fluxo é representado de forma gráfica para facilitar o entendimento.

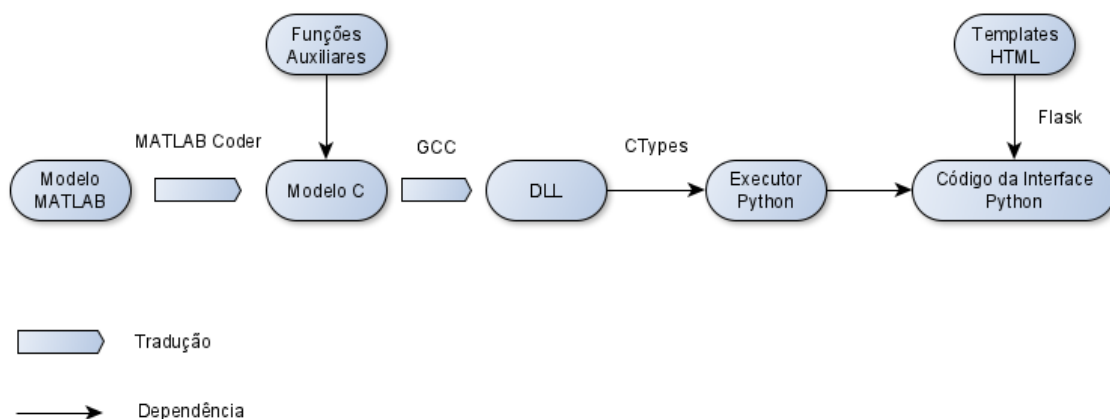


Figura 3-1 – Fluxo de geração da interface

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++, permitindo a utilização do código em outros projetos e sem a necessidade de uma licença MATLAB para executar o código, ele permite também a geração de um código MEX pré-compilado com o objetivo de que ele terá 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. É necessário:

- *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 do GCC para Windows.

3.3 Python

Como MATLAB, Python também é uma linguagem interpretada, multiparadigma, e de alto nível, mas diferentemente do primeiro o segundo 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 4.1-1 é mostrado um gráfico de dependência dos *templates* HTML e as Figuras 4.1-2 a 4.2-14 exemplificam as possíveis visualizações das telas.

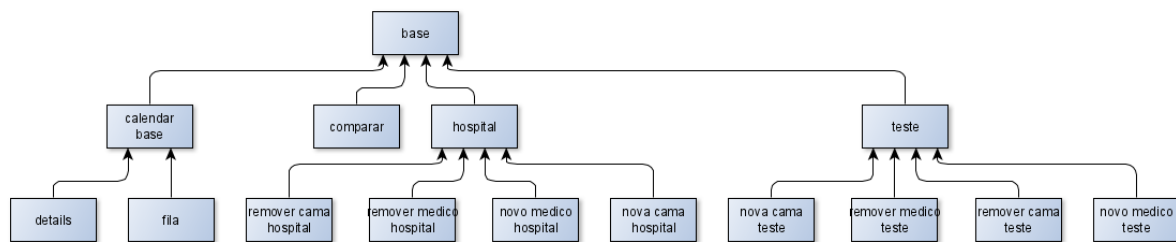


Figura 4.1-1 – Tabela de dependência dos *templates* HTML

A Figura 4.1-2 mostra página inicial da interface que permite a visualização do calendário de cirurgias, quando o modelo já está sendo rodado, uma mensagem de texto substitui o botão “Rodar Modelo” como mostrado na Figura 4.1-3. É possível filtrar para ver somente um dia, como mostrado na Figura 4.1-4 ou clicar em uma das operações para ver os requisitos necessários, como na Figura 4.1-5.



Interface do Modelo

Fila

Hospital

Testar Alterações

Comparar Alterações

< > today

January 2017

month day

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
	7a Operacao 7	1a Operacao 16	12a Operacao 20	3a Operacao 62	3a Operacao 69	2a Operacao 64
	7a Operacao 8	2a Operacao 14	2a Operacao 26	7a Operacao 31	3a Operacao 72	7a Operacao 26
	7a Operacao 1	3a Operacao 33	2a Operacao 27	7a Operacao 37	7a Operacao 45	7a Operacao 70
	+55 more	+65 more	+61 more	+62 more	+66 more	+56 more
8	9	10	11	12	13	14
Operacao 13	7a Operacao 71	1a Operacao 77	2a Operacao 96	3a Operacao 114	12a Operacao 113	1a Operacao 121
Operacao 30	7a Operacao 67	2a Operacao 83	3a Operacao 103	3a Operacao 141	1a Operacao 116	1a Operacao 123
Operacao 35	7a Operacao 76	2a Operacao 92	3a Operacao 108	3a Operacao 160	1a Operacao 120	1a Operacao 131
+31 more	+58 more	+59 more	+70 more	+67 more	+56 more	+51 more
15	16	17	18	19	20	21
Operacao 73	7a Operacao 128	2a Operacao 151	3a Operacao 189	2a Operacao 174	1a Operacao 183	2a Operacao 208
Operacao 78		3a Operacao 135	7a Operacao 150	3a Operacao 163	2a Operacao 185	3a Operacao 199
Operacao 94	7a Operacao 134	3a Operacao 153	7a Operacao 156	3a Operacao 169	2a Operacao 186	7a Operacao 187
+27 more	+52 more	+50 more	+51 more	+63 more	+61 more	+54 more
22	23	24	25	26	27	28
Operacao 128	7a Operacao 171	1a Operacao 196	1a Operacao 216	3a Operacao 221	2a Operacao 229	1a Operacao 239
Operacao 146	7a Operacao 181	2a Operacao 204	3a Operacao 226	3a Operacao 313	3a Operacao 333	2a Operacao 246
Operacao 165	7a Operacao 203	2a Operacao 209	3a Operacao 230	3a Operacao 321	7a Operacao 240	2a Operacao 256
+28 more	+63 more	+62 more	+60 more	+61 more	+60 more	+52 more
29	30	31	1	2	3	4
Operacao 181	7a Operacao 244	3a Operacao 261	2a Operacao 267	2a Operacao 286	2a Operacao 296	1a Operacao 305
Operacao 227	7a Operacao 247	3a Operacao 280	3a Operacao 271	2a Operacao 291	3a Operacao 303	1a Operacao 311
Operacao 198		3a Operacao 337	3a Operacao 273	3a Operacao 294	7a Operacao 299	3a Operacao 314
+28 more	+52 more	+64 more	+72 more	+63 more	+60 more	+57 more
5	6	7	8	9	10	11
Operacao 244	7a Operacao 317	1a Operacao 324	1a Operacao 338	2a Operacao 348	12a Operacao 368	1a Operacao 377
Operacao 275	7a Operacao 320	1a Operacao 325	3a Operacao 344	3a Operacao 353	2a Operacao 375	2a Operacao 379
Operacao 279	7a Operacao 318	2a Operacao 319	3a Operacao 352	3a Operacao 401	3a Operacao 367	3a Operacao 380
Operacao 258		2a Operacao 329	3a Operacao 362	3a Operacao 415	3a Operacao 436	3a Operacao 386
+29 more	+55 more	+57 more	+62 more	+73 more	+65 more	+54 more

Rodar Modelo

[HTML5 Templates](#)

Figura 4.1-2 – Página “Fila” sem o modelo estar rodando e no modo “month”



Interface do Modelo

[Fila](#)[Hospital](#)[Testar Alterações](#)[Comparar Alterações](#)

< > today

January 2017

month day

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
	7a Operacao 7 7a Operacao 8 7a Operacao 1 +55 more	1a Operacao 16 2a Operacao 14 3a Operacao 33 +65 more	12a Operacao 20 2a Operacao 25 2a Operacao 27 +61 more	3a Operacao 62 7a Operacao 31 7a Operacao 37 +82 more	3a Operacao 69 3a Operacao 72 7a Operacao 45 +89 more	2a Operacao 64 7a Operacao 26 7a Operacao 70 +56 more
8	9	10	11	12	13	14
Operacao 13 Operacao 30 Operacao 35 +31 more	7a Operacao 71 7a Operacao 67 7a Operacao 76 +58 more	1a Operacao 77 2a Operacao 83 2a Operacao 92 +59 more	2a Operacao 96 3a Operacao 103 3a Operacao 108 +70 more	3a Operacao 114 3a Operacao 141 3a Operacao 160 +87 more	12a Operacao 113 1a Operacao 116 1a Operacao 120 +56 more	1a Operacao 121 1a Operacao 123 1a Operacao 131 +51 more
15	16	17	18	19	20	21
Operacao 73 Operacao 78 Operacao 94 +27 more	7a Operacao 128 7a Operacao 134 +52 more	2a Operacao 151 3a Operacao 135 3a Operacao 153 +50 more	3a Operacao 189 7a Operacao 150 7a Operacao 156 +51 more	2a Operacao 174 3a Operacao 163 3a Operacao 169 +83 more	1a Operacao 183 2a Operacao 185 2a Operacao 186 +61 more	2a Operacao 208 3a Operacao 199 7a Operacao 187 +54 more
22	23	24	25	26	27	28
Operacao 128 Operacao 146 Operacao 165 +28 more	7a Operacao 171 7a Operacao 181 7a Operacao 203 +63 more	1a Operacao 196 2a Operacao 204 2a Operacao 209 +62 more	1a Operacao 216 3a Operacao 226 3a Operacao 230 +60 more	3a Operacao 221 3a Operacao 313 3a Operacao 321 +61 more	2a Operacao 229 3a Operacao 333 7a Operacao 240 +60 more	1a Operacao 239 2a Operacao 246 2a Operacao 256 +52 more
29	30	31	1	2	3	4
Operacao 181 Operacao 227 Operacao 198 +28 more	7a Operacao 244 7a Operacao 247 +52 more	3a Operacao 261 3a Operacao 280 3a Operacao 337 +64 more	2a Operacao 267 3a Operacao 271 3a Operacao 273 +72 more	2a Operacao 286 2a Operacao 291 3a Operacao 294 +63 more	2a Operacao 296 3a Operacao 303 7a Operacao 299 +60 more	1a Operacao 305 1a Operacao 311 3a Operacao 314 +57 more
5	6	7	8	9	10	11
Operacao 244 Operacao 275 Operacao 279 Operacao 268 +29 more	7a Operacao 317 7a Operacao 320 7a Operacao 318 +55 more	1a Operacao 324 1a Operacao 325 2a Operacao 319 2a Operacao 329 +57 more	1a Operacao 338 3a Operacao 344 3a Operacao 352 3a Operacao 362 +62 more	2a Operacao 348 3a Operacao 353 3a Operacao 401 3a Operacao 415 +73 more	12a Operacao 368 2a Operacao 375 3a Operacao 367 3a Operacao 436 +65 more	1a Operacao 377 2a Operacao 379 3a Operacao 380 3a Operacao 386 +54 more

Modelo rodando, por favor espere

[HTML5 Templates](#)

Figura 4.1-3 – Página “Fila” com o modelo estar rodando e no modo “month”

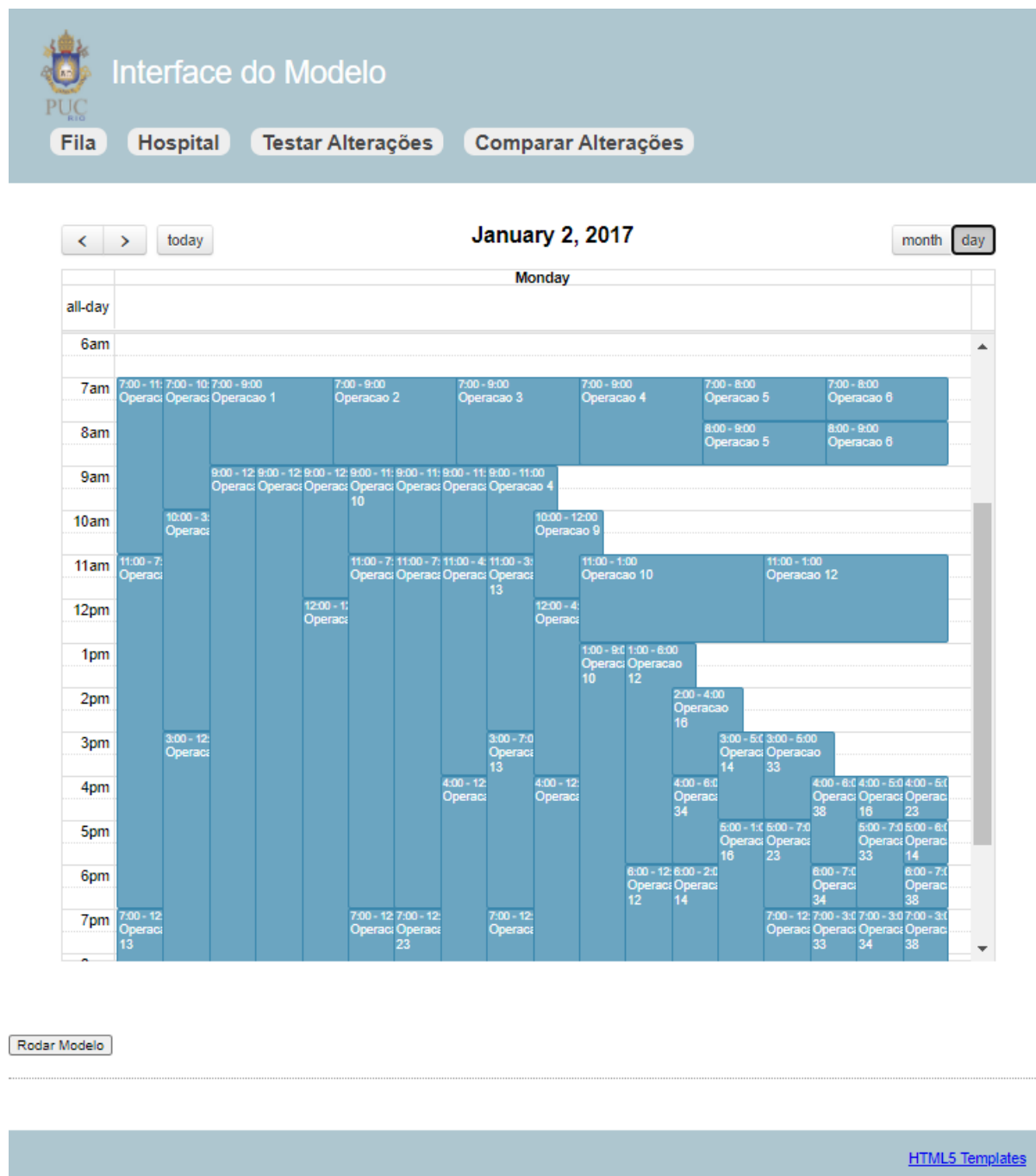


Figura 4.1-4 – Página “Fila” sem o modelo estar rodando e no modo “day”

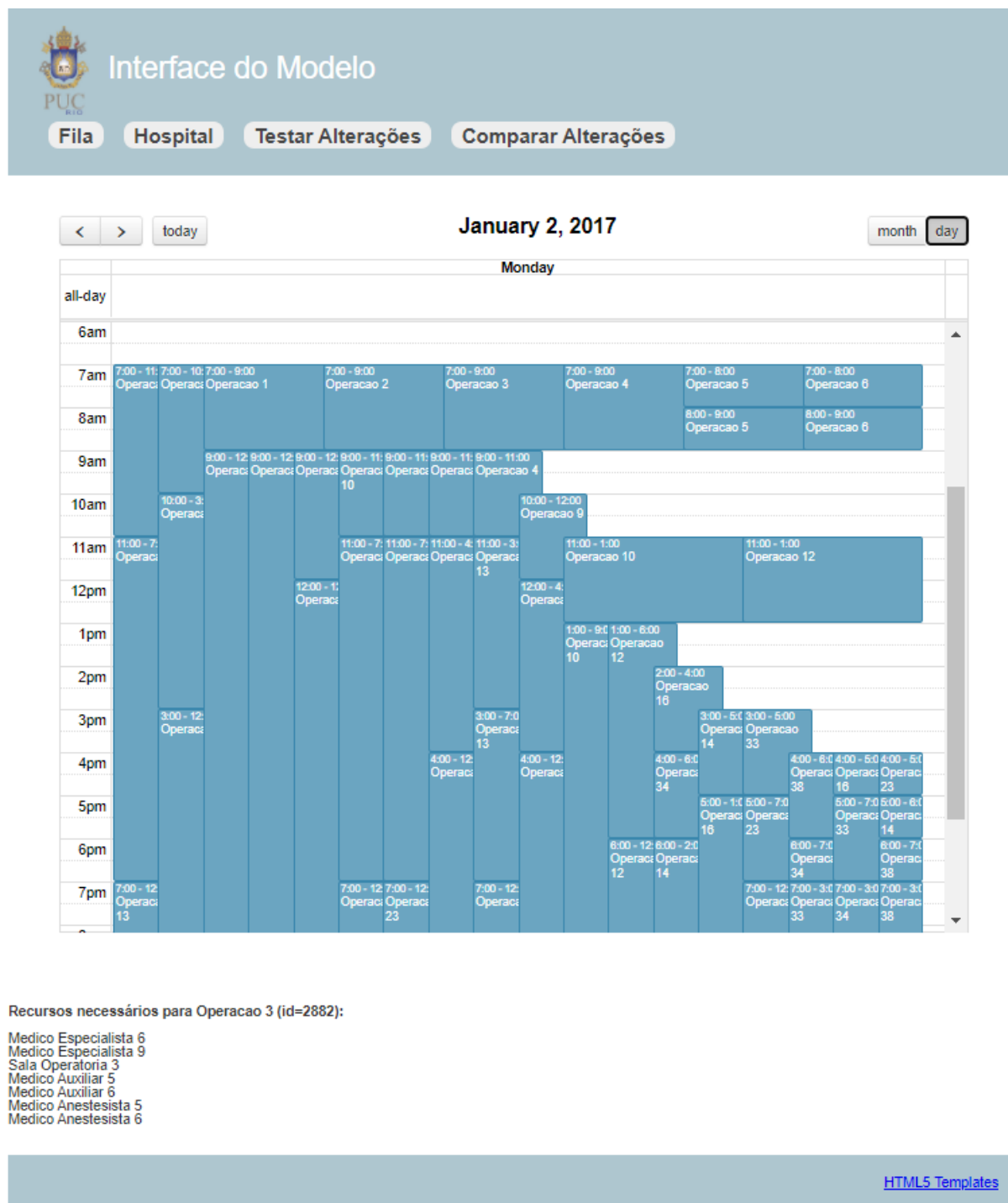


Figura 4.1-5 – Página “Details” com a Operação 3 selecionada

A página “Teste”, mostrada na Figura 4.1-6, permite que o usuário faça a configuração de um hospital hipotético para facilitar o planejamento de melhorias para as unidades de saúde. 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.

Interface do Modelo

Fila Hospital Testar Alterações Comparar Alterações

Abrir pasta: hosp Abrir

Médicos Especialistas: 30 + -

Médicos Assistentes: 10 + -

Médicos Anestesistas: 10 + -

Camas Pré Operação: 10 + -

Camas Pós Operação: 15 + -

Camas Repouso: 25 + -


Salas: 10 + -

Rodar e Salvar Como: Salvar

[HTML5 Templates](#)

Figura 4.1-6 – 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 4.1-7. No caso de adição de uma cama ou sala é perguntado para quais especialidades do hospital aquele recurso está habilitado como mostrado na Figura 4.1-8, enquanto que 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 especialista) como mostrado na Figura 4.1-9. Adicionalmente, no caso do especialista são requisitados também quais os dias de trabalho do mesmo, como mostrado na Figura 4.1-10.



Interface do Modelo

[Fila](#)
[Hospital](#)
[Testar Alterações](#)
[Comparar Alterações](#)

Abrir pasta:

Médicos Especialistas: 30
 Médicos Assistentes: 10
 Médicos Anestesistas: 10
 Camas Pré Operação: 10
 Camas Pós Operação: 15
 Camas Repouso: 25
 Salas: 10

Rodar e Salvar Como:

[HTML5 Templates](#)

Figura 4.1-7 – Página “Teste” com remoção de um médico assistente



Interface do Modelo

[Fila](#)
[Hospital](#)
[Testar Alterações](#)
[Comparar Alterações](#)

Valores das variáveis do hospital

Médicos Especialistas: 30
 Médicos Assistentes: 10
 Médicos Anestesistas: 10
 Camas Pré Operação: 10
 Camas Pós Operação: 15
 Camas Repouso: 25
 Salas: 10

Configurações da Nova Sala:


Identificador:

Habilitada para:

Especialidade 1 : ☐
 Especialidade 2 : ☐
 Especialidade 3 : ☐
 Especialidade 4 : ☐
 Especialidade 5 : ☐
 Especialidade 6 : ☐
 Especialidade 7 : ☐
 Especialidade 8 : ☐
 Especialidade 9 : ☐
 Especialidade 10 : ☐
 Especialidade 11 : ☐
 Especialidade 12 : ☐
 Especialidade 13 : ☐
 Especialidade 14 : ☐
 Especialidade 15 : ☐

[HTML5 Templates](#)

Figura 4.1-8 – Página “Teste” com adição de uma sala de operação



Interface do Modelo

Fila**Hospital****Testar Alterações****Comparar Alterações**

Abrir pasta:

Médicos Especialistas: 30

Médicos Assistentes: 10

Médicos Anestesistas: 10

Camas Pré Operação: 10

Camas Pós Operação: 15

Camas Repouso: 25

Salas: 10

Rodar e Salvar Como:

Configurações do Novo Especialista:

Identificador:

Habilidade (entre 0 e 3):

Especialidade 1 :

Especialidade 2 :

Especialidade 3 :

Especialidade 4 :

Especialidade 5 :

Especialidade 6 :

Especialidade 7 :

Especialidade 8 :

Especialidade 9 :

Especialidade 10 :

Especialidade 11 :

Especialidade 12 :

Especialidade 13 :

Especialidade 14 :

Especialidade 15 :

[HTML5 Templates](#)

Figura 4.1-9 – Página “Teste” com adição de um especialista (1ª página)



Interface do Modelo

Fila Hospital Testar Alterações Comparar Alterações

Abrir pasta:

Médicos Especialistas: 30

Médicos Assistentes: 10

Médicos Anestesistas: 10

Camas Pré Operação: 10

Camas Pós Operação: 15

Camas Repouso: 25

Salas: 10

Rodar e Salvar Como:

Configurações do Novo Especialista:
Médico "Novo Especialista":
Disponibilidade:
Seg : ☐
Ter : ☐
Qua : ☐
Qui : ☐
Sex : ☐
Sab : ☐

[HTML5 Templates](#)

Figura 4.1-10 – Página “Teste” com adição de um especialista (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 4.1-11.

Interface do Modelo

Fila Hospital Testar Alterações Comparar Alterações

Valores das variáveis do hospital

Médicos Especialistas: 30 + -

Médicos Assistentes: 10 + -

Médicos Anestesistas: 10 + -

Camas Pré Operação: 10 + -

Camas Pós Operação: 15 + -

Camas Repouso: 25 + -

Salas: 10 + -

[HTML5 Templates](#)

Figura 4.1-11 – Página “Hospital” sem nenhuma alteração

Finalmente, a página “Comparar Alterações” permite que sejam comparados dois hospitais diferentes para os quais foi rodado o modelo. Ela inicialmente pede simplesmente 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 4.1-12. 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 4.1-13. Após escolher duas pastas para fazer a comparação, a página vai passar a mostrar as penalizações do calendário feito para aquele hospital, com o peso de cada penalização em parêntesis e o resultado final para a ordenação em questão, como mostrado na Figura 4.1-14.



Interface do Modelo


Fila **Hospital** **Testar Alterações** **Comparar Alterações**

Por favor, indique quais pastas serão comparadas

Comparar

[HTML5 Templates](#)

Figura 4.1-12 – Página “Comparar Alterações” sem nenhuma pasta escolhida



Interface do Modelo

Fila **Hospital** **Testar Alterações** **Comparar Alterações**

A pasta "invalido" não é válida ou o modelo não terminou de rodar.
Por favor, indique quais pastas serão comparadas

Comparar

[HTML5 Templates](#)

Figura 4.1-12 – Página “Comparar Alterações” com mensagem de erro



Interface do Modelo

[Fila](#)[Hospital](#)[Testar Alterações](#)[Comparar Alterações](#)

Por favor, indique quais pastas serão comparadas

Para o calendário definido por "hosp", temos as seguintes penalizações:

- Tempo total necessário para as cirurgias : 5653.0 (1x)
- Número de cirurgias feitas fora do prazo : 1067.0 (0.0x)
- Tempo médio fora do prazo de tais cirurgias (caso existam) : 77.67 (0.0x)
- Número de cirurgias feitas por especialistas de segundo grau : 1079.0 (0.0x)
- Número de cirurgias feitas por especialistas de terceiro grau : 274.0 (0.0x)

Dando um score total de:

$(5653.0 \times 1) + (1067.0 \times 0.0) + (77.67 \times 0.0) + (1079.0 \times 0.0) + (274.0 \times 0.0)$

Ou seja: 5653.0

Lembrando que quanto maior o score, pior a performance do hospital simulado.

Para o calendário definido por "test", temos as seguintes penalizações:

- Tempo total necessário para as cirurgias : 2.3 (1x)
- Número de cirurgias feitas fora do prazo : 2.3 (0x)
- Tempo médio fora do prazo de tais cirurgias (caso existam) : 2.3 (0x)
- Número de cirurgias feitas por especialistas de segundo grau : 2.3 (0x)
- Número de cirurgias feitas por especialistas de terceiro grau : 2.3 (0x)

Dando um score total de:

$(2.3 \times 1) + (2.3 \times 0) + (2.3 \times 0) + (2.3 \times 0) + (2.3 \times 0)$

Ou seja: 2.3

Lembrando que quanto maior o score, pior a performance do hospital simulado.

[HTML5 Templates](#)

Figura 4.1-12 – 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 do mesmo 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 performar de forma mais lenta, 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 abrir mão das 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 rodar mais rápido 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 ainda assim ter um tempo de execução demasiado grande, 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.

Finalmente, 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.

Após isso, 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 4.2.2-1 e 4.2.2-2 é possível ver as diferenças de tipagem nos cabeçalhos das duas linguagens, e na Figura 4.2.2-3 é mostrada a tela do MATLAB Coder aonde é possível dizer os tipos dos parâmetros.

```
function [ fitness,Tt,NOFP,TmNOFP,NOE2,NOE3] = favalia( schedule,DiaOp,EP,EspMedOp,k0,k1,k2,k3 )
```

Figura 4.2.2-1 – Cabeçalho de uma função MATLAB

```
void favalia(const emxArray_int32_T *schedule, const emxArray_int32_T *DiaOp,
const emxArray_int32_T *EP, emxArray_int32_T *EspMedOp, double k0,
double k1, double k2, double k3, double *fitness, double *Tt,
double *NOFP, double *TmNOFP, double *NOE2, double *NOE3)
{

```

Figura 4.2.2-2 – Cabeçalho de uma função C

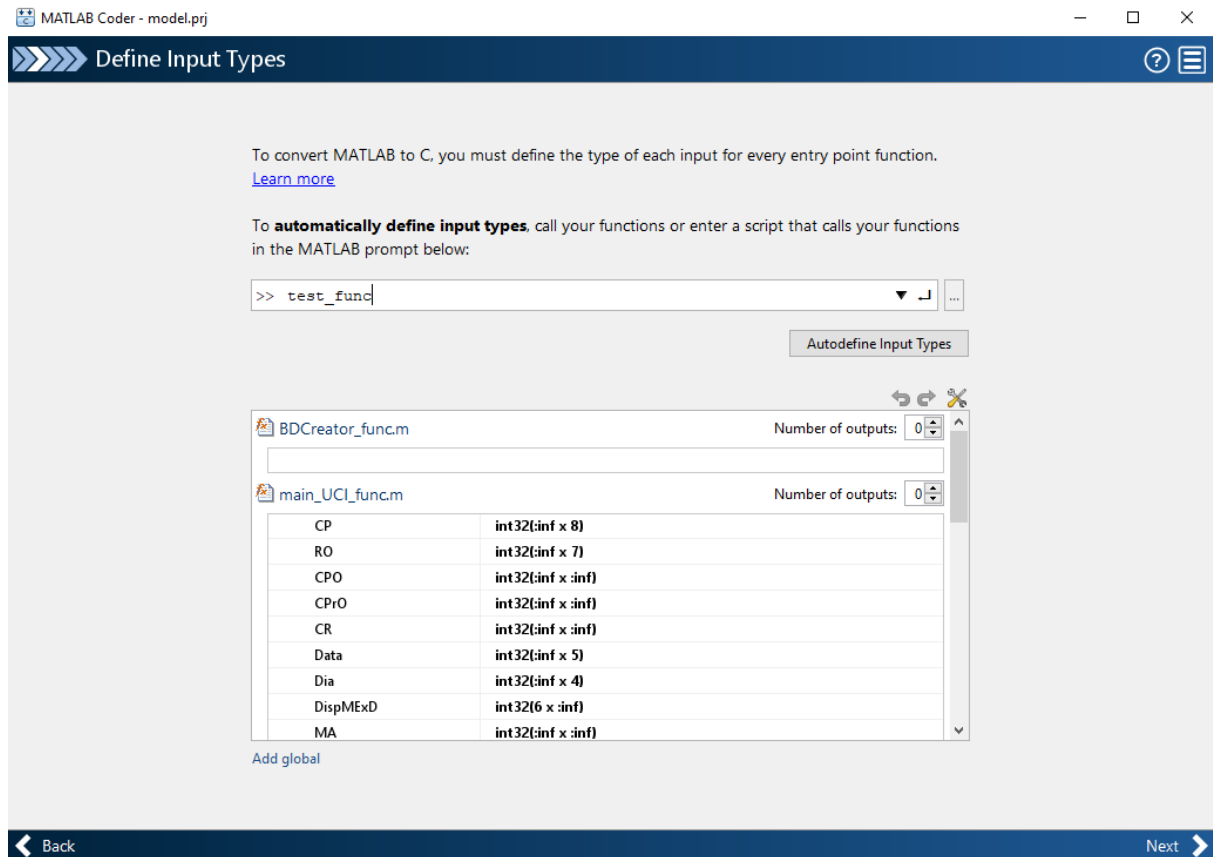


Figura 4.2.2-3 – Tela do MATLAB Coder de definição de tipo dos parâmetros

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 4.2.2-4, outro dos pré-requisitos da linguagem C na utilização de variáveis.

```

funcionCPrO (NumRec, PCPrO, PME, PMA, PMA_n, PS, PCPO, PC
tempUltPosRecXDia, DispMExD, DispME)

E = 1:size (NumRec (1)+1:sum (NumRec (1:2)), 2);
S = 1:size (sum (NumRec (1:2))+1:sum (NumRec (1:3)), 2);
A = 1:size (sum (NumRec (1:3))+1:sum (NumRec (1:4)), 2);
An = 1:size (sum (NumRec (1:4))+1:sum (NumRec (1:5)), 2);
CPO = 1:size (PCPO, 2);
CR = 1:size (PCR, 2);
CPrOA = int32 (0);
CPOA = int32 (0);
CRA = int32 (0);
IniCPrOA = int32 (0);
EndCPrOA = int32 (0);
IniSA = int32 (0);
EndSA = int32 (0);
IniCPOA = int32 (0);
EndCPOA = int32 (0);
IniCRA = int32 (0);
EndCRA = int32 (0);
EndCPrOxD = int32 (0);
EndRPxD = int32 (0);
EA_double = 1:size (PME, 2)-1;
AA_double = 1:size (PMA, 2)-1;
AnA_double = 1:size (PMA_n, 2)-1;
SA_double = 1:size (PS, 2)-1;

```

Figura 4.2.2-4 – 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 era um que 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 sutilmente transformava o segundo

parâmetro da multiplicação em um vetor (replicando o escalar até o tamanho compatível com a matriz), para que o código executasse corretamente, entretanto, quando o MATLAB Coder tentava parsear a expressão para identificar os tamanhos das variáveis auxiliares a serem utilizadas, 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 encontrados acima o MATLAB Coder passou a conseguir gerar um código C a partir do modelo. Claro que esse código não funcionava corretamente em um primeiro momento, e ainda teriam que ser feitas 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. Foi necessário então aumentar o número de gerações utilizado no código sendo 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 4.2.2-5.

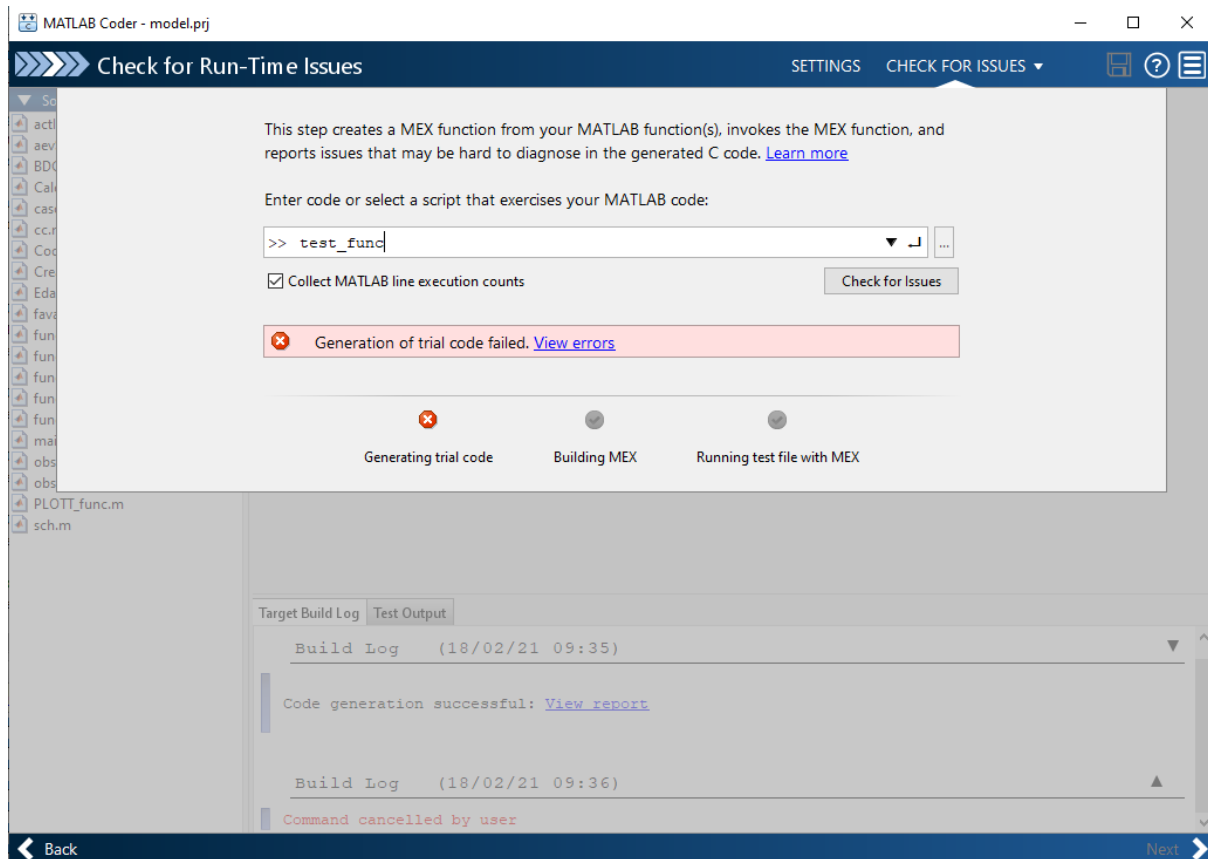


Figura 4.2.2-5 – Definição da função de teste

Outra necessidade foi alterar todos os momentos onde, 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 se enganar, 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, e ademais tornar o mesmo mais legível. Ela serve para 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 isso 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, também ocasionou da tradução ser mais rápida 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 mesmo 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 interessante 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.

```
Ger: 24 -      Ini: 6181.00000 -      Best: 5317.00000 -      Mean: 5317.00000 -      STD:      0.00000
Ger: 25 -      Ini: 6181.00000 -      Best: 5317.00000 -      Mean: 5317.00000 -      STD:      0.00000
K>> toc()
Elapsed time is 72981.961377 seconds.
```

Figura 4.2.3-1 – Execução de 25 gerações no MATLAB em 20h16min

```
Ger: 25 -      Ini: 6181.00000 -      Best: 5317.00000 -      Mean: 5317.00000 -
STD:      0.00000 -      Tt: 5317.00000 -      NOFP: 1070.00000 -      TmOFP: 75.61589 -
NOE2: 1064.00000 -      NOE3: 283.00000
Elapsed time is 18102.935613 seconds
```

Figura 4.2.3-2 – 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, aonde 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. Tudo isso utilizando diversas ferramentas que requisitaram estudo do funcionamento e por vezes a busca de outras que poderiam realizar o trabalho de forma melhor. 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 de qual a 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, mas sim um no qual foi necessário investigar diversas possíveis soluções para poder descobrir os lados positivos e negativos de cada uma delas. Outra parte difícil foi causada pelo 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 também necessário rastrear as alterações em diversos módulos para que a informação fosse coletada e passada adiante pelos caminhos corretos. Uma parte interessante e que não era esperada quando foi começado o 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 fazer 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 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] <http://www.reme.org.br/artigo/detalhes/381>
- [2] R. González, M. Vellasco, K. Figueiredo, Resource “Optimization for Elective Surgical Procedures Using Quantum-inspired Genetic Algorithms”, 2019 Genetic and Evolutionary Computation Conference (GECCO 2019)
- [3] <https://www.mathworks.com/products/MATLAB-coder.html>
- [4] <https://www.mathworks.com/products/MATLAB.html>
- [5] ISO/IEC 9899:1999 specification (PDF). [S.l.: s.n.] p. 313, § 7.20.3 "Memory management functions"
- [6] <https://gcc.gnu.org/>
- [7] Kuhlman, Dave. "A Python Book: Beginning Python, Advanced Python, and Python Exercises". Section 1.1. Archived from the original (PDF) on 23 June 2012.
- [8] <https://pypi.org/>
- [9] <https://docs.Python.org/3/library/ctypes.html>
- [10] <https://palletsprojects.com/p/flask>
- [11] Multiple (wiki). "Web application framework". Docforge. Archived from the original on 2015-07-23.
- [12] <https://www.mathworks.com/help/matlab/matlab-engine-for-python.html>
- [13] <https://github.com/awesomebytes/libermate>
- [14] <https://github.com/victorlei/smop>
- [15] <http://ompc.juricap.com/>
- [16] <https://www.mathworks.com/products/embedded-coder.html>
- [17] I. Rejer, “Genetic algorithm with aggressive mutation for feature selection in BCI feature space,” Pattern Anal. Appl., vol. 18, no. 3, pp. 485–492, Aug. 2015
- [18] G. Zhang and Gexiang, “Quantum-inspired evolutionary algorithms: a survey and empirical study,” J. Heuristics, vol. 17, no. 3, pp. 303–351, Jun. 2011.
- [19] A. Testi, E. Tanfani, R. Valente, G. L. Ansaldo, and G. C. Torre, “Prioritizing surgical waiting lists,” J. Eval. Clin. Pract., vol. 14, no. 1, pp. 59–64, Jan. 2008.
- [20] L. R. da Silveira, R. Tanscheit, and M. Vellasco, “Quantum inspired evolutionary algorithm for ordering problems,” Expert Syst. Appl., vol. 67, pp. 71–83, Jan. 2017

- [21] S. N. Chaurasia, S. Sundar, and A. Singh, “Hybrid metaheuristic approaches for the single machine total stepwise tardiness problem with release dates,” *Oper. Res.*, vol. 17, no. 1, pp. 275–295, Apr. 2017.
- [22] L. Reis da Silveira, R. Tanscheit, and M. Vellasco, “Algoritmo Genético de Ordem com Inspiração Quântica,” 2014.