MINISTÉRIO DA SAÚDE
FUNDAÇÃO OSWALDO CRUZ
INSTITUTO OSWALDO CRUZ

Mestrado em Programa de Pós-Graduação em Biologia Computacional e Sistemas

IDENTIFICAÇÃO DE ALVOS TERAPÊUTICOS PARA A BACTÉRIA MULTIRRESISTENTE *P. AERUGINOSA* CCBH4851 ATRAVÉS DA ANÁLISE DE REDES METABÓLICAS

THIAGO CASTANHEIRA MERIGUETI

Rio de Janeiro
Abril de 2018

Ministério da Saúde

FIOCRUZ
Fundação Oswaldo Cruz

INSTITUTO OSWALDO CRUZ

Programa de Pós-Graduação em Biologia Computacional e Sistemas

THIAGO CASTANHEIRA MERIGUETI

Identificação de alvos terapêuticos para a bactéria multirresistente *P. aeruginosa* CCBH4851 através da análise de redes metabólicas

Dissertação apresentada ao Instituto Oswaldo Cruz como parte dos requisitos para obtenção do título de Mestre em Ciências

Orientador (es): Prof. Dr. Fabricio Alves Barbosa da Silva
Prof. Dr. Floriano Paes Silva Jr.

RIO DE JANEIRO
Abril de 2018

Merigueti, Thiago Castanheira.

Identificação de alvos terapêuticos para a bactéria multirresistente P. aeruginosa CCBH4851 através da análise de redes metabólicas / Thiago Castanheira Merigueti. - Rio de janeiro, 2018.
144 f.

Dissertação (Mestrado) – Instituto Oswaldo Cruz, Pós-Graduação em Biologia Computacional e Sistemas, 2018.

Orientador: Fabrício Alves Barbosa da Silva.
Co-orientador: Floriano Paes Silva Júnior.

Bibliografia: f. 73-80

1. Biologia de sistemas. 2. FBA. 3. Redes metabólicas. 4. COBRA. 5. Python. I. Título.

Ministério da Saúde

FIOCRUZ
Fundação Oswaldo Cruz

INSTITUTO OSWALDO CRUZ

Programa de Pós-Graduação em Biologia Computacional e Sistemas

AUTOR: THIAGO CASTANHEIRA MERIGUETI

IDENTIFICAÇÃO DE ALVOS TERAPÊUTICOS PARA A BACTÉRIA MULTIRRESISTENTE *P. AERUGINOSA* CCBH4851 ATRAVÉS DA ANÁLISE DE REDES METABÓLICAS

ORIENTADOR (ES): **Prof. Dr. Fabrício Alves Barbosa da Silva**
Prof. Dr. Floriano Paes Silva Jr.

Aprovada em: 20/04/2018

EXAMINADORES:

Prof. Dr. Nicolas Carels (CDTS/FIOCRUZ) **– Presidente e revisor**
Prof. Dr. Hermes Senger (UFSCar)
Profª. Drª. Ana Carolina Paulo Vicente (IOC/FIOCRUZ)
Prof. Dr. Márcio Argollo Ferreira de Menezes (UFF) - **Suplente**
Profª. Drª. Ana Carolina Ramos Guimarães (IOC/FIOCRUZ) - **Suplente**

Rio de Janeiro, 20 de Abril de 2018

AGRADECIMENTOS

Primeiramente a Deus, que me deu muita saúde, paz, serenidade e forças para seguir nesse caminho árduo até o final, conciliando entre a vida acadêmica e profissional (mestrado/emprego).

À minha amada Luana, que é minha noiva, companheira, amiga, cúmplice, enfim, tudo na minha vida e sempre acreditou em mim e está sempre ao meu lado me incentivando e dando forças. Eu te amo, minha Luana!

Aos meus sogros, Maria de Lourdes e José Cláudio, que fazem tudo por mim, me tratam como um filho e sempre acreditaram na minha vitória.

Aos meus pais, Edualdo e Ana Maria (*in memoriam*), que me criaram e proporcionaram o melhor possível em minha vida. Foi por causa de vocês que me tornei o que sou hoje e cheguei até aqui. Muito obrigado por tudo!

Aos orientadores Dr. Fabrício e Dr. Floriano, que não tenho palavras para agradecer a paciência e a dedicação que tiveram comigo. Sempre confiaram no meu trabalho, me mostraram que eu posso ser melhor, ser mais crítico, mais atento, aprender mais, ler mais e sempre evoluir não apenas na vida acadêmica, na minha vida como um todo e me guiaram com extrema competência até aqui. Os recomendo a quaisquer candidatos ao programa. Gratidão eterna!

Aos amigos Cristiano Braga, Felipe Gil e Leandro Leal, que além de colegas de profissão se tornaram meus amigos e “seguraram a barra” das minhas saídas e faltas no trabalho para cursar as disciplinas no meio do expediente e me ajudavam a me manter calmo, acreditar mais e aguentar esse ritmo tão intenso.

À Fundação Oswaldo Cruz, que confiou em mim e me deu a oportunidade de realizar meu sonho de me tornar mestre, fazer parte do programa e poder aplicar meus conhecimentos em prol da sociedade, apresentando um trabalho de qualidade e um resultado que pode colaborar com a comunidade científica de alguma maneira. Minha estadia aqui mudou minha vida e vou levar comigo pra sempre, muito obrigado!

Ao Dr. Nicolas Carels por ter aceitado ser revisor deste trabalho e ser tão solícito e ter dado muitas sugestões para a melhoria deste documento.

A Lídia Maria Castanheira, minha tia e madrinha, que dedicou um pouco do seu tempo para ler este trabalho e dar uma ajuda na melhoria de escrita, analisando o português, além de sempre acreditar em mim e ser uma pessoa muito boa para mim.

A todos os envolvidos na coordenação da BCS, que estiveram comigo em todos os momentos do curso (por exemplo, no momento que precisei fazer uma cirurgia e foi necessário adiar minha qualificação naquela ocasião).

A todos os integrantes do grupo de pesquisa de bactérias multirresistentes e aos nobres colegas do PROCC.

A todos aqueles que participaram direta ou indiretamente e que, de alguma forma, contribuíram para minha formação, o meu mais sincero agradecimento.

“No fim dá tudo certo e se ainda não deu certo é porque não chegou ao fim.”

Fernando Sabino

Ministério da Saúde

FIOCRUZ
Fundação Oswaldo Cruz

INSTITUTO OSWALDO CRUZ

IDENTIFICAÇÃO DE ALVOS TERAPÊUTICOS PARA A BACTÉRIA MULTIRRESISTENTE *P. AERUGINOSA* CCBH4851 ATRAVÉS DA ANÁLISE DE REDES METABÓLICAS

RESUMO

DISSERTAÇÃO DE MESTRADO EM BIOLOGIA COMPUTACIONAL E SISTEMAS

Thiago Castanheira Merigueti

Infecções relacionadas à assistência à saúde (IRAS) é um grave problema de saúde pública. Elas podem estar associadas à morbidade e mortalidade e são responsáveis pelos aumentos da hospitalização de pacientes. Neste contexto, é importante identificar estratégias que possam impedir a propagação de bactérias em pacientes hospitalizados. Neste trabalho, propomos um novo método para identificar alvos terapêuticos, através da análise de redes metabólicas em escala genômica, que identifica alvos conhecidos e potenciais em bactérias resistentes a múltiplos fármacos. Aqui, nós automatizamos o processo de identificação de genes essenciais usando Análise de Balanço de Fluxo (do inglês Flux Balance Analysis - FBA), Análise de Variabilidade de Fluxo (do inglês Flux Variability Analysis - FVA) e consultas a vários repositórios públicos, como KEGG, Uniprot e Drugbank. A aplicação web foi desenvolvida em Python usando CobraPy e Django 1.11. Redes metabólicas em escala genômica, disponíveis no formato SBML, foram analisadas usando esta ferramenta. Foram encontradas informações de alvos terapêuticos e possíveis fármacos como resultados da análise, feita pelo sistema, de três redes metabólicas de *P. aeruginosa* disponíveis na literatura e da primeira versão da rede metabólica da cepa multirresistente CCBH4851. Foram detectados genes alvos que são relatados na literatura. O método de análise de fluxos de reações proposto fornece resultados, mesmo para redes metabólicas não curadas, incompletas ou imprecisas. Usando o método FBA, simulamos o nocaute na rede que verifica a interrupção da geração de biomassa. Este novo método pode fornecer informações sobre a identificação e descoberta de novos alvos terapêuticos para bactérias multirresistentes através da análise da rede metabólica em escala genômica.

Ministério da Saúde

FIOCRUZ
Fundação Oswaldo Cruz

INSTITUTO OSWALDO CRUZ

IDENTIFICATION OF THERAPEUTIC TARGETS FOR THE MULTIRRESISTENT BACTERIA *P. AERUGINOSA* CCBH4851 THROUGH THE ANALYSIS OF METABOLIC NETWORKS

ABSTRACT

MASTER DISSERTATION IN COMPUTATIONAL AND SYSTEMS BIOLOGY

Thiago Castanheira Merigueti

Healthcare-associated infections (HAI) is a serious public health problem. They may be associated with morbidity and mortality and are responsible for increases in patient hospitalization. In this context, it is important to identify strategies that may prevent the spread of bacteria in hospitalized patients. In this work, we propose a new method to identify therapeutic targets through the analysis of metabolic networks on genomic scale, which enables the identifies known and potential targets in multidrug resistant bacteria. Here we automate the process of identifying essential genes using Flux Balance Analysis (FBA), Flux Variability Analysis (FVA), and queries to various public repositories such as KEGG, Uniprot and Drugbank. The web application was developed in Python using CobraPy and Django 1.11. Genomic scale metabolic networks, available in the SBML format, were analyzed using this tool. Information on therapeutic targets and possible drugs was found as a result of the analysis of three metabolic networks of *P. aeruginosa* available in the literature and the first version of the metabolic network of the CCBH4851 multiresistant strain. Target genes were detected that are reported in the literature. The proposed reaction flow analysis method provides results, even for uncured, incomplete or imprecise metabolic networks. Using the FBA method, we simulate the knockout in the network that verifies the interruption of biomass generation. This new method can provide information on the identification and discovery of new therapeutic targets for multidrug resistant bacteria through the analysis of the metabolic network on genomic scale.

ÍNDICE

ÍNDICE DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

ANVISA	Agência Nacional de Vigilância Sanitária
COBRA	*COnstraint-Based Reconstruction and Analysis Toolbox*
COBRAPy	*COBRA for Python*
CRUD	*Create/Read/Update/Delete*
DCW	*Dry Cell Weight*
DNA	*Deoxyribonucleic acid*
DRY	*Don't repeat yourself*
EC	*Enzyme Commission*
EUA	Estados Unidos da América
FBA	*Flux Balance Analysis*
FC	Fibrose Cística
FDA	*Food and Drug Administration*
FIOCRUZ	Fundação Oswaldo Cruz
FVA	*Flux Variability Analysis*
GIL	*Global Interpreter Lock*
GPR	*Gene-to-protein-to-reaction*
HTML	*HyperText Markup Language*
IRAS	Infecções Relacionados à Assistência à Saúde
KEGG	*Kyoto Encyclopedia of Genes and Genomes*
MathML	*Mathematical Markup Language*
MATLAB	*MATrix LABoratory*
MTV	*Model-template-view*
OMS	Organização Mundial de Saúde
P&D	Pesquisa & Desenvolvimento
SBML	*Systems Biology Markup Language*
SGML	*Standard Generalized Markup Language*
SVG	*Scalable Vector Graphics*
UniProt	*Universal Protein Resource*
UniProtKB	*UniProt Knowledgebase*
URL	*Uniform Resource Locator*
UTI	Unidade de Terapia Intensiva
W3C	*World Wide Web Consortium*
XML	*eXtensible Markup Language*

1 INTRODUÇÃO

As Infecções Relacionadas à Assistência à Saúde (IRAS) são um grave problema de saúde pública. Entre os patógenos relacionados às IRAS, o grupo de bactérias é o que se destaca. Mais de dois milhões de casos de IRAS ocorrem todos os anos nos EUA, sendo que 50 a 60% são causados por bactérias resistentes a antimicrobianos. Em relação às ocorrências no Brasil, de acordo com os dados divulgados pela Agência Nacional de Vigilância Sanitária (ANVISA), houve um aumento considerável de notificações por parte dos hospitais, na sua maioria públicos. Assim como nos EUA, dentre os casos de IRAS no Brasil, cerca de 60% desses foram causados por bactérias multirresistentes, sendo a *P. aeruginosa* um dos seis agentes etiológicos mais encontrados em pacientes adultos hospitalizados em UTI entre 2014 e 2016. Além disso, as amostras de bactérias resistentes encontradas aumentam a cada ano (1). Em 2014, a Organização Mundial de Saúde (OMS) publicou o relatório "Aviso de resistência antimicrobiana: relatório global sobre vigilância" sobre o aumento crescente da resistência antimicrobiana no mundo (2). A resistência aos antibióticos pelas bactérias entre os agentes patogênicos hospitalares aumentou em níveis alarmantes, tanto em países desenvolvidos como em desenvolvimento. Estima-se que haverá em um futuro próximo uma propagação mundial de infecções intratáveis dentro e fora dos hospitais (1,2).

De acordo com um boletim publicado em 2017 pela OMS, existem 12 principais bactérias resistentes aos antibióticos que merecem atenção urgente em pesquisa e desenvolvimento (P&D) para busca de tratamentos antibióticos novos e efetivos. As bactérias consideradas prioritárias são Gram-negativas e resistentes a carbapenema: *Acinetobacter baumannii*, *Pseudomonas aeruginosa* e Enterobacteriaceae (1,3). A lista completa das bactérias, de acordo com o boletim, é a descrita abaixo:

Prioridade 1: CRÍTICA

Acinetobacter baumannii, resistente a carbapenema;
Pseudomonas aeruginosa, resistente a carbapenema;
Enterobacteriaceae, resistente a carbapenema, produtoras de ESBL.

Prioridade 2: ALTA

Enterococcus faecium, resistente à vancomicina;

Staphylococcus aureus, resistente à meticilina, com sensibilidade intermediária e resistência à vancomicina;

Helicobacter pylori, resistente à claritromicina;

Campylobacter spp., resistente às fluoroquinolonas;

Salmonellae, resistentes às fluoroquinolonas;

Neisseria gonorrhoeae, resistente à cefalosporina e às fluoroquinolonas.

Prioridade 3: MÉDIA

Streptococcus pneumoniae, sem sensibilidade à penicilina;

Haemophilus influenzae, resistente à ampicilina;

Shigella spp., resistente às fluoroquinolonas.

Nos seres humanos, *P. aeruginosa* é um agente patogênico oportunista que causa infecções graves em indivíduos imunocomprometidos. Este patógeno é a principal causa de morbidade em pacientes com fibrose cística (FC) (4), diabetes e outras doenças pulmonares, como bronquiectasias e pneumonia. Dada à gravidade potencial das bactérias multirresistentes e a falta de opções de tratamento, a identificação e implementação de estratégias efetivas para prevenir tais infecções são prioridades urgentes (5).

1.1 *Pseudomonas aeruginosa* - uma das bactérias da infecção hospitalar

Pseudomonas aeruginosa, bactéria Gram-negativa não fermentadora, isolada em laboratórios, é um micro-organismo ubiquitário, encontrado no solo, na água, nos vegetais, nos animais, nos alimentos e nos mais diversos ambientes hospitalares (6). Raramente, causa infecção num indivíduo imunologicamente saudável, porém é um dos principais agentes de infecção em indivíduos imunodeprimidos. Esta bactéria deve seu sucesso como patógeno em grande parte à sua versatilidade metabólica e flexibilidade (7). Sua importância clínica está baseada na difícil erradicação da infecção e contínuos fracassos terapêuticos, consequência direta da ampla expressão de fatores de virulência, assim como a resistência natural e adquirida a

muitos antibióticos e desinfetantes. Como as infecções causadas por *P. aeruginosa* envolvem diferentes órgãos e tecidos, os seus fatores de virulência são obrigatoriamente diversificados e em grande número. Os genes ligados à virulência contribuem para a sobrevivência e a aptidão dentro de um hospedeiro (8). A produção de beta-lactamase cromossomal, a impermeabilidade da membrana, a capacidade de colonizar superfícies em forma de biofilme, a presença de sistemas de efluxo, a aquisição de genes de resistência através de plasmídeos e outros elementos genéticos móveis por processos de transdução e conjugação, fazem com que poucos antibióticos sejam efetivos (6,9,10).

A *P. aeruginosa* é uma causa comum de infecção pulmonar crônica em pacientes com FC. Estas infecções contribuem consideravelmente para a morbidade e mortalidade dos pacientes com FC devido a uma intensa resposta inflamatória do hospedeiro que leva a uma perda irreversível da função pulmonar (11).

O tratamento de escolha para *P. aeruginosa* era a associação de um beta-lactâmico e um aminoglicosídeo, mas com o aumento da resistência a essas drogas, tem sido preconizada a volta da utilização de polimixinas (polimixina B e colistina) para o tratamento das infecções por bactérias multirresistentes. As polimixinas são fármacos antigos e tóxicos, cuja utilização havia sido abandonada pelo surgimento de fármacos potentes e eficazes, como as cefalosporinas e os carbapenemas, mas com o surgimento da resistência aos carbapenemas, as polimixinas voltaram a ser usadas nos hospitais do Brasil e do mundo (6,12).

Neste trabalho usamos os modelos estabelecidos para as cepas[1] de *P. aeruginosa*, que são PAO1, PA14 e CCBH4851. Essas cepas foram estudadas em laboratório e os modelos correspondentes mapeados em arquivos específicos. A cepa PAO1 foi isolada pelo departamento de engenharia biomédica da Universidade de Virgínia juntamente com o *Helmholtz Center for Infection Research (HZI),* da Alemanha (7). A cepa PA14 foi isolada pela Universidade de Virgínia, EUA, juntamente com a *Technical University of Denmark* (8). A cepa CCBH4851 é uma amostra que foi encontrada em um hospital brasileiro, isolada no Laboratório de Infecção Hospitalar da Fundação Oswaldo Cruz (LAPIH/FIOCRUZ) (12).

[1] Linhagem de micro-organismos isolada em laboratório para fins de estudos

1.2 Redes metabólicas em escala genômica

Uma rede metabólica em escala genômica fornece uma robusta estrutura para fins de compreensão das vias metabólicas dentro de um organismo (13). A integração de vias bioquímicas com sequências do genoma permitiu desenvolver as chamadas redes metabólicas em escala genômica. Essa modelagem corresponde ao mapeamento *gene-to-protein-to-reaction* (GPR) de cada via metabólica ali representada. Em geral, quanto mais informações disponíveis sobre a fisiologia, bioquímica e genética do organismo alvo, melhor será a capacidade preditiva dos modelos reconstruídos. O processo de reconstrução das redes metabólicas de eucariotos e procariotos é essencialmente o mesmo, porém as reconstruções de redes de eucariotos são geralmente mais complexas devido ao maior tamanho dos seus genomas e a multiplicidade de compartimentos celulares (14). A primeira rede metabólica em escala genômica foi gerada em 1995 para *Haemophilus influenzae* (15) e desde então, muitas reconstruções foram feitas.

A reconstrução e simulação de vias metabólicas permite uma visão detalhada de todos os mecanismos moleculares do metabolismo de um determinado organismo. Esses modelos correlacionam o genoma com a fisiologia molecular (14). Uma reconstrução reúne todas as informações metabólicas relevantes de um organismo e as converte em um modelo matemático representativo das vias metabólicas deste organismo. Usando essas modelagens, podemos identificar características essenciais do metabolismo, como o crescimento microbiano propriamente dito, a robustez da rede e os genes essenciais. Este conhecimento pode ser aplicado a processos biotecnológicos inovadores (13).

A reconstrução das vias metabólicas e seus modelos são usados para que possamos entender como funciona um organismo. Um modelo de reconstrução funciona como um primeiro passo para decifrar os mecanismos enzimáticos envolvidos em um patógeno, sendo que, estes modelos também podem, por exemplo, permitir estudar os genes mínimos necessários para uma célula manter a compatibilidade e/ou virulência, dentre outras inúmeras possibilidades de estudos de finalidades que podemos levantar com a modelagem de uma rede metabólica (13–15).

Existem reconstruções que contêm todas as reações metabólicas conhecidas em um determinado organismo com os genes que codificam cada enzima e existem

reconstruções específicas. Como exemplo temos a ênfase em fatores de virulência (13,14).

1.3 Metodologias de análise de fluxo em redes metabólicas

O *Flux Balance Analysis* (FBA), ou em português, análise de balanço de fluxo (ABF) é uma abordagem amplamente utilizada para o estudo de redes bioquímicas, em particular reconstruções de redes metabólicas de escala genômica. FBA calcula o fluxo de reações através de uma rede metabólica no estado estacionário, permitindo assim prever a taxa de crescimento de um organismo com base na produção de biomassa ou a taxa de produção de um metabolito biotecnologicamente importante (16). O estado de toda a rede metabólica é representado por fluxos para todas as reações e a soma dos fluxos de entrada deve ser igual à soma dos fluxos de saída para cada composto, onde os fluxos podem ser ponderados de acordo com os coeficientes derivados da estequiometria de cada reação (17,18).

O *Flux Variability Analysis* (FVA) é usado para encontrar o fluxo mínimo e máximo para as reações na rede, de acordo com estados predefinidos, por exemplo, suportando 90% da taxa máxima de produção de biomassa possível. As aplicações de FVA para biologia de sistemas incluem, entre outras, a melhor exploração das alternativas no estudo das distribuições de fluxo sob crescimento subótimo por meio da investigação da flexibilidade da rede e a sua redundância no ato de otimizar a formulação de propostas para a produção de antibióticos (19).

1.3.1 FBA (Flux Balance Analysis)

Desde meados da década de 1990, o uso do FBA se expandiu amplamente na elaboração de modelos metabólicos. O FBA foi desenvolvido no laboratório de Bernhard Palsson, na Universidade da Califórnia, em San Diego. A principal ideia por trás do FBA é usar a otimização linear[2] para evitar a necessidade da definição de parâmetros cinéticos (20).

[2] A otimização linear é utilizada para maximizar ou minimizar uma função linear de variáveis, denominada função objetivo, sujeita a uma série de equações lineares.

O FBA é uma ferramenta importante para aproveitar o conhecimento codificado nos modelos metabólicos e poder buscar potenciais alvos terapêuticos relevantes, cuja anulação resultaria na interrupção do crescimento microbiano (16).

O primeiro passo do FBA é representar matematicamente as reações metabólicas. A característica central desta representação é uma tabulação, na forma de uma matriz numérica, dos coeficientes estequiométricos de cada reação. Restrições como essas são um diferencial no FBA, onde não temos a necessidade de parâmetros cinéticos para medição, o que tornaria a análise da rede muito mais complexa (17,20).

O objetivo do FBA é representar matematicamente fluxos metabólicos através de uma *função objetivo* (16). A função objetivo é a expressão a ser maximizada na resolução de um problema de programação linear. Como resultado da resolução deste problema, os valores dos fluxos na rede são quantificados e o FBA procura maximizar ou minimizar esta função com a seguinte fórmula: $\mathbf{Z = c^t * v}$, que pode ser qualquer combinação linear de fluxos, onde *c* é um vetor transposto de pesos (representado pelo *t* na fórmula), indicando o quanto cada reação (como a reação de biomassa ao simular o crescimento máximo) contribui para a função (16,21). Na prática, quando apenas uma reação é desejada para maximização ou minimização, o vetor *c* é um vetor de zeros com um na posição da reação de interesse (21). O *v* é um vetor que representa o fluxo através de todas as reações em uma rede e o *Z* é o resultado da função objetivo, que, no caso aqui avaliado, é o resultado da maximização da *reação de biomassa*.

Uma *reação de biomassa* que drena os metabólitos precursores do sistema em suas estequiometrias relativas para simular a produção de biomassa é selecionada como função objetivo para prever as taxas de crescimento (16). Esta reação é consequência do processo de otimização, onde a taxa de crescimento será máxima justamente na fase exponencial do crescimento bacteriano (16). Os valores do fluxo são normalizados pelo peso da biomassa celular seca na cultura (chamado de peso seco, ou DCW, do inglês *Dry Cell Weight*) para permitir comparações ao longo do tempo. Assim, na prática, as unidades de medidas usadas para essa produção são milimoles por grama DCW por hora (mmol/g DCW/h). (20)

As reações metabólicas são representadas como uma matriz estequiométrica (*S*), de tamanho *m*n*. Cada linha dessa matriz representa um composto único (para um sistema com *m* compostos) e cada coluna representa uma reação (*n* reações). As entradas em cada coluna são os coeficientes estequiométricos dos metabolitos

que participam de uma reação. Existe um coeficiente negativo para cada metabólito consumido e um coeficiente positivo para cada metabólito produzido. Um coeficiente estequiométrico de zero é utilizado para cada metabólito que não participa de uma reação particular. *S* é uma matriz esparsa, uma vez que a maioria das reações bioquímicas envolvem apenas alguns poucos metabólitos diferentes. O fluxo através de todas as reações em uma rede é representado pelo vetor *v*, que possui um comprimento de *n*. Assume-se que o sistema de equações em estudo encontra-se em estado pseudoestacionário, ou seja, as variáveis de estado que definem seu comportamento, em relação ao tempo, permanecem invariáveis, onde trabalhamos com a derivada da concentração de metabólitos em função do tempo iguais a zero. A fórmula que resume este momento da otimização é: **S * v = 0** (16).

A distribuição de fluxo de uma rede biológica pode estar em qualquer ponto de um espaço de solução (17). Quando as restrições de equilíbrio de massa impostas pela matriz estequiométrica *S* e as restrições de capacidade impostas pelos limites inferior e superior são aplicadas a uma rede, ela definem esse espaço e a rede pode adquirir qualquer distribuição de fluxo dentro dele (16,17).

O FBA pode identificar um único valor ótimo do fluxo, que se localiza na borda do espaço de solução definido, através do problema de otimização (maximizar geração de biomassa, por exemplo) (16). Já se tratando das condições de contorno, elas estão associadas à definição do espaço de solução, onde o FBA é reduzido a um problema de programação linear (16,17,21). A forma canônica de um problema de FBA pode ser descrita da seguinte forma (22,23):

Maximizar	$\mathbf{c^t v}$
Sujeito a	**S*v = 0**
E	**limite inferior <= v <= limite superior**

A Figura 1 fornece uma visão geral das etapas envolvidas na realização de um FBA. Essencialmente, envolve quatro etapas: (i) definição do sistema (rede metabólica), (ii) obtenção de estequiometrias de reação (modelo computacional em estado pseudo-estacionário), (iii) definição da função objetivo biologicamente relevante e adição de outras restrições bioquímicas e (iv) otimização (resolução do problema de programação linear) (21)

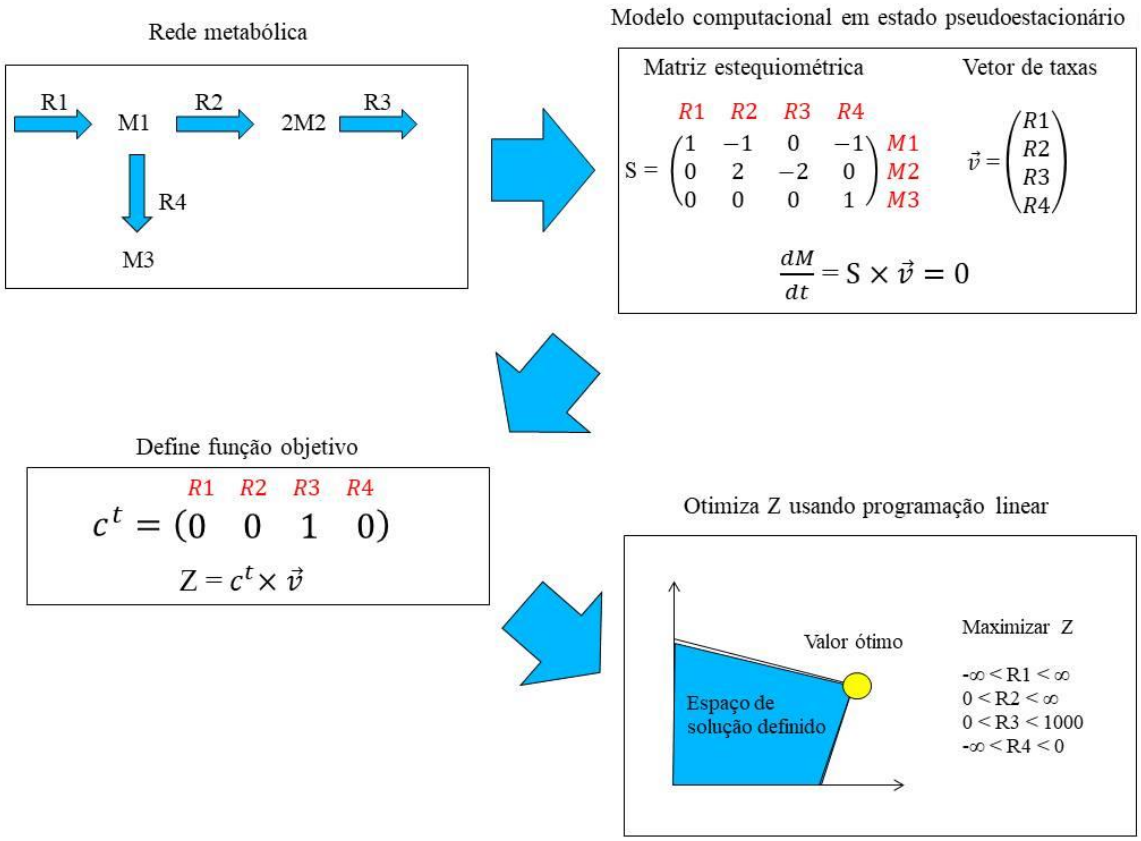

Figura 1 - Exemplo de funcionamento do FBA (adaptada de (16,24)**).**

1.3.2 FVA (Flux Variability Analysis)

O FVA (*Flux Variability Analysis*) ou, em português, análise da variabilidade do fluxo (AVF) é uma ferramenta computacional amplamente utilizada para avaliar o alcance mínimo e máximo de cada fluxo de reação que pode satisfazer as restrições usando um problema de programação linear duplo para cada reação de interesse. É frequentemente usada para determinar a robustez dos modelos metabólicos em várias condições de simulação (19). Essa abordagem é usada para encontrar os intervalos de valores permitidos para todos os fluxos em uma solução, enquanto o desempenho ideal, conforme definido pela função objetivo declarada, é cumprido. Essa abordagem é gerenciável de forma computacional e é altamente escalável (25).

FVA é uma ferramenta importante para analisar ainda mais a fundo os resultados obtidos com FBA para redes metabólicas em escala genômica (19). Para muitos modelos baseados em restrições, o FVA permite a exploração de rotas alternativas ótimas em uma rede metabólica (26). Com o FVA, novas questões

podem ser abordadas para estudar a flexibilidade das redes de reação bioquímicas em diferentes condições ambientais e genéticas (19,27).

As respostas metabólicas são essenciais para a adaptação de microrganismos à mudança das condições ambientais. O repertório de respostas de fluxo que a rede metabólica pode exibir em diferentes condições externas pode ser quantificado aplicando FVA para reconstruções metabólicas em escala genômica (28).

1.4 Aspectos técnicos em computação

1.4.1 Python

Python é uma linguagem interpretável, de alto nível, imperativa, orientada a objeto, dinâmica, de tipagem forte, criada por Guido van Rossum. A linguagem prioriza a clareza de código em relação a desempenho ou expressividade. Devido as suas características, Python é muito utilizado para processamento de texto e dados científicos (29).

A linguagem é de fácil aprendizado e muito poderosa. Possui estrutura eficiente e uma abordagem simples e eficaz. Sua sintaxe torna a linguagem ideal para scripts e desenvolvimento de aplicações rápidas em diversas áreas. O interpretador padrão e a extensa biblioteca nativa da linguagem estão disponíveis gratuitamente no site https://www.python.org e essa mesma fonte possui também uma gama de módulos, distribuições e programas de terceiros para Python, com documentação adicional. Python é extensível: se souber como programar em C, é fácil adicionar uma nova função ou módulo incorporado ao interpretador, seja para executar operações críticas a velocidade máxima, ou para vincular programas Python a bibliotecas que só podem estar disponíveis em forma binária (30).

O Python é uma linguagem que possui um controle de cada processo (*thread*), que normalmente leva uma quantidade de recursos padrão de 8 MB no Linux por requisição. Além disso, o Python possui um *Global Interpreter Lock* (GIL), que é um mecanismo dentro do interpretador da linguagem que faz o controle dos processos, independente de se ter ou não vários processadores a disposição para processamento paralelo. Porém, o controle geral de múltiplos processos (*multithreads)* é liberado a partir do momento que trabalhamos com requisições web e/ou de arquivos, o que permite que a aplicação não perca performance e possa

trabalhar corretamente em multiprocessamento, ou seja, em processamento paralelo de informação (29).

1.4.2 COBRA e COBRA for Python – Cobrapy

Os métodos COBRA (COnstraint-Based Reconstruction and Analysis Toolbox) são amplamente utilizados para modelagem de redes metabólicas em escala genômica. Devido aos sucessos com as análises do metabolismo, há um esforço crescente para aplicar o método para reconstruir e analisar modelos integrados de processos celulares. O COBRA Toolbox para MATLAB é um pacote de software líder para analisar o metabolismo em escala genômica, porém pago. O projeto openCOBRA é um esforço da comunidade para promover a pesquisa por meio da distribuição de software livre (31).

COBRApy (COBRA for Python) é um módulo Python que fornece suporte para métodos COBRA básicos. COBRApy é totalmente orientado a objetos, o que facilita a representação dos complexos processos biológicos do metabolismo. COBRApy não exige que o MATLAB funcione; no entanto, inclui uma interface para o COBRA Toolbox para MATLAB para facilitar o uso de códigos legados. A interface descrita permite utilizar comandos em MATLAB diretamente do Python. Para melhorar o desempenho, a COBRApy inclui suporte de processamento paralelo para processos computacionalmente intensivos (32).

1.4.3 Django

Django é um módulo Python para desenvolvimento rápido para web, de alto nível, que incentiva o desenvolvimento de forma mais ágil e o design limpo e pragmático. Ele cuida de muitos dos problemas do desenvolvimento Web, para que possamos nos concentrar apenas na escrita do aplicativo sem precisar reescrever métodos já existentes no Python. É gratuito e de código aberto (33,34). Ele encontra-se na versão 1.11, versão usada para desenvolver a ferramenta deste trabalho. Utiliza o padrão *model-template-view*[3] (MTV) e se concentra no máximo de

[3] Padrão de desenvolvimento de software, que separa o código em camadas, onde o model (M) é a implementação do modelo objeto-relacional usado, o template (T) é a interface do sistema, ou seja, a tela apresentada e a view (V) é a camada que implementa as regras da aplicação, processa os dados do model e passa para a camada de template.

automatização possível, dentro do princípio *don't repeat yourself*[4] (DRY), ou em português, não se repita (34).

O Django possui uma gama de características interessantes, como por exemplo, (I) a facilidade para criação de formulários, (II) URLs amigáveis, (III) um sistema robusto de cache e de templates, (IV) o suporte a aplicações multi-idioma, (V) a simplicidade para gerar um *Create/Read/Update/Delete* (CRUD) e (VI) a facilidade de gerar aplicações web complexas e de maior dinamismo para o usuário final.

Ele foi publicado em 2005, foi inspirado em um músico de jazz chamado Django Reinhardt e foi criado originalmente como sistema de gerenciamento de um portal de jornalismo no Kansas, EUA (33).

1.4.4 SBML

O *eXtensible Markup Language* (XML) é um arquivo com um formato padrão denominado linguagem de marcação, capaz de descrever diversos tipos de dados. Tem como propósito facilitar o compartilhamento de informações na web. É um subtipo do *Standard Generalized Markup Language* (SGML), ou em português, Linguagem Padronizada de Marcação Genérica, o qual é otimizado para distribuição através da web, e é definido pelo *World Wide Web Consortium* (W3C). A principal característica do XML é separar a interface dos dados estruturados (35).

Desse formato, foram criadas uma gama de linguagens de marcação baseadas no XML, entre elas o *HyperText Markup Language* (HTML) para páginas web, *Scalable Vector Graphics* (SVG) para gráficos vetoriais, *Mathematical Markup Language* (MathML) para expressões matemáticas e o *Systems Biology Markup Language* (SBML) (35,36).

O formato SBML é uma representação de modelos para biologia de sistemas. O SBML é utilizado para a descrição de sistemas biológicos, incluindo caminhos de sinalização celular, caminhos metabólicos, reações bioquímicas, regulação de genes e muitos outros (35). O SBML é definido de forma neutra em relação às linguagens de programação e codificação de software; no entanto, é principalmente orientado para permitir que os modelos sejam codificados usando XML. (36,37).

[4] Princípio do desenvolvimento de software que declara que cada parte do conhecimento deve ter uma representação única, não ambígua e definitiva dentro do sistema (66).

1.5 Repositórios utilizados neste trabalho

1.5.1 KEGG

Kyoto Encyclopedia of Genes and Genomes (KEGG) http://www.genome.ad.jp/kegg/ é uma base de conhecimento para análise sistemática de funções genéticas, ligando informações genômicas com informações funcionais de ordem superior (38). A informação genômica é armazenada no banco de dados GENES, que é uma coleção de catálogos de genes para os genomas completamente sequenciados e alguns genomas parciais com anotação atualizada de funções genéticas. A informação funcional de ordem superior é armazenada no banco de dados PATHWAY, que contém representações gráficas de processos celulares, tais como metabolismo, transporte através da membrana, transdução de sinal e ciclo celular. O banco de dados PATHWAY é complementado por um conjunto de tabelas de grupos ortólogos para a informação sobre subvias conservadas, que geralmente são codificados por genes acoplados posicionalmente no cromossomo e que são especialmente úteis na predição de funções genéticas. Um terceiro banco de dados no KEGG é LIGAND que armazena informações sobre compostos químicos, enzimas e suas reações. Os bancos de dados do KEGG são atualizados diariamente (39).

1.5.2 Drugbank

Drugbank http://www.drugbank.ca é um recurso ricamente anotado que combina dados de medicamentos detalhados com informações abrangentes de fármacos e informações de suas ações. Desde o seu primeiro lançamento em 2006, o Drugbank tem sido amplamente utilizado para facilitar a descoberta de alvos para fármacos *in silico*, o design de fármacos, a doação de fármacos ou o rastreio, a previsão do metabolismo de fármacos, a previsão de interação de fármacos e a educação farmacêutica em geral. A versão mais recente do Drugbank (versão 5.0.11, release de 20/12/2017) foi ampliada significativamente em relação a versão anterior. Com mais de 4900 insumos de medicamentos, agora contém mais 60% de moléculas pequenas e fármacos biotecnológicos aprovados pelo *Food and Drug Administration* (FDA) (40). Em comparação com a versão precedente, a versão atual do Drugbank contém três vezes mais proteínas não redundantes. O Drugbank

também melhorou significativamente a performance e a simplicidade de suas buscas de estruturas e de consulta de texto. (40,41).

1.5.3 UniProt

A base de conhecimento *Universal Protein Resource* (UniProt) é um grande recurso de sequências de proteínas e anotações detalhadas associadas. O banco de dados contém mais de 60 milhões de sequências, das quais mais de meio milhão de sequências foram curadas por especialistas que analisam criticamente os dados experimentais e previstos para cada proteína. O restante é automaticamente anotado com base em sistemas de regras que dependem do conhecimento com curadoria de especialistas. UniProt é uma longa coleção de bancos de dados que permitem aos cientistas navegar a grande quantidade de sequência e informações funcionais disponíveis para proteínas. A UniProt Knowledgebase (UniProtKB) é o recurso central que combina Swiss-Prot e TrEMBL. UniProtKB/Swiss-Prot contém mais de 550000 sequências que foram curadas por uma equipe especializada (42). Para essas entradas, a informação experimental foi extraída da literatura, organizada e resumida, facilitando enormemente o acesso dos cientistas à informação proteica. UniProtKB/TrEMBL fornece mais de 60 milhões de sequências que foram amplamente derivadas de sequenciamento de DNA de alto rendimento (43).

1.6 Estratégia de busca de alvos

No decorrer de todo o texto, usamos o termo *gene essencial*. Nesta seção, vamos apresentar algumas estratégias utilizadas para a busca dos potenciais alvos terapêuticos, ou os ditos, genes essenciais.

Uma maneira de buscar um alvo essencial é pela análise da topologia da rede metabólica. Verificar se um determinado nó possui várias ligações, o que o tornaria essencial, já que seria ligação de várias vias dentro da cepa estudada e teria como consequência interromper todo o funcionamento desta rede (44).

Outra forma de busca é fazendo um estudo comparativo na anotação da via, por enzimas análogas e específicas, onde é possível prever um número considerável de funções enzimáticas. Havendo enzimas análogas em um hospedeiro humano, por exemplo, poderá ser considerado um potencial novo alvo terapêutico ou gene essencial (45).

A forma descrita aqui neste trabalho é a análise de fluxos metabólicos, onde é feito um nocaute desta reação por meio de simulação computacional. Caso a reação nocauteada tenha como consequência a interrupção da geração de biomassa, podemos considerar que esse gene é essencial para a rede, já que ao efetuar o nocaute, a bactéria interrompe seu ciclo. Com isso temos a certeza de que o gene levantado é chamado gene essencial da rede (46).

1.7 Justificativa

Os casos de infecção hospitalar estão se tornando um problema grave de saúde pública. Com isso, se faz necessário ampliar os estudos de cada caso, com o objetivo de entendermos melhor as bactérias que causam estas infecções. Através do uso de ferramentas computacionais, poderemos levantar dados biológicos de cada bactéria e analisa-los. Dentre elas, temos a *P. aeruginosa* CCBH4851 (12), que é uma cepa multirresistente pertencente a um clone encontrado em hospitais de diferentes estados brasileiros. Ela foi selecionada porque é resistente a todos os antimicrobianos de importância clínica, com exceção da Polimixina B e possui vários mecanismos de resistência e elementos genéticos móveis. O tratamento da infecção hospitalar causado por bactérias como a *P. aeruginosa* está sendo comprometido cada vez mais, devido ao surgimento e disseminação dessa resistência. Assim, temos poucas alternativas terapêuticas (4).

Manifestações da *P. aeruginosa* ocorrem na maioria dos casos nos serviços de saúde, especialmente com indivíduos imunodeprimidos e outros altamente vulneráveis, como por exemplo, os pacientes que se encontram em unidades de terapia intensiva (UTI). As infecções podem ser associadas com uma morbidez e mortalidade significativa. Dada a gravidade potencial de infecções por *P. aeruginosa* e problemas específicos na escolha da terapia ideal, a identificação e implementação de estratégias eficazes para prevenir essas infecções são prioridades urgentes (5). Neste contexto, será importante identificar estratégias que possam impedir a disseminação desta bactéria em pacientes hospitalizados.

Outra justificativa acerca do projeto é o fato de termos a necessidade dessa ferramenta para uso não apenas para a *P. aeruginosa*, cujas redes metabólicas em escala genômica são analisadas neste trabalho, mas sim, para quaisquer bactérias mapeadas em um arquivo SBML, previamente enunciado, a fim de termos maiores levantamentos de possibilidades de estudos e apresentar novas hipóteses, visando

maior dinamismo no trabalho de bioinformática e biologia de sistemas. A aplicação aqui proposta é flexível ao ponto de poder fazer o mesmo estudo que está apresentado aqui para a *P. aeruginosa* em outros organismos.

2 OBJETIVOS

2.1 Objetivo Geral

Este trabalho tem como objetivo fazer a análise de redes metabólicas previamente mapeadas, para identificar potenciais alvos terapêuticos para a bactéria *P. aeruginosa*.

2.2 Objetivos Específicos

- Identificar potenciais alvos terapêuticos, através da inferência de genes essenciais do organismo de interesse;
- Criar uma ferramenta automatizada, em versão web, para fazer dinamicamente a análise de redes metabólicas previamente mapeadas e as devidas pesquisas em bases de dados de bioinformática;
- Apresentar possíveis hipóteses para fins de experimentos mais precisos em laboratório.

3 MATERIAL E MÉTODOS

A identificação dos alvos foi feita através de um *workflow* [5] computacional que administra a análise da rede metabólica e identifica genes essenciais cuja inibição interrompe a geração de biomassa através da técnica FBA. A lista de potenciais alvos é inferida usando FVA, e o *workflow* recupera possíveis inibidores para os genes identificados, verificando se tais inibidores estão disponíveis como fármacos aprovados e avaliam sua toxicidade para os seres humanos consultando vários repositórios. A análise foi feita mais precisamente a partir de arquivos SBML específicos de *P. aeruginosa*, para as cepas PAO1 2008 (7), PAO1 2017 (8), PA14 (8) e CCBH4851 (12).

A PAO1 versão 2008 é a versão de referência que deu origem as outras versões usadas neste trabalho, onde a PAO1 versão 2017 e a PA14 possuem toda a estrutura mapeada na referência e também contém dados referentes a fatores de virulência, que não são inclusos na cepa de referência. Já se tratando da CCBH4851, ela também usa a referência da PAO1 versão 2008 e pretende conter todas as reações envolvidas tanto na resistência bacteriana, quanto nos fatores de virulência, porém a rede metabólica dessa cepa ainda encontra-se em processo de curadoria. A Tabela 1 apresenta a quantidade de genes, reações e metabólitos encontrados nas modelagens até o momento.

Tabela 1 - Quantidade de genes/reações/metabólitos mapeados em cada modelagem usada neste trabalho.

	PAO1 2008	PAO1 2017	PA14	CCBH4851
Número de genes	1056	1148	1132	Não determinado
Número de reações	1110	1496	1495	944
Número de metabólitos	879	1284	1286	589

[5] Software que automatiza, pelo menos em algum grau, um processo ou processos, podendo ser qualquer processo que exija uma série de etapas com necessária automação via software.

O *workflow* foi implementado usando a linguagem de programação Python, versão 3.6.3 e o framework CobraPy versão 0.9.0. Esta estrutura possui todos os métodos necessários para ler o arquivo SBML (*Systems Biology Markup Language* (36,37)) que descreve a rede metabólica em escala genômica da bactéria em análise. As etapas principais do método são descritas a seguir. O código fonte encontra-se descrito na íntegra no Apêndice B deste documento.

Este método indica potenciais alvos terapêuticos que posteriormente deverão ser confirmados experimentalmente. Para avaliar a robustez e precisão dos resultados obtidos com este método para várias redes metabólicas, pesquisamos a literatura para confirmar os dados dos genes candidatos. Além disso, o tempo total de execução do método é analisado.

As etapas a seguir se iniciam após a preparação dos diretórios para armazenar os arquivos gerados no processo. A aplicação implanta um método assíncrono após preenchimento do formulário de entrada e assim temos certeza de que os arquivos que serão gerados no decorrer da execução são exatamente os resultados dos parâmetros informados no início, sem ter risco de misturar processos e resultados. É uma garantia que mesmo com o multiprocessamento aqui implantado, tenhamos os resultados organizados e enviados para o destinatário correto no final do procedimento. Podemos considerar uma solução computacional para evitar erros.

Todos os resultados de fluxos que serão apresentados aqui estão em escala de 6 casas decimais, visando apresentar uma maior precisão das medidas aqui encontradas e o tempo limite (timeout) determinado para cada requisição nas bases de dados usadas é de até 20 minutos. Caso uma requisição passe desse tempo é emitido uma mensagem de erro e o processo é interrompido.

Os dados de entrada são informados por meio de um formulário, ilustrado na Figura 2 e possui os campos de nome, e-mail, organismo a verificar e o arquivo no formato SBML, juntamente com um botão de confirmação. Todos os dados desse formulário são obrigatórios e, ao confirmar todas as informações de entrada, o sistema verifica o arquivo SBML informado, a fim de comprovar que representa um modelo de bactéria gerando biomassa, exibindo o resultado dessa verificação, conforme ilustrado também abaixo.

O formulário descrito na Figura 2 abaixo pode ser encontrado no endereço http://pseudomonas.procc.fiocruz.br/FindTargetsWEB/.

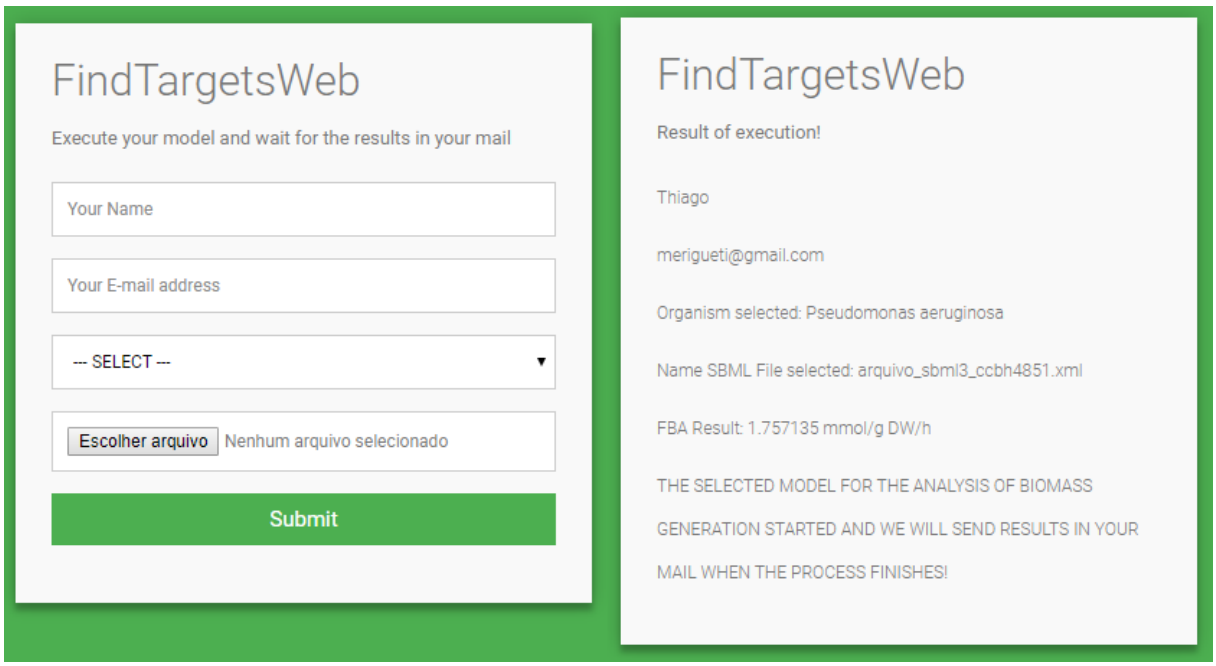

Figura 2 - Ilustração do formulário de entrada da aplicação (esquerda) e o resultado da validação do arquivo informado (direita).

Após essa verificação de geração de biomassa, caso o modelo represente uma bactéria gerando biomassa, as 9 etapas descritas na Figura 2 são ativadas e se inicia o processo e, ao final, teremos como saída uma gama de planilhas contendo os resultados. Caso o modelo informado não represente uma bactéria gerando biomassa, é exibido o resultado e o usuário é informado que não será feita a verificação e que ele deve rever seu modelo. O *workflow* por trás da tela inicial com as suas 9 etapas está descrito detalhadamente no decorrer do capítulo com o fluxograma completo do processo ilustrado na Figura 3.

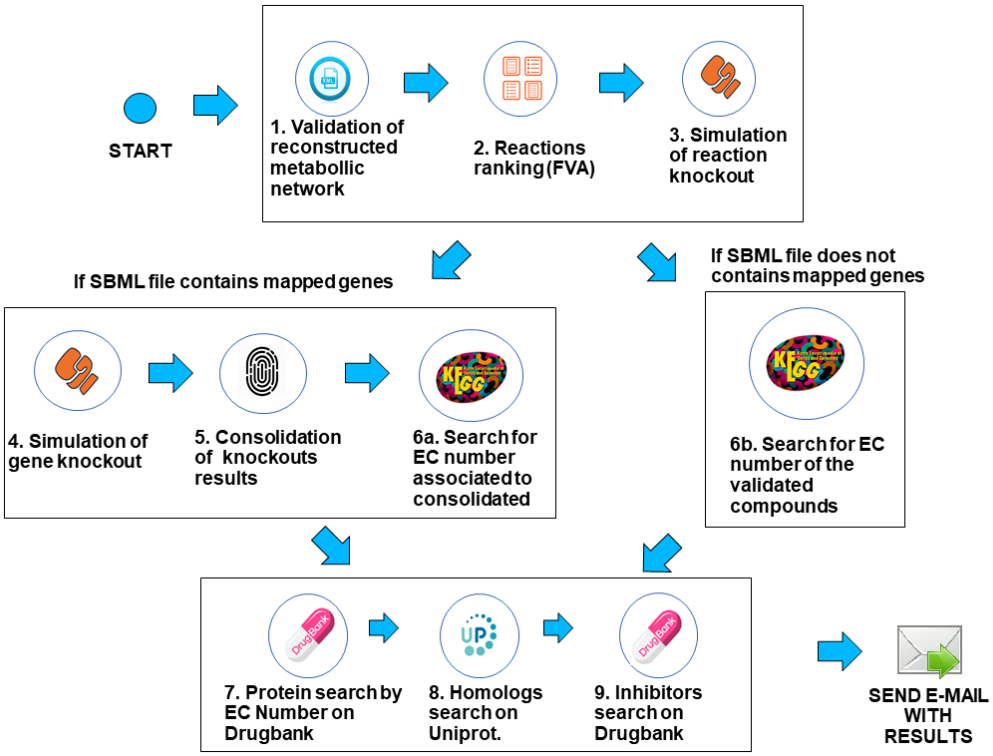

Figura 3 - Descrição do workflow computacional. EC number é para *Enzyme Commission Number*, ou seja, o número associado a uma reação de catálize enzimática.

Etapa 1 - Validação do arquivo SBML descrevendo a rede metabólica em escala genômica: Nesta etapa, o sistema verifica se a reconstrução da rede metabólica gera biomassa. Isso é feito através do método FBA, considerando como função objetivo maximizar geração de biomassa, que calcula fluxos de reação para um estado pseudoestacionário, onde as concentrações de metabólitos não apresentam variações. Se o valor da biomassa for zero, o sistema emitirá um erro para o usuário e vai interromper o processamento, apresentando uma mensagem de erro mostrando que a biomassa é igual a zero. Se o fluxo da reação da biomassa for maior do que zero, o processo de fluxo de trabalho passa para o próximo passo.

Ao constatar que a reconstrução gera biomassa, a aplicação faz a leitura do arquivo SBML, mudando para um formato interpretado pelo framework Cobrapy e o processo continua na etapa seguinte.

Além desse processo, o sistema gera uma planilha com os dados do mapeamento GPR do SBML informado que fará parte da entrega final ao usuário via e-mail.

Etapa 2 – Uso de FVA para classificar as reações: Após a validação da rede metabólica, as reações são ordenadas usando o método FVA. O objetivo é priorizar, nas seguintes etapas de processamento, as reações pelas quais a gama de possíveis valores de fluxo são menores (o mínimo é zero), dado o valor ótimo de geração de biomassa determinado na etapa anterior. O pressuposto subjacente é que as reações de pequenos intervalos de possíveis valores ótimos são menos robustas, mais suscetíveis a perturbações.

Além disso, a aplicação faz uma verificação em paralelo entre o valor do FBA de cada reação e os intervalos encontrados no FVA para essa reação. Caso tivesse algum valor de FBA fora dos limites mínimos e máximos encontrados para a reação analisada, a aplicação emite uma mensagem de erro e interrompe a execução, informando que o usuário deve verificar o arquivo SBML.

Etapa 3 – Simulação de nocaute de reação: Nesta etapa, se realiza o nocaute zerando uma única reação por vez e executando o FBA novamente. Se a geração de biomassa for anulada quando zerar o fluxo de uma determinada reação, se armazena as informações correspondentes em uma lista para posterior processamento. Além disso, ao terminar a análise de FBA com a reação nocauteada, todos os valores do intervalo referente a essa reação são repostos e o processo segue, conforme a ordem de prioridades determinadas dentro do passo anterior.

Nesse ponto, é verificado se os IDs de genes estão disponíveis no arquivo SBML. Caso positivo, o fluxo de trabalho passa para a etapa 4. Caso contrário, ele pula diretamente para a etapa 6b.

Etapa 4: Simulação de nocaute de genes: Nesta etapa, o sistema executa nocaute zerando um único gene descrito no modelo por vez. Para cumprir esta tarefa, o framework CobraPy busca pelas reações que estão ligadas ao gene selecionado e zera o valor mínimo e máximo de cada reação ligada ao gene, levando em consideração as relações GPR. Do mesmo modo que no passo anterior, se o valor da geração de biomassa tiver zerado, se armazena o gene correspondente em uma segunda lista. Vale ressaltar que um gene pode ser associado a mais de uma reação, e uma reação pode exigir a expressão de vários genes, descrito através de relações booleanas por meio de grafos GPR (25).

Da mesma maneira que no passo anterior, após a análise referente às reações envolvidas no gene analisado, todos os limites mínimos e máximos são repostos para que o processo siga adiante.

Etapa 5 – Consolidação/unificação dos resultados dos nocautes: Nessa etapa, as duas listas geradas nas etapas anteriores são utilizadas. A primeira lista é a lista de reações geradas na etapa 3 e a outra é a lista de genes gerada na etapa 4, que tem como resultado uma lista final que contém os valores encontrados em ambas as listas. Para que um gene seja incluído na lista final, ele deve estar presente na lista da etapa 4 e ser associado a, pelo menos, uma reação armazenada na etapa 3. Estes são os genes candidatos que vamos trabalhar a partir de agora no sistema. Deve-se destacar que a lista final é classificada de acordo com o processamento da FVA realizado na etapa 2.

O algoritmo 1 apresenta o pseudocódigo referente a esse passo, visando propor maior clareza e entendimento do processo aqui descrito.

Algoritmo 1: Lista unificada de genes/reações nocauteados (SBML com genes)

Entrada: Lista de genes alvos por reação nocauteada (targetGeneListFromReact) e lista de genes nocauteados (targetGeneList). Listas geradas em tempo de execução.

Saída: Lista unificada de genes alvos em um arquivo texto.

```
1:    procedure UnificationTargetsList(targetGeneListFromReact, targetGeneList)
2:        read targetGeneListFromReact
3:        read targetGeneList
4:        open file "depara.txt"
5:        for all targetgene in targetGeneListFromReact do
6:            if targetgene in targetGeneList then
7:                write targetgene in file “depara.txt”
8:            end if
9:        end for
10:       close file "depara.txt"
11:   end procedure
```

Etapa 6a – Busca do EC number de genes consolidados: Nesta etapa, consultamos a base de dados KEGG (39) para obter o *Enzyme Commission Numbers* (EC) de cada gene incluído na lista de genes final obtida no passo anterior. O resultado do processamento desta etapa é uma lista de números EC associados aos respectivos genes. Em seguida, o fluxo de trabalho prossegue para a etapa 7.

Esse passo é executado porque o número EC é bastante relevante nessa análise, pois a identificação dos fármacos no Drugbank (41) se dá pelo EC (passos seguintes) e, todas as modelagens aqui utilizadas para análise e busca de alvos terapêuticos não continham essa informação em seus arquivos SBML. Com isso, foi necessário levantar os dados dos genes alvos encontrados nos passos anteriores e ir a uma base de dados que tivesse essa associação entre gene e enzima, no nosso caso o KEGG (39).

Etapa 6b – Busca de EC Number usando informações de reação: Caso os IDs de genes não estejam disponíveis no arquivo SBML, o número EC é recuperado do KEGG (39) com base em informações de reação.

Da mesma maneira citada na etapa 6a, além de não termos os dados do número EC, não temos os dados de genes mapeados no arquivo SBML. Isso se dá pelo fato de algumas descrições de redes metabólicas em escala genômica estarem incompletas e que essa ferramenta propõe ser robusta ao ponto de também estar apta a fazer a verificação de potenciais alvos terapêuticos, mesmo com essa informação ainda em estudo e o arquivo não estando com seu mapeamento GPR completo. A busca é feita utilizando todos os compostos envolvidos na reação a analisar. Separamos a equação química em reagente e produto, separamos cada item envolvido e buscamos no KEGG pelas informações de cada um. Depois utilizamos os dados encontrados no KEGG para buscar a reação com os reagentes/produtos no padrão para, por fim, buscar pelo número EC correspondente. Com os dados encontrados, o fluxo segue para a etapa 7.

O algoritmo 2, logo abaixo, apresenta o pseudocódigo referente a esse passo, visando propor maior clareza e entendimento do processo aqui descrito, da mesma maneira que foi feito na etapa 5.

Algoritmo 2: Busca pelo número EC usando reagente/produto descrito na reação (SBML sem genes mapeados)

Entrada: Lista de compostos químicos que formam a reação (listCompoundFromSBML). Lista formada em tempo de execução.
Saida: Lista de ECs encontrados

```
1:procedure alternativeStepToGetECNumberWithoutGenes(listCompoundFromSBML)
2:      # Arquivo com todos os compostos do SBML.
3:      read listCompoundFromSBML
4:
5:      # Instância do módulo Python
6:      k <- KEGG instance
7:
8:      # Determinando o tempo de espera (em segundos)
9:      k.timeout <- 200000
10:
11:     # Composto químico no arquivo SBML. Ex.: 2 A --> B
12:     for all compound in listCompoundFromSBML do
13:
14:             # Busca todos os valores estequimétricos com regex e os remove.
15:             compound_no_stoich <- remove all stoichiometric values in compound
16:             param_splt <- empty
17:
18:             # Verifica se o composto é reversível ou irreversível para separar.
19:             if "<=>" in compound_no_stoich then
20:                     param_splt <- "<=>"
21:             else
22:                     param_splt <- "-->"
23:             end if
24:
25:             # Separa o composto entre reagente e produto. Ex.: compound_splt =
['A', 'B']
26:             compound_splt <- compound_no_stoich.split(param_splt)
27:
```

```
28:         # Se o composto não possuir produto ou reagente, volta e segue o
fluxo com outro composto.
29:         if compound_splt.length < 2 then
30:             continue
31:         end if
32:
33:         list_ec_number_0 <- initialize empty list
34:         list_ec_number_1 <- initialize empty list
35:
36:         # Inicia iteração na lista com o reagente e o produto
37:         for (x=0,1) do
38:
39:             # Pega o reagente ou o produto nesta variável
40:             item_compound <- compound_splt[x] without spaces
41:             list_id_cpd_KEGG <- initialize empty list
42:
43:             # Se o reagente e/ou produto possuírem mais de um
componente, vai no KEGG com cada informação para pegar seu ID.
44:             # Caso contrário, ele vai no KEGG com apenas um
componente.
45:             if " + " in item_compound then
46:                 item_compound_splt <- item_compound.split(" + ")
47:                 for all cpd_item in item_compound_splt do
48:                     # Busca id do componente no KEGG para item
49:                     # e o insere na lista de ids encontrados
50:                     result_id_cpd <- k.find("compound", cpd_item)
51:                     insert result_id_cpd in list_id_cpd_KEGG
52:                 end for
53:             else
54:                 result_id_cpd <- k.find("compound", item_compound)
55:                 insert item_compound in list_id_cpd_KEGG
56:             end if
57:
58:             # Here, we concat all list_id_cpd_KEGG
59:             # found to search the reaction in KEGG.
```

```
60:                 # No Python, se a lista tiver menos de 2 itens,
61:                 # não coloca o "+" no fim da string.
62:                 str_item_compound_in_cpd <- list_id_cpd_KEGG concat with
"+"
63:
64:                 # Busca todas as reações no KEGG com os ids dos
componentes juntos
65:                 result_link_reactions_cpd<-k.link("reaction",
str_item_compound_in_cpd)
66:
67:                 # Todos os resultados encontrados são inseridos nesta lista
68:                 set_id_reaction_KEGG <- insert all reactions found in KEGG.
69:
70:                 # Busca todos os ECs
71:                 # na set_id_reaction_KEGG e insere na result_list_ec
72:                 result_list_ec = k.link("enzyme", set_id_reaction_KEGG)
73:                 if x = 0 then
74:                         insert result_list_ec in list_ec_number_0
75:                 else
76:                         insert result_list_ec in list_ec_number_1
77:                 end if
78:
79:         end for
80:
81:         list_ec_number_intersect <- initialize empty list
82:         txt_file <- initialize txt file
83:
84:         # Inicia outra iteração, com as listas de ECs, buscando as
intercessões.
85:         # Se encontrar, pega o EC e armazena em um arquivo TXT.
86:         for all ec_number_0 in list_ec_number_0 do
87:                 if ec_number_0 in list_ec_number_1 then
88:                         record ec_number_0 in a txt_file
89:                 end if
90:         end for
```

```
91:
92:     end for
93:
94: end procedure
```

Etapa 7 – Busca de dados no Drugbank por EC number: Utilizando os números EC obtidos nos passos anteriores, fazemos uma busca no repositório Drugbank (41) para verificar se esse banco de dados possui qualquer registro dos números EC listados associados à bactéria alvo. Se for encontrada uma correspondência exata, recuperamos os valores do nome da proteína, organismo e Uniprot ID.

Ao executar essa pesquisa, podemos obter dados da proteína encontrada mapeados em outro organismo distinto do organismo estudado. Com isso, torna-se necessário o passo seguinte (etapa 8) para confirmarmos se a proteína encontrada nessa etapa possui uma homóloga no organismo em questão. Podemos ter também nesse momento, alguma possível ocorrência que seja exata do organismo estudado. Independente disso, esse dado é incluído para validação na etapa seguinte. No momento que essa aplicação foi concebida, não existia até então nenhum mapeamento exato para *P. aeruginosa* no Drugbank.

Chamamos de correspondência exata, o exato EC informado na busca, porque o Drugbank apresenta alguns ECs parecidos. Um caso real foi a ocorrência do EC 2.6.1.86 como candidato para a CCBH4851 e, ao fazer a busca dele no Drugbank, o resultado da pesquisa apresenta ECs como 2.6.1.1, 2.6.1.42, 2.6.1.57, entre outros, diferente do exato 2.6.1.86 que pesquisamos, tendo como consequência não prosseguirmos com a pesquisa para esse EC, já que não temos registros deles nessa base de dados.

Já se tratando do UniprotID contido nos resultados do Drugbank, são relevantes para os passos seguintes, que usam esse parâmetro como dado de entrada para andamento do processo.

Aqui é gerada uma planilha com todas as exatas ocorrências de EC, com seus respectivos Uniprot IDs encontrados.

Etapa 8 – Busca por homólogos no Uniprot: Finalmente, procuramos a semelhança de sequência entre a proteína descrita pelo UniprotID e as proteínas codificadas pelo genoma da bactéria alvo usando a ferramenta *Basic Local*

Alignment Search Tool (BLAST) (47), implementada no servidor Uniprot. Se houver um sucesso (obtermos o chamado *hit*), as informações são armazenadas e todos os dados correspondentes referentes ao homólogo encontrado são consultados. Além da pesquisa descrita, é verificado também homologia com humanos e, caso a similaridade com humanos seja maior do que com o organismo estudado, o resultado é descartado, pois pode expressar uma solução que seja prejudicial ao hospedeiro, o que pode acarretar em consequências negativas e não na cura do patógeno.

Esse ponto da aplicação é o mais crítico e demorado de todos os existentes no processo, pois é executado o BLAST de todas as ocorrências do passo anterior, o que acarreta em um processamento considerável, já que é feita uma busca de cada sequencia, usando o UniprotID como entrada e depois fazendo o BLAST propriamente dito, visando a busca pelos homólogos do organismo estudado e, após a conclusão deste processo, é feita uma conversão de cada resultado em XML, para fins de leitura por parte do sistema, além do fato de todas as buscas encontrarem até 1000 ocorrências, com todos os dados já conhecidos de resultados BLAST executados via web. Esta etapa corresponde a 90% do tempo total médio de execução da aplicação.

Nesse ponto é gerada uma planilha com os dados encontrados de cada Uniprot ID, que contenha seu respectivo hit do organismo estudado, juntamente com uma gama de outros dados relevantes como via metabólica, função, localização, dentre outros.

Etapa 9 – Busca pelos inibidores existentes: O último passo é voltar ao Drugbank [36] e consultar o repositório, usando os UniprotID armazenados, com o objetivo de recuperar os inibidores conhecidos para os IDs encontrados, cuja homologia foi confirmada pelo passo anterior, se disponível. Nesse ponto, foi feito também um filtro para os fármacos encontrados, considerando apenas os fármacos com status de aprovado (*Aprovado*), investigado (Investigational) e experimental (*Experimental*).

Aqui é gerada uma planilha com os dados de fármacos encontrados para cada ocorrência identificada no passo anterior e em seguida montada uma planilha com todos os resultados dos três últimos passos concatenados em uma planilha para envio ao usuário.

4 RESULTADOS

4.1 Implementação computacional

Conforme descrito no capítulo anterior, o *workflow* foi dividido em 9 etapas (Figura 1). Primeiramente, o aplicativo lê os dados de entrada: nome do usuário, e-mail, espécies de bactérias e a rede metabólica reconstruída descrita no formato SBML. Após a submissão do formulário de entrada, o sistema gera uma planilha contendo os dados sobre genes, reação e metabólitos obtidos do arquivo SBML e verifica se a rede metabólica reconstruída gera biomassa. Para este fim, o programa executa o FBA para obter o valor ótimo da biomassa. Se o valor da biomassa for positivo, o sistema passa para o próximo passo. Caso contrário, informa ao usuário que a rede metabólica reconstruída não gera biomassa e que o sistema irá interromper a execução atual.

Com o segundo passo, o aplicativo executa o método FVA. O FVA verifica quais reações são ativas analisando os valores de reação máxima e mínima. Se ambos forem diferentes de zero, a reação correspondente será analisada no fluxo de processamento. Em seguida, classificam-se as reações, priorizando as reações com intervalos menores entre valores mínimos e máximos.

Após a obtenção da lista de reações ativas, um nocaute é simulado para cada uma delas, ajustando os seus valores máximos e mínimos a zero, se executa o FBA novamente para verificar o resultado da geração de biomassa. Em seguida, se a geração de biomassa for igual a zero, a reação correspondente é uma candidata e é armazenada separadamente de acordo com sua classificação.

Reiterando o processo de nocaute com os genes depois das reações, o aplicativo escolhe cada gene mapeado relacionado à reconstrução metabólica, simula o nocaute das reações associadas a esse gene e executa novamente o FBA. Se o valor da biomassa for zero para o nocaute de um determinado gene, a informação deste gene é armazenada para uso nas etapas seguintes do processo.

Durante o passo seguinte, a lista dos genes e de reações obtidas nos passos anteriores são comparadas. Os genes que estão presentes em ambas as listas são considerados como possíveis alvos. Em seguida, o programa consulta o repositório KEGG para obter o número EC de cada potencial alvo. Em alguns arquivos SBML, os IDs de genes não estão disponíveis. Nesse caso, o número EC é recuperado

através de consultas com base nos componentes da reação. Com todos os números ECs disponíveis, buscamos resolver a premissa central deste trabalho, que é encontrar o EC da enzima/reação (cujo nocaute zera a produção de biomassa) no drugbank, mostrando que esta é (ou está sendo considerada como) um alvo de fármaco, descrito no Drugbank. Os dados recuperados do Drugbank são o nome da proteína, o organismo associado e o ID Uniprot. Porém as informações cadastradas no Drugbank podem estar associadas a outros organismos que o sob estudo.

Então, o Uniprot ID no Drugbank é usado para consultar o repositório Uniprot para recuperar a sequência proteica correspondente no formato FASTA. A sequência de proteína é usada para executar uma pesquisa BLAST contra o genoma da bactéria-alvo, usando o banco de dados de proteínas de referência Uniprotkb.

Os números de acessos específicos da bactéria em análise são armazenados, a menos que o nível de similaridade de sequência da proteína correspondente seja maior para humanos do que para a bactéria. O último passo é consultar novamente o repositório Drugbank, usando os UniprotIDs armazenados, para recuperar inibidores conhecidos. Este último passo é feito apenas para os ECs já encontrados anteriormente cujas enzimas apresentaram *hits* com a bactéria estudada, no caso *P. aeruginosa*, durante a pesquisa BLAST. O sistema gera planilhas com todos os resultados que são enviados por e-mail ao usuário em um arquivo compactado.

4.2 Desempenho

Todos esses dados foram resultados da extração das bases aqui citadas no capítulo anterior e o tempo total de execução de cada análise foi medido em um servidor Linux com processador Intel Xeon CPU E5-2620 2,0 GHz, com 6 cores HT, 64 Gb de memória RAM, hospedado em um servidor Apache 2.4 e localizado no endereço http://pseudomonas.procc.fiocruz.br/FindTargetsWEB/, denominado servidor Pseudomonas e mantido pelo PROCC/FIOCRUZ (Programa de Computação Científica), tendo a internet com médias de velocidade de download em 90 Mbps (megabit por segundo[6]) e velocidade de upload em 94 Mbps, medidos

[6] Unidade de transmissão de dados equivalente a 1.000 quilobits por segundo ou 1.000.000 bits por segundo.

utilizando a ferramenta disponibilizada em https://simet.nic.br/, que é um portal de medição mantido pelo Núcleo de Informação e Coordenação do Ponto BR (nic.br). O tempo médio de cada execução encontra-se descrito na tabela abaixo.

Tabela 2 - Tempo total médio de execução da aplicação no servidor.

	PAO1 2008	PAO1 2017	PA14	CCBH4851
Execução 1 (minutos)	85	65	100	150
Execução 2 (minutos)	92	80	90	155
Execução 3 (minutos)	100	95	116	170
Execução 4 (minutos)	170	150	160	210
Execução 5 (minutos)	155	170	180	215
Tempo médio e desvio padrão (minutos)	120 (39)	112 (45)	129 (39)	180 (30)

O tempo médio descrito na Tabela 2 foi calculado utilizando 5 execuções realizadas em tempos distintos, onde as 3 primeiras foram feitas na parte da manhã e as 2 últimas foram realizadas na parte da noite.

4.3 Avaliação do workflow e dos modelos das cepas

Para a avaliação da ferramenta proposta neste trabalho, realizamos a análise de quatro redes metabólicas de *P. aeruginosa*, onde para uma delas, a PAO1, foi utilizada 2 versões, uma de 2008 modelada por Oberhardt et al e outra de 2017, modelada por Bartell et al. Além destas, utilizou-se a PA14 também sob a responsabilidade de Bartell et al. e a CCBH4851, uma rede metabólica ainda não publicada, que está sendo modelada pelo nosso grupo na Fundação Oswaldo Cruz e representa uma cepa multirresistente encontrada em um cateter de um paciente hospitalizado em um hospital do estado de Goiás, Brasil (12).

Para a validação da ferramenta, foram feitos testes em outras 2 bactérias diferentes de *P. aeruginosa*, ou seja, *Klebsiella pneumoniae* e *Haemophilus influenzae*, obtendo também a detecção de potenciais alvos terapêuticos, conforme apresentado na tabela abaixo. Essas cepas foram escolhidas, por já possuírem uma versão curada, que foi usada para comprovar o funcionamento do Cobrapy.

Tabela 3 - Organismos testados diferentes de *P. aeruginosa*.

Organismos testados	Número de alvos encontrados
Klebsiella pneumoniae	21
Haemophilus influenzae	9

A rede PAO1, versão 2008 de Oberhardt et al., é a modelagem de referência existente em *P. aeruginosa,* para todas as outras já criadas e publicadas. Já a rede PAO1 de 2017 é a rede mais próxima da realidade em relação a fatores de virulência, assim como a PA14. A PA14 foi criada em paralelo à versão de 2017 e também contempla fatores de virulência. Já se tratando da CCBH4851, ela foi escolhida por ser uma amostra encontrada no Brasil e se mostrar resistente a todas as formas de tratamento possíveis até então.

Vale ressaltar que cada reconstrução indica um valor diferente para a taxa de crescimento após a execução do FBA, conforme mostrado na tabela abaixo. A diferença na taxa de crescimento é devida à equação da biomassa, bem como ao número de genes, reações e metabólitos nos modelos, conforme descrito na Tabela 3. Geralmente a duração do ciclo celular descrita na literatura para a *P. aeruginosa* é entre 45 minutos e 1 hora. (48)

A equação de biomassa, extraída de cada cepa testada, encontra-se descrita na íntegra no Apêndice A deste documento.

Tabela 4 - Fluxo ótimo de reação de biomassa (taxa de crescimento) nas redes metabólicas testadas.

	PAO1 2008 (7)	PAO1 2017 (8)	PA14 (8)	CCBH4851
Taxa de crescimento (h^{-1})	1,047929	15,50964	15,50838	1,757

A confirmação inicial visa verificar se a rede utilizada aqui representa uma bactéria que gera biomassa. Para isso, usamos a ferramenta e executamos o FBA para verificar esse valor ideal, na escala de log. Todas as redes geraram biomassa, conforme apresentado na Tabela 3 e esse resultado demonstra que, nesta simulação, as bactérias levariam menos de 1 hora para se reproduzir. Ressaltamos que uma geração de biomassa diferente de zero é suficiente para o algoritmo processar a rede informada.

Uma consideração a fazer nesse ponto é o fato dos resultados apresentados para a PAO1 2017 e PA14 estarem totalmente equivocados. Podemos provar isso utilizando a fórmula que determina o tempo de duplicação da bactéria, conforme descrito em (6), que após tratamento algébrico, é possível obter por meio da razão entre o log natural (ln) de 2 e a taxa de crescimento (h^{-1}), obtendo um resultado em hora. Após multiplicar por 60, teremos o tempo de duplicação da bactéria em minutos.

Ao aplicar a fórmula para a PA14, por exemplo, faz-se a razão ln(2) / h^{-1} = 0,7 / 15,50838 ≈ 0,045 * 60 ≈ 3 minutos de duplicação, o que podemos considerar absurdo. Já aplicando para a CCBH4851, em outro exemplo, temos 0,7 / 1,757 ≈ 0,40 * 60 ≈ 24 minutos, o que seria mais próximo da realidade. O tempo médio de crescimento bacteriano gira em torno de 20 minutos, dependendo do organismo, do meio e das condições que se encontra naquele momento. (6,49)

A seguir apresentamos o tempo total de geração de cada cepa testada, obtido por meio da aplicação da fórmula de tempo de duplicação bacteriano.

Tabela 5 - Tempo total de duplicação bacteriana (tempo de geração), em função da biomassa, inferido a partir dos 4 modelos analisados.

	PAO1 2008 (7)	PAO1 2017 (8)	PA14 (8)	CCBH4851
Tempo total de geração (minutos)	40	3	3	24

Conclui-se que o resultado de biomassa aqui apresentado não representa uma realidade, conforme provado anteriormente. Entramos em contato com os criadores do artigo de referência e eles alegaram terem focado nos fatores de virulência e que tais taxas de crescimento descritas na Tabela 3 acima estão realmente incorretas. Apesar disso, este *workflow* permanece estável, exibindo a

robustez necessária que essa análise precisa. Além disso, a rede metabólica da cepa multirresistente CCBH4851 ainda está incompleta, buscando um modelo mais próximo de representação de bactérias. Entretanto, a taxa de crescimento da rede metabólica da cepa CCBH4851 é mais próxima da realidade que a taxa das redes das cepas criadas em 2017.

Obtivemos uma gama de alvos inferidos nos modelos das 4 cepas testadas, com as mais diversas combinações. Encontramos os mesmos alvos particulares e comuns em modelos de 2, 3 e 4 cepas, conforme apresentamos abaixo por meio do diagrama de Venn, para fins de ilustração da quantidade de números ECs que foram encontrados dentre os 4 modelos (Figura 4).

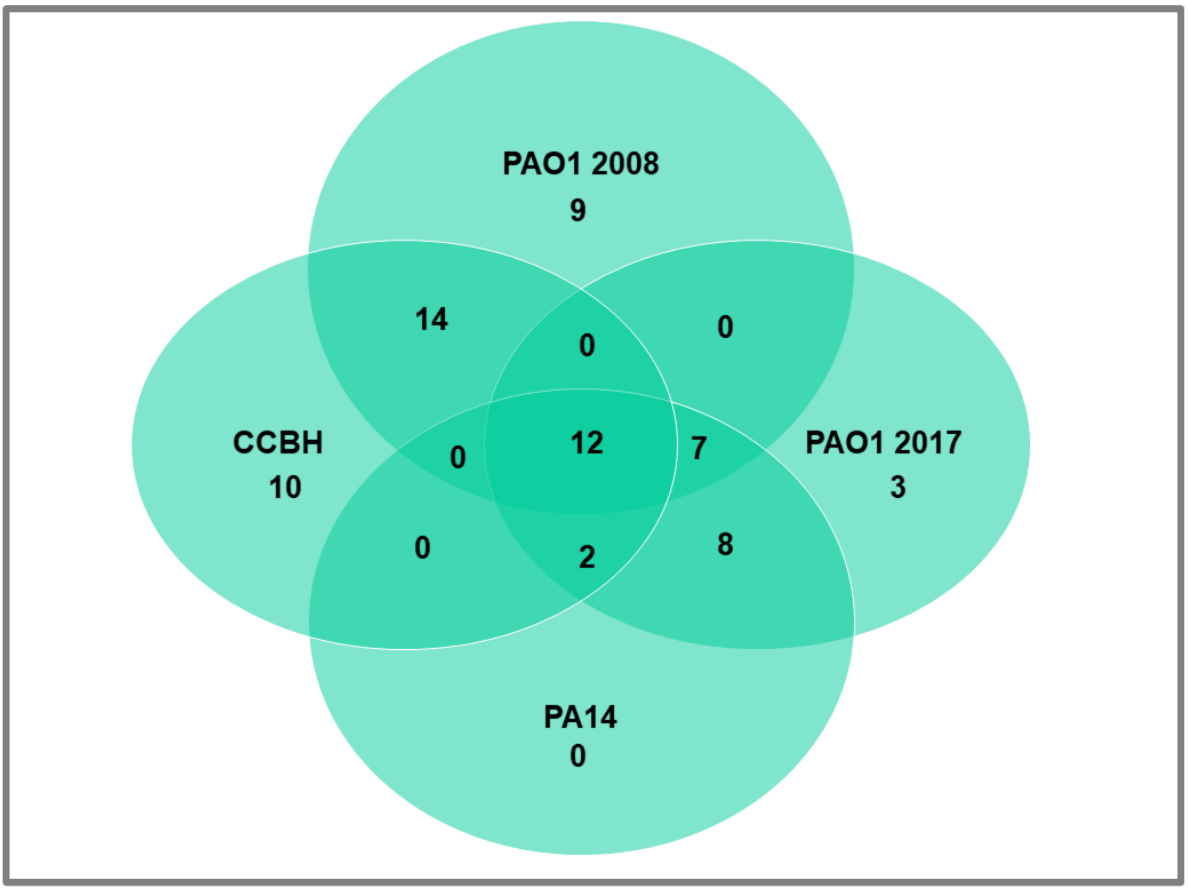

Figura 4 - Diagrama de Venn representando o número de ocorrências dentre os 4 modelos. Total de alvos = 65.

Abaixo apresenta-se uma série de seções referentes aos resultados de cada modelo testado neste trabalho. Cada seção é formada por 2 tabelas, onde a primeira tabela tem os dados de EC, proteína, gene, e-value e via metabólica. Foi criado o apêndice C para apresentar dados adicionais da primeira tabela descrita, tais como função e localização de cada ocorrência apresentada. A segunda tabela tem as

informações dos fármacos para cada alvo mapeado na tabela anterior, com os dados de nome do fármaco, grupo e o organismo alvo conhecido.

4.4 Alvos comuns a *P. aeruginosa*

Nesta seção apresentamos os resultados conforme as combinações apresentadas no diagrama de Venn acima (Figura 4), sendo que, assim como na Figura, as combinações que não aparecem, não possuem nenhuma ocorrência entre si, como por exemplo, PA14 e CCBH4851 que não tiveram nenhum resultado em comum.

4.4.1 Alvos e fármacos comuns aos quatro modelos

Aqui apresentamos os resultados obtidos para todos os cenários testados, ou seja, os 12 potenciais alvos terapêuticos aqui mostrados foram inferidos em quaisquer dos modelos analisados, apresentadas pela priorização média do intervalo encontrado no FVA.

Tabela 6 - Lista de potenciais alvos terapêuticos determinados a partir dos 4 modelos testados.

EC	Proteína	Gene	E-Value	Via
1.1.1.25	Shikimate desidrogenase	aroE PA0025	6E-84	Biossíntese intermediária metabólica; biossíntese de cororateamento; corismato de D-erythrose 4-fosfato e fosfoenolpiruvato: passo 4/7.
1.3.1.98	UDP-N-acetilenolpiruvoilglucosamina redutase	murB PA2977	4,7E-79	Biogênese da parede celular; biossíntese de peptidoglicano.
1.5.1.3	Diidrofolato redutase	folA PA0350	2E-38	Biossíntese de cofactores; biossíntese de tetrahidrofolato; 5,6,7,8-tetra-hidrofolato a partir de 7,8-di-hidrofolato: passo 1/1.
2.1.1.45	Timidilato sintase	thyA PA0342	4,3E-141	Metabolismo de pirimidina; biossíntese de TDT.
2.4.1.227	UDP-N-acetilglucosamina--N-acetilmuramil-(pentapeptídeo) pirofosforil-undecaprenol N-acetilglucosamina transferase	murG PA4412	2,3E-88	Biogênese da parede celular; biossíntese de peptidoglicano.
2.5.1.15	Dihidropteroato sintase 1	folP PA4750	5,8E-103	Biossíntese de cofactores; biossíntese de tetrahidrofolato; 7,8-di-hidrofolato a partir de difosfato de 2-amino-4-hidroxi-6-hidroximetil-7,8-dihidropteridina e 4-aminobenzoato: passo 1/2.
2.5.1.7	UDP-N-acetilglucosamina 1-carboxiviniltransferase	murA PA4450	5,1E-171	Biogênese da parede celular; biossíntese de peptidoglycan.
2.6.1.85	Para-aminobenzoato sintase componente 1	pabB PA1758	3,2E-119	N/A
2.7.4.25	Cytidylate quinase	cmk PA3163	4,9E-86	N/A
6.3.2.13	UDP-N-acetilmuramil-L-	murE PA4417	4,7E-131	Biogênese da parede celular; biossíntese de peptidoglycan.

	alanil-D-glutamato--2,6-diaminopimelato-ligase			
6.3.2.9	UDP-N-acetilmuramoilalanina--D-glutamato-ligase	murD PA4414	5,8E-123	Biogênese da parede celular; biossíntese de peptidoglycan.
1.1.1.100	3-oxoacil-[ACP] redutase FabG	fabG PA2967	1,5E-99	Metabolismo lipídico; biossíntese de ácidos graxos.

Abaixo seguimos apresentando os fármacos encontrados para cada alvo terapêutico mapeado na Tabela 6.

Tabela 7 - Fármacos mapeados para cada alvo considerando todos os modelos.

EC 1.1.1.100 - 3-oxoacil-[ACP] reductase FabG		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
2'-Monophosphoadenosine 5'-Diphosphoribose	experimental	*E. coli*
EC 1.1.1.25 - Shikimate desidrogenase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
2'-Monophosphoadenosine-5'-Diphosphate	experimental	*H. influenzae*
1,4-Dithiothreitol	experimental	*E. coli*
2'-Monophosphoadenosine 5'-Diphosphoribose	experimental	*E. coli*
EC 1.3.1.98 - UDP-N-acetilenolpiruvoilglucosamina redutase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
(5Z)-3-(4-CHLOROPHENYL)-4-HYDROXY-5-(1-NAPHTHYLMETHYLENE) FURAN-2(5H)-ONE	experimental	*E. coli*
Flavin adenine dinucleotide	aprovado	*E. coli*
EC 1.5.1.3 - Diidrofolato redutase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
2'-Monophosphoadenosine 5'-Diphosphoribose	experimental	*E. coli*
Urea	aprovado	*E. coli*
Lometrexol	investigacional	*E. coli*
Dihydrofolic Acid	experimental	*E. coli*
2'-Monophosphoadenosine-5'-Diphosphate	experimental	*E. coli*
Levoleucovorin	aprovado	*E. coli*
1-{[N-(1-IMINO-GUANIDINO-METHYL)]SULFANYLMETHYL}-3-	experimental	*E. coli*

TRIFLUOROMETHYL-BENZENE		
Isoniazid	aprovado	*M. tuberculosis*
Bromo-WR99210	experimental	*M. tuberculosis*
EC 2.1.1.45 - Timidilato sintase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Tosyl-D-Proline	experimental	*E. coli*
L-methionine (S)-S-oxide	experimental	*E. coli*
(6S)-5,6,7,8-tetrahydrofolate	experimental	*E. coli*
2'-Deoxyuridine	experimental	*E. coli*
N-Carboxymethionine	experimental	*E. coli*
5,10-Methylene-6-Hydrofolic Acid	experimental	*E. coli*
N,O-Didansyl-L-Tyrosine	experimental	*E. coli*
2'-5'dideoxyuridine	experimental	*E. coli*
10-Propargyl-5,8-Dideazafolic Acid	experimental	*E. coli*
Sp-876	experimental	*E. coli*
5-Fluoro-2'-Deoxyuridine-5'-Monophosphate	experimental	*E. coli*
Ly231514 Tetra Glu	experimental	*E. coli*
2'-deoxyuridylic acid	experimental	*E. coli*
1,4-Dithiothreitol	experimental	*E. coli*
2'-Deoxyguanosine-5'-Monophosphate	experimental	*E. coli*
Sp-722	experimental	*E. coli*
S,S-(2-Hydroxyethyl)Thiocysteine	experimental	*E. coli*
2-bromophenol	experimental	*E. coli*
4-CHLORO-3',3''-DIBROMOPHENOL-1,8-NAPHTHALEIN	experimental	*E. coli*
2-{4-[2-(2-AMINO-4-OXO-4,7-DIHYDRO-3H-PYRROLO[2,3-D]PYRIMIDIN-5-YL)-ETHYL]-BENZOYLAMINO}-3-METHYL-BUTYRIC ACID	experimental	*E. coli*
N-[Tosyl-D-Prolinyl]Amino-Ethanethiol	experimental	*E. coli*
LY341770	experimental	*E. coli*

5-Fluoro-2'-Deoxyuridine-5'-Monophosphate	experimental	*B. subtilis*
EC 2.4.1.227 - UDP-N-acetilglucosamina--N-acetilmuramil-(pentapeptídeo) pirofosforil-undecaprenol N-acetilglucosamina transferase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Uridine-Diphosphate-N-Acetylgalactosamine	experimental	*E. coli*
EC 2.5.1.15 - Dihidropteroato sintase 1		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Sulfanilamide	aprovado	*E. coli*
Sulfacytine	aprovado	*E. coli*
Sulfisoxazole	aprovado, vet_aprovado	*E. coli*
Sulfaphenazole	aprovado	*E. coli*
Sulfamethazine	aprovado, investigational, vet_aprovado	*E. coli*
Sulfamerazine	aprovado, vet_aprovado	*E. coli*
Sulfamethoxazole	aprovado	*E. coli*
Sulfamethizole	aprovado, investigational, vet_aprovado	*E. coli*
Sulfacetamide	aprovado	*E. coli*
Dapsone	aprovado, investigational	*M. leprae*
EC 2.5.1.7 - UDP-N-acetilglucosamina 1-carboxiviniltransferase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Fosfomycin	aprovado	*E. coli*
(S)-2-{Methyl-[2-(Naphthalene-2-Sulfonylamino)-5-(Naphthalene-2-Sulfonyloxy)-Benzoyl]-Amino}-Succinicacid	experimental	*Enterobacter cloacae subsp. cloacae*
Aminomethylcyclohexane	experimental	*Enterobacter cloacae subsp. cloacae*
Cyclohexylammonium Ion	experimental	*Enterobacter cloacae subsp. cloacae*
3'-1-Carboxy-1-Phosphonooxy-Ethoxy-Uridine-Diphosphate-N-Acetylglucosamine	experimental	*Enterobacter cloacae subsp. cloacae*
1-Anilino-8-Naphthalene Sulfonate	experimental	*Enterobacter cloacae subsp. cloacae*
Uridine-Diphosphate-N-Acetylglucosamine	experimental	*S. flexneri*
EC 2.6.1.85 - Para-aminobenzoato sintase componente 1		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**

Formic Acid	aprovado, experimental	*E. coli*
EC 2.7.4.25 - Cytidylate quinase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
2',3'-Dideoxycytidine-5'-Monophosphate	experimental	*E. coli*
Aracytidine 5'-monophosphate	experimental	*E. coli*
Cytidine-5'-Diphosphate	experimental	*E. coli*
Cytidine-5'-Monophosphate	experimental	*E. coli*
EC 6.3.2.13 - UDP-N-acetilmuramil-L-alanil-D-glutamato--2,6-diaminopimelato-ligase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Lysine Nz-Carboxylic Acid	experimental	*E. coli*
Uridine-5'-Diphosphate-N-Acetylmuramoyl-L-Alanine-D-Glutamate	experimental	*E. coli*
2,6-Diaminopimelic Acid	experimental	*E. coli*
EC 6.3.2.9 - UDP-N-acetilmuramoilalanina--D-glutamato-ligase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Uridine-5'-Diphosphate-N-Acetylmuramoyl-L-Alanine	experimental	*E. coli*
Uridine-5'-Diphosphate-N-Acetylmuramoyl-L-Alanine-D-Glutamate	experimental	*E. coli*
Lysine Nz-Carboxylic Acid	experimental	*E. coli*
N-[(6-butoxynaphthalen-2-yl)sulfonyl]-D-glutamic acid	experimental	*E. coli*
N-{[6-(PENTYLOXY)NAPHTHALEN-2-YL]SULFONYL}-D-GLUTAMIC ACID	experimental	*E. coli*
N-({6-[(4-CYANOBENZYL)OXY]NAPHTHALEN-2-YL}SULFONYL)-D-GLUTAMIC ACID	experimental	*E. coli*
N-({6-[(4-CYANO-2-FLUOROBENZYL)OXY]NAPHTHALEN-2-YL}SULFONYL)-D-GLUTAMIC ACID	experimental	*E. coli*
N-[(6-butoxynaphthalen-2-yl)sulfonyl]-L-glutamic acid	experimental	*E. coli*

4.4.2 Alvos e fármacos comuns aos modelos PAO1 2008, PAO1 2017 e PA14

Além dos resultados comuns a todos, tivemos resultados comuns a 3 dos 4 cenários testados. Neste caso tivemos 7 ocorrências em comum, apresentadas abaixo.

Tabela 8 - Lista de potenciais alvos encontrados para os modelos PAO1 2008, PAO1 2017 e PA14.

EC	Proteína	Gene	E-Value	Via
1.17.1.8	4-hidroxi-tetra-hidrodipicolinato redutase	dapB PA4759	1,5E-107	Biossíntese de aminoácidos; Biossíntese de L-lisina via via DAP; (S) -tetra-hidrodipicolinato de L-aspartato: passo 4/4.
2.5.1.61	Porphobilinogen deaminase	hemC PA5260	5,8E-137	Metabolismo composto contendo porfirina; biossíntese de protoporfirina-IX; coproporfirinógeno-III a partir de 5-aminolevulinato: passo 2/4.
2.7.7.23	Proteína bifuncional GlmU	glmU PA5552	1,2E-164	Biossíntese de açúcares nucleotídicos; Biossíntese de UDP-N-acetil-alfa-D-glucosamina; N-acetil-alfa-D-glucosamina 1-fosfato a partir de 6-fosfato de alfa-D-glucosamina (via II): passo 2/2. Biossíntese de açúcares nucleotídicos; Biossíntese de UDP-N-acetil-alfa-D-glucosamina; UDP-N-acetil-alfa-D-glucosamina a partir de 1-fosfato de N-acetil-alfa-D-glucosamina: passo 1/1. Biogênese da membrana externa bacteriana; Biossíntese LPS lipídica A.
4.1.3.38	Aminodeoxicorismato liase	pabC PA2964	1,4E-35	N/A
4.2.3.5	Cororato sintase	aroC PA1681	3,6E-95	Biossíntese intermediária metabólica; biossíntese de cororateamento; corismato de D-erythrose 4-fosfato e fosfoenolpiruvato: passo 7/7.
5.3.1.1	Triosephosphate isomerase	tpiA PA4748	2E-74	Biossíntese de carboidratos; gluconeogênese. Degradação de carboidratos; glicolise; D-gliceraldeído 3-fosfato a partir de fosfato de glicerona: passo 1/1.
5.3.1.6	Ribose-5-fosfato isomerase A	rpiA PA0330	4,6E-88	Degradação de carboidratos; caminho de pentose fosfato; 5-fosfato de D-ribose a partir de 5-fosfato de D-ribulose (estádio não oxidativo): passo 1/1.

Abaixo seguimos apresentando os fármacos encontrados para cada alvo terapêutico mapeado na Tabela 8.

Tabela 9 - Fármacos mapeados para cada alvo inferido a partir dos modelos PAO1 2008, PAO1 2017 e PA14

EC 1.17.1.8 - 4-hidroxi-tetra-hidrodipicolinato redutase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Dipicolinic Acid	experimental	*E. coli*
3-Acetylpyridine Adenine Dinucleotide	experimental	*E. coli*
EC 2.5.1.61 - Porphobilinogen deaminase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Dipyrromethane Cofactor	experimental	*E. coli*
EC 2.7.7.23 - Proteína bifuncional GlmU		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
4-chloro-N-(3-methoxypropyl)-N-[(3S)-1-(2-phenylethyl)piperidin-3-yl]benzamide	experimental	*H. influenzae*
EC 4.1.3.38 - Aminodeoxicorismato liase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
D-[3-hydroxy-2-methyl-5-phosphonooxymethyl-pyridin-4-ylmethyl]-N,O-cycloserylamide	experimental	*E. coli*
EC 4.2.3.5 - Cororato sintase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Riboflavin Monophosphate	experimental	*H. pylori*
EC 5.3.1.1 - Triosephosphate isomerase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
2-Phosphoglycolic Acid	experimental	*V. marinus*
EC 5.3.1.6 - Ribose-5-fosfato isomerase A		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
beta-D-arabinofuranose 5-phosphate	experimental	*S. flexneri*
Citric Acid	aprovado, nutraceutical, vet_aprovado	*H. influenzae*

4.4.3 Alvos e fármacos comuns aos modelos PAO1 2017, PA14 e CCBH4851

Se tratando desta combinação, obtivemos apenas 2 ocorrências em comum, apresentadas abaixo.

Tabela 10 - Lista de potenciais alvos encontrados para os modelos PAO1 2017, PA14 e CCBH4851.

EC	Proteína	Gene	E-Value	Via
2.4.2.10	Orotate phosphoribosyltransferase	pyrE PA5331	1,4E-103	Metabolismo de pirimidina; Biossíntese UMP via via de novo; UMP do orotate: passo 1/2.
4.3.3.7	4-hidroxi-tetrahidrodipicolinato sintase	dapA PA1010	3,6E-79	Biossíntese de aminoácidos; Biossíntese de L-lisina via via DAP; (S) -tetra-hidrodipicolinato de L-aspartato: passo 3/4.

Continuando a seção, apresentamos a seguir a tabela contendo os dados de fármacos para os alvos encontrados.

Tabela 11 - Fármacos mapeados para cada alvo inferido a partir dos modelos PAO1 2017, PA14 e CCBH4851.

EC 2.4.2.10 - Orotate phosphoribosyltransferase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Orotic Acid	experimental, investigational	*S. typhimurium*
Alpha-Phosphoribosylpyrophosphoric Acid	experimental, investigational	*S. typhimurium*
EC 4.3.3.7 - 4-hidroxi-tetrahidrodipicolinato sintase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Nz-(1-Carboxyethyl)-Lysine	experimental	*T. maritima*

4.4.4 Alvos e fármacos comuns aos modelos PAO1 2008 e CCBH4851

Tivemos ocorrência em comum a pelo menos 2 das 4 cepas testadas. Neste caso tivemos 14 ocorrências em comum, apresentadas na tabela abaixo.

Tabela 12 - Lista de potenciais alvos encontrados para os modelos PAO1 2008 e CCBH4851.

EC	Proteína	Gene	E-Value	Via
1.1.1.133	dTDP-4-deshidrorhamnose redutase	rmlD PA5162	1,2E-95	Biossíntese de carboidratos; biossíntese dTDP-L-Rhamnose.
1.3.1.9	Enoyl-[ACP] redutase [NADH] FabI	fabI PA1806	1,1E-121	Metabolismo lipídico; biossíntese de ácidos graxos.
2.1.2.11	3-metil-2-oxobutanoato hidroximetiltransferase	panB2 PA4729	1,1E-93	Biossíntese de cofactores; Biossíntese de pantotenato (R); (R) -panoato de 3-metil-2-oxobutanoato: passo 1/2.
2.5.1.55	2-desidro-3-desoxifosocótono aldolase	kdsA PA3636	2,6E-137	Biossíntese de carboidratos; Biossíntese de 3-desoxi-D-manno-octulosonato; 3-desoxi-D-manno-octulosonato de D-ribulose 5-fosfato: passo 2/3. Biogênese da membrana externa bacteriana; biossíntese de lipopolissacarídeos.
2.7.1.24	Dephospho-CoA quinase	coaE PA4529	1,9E-56	Biossíntese de cofactores; biossíntese de coenzima A; CoA de (R) -pantotenato: passo 5/5.
2.7.1.26	Proteína de biossíntese de riboflavina RibF	ribF PA4561	5,2E-42	Biossíntese de cofactores; Biossíntese de FAD; FAD da FMN: passo 1/1. Biossíntese de cofactores; Biossíntese FMN; FMN da riboflavina (via ATP): passo 1/1.
2.7.4.9	Timidilato quinase	tmk PA2962	8,6E-46	N/A
2.7.7.18	Nicotinato-nucleótido adenililtransferase	nadD PA4006	1,8E-38	Biossíntese de cofactores; Bensíntese de NAD (+); deamido-NAD (+) do nicotinato D-ribonucleótido: passo 1/1.
2.7.7.3	Phosphopantetheine adenylyltransferase	coaD PA0363	1,4E-48	Biossíntese de cofactores; biossíntese de coenzima A; CoA do (R) -pantotenato: passo 4/5.
2.7.7.38	3-desoxi-manno-octulosonato cytidylytransferase	kdsB PA2979	2,4E-84	Biossíntese de açúcares nucleotídicos; Biossíntese de CMP-3-desoxi-D-manno-octulosonato; CMP-3-desoxi-D-mano-octulosonato a partir de 3-desoxi-D-mano-octosonato e CTP: passo 1/1. Biogênese da membrana externa bacteriana; biossíntese de lipopolissacarídeos.
3.1.3.45	3-desoxi-D-manno-octulosonato 8-fosfato fosfatase KdsC	PA4458	5,5E-34	N/A
3.6.1.1	Pirofosfatase inorgânica	ppa PA4031	3,9E-73	N/A
4.2.1.46	dTDP-glucose 4,6-desidratase	rmlB PA5161	2,1E-152	N/A

6.3.2.1	Pantotenato sintetase	panC PA4730	7,4E-77	Biossíntese de cofactores; Biossíntese de pantotenato (R); (R) -pantotenato de (R) -pantoato e beta-alanina: passo 1/1.

Abaixo apresentando os fármacos encontrados para os alvos mapeados na Tabela 12.

Tabela 13 - Fármacos mapeados para cada alvo inferido a partir dos modelos PAO1 2008 e CCBH4851.

EC 1.1.1.133 - dTDP-4-deshidrorhamnose redutase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
2'-Deoxy-Thymidine-Beta-L-Rhamnose	experimental	*S. typhimurium*
EC 1.3.1.9 - Enoyl-[ACP] redutase [NADH] FabI		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Beta-D-Glucose	experimental	*E. coli*
Indole Naphthyridinone	experimental	*E. coli*
3-(6-Aminopyridin-3-Yl)-N-Methyl-N-[(1-Methyl-1h-Indol-2-Yl)Methyl]Acrylamide	experimental	*E. coli*
4-(2-Thienyl)-1-(4-Methylbenzyl)-1h-Imidazole	experimental	*E. coli*
3-[(Acetyl-Methyl-Amino)-Methyl]-4-Amino-N-Methyl-N-(1-Methyl-1h-Indol-2-Ylmethyl)-Benzamide	experimental	*E. coli*
1,3,4,9-Tetrahydro-2-(Hydroxybenzoyl)-9-[(4-Hydroxyphenyl)Methyl]-6-Methoxy-2h-Pyrido[3,4-B]Indole	experimental	*E. coli*
6-METHYL-2(PROPANE-1-SULFONYL)-2H-THIENO[3,2-D][1,2,3]DIAZABORININ-1-OL	experimental	*E. coli*
Triclosan	aprovado	*E. coli*
2-(TOLUENE-4-SULFONYL)-2H-BENZO[D][1,2,3]DIAZABORININ-1-OL	experimental	*E. coli*
EC 2.1.2.11 - 3-metil-2-oxobutanoato hidroximetiltransferase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
2-Dehydropantoate	experimental	*E. coli*

Alpha-ketoisovalerate	experimental	*Neisseria meningitidis serogroup B*
EC 2.5.1.55 - 2-desidro-3-desoxifosocótono aldolase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
1-Deoxy-Ribofuranose-5'-Phosphate	experimental	*S. flexneri*
3-Fluoro-2-(Phosphonooxy)Propanoic Acid	experimental	*S. flexneri*
{[(2,2-Dihydroxy-Ethyl)-(2,3,4,5-Tetrahydroxy-6-Phosphonooxy-Hexyl)-Amino]-Methyl}-Phosphonic Acid	experimental	*S. flexneri*
Phosphoenolpyruvate	experimental	*S. flexneri*
EC 2.7.1.24 - Dephospho-CoA quinase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Bis(Adenosine)-5'-Triphosphate	experimental	*E. coli*
EC 2.7.1.26 - Proteína de biossíntese de riboflavina RibF		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
7,8-dimethylalloxazine	experimental	*T. maritima*
Riboflavin Monophosphate	experimental	*T. maritima*
Citric Acid	aprovado, nutraceutical, vet_aprovado	*T. maritima*
EC 2.7.4.9 - Timidilato quinase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
P1-(5'-Adenosyl)P5-(5'-Thymidyl)Pentaphosphate	experimental	*S. flexneri*
EC 2.7.7.18 - Nicotinato-nucleótido adenililtransferase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Citric Acid	aprovado, nutraceutical, vet_aprovado	*S. flexneri*
EC 2.7.7.3 - Phosphopantetheine adenylyltransferase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Dephospho Coenzyme A	experimental	*S. flexneri*
Coenzyme A	investigational, nutraceutical	*S. flexneri*
4'-Phosphopantetheine	experimental	*S. flexneri*
4'-Phosphopantetheine	experimental	*T. thermophilus*

EC 2.7.7.38 - 3-deoxi-manno-octulosonato cytidylytransferase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Cmp-2-Keto-3-Deoxy-Octulosonic Acid	experimental	*H. influenzae*
EC 3.1.3.45 - 3-desoxi-D-manno-octulosonato 8-fosfato fosfatase KdsC		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
2-(N-Morpholino)-Ethanesulfonic Acid	experimental	*H. influenzae*
EC 3.6.1.1 - Pirofosfatase inorgânica		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
N-(pyridin-3-ylmethyl)aniline	experimental	*B. pseudomallei*
5-amino-1-(4-chlorophenyl)-1H-pyrazole-4-carbonitrile	experimental	*B. pseudomallei*
EC 4.2.1.46 - dTDP-glucose 4,6-desidratase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
2'deoxy-Thymidine-5'-Diphospho-Alpha-D-Glucose	experimental	*S. typhimurium*
EC 6.3.2.1 - Pantotenato sintetase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Tris-Hydroxymethyl-Methyl-Ammonium	experimental	*T. thermophilus*
Pantoyl Adenylate	experimental	*M. tuberculosis*
Alpha,Beta-Methyleneadenosine-5'-Triphosphate	experimental	*M. tuberculosis*
2,4-Dihydroxy-3,3-Dimethyl-Butyrate	experimental	*M. tuberculosis*
Beta-Alanine	experimental	*M. tuberculosis*

4.4.5 Alvos e fármacos comuns aos modelos PAO1 2017 e PA14

Neste caso tivemos 8 ocorrências para os 2 dos 4 cenários testados. Destaco que esses 2 modelos de rede metabólica da *P. aeruginosa* foram feitos pela mesma equipe, no caso Bartell et al. 2017.

Tabela 14 - Lista de potenciais alvos encontrados para os modelos PAO1 2017 e PA14.

EC	Proteína	Gene	E-Value	Via
2.5.1.10	Farnesil difosfato sintase	ispA PA4043	1,9E-81	N/A
2.7.1.148	4-difosfocitil-2-C-metil-D-eritritol quinase	ispE PA4669	1,1E-97	Biossíntese isoprenóidea; biossíntese de isopentenil difosfato via via DXP; difosfato de isopentenilo a partir de 1-desoxi-D-xilulose 5-fosfato: passo 3/6.
2.7.7.60	2-C-metil-D-eritritol 4-fosfato cytidylytransferase	ispD PA3633	9,9E-52	Biossíntese isoprenóidea; biossíntese de isopentenil difosfato via via DXP; difosfato de isopentenilo a partir de 1-desoxi-D-xilulose 5-fosfato: passo 2/6.
3.4.16.4	D-alanyl-D-alanine carboxypeptidase	dacC PA3999, PA3047, ftsI pbpB PA4418, pbpC PA2272	4,7E-156	N/A
3.5.2.3	Dihydroorotase	pyrC PA3527	5,6E-130	Metabolismo de pirimidina; Biossíntese UMP via via de novo; (S) -dihidroorotato de bicarbonato: passo 3/3.
4.1.1.23	Orotidina 5'-fosfato descarboxilase	pyrF PA2876	4,2E-85	Metabolismo de pirimidina; Biossíntese UMP via via de novo; UMP do orotate: passo 2/2.
4.2.1.59	3-hidroxidecanoil-[ACP] desidratada	fabA PA1610	2,9E-81	Metabolismo lipídico; biossíntese de ácidos graxos.
4.6.1.12	2-C-metil-D-eritritol 2,4-ciclodifosfato sintase	ispF PA3627	2,9E-69	Biossíntese isoprenóidea; biossíntese de isopentenil difosfato via via DXP; difosfato de isopentenilo a partir de 5-fosfato de 1-desoxi-D-xilulose: passo 4/6.

Agora apresentamos a tabela contendo os fármacos encontrados para os alvos descritos na tabela anterior.

Tabela 15 - Fármacos mapeados para cada alvo inferido a partir dos modelos PAO1 2017 e PA14.

EC 2.5.1.10 - Farnesil difosfato sintase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Dimethylallyl S-Thiolodiphosphate	experimental	*E. coli*
Pyrophosphoric acid	aprovado, experimental	*E. coli*
Isopentyl Pyrophosphate	experimental	*E. coli*
EC 2.7.1.148 - 4-difosfocitil-2-C-metil-D-eritritol quinase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
4-Diphosphocytidyl-2-C-Methyl-D-Erythritol	experimental	*S. flexneri*
Phosphoaminophosphonic Acid-Adenylate Ester	experimental	*S. flexneri*
EC 2.7.7.60 - 2-C-metil-D-eritritol 4-fosfato cytidylytransferase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Pentane-1,5-Diamine	experimental	*E. coli*
4-Diphosphocytidyl-2-C-Methyl-D-Erythritol	experimental	*E. coli*
Cytidine-5'-Triphosphate	experimental	*E. coli*
EC 3.4.16.4 - D-alanil-D-alanina carboxipeptidase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Cloxacillin	aprovado, vet_aprovado	*E. coli*
BOC-GAMMA-D-GLU-L-LYS(CBZ)-D-BOROALA	experimental	*E. coli*
(2R)-2-{(1R)-2-oxo-1-[(2-thienylacetyl)amino]ethyl}-5,6-dihydro-2h-1,3-thiazine-4-carboxylic acid	experimental	*Actinomadura sp.*
Cefmetazole	aprovado, investigational	*E. coli*
Ertapenem	aprovado, investigational	*E. coli*
Cefpiramide	aprovado	*E. coli*
Cefoxitin	aprovado	*E. coli*
Cefoperazone	aprovado, investigational	*E. coli*
Cefoxitin	aprovado	*E. coli*
Cefoperazone	aprovado, investigational	*E. coli*

BOC-GAMMA-D-GLU-L-LYS(CBZ)-D-BOROALA	experimental	*E. coli*
Cefmetazole	aprovado, investigational	*E. coli*
faropenem medoxomil	investigational	*E. coli*
faropenem medoxomil	investigational	*E. coli*
EC 3.5.2.3 - Dihidroorase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Lysine Nz-Carboxylic Acid	experimental	*E. coli*
Orotic Acid	experimental, investigational	*E. coli*
Dihydroorotic Acid	experimental	*E. coli*
N-Carbamoyl-L-Aspartate	experimental	*E. coli*
EC 4.1.1.23 - Orotidina 5'-fosfato descarboxilase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
1-(5'-Phospho-Beta-D-Ribofuranosyl)Barbituric Acid	experimental	E. coli
6-Hydroxyuridine-5'-Phosphate	experimental	E. coli
EC 4.2.1.59 - 3-hidroxidecanoil-[ACP] desidratada		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
2-Decenoyl N-Acetyl Cysteamine	experimental	*E. coli*
2-Decenoyl N-Acetyl Cysteamine	experimental	*S. flexneri*
EC 4.6.1.12 - 2-C-metil-D-eritritol 2,4-ciclodifosfato sintase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
4-Diphosphocytidyl-2-C-Methyl-D-Erythritol	experimental	*S. flexneri*
Cytidine-5'-Diphosphate	experimental	*S. flexneri*
2c-Methyl-D-Erythritol 2,4-Cyclodiphosphate	experimental	*S. flexneri*
4-Diphosphocytidyl-2-C-Methyl-D-Erythritol 2-Phosphate	experimental	*S. flexneri*
Cytidine-5'-Monophosphate	experimental	*S. flexneri*
Geranyl Diphosphate	experimental	*S. flexneri*
FARNESYL DIPHOSPHATE	experimental	*Shewanella oneidensis*
Geranyl Diphosphate	experimental	*E. coli*

4.5 Alvos exclusivos de um determinado modelo

Nesta seção apresentamos os resultados encontrados apenas em uma das cepas testadas. Apenas a cepa PA14 não obteve nenhum resultado particular a ela.

4.5.1 Alvos exclusivos do modelo da PAO1 2008

Apenas na cepa PAO1 versão 2008, que foi a 1º versão usada neste trabalho, obtivemos 9 potenciais alvos terapêuticos no total, priorizados pelo intervalo do FVA e apresentados na Tabela 16.

Tabela 16 - Lista de potenciais alvos terapêuticos encontrados para o modelo PAO1 2008.

EC	Proteína	Gene	E-Value	Via
1.1.1.85	3-isopropilmalato desidrogenase	leuB PA3118	1,1E-140	Biossíntese de aminoácidos; Biossíntese de L-leucina; L-leucina a partir de 3-metil-2-oxobutanoato: passo 3/4.
2.2.1.6	Acetolactato sintase, catabólico	PA4180	1,8E-95	N/A
2.4.2.19	Nicotinato-nucleótido pirofosforilase [carboxilante]	nadC PA4524	5,9E-96	Biossíntese de cofactores; Bensíntese de NAD (+); D-ribonucleótido de nicotinato a partir de quinolinato: passo 1/1.
4.2.1.20	Cadeia alfa/beta de triptofano sintase	trpA PA0035 / trpB PA0036	4,5E-53	Biossíntese de aminoácidos; Biossíntese de L-triptofano; L-triptofano do corisato: passo 5/5.
5.1.3.13	dTDP-4-dehydrorhamnose 3,5-epimerase	rmlC PA5164	9,7E-36	Biossíntese de carboidratos; biossíntese dTDP-L-Rhamnose. Biogênese da membrana externa bacteriana; biossíntese de lipopolissacarídeos.
5.1.3.20	ADP-L-glicero-D-manno-heptose-6-epimerase	hldD rfaD PA3337	9,1E-103	Biossíntese de açúcares nucleotídicos; Biossíntese de ADP-L-glicero-beta-D-manno-heptose; ADP-L-glicero-beta-D-manno-heptose a partir de D-glicero-beta-D-manno-heptose 7-fosfato: passo 4/4. Biogênese da membrana externa bacteriana; Biossíntese do núcleo LPS.
5.3.1.24	N- (5'-fosforibosil) antranilato isomerase	trpF PA3113	2,6E-38	Biossíntese de aminoácidos; Biossíntese de L-triptofano; L-triptofano do corismato: passo 3/5.
5.3.1.28	Fosfopentose isomerase	gmhA PA4425	1,2E-132	Biossíntese de carboidratos; Biossíntese de 7-fosfato de D-glicero-D-manno-heptose; D-glicero-alfa-D-manno-heptose 7-fosfato e D-glicero-beta-D-manno-heptose 7-fosfato a partir de 7-fosfato de sedoheptulose: passo 1/1. Biogênese da membrana externa bacteriana; Biossíntese do núcleo LPS.
5.4.99.5	P-proteína / Secrease chorismate mutase	pheA PA3166 /aroQ PA5184	5,6E-46	Biossíntese de aminoácidos; Biossíntese de L-fenilalanina; fenilpiruvato do pré-fenado: passo 1/1. Biossíntese intermediária metabólica; Prevenção da biossíntese; Prefenate from chorismate: passo 1/1.

A imensa maioria das informações aqui citadas foi encontrada mapeada em *E. coli* e *hits* de *e-value* menores que 10^{-3} com *P. aeruginosa*.

A tabela 17 apresenta os fármacos encontrados para os alvos exclusivos da PAO1 2008 citados na Tabela 16.

Tabela 17 - Fármacos mapeados para cada alvo inferido a partir do modelo PAO1 2008.

EC 1.1.1.85 - 3-isopropilmalato desidrogenase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
3-Isopropylmalic Acid	experimental	*T. ferrooxidans*
EC 2.2.1.6 - Acetolactato sintase, catabólico		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Cocarboxylase	aprovado, experimental	*K. pneumoniae*
2-{(9as)-9a-[(1s)-1-Hydroxyethyl]-2,7-Dimethyl-9a,10-Dihydro-5h-Pyrimido[4,5-D][1,3]Thiazolo[3,2-a]Pyrimidin-8-Yl}Ethyl Trihydrogen Diphosphate	experimental	*K. pneumoniae*
Triethylene glycol	experimental	*K. pneumoniae*
EC 2.4.2.19 - Nicotinato-nucleótido pirofosforilase [carboxilante]		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Quinolinic Acid	experimental	*S. typhimurium*
EC 4.2.1.20 – Cadeia alfa/beta de triptofano sintase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Indole-3-Propanol Phosphate	experimental	*T. thermophilus*
Citric Acid	aprovado, nutraceutical, vet_aprovado	*T. thermophilus*
Indole-3-Glycerol Phosphate	experimental	*S. typhimurium*
4-(2-HYDROXYPHENYLSULFINYL)-BUTYLPHOSPHONIC ACID	experimental	*S. typhimurium*
N-[1H-INDOL-3-YL-ACETYL]ASPARTIC ACID	experimental	*S. typhimurium*

N-[1H-INDOL-3-YL-ACETYL]GLYCINE ACID	experimental	*S. typhimurium*
2-{[4-(TRIFLUOROMETHOXY) BENZOYL]AMINO}ETHYL DIHYDROGEN PHOSPHATE	experimental	*S. typhimurium*
2-({[4-(TRIFLUOROMETHOXY) PHENYL]SULFONYL}AMINO)ETHYL DIHYDROGEN PHOSPHATE	experimental	*S. typhimurium*
N-[1H-INDOL-3-YL-ACETYL]VALINE ACID	experimental	*S. typhimurium*
2-[(2-NAPHTHYLSULFONYL)AMINO]ETHYL DIHYDROGEN PHOSPHATE	experimental	*S. typhimurium*
4-(2-HYDROXY-4-FLUOROPHENYLTHIO)-BUTYLPHOSPHONIC ACID	experimental	*S. typhimurium*
5-FLUOROINDOLE PROPANOL PHOSPHATE	experimental	*S. typhimurium*
4-(2-HYDROXYPHENYLTHIO)-1-BUTENYLPHOSPHONIC ACID	experimental	*S. typhimurium*
Indole-3-Propanol Phosphate	experimental	*S. typhimurium*
EC 5.1.3.13 - dTDP-4-dehydrorhamnose 3,5-epimerase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
S,S-(2-Hydroxyethyl)Thiocysteine	experimental	*M. tuberculosis*
D-tartaric acid	experimental	*P. aeruginosa*
3'-O-Acetylthymidine-5'-Diphosphate	experimental	*S. typhimurium*
EC 5.1.3.20 - ADP-L-glicero-D-manno-heptose-6-epimerase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
2'-Monophosphoadenosine 5'-Diphosphoribose	experimental	*E. coli*

Adenosine-5'-Monophosphate Glucopyranosyl-Monophosphate Ester	experimental	*E. coli*
EC 5.3.1.24 - N-(5'-fosforibosil) antranilato isomerase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
1-(O-Carboxy-Phenylamino)-1-Deoxy-D-Ribulose-5-Phosphate	experimental	*T. maritima*
EC 5.3.1.28 - Fosfopentose isomerase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
D-Glycero-D-Mannopyranose-7-Phosphate	experimental	*P. aeruginosa*
EC 5.4.99.5 - P-proteína / Secrease chorismate mutase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
(1R,3S,5S,8R)-8-Hydroxy-2-oxabicyclo[3.3.1]non-6-ene-3,5-dicarboxylic acid	experimental	*E. coli*
(1R,3S,5S,8R)-8-Hydroxy-2-oxabicyclo[3.3.1]non-6-ene-3,5-dicarboxylic acid	experimental	*M. tuberculosis*

4.5.2 Alvos exclusivos do modelo da PAO1 2017

Para a cepa PAO1 versão 2017, que foi a 2º versão usada neste trabalho, principalmente para fins de comparação com a PAO1 versão 2008, obtivemos apenas 3 potenciais alvos terapêuticos exclusivos, priorizados pelo intervalo do FVA e apresentados na tabela abaixo.

Tabela 18 - Lista de potenciais alvos terapêuticos encontrados para o modelo PAO1 2017.

EC	Proteína	Gene	E-Value	Via
1.1.1.267	1-desoxi-D-xilulose 5-fosfato reductoisomerase	dxr PA3650	4,5E-147	Biossíntese isoprenóidea; biossíntese de isopentenil difosfato via via DXP; difosfato de isopentenilo a partir de 1-desoxi-D-xilulose 5-fosfato: passo 1/6.
2.7.4.3	Adenilato quinase	adk PA3686	6,3E-90	Metabolismo de purina; Biossíntese de AMP via caminho de salvamento; AMP da ADP: passo 1/1.
4.1.3.40	Chorismate--pyruvate lyase	ubiC PA5357	1E-12	Biossíntese de cofactores; biossíntese de ubiquinona.

Abaixo apresentamos uma tabela referente aos fármacos mapeados para os alvos apresentados na Tabela 18.

Tabela 19 - Fármacos mapeados para cada alvo inferido a partir do modelo PAO1 2017.

EC 1.1.1.267 - 1-desoxi-D-xilulose 5-fosfato reductoisomerase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Fosmidomycin	experimental, investigational	*E. coli*
[{(5-Chloro-2-Pyridinyl)Amino} Methylene]-1,1-Bisphosphonate	experimental	*E. coli*
1-Deoxy-D-xylulose 5-phosphate	experimental	*E. coli*
Citric Acid	aprovado, nutraceutical, vet_aprovado	*E. coli*
EC 2.7.4.3 - Adenilato quinase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Bis(Adenosine)-5'-Pentaphosphate	experimental	*E. coli*
EC 4.1.3.40 - Chorismate--pyruvate lyase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Vanillic acid	experimental	*E. coli*
P-Hydroxybenzoic Acid	experimental	*E. coli*

4.5.3 Alvos exclusivos do modelo da CCBH4851

Para a cepa CCBH4851, cuja cepa não possui nenhuma análise publicada nem mesmo a própria cepa, pois ainda está em andamento à curadoria, por parte da Fundação Oswaldo Cruz, obtivemos 10 potenciais alvos terapêuticos no total encontrados apenas nessa cepa, priorizados pelo intervalo do FVA e apresentados abaixo.

Tabela 20 - Lista de potenciais alvos terapêuticos encontrados para o modelo CCBH4851.

EC	Proteína	Gene	E-Value	Via
1.2.5.1	Pyruvate dehydrogenase [ubiquinone]	poxB PA5297	1,3E-169	N/A
1.4.1.20	L-phenylalanine dehydrogenase	ldh PA3418	5,5E-55	N/A
1.8.1.9	Thioredoxin reductase	trxB1 PA2616, trxB2 PA0849	4E-160	N/A
2.1.2.2	Phosphoribosylglycinamide formyltransferase	purN PA0944	1,5E-81	Metabolismo de purina; Biossíntese de IMP via via de novo; N (2) -formil-N (1) - (5-fosfo-D-ribosil) glicinamida a partir de N (1) - (5-fosfo-D-ribosil) glicinamida (via 10-formil THF): passo 1/1 .
2.7.6.3	2-amino-4-hydroxy-6-hydroxymethyldihydropteridine pyrophosphokinase	folK PA4728	3,2E-45	Biossíntese de cofactores; biossíntese de tetrahidrofolato; Difosfato de 2-amino-4-hidroxi-6-hidroximetil-7,8-dihidropteridina a partir de 7,8-dihidroneopterina trifosfato: passo 4/4.
2.8.1.6	Biotin synthase	bioB PA0500	9E-170	Biossíntese de cofactores; biossíntese de biotina; biotina a partir de 7,8-diaminononanoato: passo 2/2.
3.5.3.23	N-succinylarginine dihydrolase	astB aruB PA0899	6E-174	Degradação de aminoácidos; Degradação de L-arginina via via AST; L-glutamato e succinato da L-arginina: passo 2/5.
3.6.1.23	Deoxyuridine 5'-triphosphate nucleotidohydrolase	dut PA5321	7,6E-71	Metabolismo de pirimidina; biossíntese de DUMP; DUMP de dCTP (rota DUTP): etapa 2/2.
3.6.1.55	8-oxo-dGTP diphosphatase	PA4400	5,8E-22	N/A
6.3.1.5	NH(3)-dependent NAD(+) synthetase	nadE PA4920	3,6E-71	Biossíntese de cofactores; Bensíntese de NAD (+); NAD (+) de deamido-NAD (+) (via amoníaca): passo 1/1.

Abaixo seguimos apresentando os fármacos encontrados para os alvos terapêuticos mapeados na Tabela 20.

Tabela 21 - Fármacos mapeados para cada alvo inferido a partir do modelo CCBH4851.

EC 1.2.5.1 - Pyruvate dehydrogenase [ubiquinone]		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Nitrofural	aprovado, investigational, vet_aprovado	*E. coli*
EC 1.4.1.20 - L-phenylalanine dehydrogenase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Phenylpyruvic acid	experimental	*Rhodococcus sp.*
Alpha-Hydroxy-Beta-Phenyl-Propionic Acid	experimental	*Rhodococcus sp.*
EC 1.8.1.9 - Thioredoxin reductase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Flavin adenine dinucleotide	aprovado	*E. coli*
EC 2.1.2.2 - Phosphoribosylglycinamide formyltransferase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Glycinamide Ribonucleotide	experimental	*E. coli*
(10R)-10-Formyl-5,8,10-Trideazafolic Acid	experimental	*E. coli*
(10S)-10-Formyl-5,8,10-Trideazafolic Acid	experimental	*E. coli*
EC 2.7.6.3 - 2-amino-4-hydroxy-6-hydroxymethyldihydropteridine pyrophosphokinase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
7,8-Dihydro-7,7-Dimethyl-6-Hydroxypterin	experimental	*H. influenzae*
[Pterin-6-Yl Methanyl]-Phosphonophosphate	experimental	*E. coli*
6-Hydroxymethyl-7,8-Dihydropterin	experimental	*E. coli*
Alpha,Beta-Methyleneadenosine-5'-Triphosphate	experimental	*E. coli*
6-Hydroxymethylpterin	experimental	*E. coli*

7,8-dihydro-6-hydroxymethyl-7-methyl-7-[2-phenylethyl]-pterin	experimental	*E. coli*
EC 2.8.1.6 - Biotin synthase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
D-Dethiobiotin	experimental	*E. coli*
Tromethamine	aprovado	*E. coli*
EC 3.5.3.23 - N-succinylarginine dihydrolase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
N~2~-Succinylarginine	experimental	*E. coli*
EC 3.6.1.23 - Deoxyuridine 5'-triphosphate nucleotidohydrolase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Deoxyuridine-5'-Triphosphate	experimental	*E. coli*
2'-deoxyuridylic acid	experimental	*E. coli*
Deoxyuridine-5'-Diphosphate	experimental	*E. coli*
2'-Deoxyuridine 5'-Alpha,Beta-Imido-Triphosphate	experimental	*E. coli*
EC 3.6.1.55 - 8-oxo-dGTP diphosphatase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
8-Oxo-2'-Deoxy-Guanosine-5'-Monophosphate	experimental	*E. coli*
EC 6.3.1.5 - NH(3)-dependent NAD(+) synthetase		
Nome do fármaco	**Grupo**	**Organismo alvo conhecido**
Gamma-Arsono-Beta, Gamma-Methyleneadenosine-5'-Diphosphate	experimental	*E. coli*
Deamido-Nad+	experimental	*E. coli*
Pyrophosphoric acid	aprovado, experimental	*E. coli*
Gentamicin	aprovado, vet_aprovado	*Bacillus subtilis*
Deamido-Nad+	experimental	*Bacillus subtilis*
Alpha,Beta-Methyleneadenosine-5'-Triphosphate	experimental	*Bacillus subtilis*

5 DISCUSSÃO

Visando tornar esse tópico mais claro, a discussão foi dividida em 2 partes, onde discutimos separadamente os resultados computacionais e biológicos apresentados.

5.1 Discussão computacional

A aplicação descrita neste trabalho é genérica ao ponto de estarmos aptos a, facilmente, utilizá-la em novas versões da rede metabólica, neste caso, em *P. aeruginosa*. Caso tenhamos necessidade de alterar os arquivos de mapeamento da rede metabólica, acrescentando e/ou retirando alguma informação mapeada, teremos capacidade de fazer uma nova análise e ter novos resultados, conforme descrito acima nos resultados obtidos, onde foram feitas análises de versões distintas da rede metabólica da *P. aeruginosa* PAO1, na PA14 e na CCBH4851 e encontrados resultados comuns as 4 versões, comuns a pelo menos 2 ou 3 versões e resultados exclusivos de cada versão.

O Cobrapy foi escolhido por ser uma versão gratuita do framework COBRA, que é popularmente conhecido pelo nome de COBRA Toolbox, do MATLAB e é o mais utilizado em análise de redes metabólicas em escala genômica. O Cobrapy está escrito totalmente em Python.

Várias vantagens do método proposto podem ser destacadas: a robustez do sistema, que pode identificar potenciais alvos mesmo para redes incompletas e/ou imprecisas, ressaltando que tais redes são muito mais comuns do que se espera, o que não inviabiliza a execução deste sistema, conforme demonstrado. O sistema é implantado como uma aplicação web e é assíncrono: o usuário é notificado quando os resultados estão disponíveis. O desempenho do sistema é otimizado, e é capaz de processar a rede metabólica de bactérias diferentes de *P. aeruginosa*. O único requisito é a disponibilidade de um arquivo SBML descrevendo a rede metabólica em escala genômica correspondente.

Além da dinâmica da aplicação, temos também outra vantagem da ferramenta, que é o tempo de resposta, que se encontra entre 2 e 3 horas para uma bactéria, conforme apresentado na tabela 3, descrita na seção de resultados. Isso pode ser considerado interessante, dado o fato de termos uma resposta no mesmo

dia, para as possíveis perguntas biológicas que venhamos a fazer, através do uso da aplicação, o que permite maior agilidade e flexibilidade nos ajustes dos modelos e na descoberta de resultados. O tempo foi determinado com execuções em períodos diferentes onde foram feitas as mesmas em horários distintos, onde apresentou uma alteração de tempo e aumento da média total de execução da aplicação, devido a uma possível atualização e/ou realização de *backups* por parte dos servidores usados nessa ferramenta. Além disso, devemos considerar o fato da rede CCBH4851 ter uma demora maior do que as outras, já que a mesma não teve nenhum gene mapeado, o que faz com que a aplicação siga pelo passo 6b, apresentado na seção de métodos, acarretando em um maior número de requisições no KEGG para buscar o número EC exato correspondente ao composto químico mapeado no modelo dessa cepa, onde a média total de execução da ferramenta aumentou consideravelmente.

Aqui citamos detalhadamente apenas os resultados de modelos de cepa de *P. aeruginosa*, mas podemos afirmar que a aplicação é genérica suficiente para estudar bactérias, onde podemos, por exemplo, buscar uma modelagem de rede metabólica em escala genômica de uma cepa de *E. coli*, testar na aplicação e teremos os resultados propostos nesta ferramenta, na mesma estrutura dos resultados apresentados. Nesse caso, para testar e comprovar a generalidade desta ferramenta foi utilizado também os modelos de *Klebsiella pneumoniae* (50) e de *Haemophilus influenzae* (51), onde obtivemos o retorno com sucesso de todos esses testes, conforme descrito na Tabela 2, na seção de Resultados.

Esta aplicação também apresenta robustez, pois tivemos problemas com os mapeamentos feitos para a PAO1 2017 e PA14, cujos resultados de tempo de duplicação eram valores considerados equivocados, dado o tempo de crescimento microbiano (aprox. 3 minutos provados por meio do uso da fórmula de cálculo da taxa de crescimento) e, mesmo com esta limitação, a aplicação é capaz de analisar a rede como um todo e apresentar resultados, mostrando os potenciais alvos terapêuticos.

Além da robustez, da generalidade e do dinamismo já apresentados, podemos citar também concorrência, que apresenta outra característica positiva desta ferramenta criada para este trabalho. Além do usuário não ter a necessidade de ficar esperando terminar todo o processo para obter seu resultado, ele pode fazer novas execuções e esperar todos os resultados em seu e-mail, que tem a necessidade de ser previamente informado no formulário de entrada da aplicação.

Essa ferramenta faz chamadas a serviços web e bancos de dados e a performance da aplicação aqui medida fica totalmente em função da velocidade da internet ao qual a aplicação estará hospedada, podendo ter variações na média, devido a instabilidade do servidor onde a aplicação está hospedada e/ou instabilidade dos bancos acessados pela ferramenta. Essa é uma vantagem da aplicação, pois ela oferece suporte para múltiplas análises concorrentes, ou seja, enquanto é feita uma verificação de um organismo, a aplicação está disponível para realizar outra análise em paralelo.

Todos os resultados já vêm organizados em formato de planilhas. Elas são armazenadas em arquivos compactados e enviados para o e-mail do usuário, além do fato da aplicação estar em uma versão web, totalmente intuitiva, oferecendo ao usuário uma possibilidade de fazer seus estudos e focar apenas nas respostas das suas perguntas biológicas. A aplicação faz a verificação do modelo informado e valida o mesmo, mantendo o usuário informado do status atual do arquivo que ele está usando. Ao terminar essa validação e a rede estar correta, é iniciado o serviço por trás da ferramenta e o processo vai até o final.

O tempo total descrito de execução desta ferramenta é contado a partir da ativação do serviço assíncrono até a geração das planilhas e envio do e-mail.

Foram encontradas outras propostas de análise de redes metabólicas em escala genômica. Uma proposta publicada na literatura é a descrita em (52), que descreve a criação do modelo SBML, propõe uso do FBA e do FVA para priorização de genes candidatos, porém não propõe nenhum fármaco para os alvos encontrados, nem tampouco descrevem os alvos nem os modelos trabalhados. O procedimento descrito em (53) faz todo o processo até o mapeamento dos números ECs e utiliza ferramentas gráficas para apenas identificar os potenciais alvos, sem descrever nenhum fármaco. Nesse caso, os autores trabalharam com *E. coli* e *B. subtilis*.

Todos os procedimentos descritos em (52) e (53) descrevem também a criação dos modelos das bactérias, processo este que não foi contemplado neste trabalho.

5.2 Discussão biológica

Após argumentarmos a parte tecnológica da ferramenta, vamos falar da parte biológica do trabalho, onde obtivemos 12 alvos terapêuticos em comum a todas as 4

cepas aqui testadas e, destacando o tema desse trabalho (a cepa CCBH4851), 10 alvos foram encontrados apenas na cepa CCBH4851.

Os mapeamentos feitos da PAO1 2017 e PA14 apresentam o tempo de crescimento microbiano fora da realidade (aprox. 3 minutos). Apesar de termos os resultados de alvos bem próximos das redes PAO1 2008 e CCBH4851, podemos ter equívocos na rede criada por (8), devido a terem focado no conhecimento dos fatores de virulência. Um argumento que nos permite dizer que, mesmo com a taxa de geração de biomassa estando equivocada, a aplicação deu um resultado plausível e foram encontrados 12 alvos em comum a todos os modelos testados.

Corroborando a validade dos resultados obtidos, podemos citar também o fato dos modelos PAO1 2017 e PA14 terem dado resultados bem próximos. Acreditamos que isso se deve ao fato deles terem sido criados pela mesma equipe, enquanto a CCBH4851 utiliza a PAO1 2008 como referência em sua concepção.

Queremos destacar alguns dos genes identificados como alvos comuns às quatro modelagens. O gene murA (PA4450 - EC 2.5.1.7) e murB (PA2977 - EC 1.3.1.98) codificam enzimas envolvidas na síntese da parede celular bacteriana, tendo sido identificadas como essenciais tanto em *Pseudomonas sp.* quanto em *E. coli* (54). O gene folP (PA4750 - EC 2.5.1.15) foi clonado e sequenciado em *E. coli*, conforme dados da literatura dos anos 90 e tem grande atuação na biossíntese de ácido fólico, que é fundamental para a reprodução da bactéria (55). Já o gene folA (PA0350 – EC 1.5.1.3) encontra-se na biossintese de cofatores, sendo um importante intermediário do metabolismo de folato, sendo considerada a enzima chave deste processo, além de ser considerada essencial para o crescimento microbiano (56). Outro alvo também que vale destacar aqui, dentro dos 12 em comuns a todos, é o gene aroE (PA0025 – EC 1.1.1.25) que foi descrito como potencial alvo terapêutico tanto de *P. putida* quanto de *E. coli* (57). Ressaltamos que os genes descritos são fundamentais para a geração de biomassa da bactéria e são comuns às quatro modelagens analisadas.

Temos outros exemplos de alvos detectados nas cepas de *P. aeruginosa* testadas, como por exemplo o gene rmlC (PA5164 – EC 5.1.3.13), encontrado apenas na PAO1 versão 2008, que é considerado gene essencial na *M. tuberculosis*, envolvido na biossíntese de carboidratos (58). O gene dxr (PA3650 – EC 1.1.1.267), detectado apenas na PAO1 versão 2017, também é considerado um potencial alvo terapêutico em bactérias e até mesmo no parasito da malária e foi constatado que é uma enzima chave para início da biossíntese de isoprenóides e

considerada um bom alvo, pois ela não possui nenhuma equivalência em humanos (59), logo obtendo informações de inibidores para essa enzima, já sabemos que não terá nenhum efeito adverso no humano.

Vamos destacar também alguns alvos terapêuticos identificados apenas para a *P. aeruginosa* CCBH4851, como por exemplo o gene poxB (PA5297 – EC 1.2.5.1), que foi identificado como um gene essencial e responsável pela contribuição a resistência da bactéria a um amplo espectro de antibióticos beta-lactâmicos (60). O gene purN (PA0944 – EC 2.1.2.2) é conhecido como potencial alvo terapêutico e foi identificado como uma enzima da via biossintética de purinas, que representa um novo alvo terapêutico em bactérias e foi identificada em *M. tuberculosis* como essencial para o crescimento e sobrevivência desta bactéria (61). Podemos também destacar o gene nadE (PA4920 – EC 6.3.1.5), que é um dos principais genes envolvidos na via de biossíntese de NAD+ e, por ele ser essencial nessa via, torna-se um potencial alvo terapêutico, que já foi testado em *E. coli* e em *M. tuberculosis* (62).

Ainda se tratando apenas de alvos detectados na cepa CCBH4851, um fármaco aprovado que foi encontrado para o gene poxB (PA5297 – EC 1.2.5.1) foi o Nitrofural, que já é usado para combate a *E. coli* e está com status de aprovado como inibidor na base Drugbank (63,64). Logo pode ser uma proposta para se utilizar também com *P.* aeruginosa, pois o gene mapeado em *E. coli* possui similaridade de quase 50%. Outro fármaco que podemos destacar dos encontrados apenas para a CCBH4851 é o Tromethamine, que também se encontra com status de aprovado na base Drugbank para combate a *E. coli* e o gene alvo desse fármaco, o gene bioB (PA0500 – EC 2.8.1.6). Temos também o fármaco Gentamicin com status de aprovado para combate a bactéria *Bacillus subtilis* (65) e, conforme encontrado com detalhes na descrição completa do fármaco no Drugbank, é indicado para tratamento de infecções graves causadas por *P. aeruginosa*, *E. coli*, entre outras citadas no Drugbank. Apesar de termos algumas cepas de *P. aeruginosa* que são resistentes a este fármaco, é um antibiótico utilizado há muito tempo para tratamento de *P. aeruginosa*.

As propostas aqui descritas têm como objetivo apresentar hipóteses para fins de testes em bancada. De acordo com este caminho, podemos realizar uma varredura de alvos potenciais utilizando as informações levantadas por ferramentas de tecnologia da informação para minimizar custos associados à experimentação na bancada. O critério de geração de biomassa não é uma garantia da inibição da

bactéria com as propostas aqui apresentadas, já que não temos a garantia de que os modelos usados estão completos e refletem a realidade, bem como algumas bactérias ativam outras vias alternativas para fins de combate aos antibióticos aplicados nelas, além da possibilidade delas terem mecanismos que não permitam que o fármaco atinja seu alvo.

Uma possível ação que pode ser aplicada utilizando a ferramenta proposta neste trabalho é combinar os possíveis resultados aqui apresentados, visando uma maior abrangência de combate a bactéria em estudo, fazendo multiplos nocautes de genes, podendo assim, ter outros resultados e/ou hipóteses a aplicar, visando combate-las com maior afinco as bactérias em estudos e propor novas formas de inibição destas. Enfatizamos que esta ação é uma evolução viável desta ferramenta.

6 CONCLUSÕES

Este trabalho realiza um processo de descoberta combinando bioinformática e biologia de sistemas, integrando dados matemáticos, estatísticos e de análise biológica. É uma poderosa ferramenta criada para encontrar potenciais alvos terapêuticos através da análise de redes metabólicas. Este método irá fornecer informações sobre novas opções de inibição de bactérias resistentes a antibióticos, podendo ser aplicado como uma proposta de combate a infecção hospitalar de uma maneira diferente e dinâmica, aumentando as opções para combater a bactéria *P. aeruginosa*.

Podemos citar como um dos principais resultados deste trabalho a criação de uma ferramenta totalmente web, que propicia a identificação de alvos em comum às quatro cepas testadas, sendo os alvos já mapeados e descritos na literatura para outros organismos, além de ter fármacos já aprovados na lista dos candidatos e recomendados para uso também em *P. aeruginosa*.

Acreditamos que este projeto poderá ser usado como modelo para estudo de outras bactérias, devido à flexibilidade proposta pela computação, e podemos torná-lo uma base para que possam ser iniciadas outras análises relevantes para a área.

Acreditamos também que esta proposta servirá como ponto de partida para a criação de aplicações ainda mais completas, em ambiente web, como a leitura de modelos integrados, busca em outras bases de dados existentes e resultados com maior número de informações e/ou integrações, mais próximas ainda da realidade. Além disso, essa ferramenta traz uma nova opção no meio acadêmico para busca de inibidores de bactérias, propondo novas hipóteses mais precisas e objetivas, passiveis de testes em bancada, com diminuição dos custos associados.

Ressaltamos que a automação do processo aqui descrito constitue uma importante ferramenta para o estudo de outros organismos, onde este trabalho poderá ser usado como base para uma aplicação web, totalmente intuitiva, que pode ser um marco para aprimorar ainda mais esse tipo de análise e abrir possibilidades dentro do campo da biologia.

7 REFERÊNCIAS BIBLIOGRÁFICAS

1. Anvisa. Segurança do Paciente e Qualidade em Serviços de Saúde. Bol Inf. 2014;1:1–12.

2. OMS. OMS. In: El primer informe mundial de la OMS sobre la resistencia a los antibióticos pone de manifiesto una grave amenaza para la salud pública en todo el mundo. 2014. p. 5.

3. OMS. La OMS publica la lista de las bacterias para las que se necesitan urgentemente nuevos antibióticos [Internet]. 27 de febrero. 2017 [cited 2018 Mar 1]. p. 1. Available from: http://www.who.int/mediacentre/news/releases/2017/bacteria-antibiotics-needed/es/

4. Vallet-Gely I, Boccard F. Chromosomal organization and segregation in Pseudomonas aeruginosa. PLoS Genet. 2013;9(5).

5. Kerr KG, Snelling AM. Pseudomonas aeruginosa: a formidable and ever-present adversary. Vol. 73, Journal of Hospital Infection. 2009. p. 338–44.

6. Trabulsi L, Alterthum F. Microbiologia. Atheneu Editora, editor. ISBN: 9788573799811; 2008. 780 p.

7. Oberhardt MA, Puchałka J, Fryer KE, Martins Dos Santos VAP, Papin JA. Genome-scale metabolic network analysis of the opportunistic pathogen Pseudomonas aeruginosa PAO1. J Bacteriol. 2008;190(8):2790–803.

8. Bartell JA, Blazier AS, Yen P, Thøgersen JC, Jelsbak L, Goldberg JB, et al. Reconstruction of the metabolic network of Pseudomonas aeruginosa to interrogate virulence factor synthesis. Nat Commun. 2017;8:13.

9. Neves PR, Mamizuka EM, Levy CE, Lincopan N. Pseudomonas aeruginosa multirresistente: um problema endêmico no Brasil. J Bras Patol e Med Lab. 2011;47(4):409–20.

10. de Freitas ALP, Barth AL. Antibiotic resistance and molecular typing of Pseudomonas aeruginosa: focus on imipenem. Brazilian J Infect Dis [Internet]. 2002;6(1):1–7. Available from: http://www.ncbi.nlm.nih.gov/pubmed/11980597

11. Leão RS, Carvalho-Assef APD, Ferreira AG, Folescu TW, Barth AL, Pitt TL, et al. Comparison of the worldwide transmissible Pseudomonas aeruginosa with isolates from Brazilian cystic fibrosis patients. Brazilian J Microbiol. 2010;41(4):1079–81.

12. Silveira M, Albano R, Asensi M, Assef APC. The draft genome sequence of multidrug-resistant Pseudomonas aeruginosa strain CCBH4851, a nosocomial isolate belonging to clone SP (ST277) that is prevalent in Brazil. Mem Inst Oswaldo Cruz. 2014;109(8):1086–7.

13. Francke C, Siezen RJ, Teusink B. Reconstructing the metabolic network of a bacterium from its genome. Vol. 13, Trends in Microbiology. 2005. p. 550–8.

14. Thiele I, Palsson B. A protocol for generating a high-quality genome-scale metabolic reconstruction. Nat Protoc. 2010;5(1):93–121.

15. Fleischmann R, Adams M, White O, Clayton R, Kirkness E, Kerlavage A, et al. Whole-genome random sequencing and assembly of Haemophilus influenzae Rd. Science (80-) [Internet]. 1995;269(5223):496–512. Available from: http://www.sciencemag.org/cgi/doi/10.1126/science.7542800

16. Orth JD, Thiele I, Palsson BØ. What is flux balance analysis? Nat Biotechnol [Internet]. 2010;28(3):245–8. Available from: http://www.nature.com/doifinder/10.1038/nbt.1614

17. Tamura T, Lu W, Akutsu T. Computational methods for modification of metabolic networks. Vol. 13, Computational and Structural Biotechnology Journal. 2015. p. 376–81.

18. Chapman SP, Paget CM, Johnson GN, Schwartz J-M. Flux balance analysis reveals acetate metabolism modulates cyclic electron flow and alternative glycolytic pathways in Chlamydomonas reinhardtii. Front Plant Sci [Internet]. 2015;6. Available from: http://journal.frontiersin.org/Article/10.3389/fpls.2015.00474/abstract

19. Gudmundsson S, Thiele I. Computationally efficient flux variability analysis. BMC Bioinformatics. 2010;11.

20. Covert M. Fundamentals of Systems Biology: From Synthetic Circuits to Whole-cell Models. Press C, editor. ISBN: 9781449369330; 2015. 367 p.

21. Raman K, Chandra N. Flux balance analysis of biological systems: Applications and challenges. Vol. 10, Briefings in Bioinformatics. 2009. p. 435–49.

22. Alves Barbosa da Silva F, Carels N, Paes Silva Jr. F. Theoretical and Applied Aspects of Systems Biology. Springer International Publishing; 2018. VIII, 203.

23. Zomorrodi AR, Suthers PF, Ranganathan S, Maranas CD. Mathematical optimization applications in metabolic networks. Vol. 14, Metabolic Engineering. 2012. p. 672–86.

24. Nikoloski Z, Perez-Storey R, Sweetlove LJ. Inference and prediction of metabolic network fluxes. Plant Physiol [Internet]. 2015;pp.01082.2015. Available from: http://www.plantphysiol.org/lookup/doi/10.1104/pp.15.01082

25. Palsson BO. Systems biology: Constraint-based reconstruction and analysis.

Systems Biology: Constraint-Based Reconstruction and Analysis. 2015. 1-531 p.

26. Hay JO, Schwender J. Flux variability analysis: Application to developing oilseed rape embryos using toolboxes for constraint-based modeling. Methods Mol Biol. 2014;1090:301–16.

27. Müller AC, Bockmayr A. Fast thermodynamically constrained flux variability analysis. Vol. 29, Bioinformatics. 2013. p. 903–9.

28. San Román M, Cancela H, Acerenza L. Source and regulation of flux variability in Escherichia coli. BMC Syst Biol. 2014;8(1).

29. Rossum G Van, Drake FL. The Python Library Reference. October. 2010;1–1144.

30. Rossum G Van, Drake FL. Python Tutorial. History. 2010;42(4):1–122.

31. Hyduke D, Hyduke D, Schellenberger J, Que R, Fleming R, Thiele I, et al. COBRA Toolbox 2.0. Protoc Exch [Internet]. 2011; Available from: http://www.nature.com/protocolexchange/protocols/2097

32. Ebrahim A, Lerman JA, Palsson BO, Hyduke DR. COBRApy: COnstraints-Based Reconstruction and Analysis for Python. BMC Syst Biol. 2013;7.

33. Django Software Foundation. Django: The Web framework for perfectionists with deadlines. DjangoprojectCom. 2013;1–3.

34. Holovaty A, Kaplan-Moss J. The Definitive Guide to Django: Web Development Done Right. Development. 2009. 499 p.

35. Cerami E. XML for bioinformatics. XML for Bioinformatics. 2005. 1-304 p.

36. Hucka M, Hucka M, Bergmann F, Hoops S, Keating S, Sahle S, et al. The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core. Nat Preced. 2010;

37. Le Novère N, Hucka M, Hoops S, Keating S, Sahle S, Wilkinson D. Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions. Nat Preced. 2008;1–38.

38. Kanehisa M, Sato Y, Kawashima M, Furumichi M, Tanabe M. KEGG as a reference resource for gene and protein annotation. Nucleic Acids Res. 2016;44(D1):D457–62.

39. Ogata H, Goto S, Sato K, Fujibuchi W, Bono H, Kanehisa M. KEGG: Kyoto encyclopedia of genes and genomes. Vol. 27, Nucleic Acids Research. 1999. p. 29–34.

40. Wishart DS, Knox C, Guo AC, Cheng D, Shrivastava S, Tzur D, et al. DrugBank: A knowledgebase for drugs, drug actions and drug targets. Nucleic Acids Res. 2008;36(SUPPL. 1).

41. DrugBank. DrugBank. DrugBank Version 4.3. 2013.

42. The UniProt Consortium. UniProt: a hub for protein information. Nucleic Acids Res [Internet]. 2015;43(Database issue):D204-12. Available from: http://nar.oxfordjournals.org/cgi/content/long/43/D1/D204

43. Bateman A, Martin MJ, O'Donovan C, Magrane M, Alpi E, Antunes R, et al. UniProt: The universal protein knowledgebase. Nucleic Acids Res. 2017;45(D1):D158–69.

44. Kotlyar M, Fortney K, Jurisica I. Network-based characterization of drug-regulated genes, drug targets, and toxicity. Vol. 57, Methods. 2012. p. 499–507.

45. Alves-Ferreira M, Guimarães ACR, Capriles PV da SZ, Dardenne LE, Degrave WM, Guimaraes ACR, et al. A new approach for potential drug target discovery through in silico metabolic pathway analysis using Trypanosoma cruzi genome information. Mem Inst Oswaldo Cruz [Internet]. 2009;104(8):1100–10. Available from: http://www.ncbi.nlm.nih.gov/pubmed/20140370

46. Joyce AR, Palsson BØ. Predicting Gene Essentiality Using Genome-Scale in Silico Models. In: Methods in molecular biology (Clifton, NJ) [Internet]. 2008. p. 433–57. Available from: http://link.springer.com/10.1007/978-1-59745-321-9_30

47. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990;215(3):403–10.

48. LaBauve AE, Wargo MJ. Growth and laboratory maintenance of Pseudomonas aeruginosa. Curr Protoc Microbiol. 2012;(SUPPL.25).

49. Maier RM. Environmental Microbiology [Internet]. Environmental Microbiology. 2015. 213-243 p. Available from: http://www.sciencedirect.com/science/article/pii/B9780123946263000119

50. Liao YC, Huang TW, Chen FC, Charusanti P, Hong JSJ, Chang HY, et al. An experimentally validated genome-scale metabolic reconstruction of Klebsiella pneumoniae MGH 78578, iYL1228. J Bacteriol. 2011;193(7):1710–7.

51. Schilling CH, Palsson B. Assessment of the metabolic capabilities of Haemophilus influenzae Rd through a genome-scale pathway analysis. J Theor Biol. 2000;203(3):249–83.

52. Chavali AK, D'Auria KM, Hewlett EL, Pearson RD, Papin JA. A metabolic network approach for the identification and prioritization of antimicrobial drug targets. Vol. 20, Trends in Microbiology. 2012. p. 113–23.

53. Ahn YY, Lee DS, Burd H, Blank W, Kapatral V. Metabolic network analysis-based identification of antimicrobial drug targets in Category A bioterrorism agents. PLoS One. 2014;9(1).

54. Benson TE, Walsh CT, Hogle JM. The structure of the substrate-free form of MurB, an essential enzyme for the synthesis of bacterial cell walls. Structure. 1996;4(1):47–54.

55. Dallas WS, Gowen JE, Ray PH, Cox MJ, Dev IK. Cloning, sequencing, and enhanced expression of the dihydropteroate synthase gene of Escherichia coli MC4100. J Bacteriol. 1992;174(18):5961–70.

56. Myllykallio H, Leduc D, Filee J, Liebl U. Life without dihydrofolate reductase FolA. Vol. 11, Trends in Microbiology. 2003. p. 220–3.

57. Peek J, Shi T, Christendat D. Identification of novel polyphenolic inhibitors of shikimate dehydrogenase (AroE). J Biomol Screen. 2014;19(7):1090–8.

58. Kantardjieff KA, Kim CY, Naranjo C, Waldo GS, Lekin T, Segelke BW, et al. Mycobacterium tuberculosis RmlC epimerase (Rv3465): A promising drug-target structure in the rhamnose pathway. Acta Crystallogr Sect D Biol Crystallogr. 2004;60(5):895–902.

59. Singh N, Chevé G, Avery M a, McCurdy CR. Targeting the methyl erythritol phosphate (MEP) pathway for novel antimalarial, antibacterial and herbicidal drug discovery: inhibition of 1-deoxy-D-xylulose-5-phosphate reductoisomerase (DXR) enzyme. Curr Pharm Des. 2007;13(11):1161–77.

60. Kong KF, Jayawardena SR, Del Puerto A, Wiehlmann L, Laabs U, Tümmler B, et al. Characterization of poxB, a chromosomal-encoded Pseudomonas aeruginosa oxacillinase. Gene. 2005;358(1–2):82–92.

61. Zhang Z, Caradoc-Davies TT, Dickson JM, Baker EN, Squire CJ. Structures of Glycinamide Ribonucleotide Transformylase (PurN) from Mycobacterium tuberculosis Reveal a Novel Dimer with Relevance to Drug Discovery. J Mol Biol. 2009;389(4):722–33.

62. Rodionova IA, Schuster BM, Guinn KM, Sorci L, Scott DA, Li X, et al. Metabolic and bactericidal effects of targeted suppression of NadD and NadE enzymes in mycobacteria. MBio. 2014;5(1).

63. Overington JP, Al-Lazikani B, Hopkins AL. How many drug targets are there? Nat Rev Drug Discov. 2006;5(12):993–6.

64. Imming P, Sinning C, Meyer A. Drugs, their targets and the nature and number of drug targets. Nat Rev Drug Discov. 2006;5(10):821–34.

65. Velu SE, Cristofoli WA, Garcia GJ, Brouillette CG, Pierson MC, Luan CH, et al. Tethered dimers as NAD synthetase inhibitors with antibacterial activity. J Med Chem. 2003;46(15):3371–81.

66. Hunt A, Thomas D. The Pragmatic Programmer. October. 1999. 352 p.

8 APÊNDICES E/OU ANEXOS

Esta sessão foi criada com o objetivo de apresentar alguns materiais adicionais, considerados relevantes e complementares as informações apresentadas neste trabalho.

APÊNDICE A - EQUAÇÕES DE BIOMASSA

Nesse item, foram incluídas todas as equações de biomassa dos cenários testados neste trabalho, em sua íntegra, conforme extraídos dos arquivos SBML das cepas usadas.

Equação biomassa - PAO1 versão 2008

0.045754 5mthf_c + 0.007686 PA_core_lipidA_c + 4.5e-05 accoa_c + 0.446559 ala__L_c + 0.000915 amp_c + 0.25714 arg__L_c + 0.20955 asn__L_c + 0.20955 asp__L_c + 41.84827 atp_c + 9.6e-05 cdlp_PA_c + 5e-06 coa_c + 0.1153 ctp_c + 0.079612 cys__L_c + 0.022602 datp_c + 0.023243 dctp_c + 0.023243 dgtp_c + 0.022602 dttp_c + 4.5e-05 fad_c + 0.22877 gln__L_c + 0.22877 glu__L_c + 0.532577 gly_c + 0.140922 glycogen_c + 0.185761 gtp_c + 41.6918 h2o_c + 0.082357 his__L_c + 0.252562 ile__L_c + 0.391654 leu__L_c + 0.298316 lys__L_c + 0.133601 met__L_c + 0.001967 nad_c + 4.5e-05 nadh_c + 0.000118 nadp_c + 0.000366 nadph_c + 0.001462 pe_PA_c + 0.025256 peptido_EC_c + 0.000353 pg_PA_c + 0.161054 phe__L_c + 1e-06 pheme_c + 0.192166 pro__L_c + 3.9e-05 ps_PA_c + 0.032027 ptrc_c + 0.18759 ser__L_c + 0.006405 spmd_c + 2e-06 succoa_c + 0.220534 thr__L_c + 0.049414 trp__L_c + 0.119875 tyr__L_c + 0.002745 udpg_c + 0.124451 utp_c + 0.367862 val__L_c --> 41.6918 adp_c + 41.691389 h_c + 41.693631 pi_c + 0.668191 ppi_c

Equação biomassa - PAO1 versão 2017

0.0001 cJB00125_c + 0.0013602 cPY00124_c + 0.0030243 cPY00129_c + 0.00079222 cPY00132_c + 0.0015844 cPY00135_c + 0.00053681 cPY00138_c + 0.00097843 cPY00140_c + 59.81 cpd00001_c + 59.81 cpd00002_c + 0.10147 cpd00018_c + 0.22711 cpd00023_c + 0.0001 cpd00028_c + 0.31517 cpd00033_c + 0.43505 cpd00035_c + 0.10702 cpd00039_c + 0.19852 cpd00041_c + 0.20202 cpd00046_c + 0.28478 cpd00051_c + 0.15885 cpd00053_c + 0.20628 cpd00054_c + 0.075597 cpd00060_c + 0.055547 cpd00065_c + 0.13262 cpd00066_c + 0.094677 cpd00069_c + 0.037522 cpd00084_c + 0.099817 cpd00091_c + 0.46473 cpd00107_c + 0.56945 cpd00118_c + 0.080962 cpd00119_c + 0.0001 cpd00125_c + 0.19855 cpd00126_c + 0.18967 cpd00129_c + 0.098535 cpd00132_c + 2.5e-05 cpd00155_c + 0.25796 cpd00156_c + 0.15587 cpd00161_c + 0.03128 cpd00206_c +

0.016114 cpd00294_c + 0.03128 cpd00296_c + 0.016114 cpd00298_c + 0.15571 cpd00322_c + 0.0078591 cpd15428_c + 0.0041649 cpd15429_c + 0.00037815 cpd15430_c + 0.011889 cpd15431_c + 0.047824 cpd15531_c + 0.020287 cpd15532_c + 0.0020692 cpd15533_c + 0.068198 cpd15534_c + 0.012557 cpd15538_c + 0.0037982 cpd15539_c + 0.00042166 cpd15540_c + 0.011557 cpd15541_c + 0.053367 cpd15665_c + 0.025523 cpd17066_c --> 59.81 cpd00008_c + 59.81 cpd00009_c + 0.69664 cpd00012_c + 59.81 cpd00067_c

Equação de biomassa - PA14

0.0001 cJB00125_c + 0.0013602 cPY00124_c + 0.0030243 cPY00129_c + 0.00079222 cPY00132_c + 0.0015844 cPY00135_c + 0.00053681 cPY00138_c + 0.00097843 cPY00140_c + 59.81 cpd00001_c + 59.81 cpd00002_c + 0.10146 cpd00018_c + 0.22687 cpd00023_c + 0.0001 cpd00028_c + 0.31256 cpd00033_c + 0.43461 cpd00035_c + 0.10776 cpd00039_c + 0.19864 cpd00041_c + 0.19949 cpd00046_c + 0.28443 cpd00051_c + 0.16033 cpd00053_c + 0.20723 cpd00054_c + 0.076901 cpd00060_c + 0.055748 cpd00065_c + 0.13212 cpd00066_c + 0.094418 cpd00069_c + 0.037574 cpd00084_c + 0.10137 cpd00091_c + 0.46321 cpd00107_c + 0.56945 cpd00118_c + 0.081188 cpd00119_c + 0.0001 cpd00125_c + 0.19942 cpd00126_c + 0.18965 cpd00129_c + 0.098829 cpd00132_c + 2.5e-05 cpd00155_c + 0.25557 cpd00156_c + 0.15734 cpd00161_c + 0.031422 cpd00206_c + 0.015972 cpd00294_c + 0.031422 cpd00296_c + 0.015972 cpd00298_c + 0.15583 cpd00322_c + 0.0078591 cpd15428_c + 0.0041649 cpd15429_c + 0.00037815 cpd15430_c + 0.011889 cpd15431_c + 0.047824 cpd15531_c + 0.020287 cpd15532_c + 0.0020692 cpd15533_c + 0.068198 cpd15534_c + 0.012557 cpd15538_c + 0.0037982 cpd15539_c + 0.00042166 cpd15540_c + 0.011557 cpd15541_c + 0.053367 cpd15665_c + 0.025523 cpd17066_c --> 59.81 cpd00008_c + 59.81 cpd00009_c + 0.69653 cpd00012_c + 59.81 cpd00067_c

Equação biomassa – CCBH4851

1.89832 5mthf_c_ + 1.65071304347826 accoa_c_ + 3.79664 amp_c_ + 37.0423 atp_c_ + 0.491029487842732 cdlp_pa_c_ + 1.80792380952381 coa_c_ + 42.4822647420835 dna_c_ + 1.40616296296296 fad_c_ + 6.32773333333333 glycogen_c_ + 37.0423 h2o_c_ + 1.80792380952381 nad_c_ + 1.80792380952381 nadh_c_ + 1.80792380952381 nadp_c_ + 1.80792380952381 nadph_c_ + 0.243374358974359 pa_core_lipida_c_ + 0.969519918283963 pe_pa_c_ + 0.94916

peptido_ec_c_ + 0.945378486055777 pg_pa_c_ + 1.11665882352941 pheme_c_ + 1.70368276276761 protein_c_ + 0.945378486055777 ps_pa_c_ + 9.4916 ptrc_c_ + 6.80389240336195 rna_c_ + 5.42377142857143 spmd_c_ + 1.518656 succoa_c_ + 3.16386666666667 thmpp_c_ + 2.53109333333333 udpg_c_ --> 37.0423 adp_c_ + biomass_e_ + 37.0423 h_c_ + 37.0423 pi_c_

APÊNDICE B - CÓDIGO FONTE NA ÍNTEGRA, REFERENTE AO PROCESSO CRIADO EM PYTHON

Aqui nesse anexo apresento na íntegra toda a codificação criada para formar a aplicação aqui proposta neste trabalho. Toda ela foi concebida em Python 3, conforme explicado no decorrer do documento e as linhas de código abaixo contemplam isso. Só esse arquivo tem no total 1123 linhas de código, sem contar os outros arquivos python que fazem parte das configurações e layouts do Django.

Arquivo **pipelineFindTargets.py**

```
from __future__ import print_function

import traceback
from datetime import datetime
from bioservices.kegg import KEGG
from bioservices.uniprot import UniProt
from bioservices.ncbiblast import NCBIblast
from bs4 import BeautifulSoup
from zipfile import ZipFile, ZIP_DEFLATED
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders

import xlwt
import requests
import os
import re
import sys
import cobra.manipulation
import pandas as pd
import time
import threading
```

```
import smtplib

# CLASSE CRIADA COM O OBJETIVO DE MANTER A CHAMADA ASSINCRONA DO METODO DA MODELAGEM
class MyThread(threading.Thread):

    name = None
    organism = None
    email = None
    model = None

    def __init__(self, name, organism, email, modelParam):
        threading.Thread.__init__(self)
        self.model = modelParam
        self.organism = organism
        self.name = name
        self.email = email

    def run(self):
        #print("A THREAD COMECOU!!!!!!!!")
        FindTargets().mainMethod(self.model, self.organism, self.name, self.email)


class FindTargets:

    # ADD ALL ORGANISMS MAPPED FROM KEGG TO ATTRIBUTE IN COMBOBOX FORM
    # USED TO FULL OUR COMBOBOX FROM INDEX APP
    def list_organism(self):
        dir_path = os.path.dirname(os.path.realpath(__file__)) # identifica o local real onde esse arquivo esta
        with open(dir_path+"/organismos.txt", "r") as infile:
            data = infile.read()
        my_list = data.splitlines()
```

```
            my_list_aux = []
            my_list_aux.append(("", " --- SELECIONE --- "))

            for itens in my_list:
                itens_splt = itens.split(",")
                tuple_add = (itens_splt[0], itens_splt[1])
                my_list_aux.append(tuple_add)

            return tuple(my_list_aux)

        # METODO PARA MONTAR O OBJETO COBRA PARA TRAFEGO NA
APLICACAO!

        def readModel(self, sbmlfile):
            return cobra.io.read_sbml_model(sbmlfile)

        # MODEL VALIDATION METHOD TO CONTINUE OUR EXECUTION

        def validateModel(self, model):
            return_dict_val_model = {}

            numCasasDec = 6

            initialSolution = model.optimize()
            valInitialSolutionFmt = round(float(initialSolution.f), numCasasDec)

            return_dict_val_model['valInitialSolutionFmt'] = valInitialSolutionFmt

            if initialSolution.status != 'optimal':
                return_dict_val_model['message'] = "ERROR! SELECTED MODEL
HAS AN ERROR. PLEASE VERIFY AND TRY AGAIN!"
                return_dict_val_model['ehParaFazer'] = False
            else:
                if initialSolution.f == 0.0 or initialSolution.f == -0.0:
```

```
                return_dict_val_model['message'] = "ERROR! SELECTED MODEL
DOES NOT GENERATE BIOMASS. FINISHED EXECUTION!"
                return_dict_val_model['ehParaFazer'] = False
            else:
                return_dict_val_model['message'] = "OK! SELECTED MODEL
GENERATE BIOMASS. ANALYSIS STARTED AND WE WILL SEND RESULTS IN
YOUR MAIL WHEN FINISH PROCESS!"
                return_dict_val_model['ehParaFazer'] = True

            return return_dict_val_model


        #
        # MAIN METHOD THAT REALIZE ANALYSIS OF SELECTED NETWORK
SBML FILE
        #

        def mainMethod(self, model, organismParam, name, email):

            numCasasDec = 6
            TIMEOUT_SECONDS = 200000
            model = model

            # BEFORE, WE NEED TO CREATE FOLDERS TO INCLUDE OUR
FILES AND AFTER ZIP THEM
            dataHoraAtual = datetime.now()
            dataHoraAtualFmt = dataHoraAtual.strftime('%Y-%m-
%d_%H.%M.%S.%f')
            dir_path = os.path.dirname(os.path.realpath(__file__))
            directory = dir_path+"/results/"+dataHoraAtualFmt

            # TESTS LINES!
            #directory = dir_path+"/results/"+"testesMeriguetiCCBH"
            #directory = dir_path+"/results/"+"testesMeriguetiPAO1"
            #directory = dir_path+"/results/"+"testesMeriguetiPAO1_2017"
```

```
            dir_blasts = directory+"/blasts"

            if not os.path.exists(directory):
                os.makedirs(directory)

            if not os.path.exists(dir_blasts):
                os.makedirs(dir_blasts)

            try:
                ######################################################
                # ITEM 1 - ANALYSIS OF P. AERUGINOSA MODEL SELECTED
                ######################################################
                self.reportModel(model,
directory+"//dados_modelo_"+str(model)+"_.xls")

                # FBA EXECUTION
                initialSolution = model.optimize()
                valInitialSolutionFmt = round(float(initialSolution.f), numCasasDec)

                ##################################################
                # ITEM 2 - PRIORITIZATION OF EXISTING REACTIONS
                ##################################################
                fva_result = cobra.flux_analysis.flux_variability_analysis(model,
model.reactions[:len(model.reactions)])

pd.DataFrame.from_dict(fva_result).round(numCasasDec).to_csv(directory+"/01-
fva.csv")

                listReactPerturb = []
                reacoesFVADifZero = open(directory+"//"+"02-reacoesFVADifZero.txt",
"w")

                df_fva_result = pd.DataFrame(fva_result)
                itemsFVAResult = df_fva_result.iterrows()
                itemsFBAResult = sorted(initialSolution.x_dict.iteritems())
```

```
                for (keyFVA, valueFVA), (keyFBA, valueFBA) in
list(zip(itemsFVAResult, itemsFBAResult)):
                    minFVA = round(float(valueFVA["minimum"]), numCasasDec)
                    maxFVA = round(float(valueFVA["maximum"]), numCasasDec)
                    valueFBA = round(float(valueFBA), numCasasDec)

                    # Filtrando reacoes que estao ativas no momento
                    if maxFVA != 0 or minFVA != 0:
                        reacoesFVADifZero.write(str(keyFBA) + '\n')
                        #diferencaReacao = maxFVA - minFVA
                        #reacoesFVADifZero.write("Resultado da diferenca entre max e
min = " + str(diferencaReacao))
                        listReactPerturb.append(keyFBA)
                        #reacoesFVADifZero.write('\n\n')

                    if keyFBA != keyFVA:
                        raise Exception("Por favor, verifique, chave FBA/FVA nao batem
na comparacao")

                    else:
                        if valueFBA < minFVA:
                            #print(keyFBA)
                            #print(valueFBA)
                            #print(minFVA)
                            #print(maxFVA)
                            raise Exception("Por favor, verifique o modelo. Valor do FBA
dessa reacao ser menor que o minimo descrito no FVA")

                        if valueFBA > maxFVA:
                            ##print(keyFBA)
                            #print(valueFBA)
                            #print(minFVA)
                            #print(maxFVA)
```

```
                    raise Exception("Por favor, verifique o modelo. Valor do FBA
dessa reacao ser maior que o maximo descrito no FVA")

                reacoesFVADifZero.close()
                #print("Lista de reacoes a analisar = " + str(len(listReactPerturb)))

                #print("GENERATED FILE 01-fva.csv")
                #print("GENERATED FILE %s" % reacoesFVADifZero.name)

                ############################################################
                # ITEM 3 - SIMULATION OF SINGLE KNOCKOUT OF REACTION
                ############################################################
                contador = 0
                lbAux = ubAux = 0

                reacoesZerandoFBA = []

                file_compound_reaction_ko =
open(directory+"//react_biomass_zero_sbml_no_genes.txt", 'w', encoding="ISO-
8859-1")
                for i in listReactPerturb:
                    reacao = model.reactions.get_by_id(i)
                    contador += 1
                    FBABeforeDelete = model.optimize()
                    valFBeforeDelete = round(float(FBABeforeDelete.f), numCasasDec)
                    if valFBeforeDelete != valInitialSolutionFmt:
                        raise Exception("1o IF - Erro! FBA nao bate com a primeira
execucao")

                    lbAux = reacao.lower_bound
                    ubAux = reacao.upper_bound
                    reacao.lower_bound = 0
                    reacao.upper_bound = 0

                    FBAAfterDelete = model.optimize()
```

```
                    valFAfterDelete = round(float(FBAAfterDelete.f), numCasasDec)
                    if valFAfterDelete == 0.0 or valFAfterDelete == -0.0:

file_compound_reaction_ko.write("{0}\n".format(reacao.build_reaction_string(reacao.r
eaction)))
                        reacoesZerandoFBA.append(reacao)

                    reacao.lower_bound = lbAux
                    reacao.upper_bound = ubAux
                    lbAux = 0
                    ubAux = 0

                    FBAAfterRestoreModel = model.optimize()
                    valFFormatAfterRestore = round(float(FBAAfterRestoreModel.f),
numCasasDec)
                    if valFFormatAfterRestore != valInitialSolutionFmt:
                        raise Exception("2o IF - Erro! FBA nao bate com a primeira
execucao")

                #print("Total de reacoes cujo FBA zerou = " +
str(len(reacoesZerandoFBA)))
                file_compound_reaction_ko.close()

                # AQUI ACONTECE UMA BIFURCACAO NA APLICACAO, ONDE
ELE FAZ ESSES PASSOS CASO TENHAMOS GENES PARA
                # TRABALHAR, CASO CONTRARIO, ELE PULA ISSO TUDO, FAZ O
CENARIO DESCRITO NO ELSE E SEGUE O FLUXO NO PASSO 7
                if len(model.genes) > 0:
                    #print("Extracao dos genes referentes as reacoes cujo FBA zerou -
inicio")
                    reacoesZerandoFBAArq = open(directory+"//"+"03-
relacao_nocaute_reacao-gene.txt", "w")

                    # Aqui pegamos as reacoes e extraimos os genes envolvidos com
cada uma
```

```
                listaGenesAlvos = []
                for reacao in reacoesZerandoFBA:
                    reacoesZerandoFBAArq.write(str(reacao) + '\n')
                    for gene in reacao.genes:
                        listaGenesAlvos.append(str(gene))
                        reacoesZerandoFBAArq.write(str(gene) + '\n')
                    reacoesZerandoFBAArq.write('----------\n\n')
                reacoesZerandoFBAArq.close()

                # Aqui com a lista de genes gerada a partir das reacoes,
ordenamos e colocamos em um arquivo
                listaGenesAlvosSet = list(set(listaGenesAlvos))
                listaGenesAlvosSet.sort()
                arqGenesAlvos = open(directory+"//"+"04-genesAlvos.txt", "w")
                contador = 0
                for gene in listaGenesAlvosSet:
                    contador += 1
                    arqGenesAlvos.write(str(contador) + " - " + str(gene) + "\n")
                arqGenesAlvos.close()
                #print("Total de genes encontrados por reacao = " +
str(len(listaGenesAlvosSet)))
                #print("Extracao dos genes referentes as reacoes cujo FBA zerou -
final")

                #print("GENERATED FILE %s" % reacoesZerandoFBAArq.name)
                #print("GENERATED FILE %s" % arqGenesAlvos.name)

                ########################################################
                # ITEM 4 - SIMULATION OF SINGLE KNOCKOUT OF GENES
                ########################################################
                fo = open(directory+"//"+"05-relacao_nocaute_gene-reacao.txt", "w")
                contador = 0
                listaGenesAlvosSet02 = []
                for i in model.genes:
                    FBABeforeDelete = model.optimize()
```

```
                    valFBeforeDelete = round(float(FBABeforeDelete.f),
numCasasDec)
                    if valFBeforeDelete != valInitialSolutionFmt:
                        raise Exception("1o IF - Erro! FBA nao bate com a primeira
execucao")

                    cobra.manipulation.delete_model_genes(model, [str(i)])

                    FBAAfterDelete = model.optimize()
                    valFAfterDelete = round(float(FBAAfterDelete.f), numCasasDec)
                    if valFAfterDelete == 0.0:
                        contador += 1
                        # reacoes_assoc = (l.reaction for l in i.reactions)
                        fo.write(str(contador) + ". Gene " + str(i) + " inibido.\n")
                        for x in i.reactions:
                            fo.write("%s : %s" % (x.id, x.reaction))
                        # fo.write("Reacoes associadas:\n%s" % ("{\n" +
",\n".join(reacoes_assoc) + "\n}"))
                        fo.write('\n\n')
                        listaGenesAlvosSet02.append(str(i))

                    cobra.manipulation.undelete_model_genes(model)

                    FBAAfterRestoreModel = model.optimize()
                    valFFormatAfterRestore = round(float(FBAAfterRestoreModel.f),
numCasasDec)
                    if valFFormatAfterRestore != valInitialSolutionFmt:
                        raise Exception("2o IF -Erro! FBA nao bate com a primeira
execucao")

                fo.write("Total de genes inibidos que interrompem a geracao de
biomassa = " + str(contador))
                fo.close()
```

```
                #print("Total de genes encontrados por nocaute direto = " +
str(len(listaGenesAlvosSet02)))
                #print("GENERATED FILE %s" % fo.name)



#################################################################
                # ITEM 5 - VERIFICATION AND UNIFICATION OF KNOCKOUT
RESULTS

#################################################################
                count = 0
                deParaGenes = open(directory+"//"+"06-deparaGenes.txt", "w")
                for geneAlvo in listaGenesAlvosSet:
                    if geneAlvo in listaGenesAlvosSet02:
                        count += 1
                        deParaGenes.write(geneAlvo)
                        deParaGenes.write('\n')
                deParaGenes.close()
                #print("Total de genes encontrados apos de/para = " + str(count))
                #print("GENERATED FILE %s" % deParaGenes.name)



#################################################################
                # ITEM 6 - SEARCH FOR THE CORRESPONDING EC NUMBER
OF THE GENE

#################################################################
                k = KEGG(cache=True)
                k.TIMEOUT = TIMEOUT_SECONDS

                contador = 0
                with open(directory+"//"+"06-deparaGenes.txt", "r") as infile:
                    data = infile.read()
```

```
                my_list_genes = data.splitlines()

                fileGenesWithEC = open(directory+"//"+"07-genes_ECNumbers.txt",
"w")
                fileAssocGenesEC = open(directory+"//"+"07-1-
assoc_genes_ECNumbers.txt", "w")

                # AQUI ELE BUSCA TODOS OS ACRONIMOS DO KEGG
RELACIONADOS COM O ORGANISMO SELECIONADO NA TELA
                df_list_organism =
pd.read_excel(dir_path+"//list_organism_kegg.xls")
                df_list_organism_filter =
df_list_organism[df_list_organism['name_organism'].str.contains(organismParam)]
                list_acron_kegg = df_list_organism_filter['acron_organism'].tolist()

                for acron in list_acron_kegg:
                    for gene in my_list_genes:
                        source = acron+":"+gene
                        #source = "pae:"+gene # pae:PA0005
                        ec = k.link("enzyme", source)

                        returnKeggEC = str(ec).split() # split por espaco para separar o
que veio do unicode
                        if (len(returnKeggEC)) == 0 :
                            continue
                        contador += 1

                        for ecnumber in returnKeggEC:
                            if "ec:" in ecnumber: # caso tenha o inicio "ec:"
                                ecnumbersplitFinal = ecnumber.split(":") # novo split para
prevalecer apenas o numero
                                fileGenesWithEC.write(ecnumbersplitFinal[-1])
                                fileGenesWithEC.write('\n')
                                fileAssocGenesEC.write('{0};{1}\n'.format(gene,
ecnumbersplitFinal[-1]))
```

```
                fileGenesWithEC.close()
                fileAssocGenesEC.close()

                #print("GENERATED FILE %s" % fileGenesWithEC.name)
                #print("GENERATED FILE %s" % fileAssocGenesEC.name)

            else:
                self.alternativeStepToGetECNumberWithoutGenes(directory)

            ############################################################
            # ITEM 7 - SEARCH FOR PROTEIN BY EC NUMBER NO
DRUGBANK
            ############################################################
            with open(directory+"//"+"07-genes_ECNumbers.txt", "r") as infile:
                data = infile.read()
            my_list_ecnumbers = data.splitlines()
            my_list_ecnumbers = list(set(my_list_ecnumbers))
            my_list_ecnumbers.sort()

            # my_list_ecnumbers = ['2.5.1.7']

            # Aqui filtra os ECs encontrados no drugbank e simula a navegacao
na pagina
            fileFilterEC = open(directory+"//"+"08-filter_ECNumbers_drugbank.txt",
"w")

            for ec in my_list_ecnumbers:
                #print(ec)
                time.sleep(1) # esperar 1 segundo para acessar o link

                link01 =
"https://www.drugbank.ca/unearth/q?utf8=%E2%9C%93&searcher=targets&query="+
str(ec)+"&aprovado=1&illicit=1&investigational=1&withdrawn=1&experimental=1&us=
1&canada=1&eu=1&commit=Apply+Filter"
```

```
                r = requests.get(link01)

                # verificar se houve retorno da pagina acessada
                if r.status_code == 200:

                    # Instancia de bs4 do resultado da 1a tela do drugbank
                    soup = BeautifulSoup(r.text)

                    # filtro para a lista de todos os itens encontrados na busca do
requests
                    list_found_ec = soup.findAll(attrs={"class": "search-result p-2 mb-
4 p-sm-3 mb-sm-2"})

                    # caso tenha item, inicia as buscas para pegar o resto da
informacao
                    if len(list_found_ec) > 0:

                        # itera na lista de itens encontrados no drugbank
                        for item in list_found_ec:
                            em = item.find('em')
                            ec_encontrado = em.text

                            #print(ec_encontrado, ec, ec_encontrado == ec)
                            if ec_encontrado != ec:
                                continue

                            # busca pelo link para acesso a 2a tela do drugbank
                            var = item.find('a')

                            # Montagem do link e acesso a 2a tela do drugbank para
buscar as informacoes necessarias
                            link02 = "https://www.drugbank.ca"+str(var['href'])
                            r2 = requests.get(link02)
                            if r2.status_code == 200:
```

```
                                # Nova instancia de bs4 com as informacoes da 2a tela
                                # e pegando nome da proteina, organismo, uniprot id e
drugbank id
                                soup2 = BeautifulSoup(r2.text)
                                organism_data_class = soup2.find(attrs={"class":
"content-container"})
                                protein_name = organism_data_class.findAll('dd')[0].text
                                organism_name =
organism_data_class.findAll('dd')[2].text
                                uniprot_id = organism_data_class.find('a').text
                                drugbank_id = str(var['href']).split("/")[-1]
                                fileFilterEC.write('{0};{1};{2};{3};{4}\n'.format(ec,
protein_name, organism_name, uniprot_id, drugbank_id))

                            else:
                                fileFilterEC.write('{0};{1}\n'.format(ec, r.raise_for_status()))

                        else:
                            fileFilterEC.write('{0};{1}\n'.format(ec, r.raise_for_status()))

                    fileFilterEC.close()

                    #print("GENERATED FILE %s" % fileFilterEC.name)


                    ##################################################
                    # ITEM 8 - SEARCH FOR THE HOMOLOGUES AT UNIPROT
                    ##################################################
                    with open(directory+"//"+"08-filter_ECNumbers_drugbank.txt", "r") as
infile:
                        data = infile.read()
                    my_list_uniprotid = data.splitlines()
                    my_list_uniprotid = list(set(my_list_uniprotid))
                    my_list_uniprotid.sort()
```

```
                # Instancia de chamada ao UniProt
                u = UniProt()
                s = NCBIblast()

                u.TIMEOUT = TIMEOUT_SECONDS
                s.TIMEOUT = TIMEOUT_SECONDS

                #my_list_uniprotid = ["P42898"]

                fileResultBlast = open(directory+"//"+"11-hitsEncontradosUniprot.txt",
"w")

                # GET ALL JOBID FROM UNIPROT FOUND
                file_jobid = open(directory+"//"+"list_jobid.txt", "w", encoding="ISO-
8859-1")
                start_time_first_req = time.time()
                for uniprotid in my_list_uniprotid:

                    uniprotid = uniprotid.split(';')[3]
                    #print(uniprotid)

                    # buscando a sequencia da proteina por ID, vindo do drugbank
                    sequence = u.retrieve(uniprotid, "fasta")
                    sequence = sequence.split("\n", 1)[1].strip("\n")

                    # executando o job que faz o blast
                    jobid = s.run(program="blastp",
database="uniprotkb_reference_proteomes", sequence=sequence, stype="protein",
email="thiago.merigueti@ioc.fiocruz.br", alignments='1000')
                    #print(jobid)
                    file_jobid.write("{0};{1}\n".format(uniprotid, jobid))

                elapsed_time = time.time() - start_time_first_req
                #print("TEMPO TOTAL PRIMEIRO PASSO = ",
time.strftime("%H:%M:%S", time.gmtime(elapsed_time)))
```

```
                file_jobid.close()

                # AFTER GENERATE ALL JOBID FROM BLAST, WE ITERATE ALL
ITENS AND GET ALL XML RETURN
                with open(directory+"//"+"list_jobid.txt", "r") as infile:
                    data = infile.read()
                my_list_jobid = data.splitlines()
                #my_list_jobid = list(set(my_list_jobid))

                arquivo = open(directory+"//"+"list_jobid.txt", 'r')

                for item in arquivo:
                    item = item.strip()
                    uniprotid = item.split(';')[0]
                    jobid = item.split(';')[1]

                    url_status_blast =
"http://www.ebi.ac.uk/Tools/services/rest/ncbiblast/status/"
                    url_result_blast =
"http://www.ebi.ac.uk/Tools/services/rest/ncbiblast/result/"
                    last_url_result_blast = "/xml"

                    start_time_first_req = time.time()

                    condition = True
                    count_error = 0
                    url_1 = url_status_blast+jobid
                    #print(url_1)
                    count_refresh = 0
                    while condition:
                        response = requests.get(url_1)
                        #print(response.text)
                        if response.text != "RUNNING":
                            if response.text == "FINISHED":
                                condition = False
```

```
                    if response.text == "ERROR":
                        count_error += 1
                        #print("ERRO!", count_error)
                        if count_error == 3:
                            condition = False

                time.sleep(3)
                if count_refresh > 60:
                    condition = False

                count_refresh += 1

            elapsed_time = time.time() - start_time_first_req
            #print("TEMPO TOTAL PRIMEIRO LINK = ",
time.strftime("%H:%M:%S", time.gmtime(elapsed_time)))

            start_time_second_req = time.time()

            if response.text != 'FINISHED':

fileResultBlast.write("{0};;{1};;{2};;{3};;{4};;{5};;{6};;{7};;{8};;{9}\n".format(
                    uniprotid, "ERRO", "ERRO", 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
                ))
                continue

            url_2 = url_result_blast+jobid+last_url_result_blast
            #print(url_2)
            response = requests.get(url_2)
            soup = BeautifulSoup(response.text)

            elapsed_time_2 = time.time() - start_time_second_req
            #print("TEMPO TOTAL SEGUNDO LINK = ",
time.strftime("%H:%M:%S", time.gmtime(elapsed_time_2)))
```

```
                # ARMAZENO EM OUTRA PASTA OS ARQUIVOS DO BLAST NA
INTEGRA
                fileHitsBlastIntegra = open(dir_blasts+"//all_hits_"+uniprotid+".xml",
"w")
                fileHitsBlastIntegra.write(str(soup))
                fileHitsBlastIntegra.close()

                # busca por todos os hits encontrados
                listAllHits = soup.findAll('hit')

                # itera os hits e, caso encontre pseudomonas, guarda as
informacoes dele
                fileListAllHist =
open(dir_blasts+"//hit_organism_found_"+uniprotid+".txt", 'w')

                percentSimilarHuman = 0.0
                for hit in listAllHits:
                    organism = hit['description'].split("=")[1].split("GN")[0].strip()
                    if "Homo sapiens" in organism:
                        percentSimilarHuman = float(hit.find('identity').string.strip())
                        break

                for hit in listAllHits:
                    organism = hit['description'].split("=")[1].split("GN")[0].strip()
                    fileListAllHist.write("{0}\n".format(organism))

                    #if organism == 'Pseudomonas aeruginosa (strain ATCC 15692 /
DSM 22644 / CIP 104116 / JCM 14847 / LMG 12228 / 1C / PRS 101 / PAO1)' :
                    if organismParam in organism: # se o valor da combo for
encontrado no hit, armazena
                        idPAO1 = hit['ac']
                        percentBlast = hit.find('identity').string.strip()
                        eValue = hit.find('expectation').string.strip()
```

```
                    foundID = u.search(idPAO1, columns='entry name, id, genes,
pathway, comment(FUNCTION), comment(CATALYTIC ACTIVITY), database(PDB),
database(PSEUDOCAP), comment(SUBCELLULAR LOCATION), ec')
                    splitForListFoundID = list(str(foundID).split("\t"))

                    # 0-uniprotid;;1-hit pseudomonas;;2-percentidentityblast;;3-
evalue;;
                    # 4-Gene names;;5-Pathway;;6-Function;;7-Catalitic Activity;;
                    # 8-Localization;;9-ID PDB
                    if float(percentSimilarHuman) < float(percentBlast): # so grava
se o percentual de similaridade for menor do que com humanos

fileResultBlast.write("{0};;{1};;{2};;{3};;{4};;{5};;{6};;{7};;{8};;{9}\n".format(
                        uniprotid, idPAO1, percentBlast, eValue,
splitForListFoundID[11], splitForListFoundID[12],
                        splitForListFoundID[13], splitForListFoundID[14],
splitForListFoundID[17], splitForListFoundID[15]
                    ))
                    else: # Caso tenha hit com percent de humano maior, poe tudo
zerado

fileResultBlast.write("{0};;{1};;{2};;{3};;{4};;{5};;{6};;{7};;{8};;{9}\n".format(
                        uniprotid, idPAO1, percentBlast, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0
                    ))

                fileListAllHist.close()

            fileResultBlast.close()
            arquivo.close()

            #print("GENERATED FILE %s" % fileListAllHist.name)
            #print("GENERATED FILE %s" % fileResultBlast.name)

            ##########################################################
```

```
                # ITEM 9 - SEARCH FOR THE INHIBITORS FOR THE BACTERIA
                ########################################################
                # PEGANDO A ENTRADA DO ITEM POR MEIO DOS ARQUIVOS
GERADOS E CONVERTENDO A DATAFRAME PARA FACILITAR
                data08 = pd.read_csv(directory + "//" + "08-
filter_ECNumbers_drugbank.txt", sep=";", header=None)
                data08.columns = ["EC NUMBER", "PROTEIN NAME", "ORGANISM
NAME", "UNIPROTID", "DRUGBANKID"]
                data08 = data08.sort_values("EC NUMBER")
                my_list_uniprotid_drugbankid = data08["UNIPROTID"].tolist()
                data08.to_excel(directory + "//" + "08-filter_ECNumbers_drugbank.xls")

                data11 = pd.read_csv(directory + "//" + "11-
hitsEncontradosUniprot.txt", sep=";;", header=None)
                data11.columns = ["UNIPROTID", "HIT_UNIPROTID",
"PERCENT_IDENT_BLAST", "EVALUE", "GENE_NAME",
                                  "PATHWAY", "FUNCTION", "CATALYTIC ACTIVITY",
"LOCALIZATION", "ID PDB"]
                data11 = data11.sort_values("UNIPROTID")
                my_list_uniprotid_hit = data11["UNIPROTID"].tolist()
                data11.to_excel(directory + "//" + "11-hitsEncontradosUniprot.xls")

                # GERACAO DOS ARQUIVOS DE SAIDA DESSE ITEM
                fileDataDrugs = open(directory + "//" + "13-
list_inhibitors_per_target.txt", "w")
                fileInhibitorsDrugs = open(directory + "//" + "14-
list_inhibitors_aprovado.txt", "w")

                for uniprot_drugbank in my_list_uniprotid_drugbankid:

                    if uniprot_drugbank in my_list_uniprotid_hit:
                        data11Aux = data11[data11['UNIPROTID'] == uniprot_drugbank]

                        hit_toxicity = False
                        for item in data11Aux.itertuples():
```

```
                    validate = item.GENE_NAME
                    if validate == '0.0':
                        hit_toxicity = True
                        break

                # CASO TENHA HIT MENOR QUE HUMANO, ESTARA CHEIO
DE ZEROS NO RESULTADO
                # SENDO ASSIM, ELE NAO DEVE SER CONSIDERADO UM
BOM RESULTADO E NAO DEVE SER ARMAZENADO
                if hit_toxicity:
                    continue

                #print(uniprot_drugbank)
                data08Aux = data08[data08["UNIPROTID"] == uniprot_drugbank]
                drugbankid = data08Aux.iloc[0][4]

                linkAccessForDrugsID =
"https://www.drugbank.ca/biodb/bio_entities/" + str(drugbankid)
                r = requests.get(linkAccessForDrugsID)

                if r.status_code == 200:
                    soup = BeautifulSoup(r.text)
                    listDrugsFound = soup.find(attrs={"class": "table table-sm table-
bordered datatable dt-responsive"})
                    listDrugsFound = listDrugsFound.find('tbody')
                    listDrugsFound = listDrugsFound.findAll('tr')  # lista com todos
os farmacos para o drugbankid informado

                    for item in listDrugsFound:
                        drugbank_drug_id = item('td')[0].text
                        drug_name = item('td')[1].text
                        drug_group = item('td')[2].text
                        pharma_action = item('td')[3].text
                        actions = item('td')[4].text
```

```
                        # 0-uniprotid;;1-drugbankid;;2-drugbank_drug_id;;3-
drug_name;;
                        # 4-drug_group;;5-pharma_action;;6-actions
                        fileDataDrugs.write("{0};{1};{2};{3};{4};{5};{6}\n".format(
                            uniprot_drugbank, drugbankid, drugbank_drug_id,
drug_name,
                            drug_group, pharma_action, actions
                        ))

                        if pharma_action == 'yes' and actions == 'inhibitor':
                            # 0-uniprotid;;1-drugbankid;;2-drugbank_drug_id;;3-
drug_name;;
                            # 4-drug_group;;5-pharma_action;;6-actions

fileInhibitorsDrugs.write("{0};{1};{2};{3};{4};{5};{6}\n".format(
                                uniprot_drugbank, drugbankid, drugbank_drug_id,
drug_name,
                                drug_group, pharma_action, actions
                            ))

                    else:
                        fileDataDrugs.write("{0};{1};{2};{3};{4};{5};{6}\n".format(
                            uniprot_drugbank, drugbankid, r.raise_for_status(),
r.raise_for_status(),
                            r.raise_for_status(), r.raise_for_status(), r.raise_for_status()
                        ))

                fileDataDrugs.close()
                fileInhibitorsDrugs.close()

                data13 = pd.read_csv(directory + "//" + "13-
list_inhibitors_per_target.txt", sep=";", header=None)
                data13.columns = ["UNIPROTID", "DRUGBANKID",
"DRUGBANKDRUGID", "DRUGNAME", "DRUGGROUP", "PHARMAACTION",
"ACTIONS"]
```

```
                data13 = data13.sort_values("UNIPROTID")
                data13.to_excel(directory + "//" + "13-list_inhibitors_per_target.xls")

                data14 = pd.read_csv(directory + "//" + "14-
list_inhibitors_aprovado.txt", sep=";", header=None)
                data14.columns = ["UNIPROTID", "DRUGBANKID",
"DRUGBANKDRUGID", "DRUGNAME", "DRUGGROUP", "PHARMAACTION",
"ACTIONS"]
                data14 = data14.sort_values("UNIPROTID")
                data14.to_excel(directory + "//" + "14-list_inhibitors_aprovado.xls")

                #print("GENERATED FILE %s" % fileDataDrugs.name)
                #print("GENERATED FILE %s" % fileInhibitorsDrugs.name)

                # CRIANDO A PLANILHA UNICA COM TODOS OS RESULTADOS
JUNTOS
                # CONTENDO APENAS AS OCORRENCIAS EM TODAS AS 3
PLANILHAS GERADAS!
                df_08 = pd.read_excel(directory + "//" + "08-
filter_ECNumbers_drugbank.xls")
                df_11 = pd.read_excel(directory + "//" + "11-
hitsEncontradosUniprot.xls")
                df_13 = pd.read_excel(directory + "//" + "13-
list_inhibitors_per_target.xls")

                df_merged = pd.merge(pd.merge(df_08, df_11, on='UNIPROTID',
how='inner'), df_13, on='UNIPROTID', how='inner')
                df_merged.to_excel(directory + "//" + "resultado_final_unificado.xls")


##########################################################################
################
                # ITEM 10 - LAST ITENS FOR THE METHOD WHERE ZIP ALL
DATAS GENERATED AND SEND MAIL
```

```
##########################################################################
################
                zip_file_report = self.zipReportsToSendMail(dir_path,
dataHoraAtualFmt, directory)
                self.sendMailWithReportAttached(name, email, zip_file_report)
                #print("ACABOU!!! SEJA FELIZ!!!")

            except Exception as e:
                self.sendMailWithError(name, email,
str(traceback.format_exception(None, e, e.__traceback__), file=sys.stderr,
flush=True))


        # GENERATE ZIP FILE TO SEND MAIL TO USER

        def zipReportsToSendMail(self, dir_path, dataHoraAtualFmt, directory):
            zf = ZipFile(dir_path+"/results/"+dataHoraAtualFmt+'_resultado.zip', "w")
            for root, subdirs, files in os.walk(directory):
                for filename in files:
                    file_extension = filename.split(".")[-1]
                    if file_extension == 'xls':
                        zf.write(os.path.join(root, filename), filename, ZIP_DEFLATED)
            zf.close()

            return zf


        # SEND MAIL WITH RESULTS

        def sendMailWithReportAttached(self, name, email, zip_file_report):
            mensagem = "FINAL REPORT - FIND TARGETS WEB \n\n\n" \
                    "" \
                    "YOUR ANALYSIS WAS FINISHED WITH SUCCESS AND
GENERATE THE FILES:\n" \
```

```
                    "" \
                    "08-filter_ECNumbers_drugbank.xls\n" \
                    "FIELDS: EC NUMBER, PROTEIN NAME, ORGANISM NAME,
UNIPROTID, DRUGBANKID\n\n" \
                    "" \
                    "11-hitsEncontradosUniprot.xls\n" \
                    "FIELDS: UNIPROTID, HIT_UNIPROTID,
PERCENT_IDENT_BLAST, EVALUE, GENE_NAME, PATHWAY, FUNCTION,
CATALYTIC ACTIVITY, LOCALIZATION, ID PDB\n\n" \
                    "" \
                    "13-list_inhibitors_per_target.xls AND 14-
list_inhibitors_aprovado.xls\n" \
                    "FIELDS: UNIPROTID          DRUGBANKID
        DRUGBANKDRUGID        DRUGNAME DRUGGROUP
        PHARMAACTION    ACTIONS\n\n" \
                    "" \
                    "dados_modelo.xls\n" \
                    "ALL GENES, REACTIONS AND METABOLITES IN SBML FILE
USED IN ANALYSIS\n\n" \
                    "" \
                    "resultado_final.xls\n" \
                    "FILE WITH ALL DATA FOUND IN 08, 11 AND 13 XLS
FILES.\n\n" \

            remetente    = 'merigueti@hotmail.com'
            senha        = senha

            # Informacoes da mensagem
            destinatario = email
            assunto      = 'REPORT THERAPEUTICS TARGETS FROM YOUR
NETWORK MODEL'

            msg = MIMEMultipart()

            msg['From'] = remetente
```

```
            msg['To'] = destinatario
            msg['Subject'] = assunto

            # Preparando a mensagem
            '''
            mensagem = '\r\n'.join([
              'From: %s' % remetente,
              'To: %s' % destinatario,
              'Subject: %s' % assunto,
              '',
              '%s' % mensagem
              ])
            '''
            mensagem = '\r\n'.join([
                '%s' % mensagem
            ])

            mensagem = mensagem.encode("UTF-8")

            msg.attach(MIMEText(mensagem.decode("UTF-8"), 'plain'))

            filename = zip_file_report.filename
            attachment = open(filename, "rb")

            part = MIMEBase('application', 'zip')
            part.set_payload((attachment).read())
            encoders.encode_base64(part)
            part.add_header('Content-Disposition', "attachment",
filename=os.path.basename(filename))
            msg.attach(part)

            # Enviando o email (USANDO O SMTP DO HOTMAIL PARA ENVIAR)
            server = smtplib.SMTP("smtp.live.com",587)
            server.starttls()
            server.login(remetente,senha)
```

```
            text = msg.as_string()
            server.sendmail(remetente, destinatario, text)
            server.quit()


        # SEND MAIL WITH ERRORS!

        def sendMailWithError(self, name, email, dsc_exception):
            mensagem = "ERROR! PLEASE CONTACT ADMINISTRATOR OR TRY
AGAIN\n\n"
            mensagem = mensagem + dsc_exception

            remetente = 'merigueti@hotmail.com'
            senha     = senha

            # Informacoes da mensagem
            destinatario = email
            assunto = 'REPORT THERAPEUTICS TARGETS FROM YOUR
NETWORK MODEL - ERROR!'

            msg = MIMEMultipart()

            msg['From'] = remetente
            msg['To'] = destinatario
            msg['Subject'] = assunto

            # Preparando a mensagem
            mensagem = '\r\n'.join([
                '%s' % mensagem
            ])

            mensagem = mensagem.encode("UTF-8")

            msg.attach(MIMEText(mensagem.decode("UTF-8"), 'plain'))
```

```
            # Enviando o email (USANDO O SMTP DO HOTMAIL PARA ENVIAR)
            server = smtplib.SMTP("smtp.live.com", 587)
            server.starttls()
            server.login(remetente, senha)
            text = msg.as_string()
            server.sendmail(remetente, destinatario, text)
            server.quit()


        # METODO QUE ARMAZENA EM PLANILHA OS DADOS DE GENE E REACAO DE UM MODELO AVALIADO
        # @PARAMS => model = modelo | nomeArquivoModelo = nome do arquivo sbml

        def reportModel(self, model, nomeArquivoModelo):
            workbook_dados_modelo = xlwt.Workbook()
            sheet_genes = workbook_dados_modelo.add_sheet("GENES")
            sheet_genes.write(0, 0, "ID GENE")
            sheet_genes.write(0, 1, "NOME GENE")
            sheet_genes.write(0, 2, "REACOES ASSOC")
            contador_genes = 1
            for gene in model.genes:
                assoc = (l.id for l in gene.reactions)
                sheet_genes.write(contador_genes, 0, gene.id)
                sheet_genes.write(contador_genes, 1, gene.name)
                sheet_genes.write(contador_genes, 2, ", ".join(assoc))
                contador_genes += 1

            sheet_react = workbook_dados_modelo.add_sheet("REACOES")
            sheet_react.write(0, 0, "ID REACAO")
            sheet_react.write(0, 1, "NOME REACAO")
            sheet_react.write(0, 2, "COMPOSICAO")
            sheet_react.write(0, 3, "LIMITE INF/SUP")
            sheet_react.write(0, 4, "SUBSISTEMA")
            sheet_react.write(0, 5, "GENES ASSOC")
```

```
            sheet_react.write(0, 6, "METABOLITOS ASSOC")
            contador_react = 1

            for reacao in model.reactions:
                assoc_genes = (l.id for l in reacao.genes)
                assoc_metab = (l.id for l in reacao.metabolites)
                sheet_react.write(contador_react, 0, reacao.id)
                sheet_react.write(contador_react, 1, reacao.name)
                sheet_react.write(contador_react, 2,
reacao.build_reaction_string(reacao.reaction))
                sheet_react.write(contador_react, 3, str(reacao.bounds))
                sheet_react.write(contador_react, 4, reacao.subsystem)
                sheet_react.write(contador_react, 5, ", ".join(assoc_genes))
                sheet_react.write(contador_react, 6, ", ".join(assoc_metab))
                contador_react += 1

            workbook_dados_modelo.save(str(nomeArquivoModelo))



        # METHOD TO GET EC NUMBER FROM SBML WITHOUT GENES
MAPPED

        def alternativeStepToGetECNumberWithoutGenes(self, directory):
            with open(directory+"//react_biomass_zero_sbml_no_genes.txt", "r") as
infile:
                data = infile.read()
            my_list_compound_sbml = data.splitlines()

            k = KEGG(False, True)
            k.TIMEOUT = 200000
            pd.options.display.max_colwidth = 1000
            pd.options.display.max_rows = 1000
```

```
            file_ecs = open(directory+"//07-genes_ECNumbers.txt", "w",
encoding="ISO-8859-1")
            file_ecs_compound = open(directory + "//07-1-
assoc_EC_compounds.txt", "w", encoding="ISO-8859-1")
            file_comp_not_found_from_reactant_sbml =
open(directory+"//idcomp_notfound_kegg.txt", "w", encoding="ISO-8859-1")
            file_reaction_not_found_from_reactant_sbml =
open(directory+"//idreaction_not_found_kegg.txt", "w", encoding="ISO-8859-1")

            #print("PARTE 1 METODO ALTERNATIVO")
            for compound in my_list_compound_sbml:
                #print("parte1", compound)
                compound_no_stoich = re.sub("\d+\.\d+", "", compound) # retirada de
todos os valores estequiometricos

                if " - reduced " in compound:
                    compound_no_stoich = compound_no_stoich.replace(" - reduced ",
" ")

                param_splt = "" # preparo do split para separar a composicao quimica
                if "<=>" in compound_no_stoich:
                    param_splt = "<=>"
                else:
                    param_splt = "-->"

                compound_splt = compound_no_stoich.split(param_splt) # separacao
da composicao quimica pelo reactante(0) e o produto(1)

                # Caso seja uma reacao de insercao ou uma reacao de excrecao, ele
ignora
                if compound_splt[0].strip() == "" or compound_splt[1].strip() == "":
                    continue

                # TRATAMENTO DOS DADOS DO REAGENTE DA COMPOSICAO
ENCONTRADA
```

```
                reactant_sbml = compound_splt[0].strip()

                list_id_cpd_kegg = []
                len_reactant_sbml = 0

                # verifica os reagentes que possuem mais de um composto envolvido
                if " + " in reactant_sbml: #(ex.: acetyl-coa + atp + bicarbonate)
                    reactant_sbml_splt = reactant_sbml.split(" + ")
                    len_reactant_sbml = len(reactant_sbml_splt)

                    # iteracao dentro dos compostos dos reactants (acetyl-coa + atp +
bicarbonate)
                    for cpd_reactant_sbml in reactant_sbml_splt:
                        #http://rest.kegg.jp/find/compound/acetyl-coa
                        result_id_cpd = k.find("compound", cpd_reactant_sbml)
                        result_id_cpd_splt = result_id_cpd.split('\n')
                        result_id_cpd_splt = filter(None, result_id_cpd_splt)
                        result_id_cpd_splt_filter = list(result_id_cpd_splt)

                        # iteracao dentro do resultado encontrado para um dos
compostos do reagente
                        # (cpd:C00024 Acetyl-CoA; Acetyl coenzyme A)
                        if len(result_id_cpd_splt_filter) > 0:
                            for result_cpd in result_id_cpd_splt_filter:
                                local_result_cpd_splt = result_cpd.split('\t')
                                if ";" in local_result_cpd_splt[1]:
                                    dsc_cpd_splt = local_result_cpd_splt[1].split(';')
                                    for cpd_splt in dsc_cpd_splt:
                                        if cpd_splt.strip().lower() ==
cpd_reactant_sbml.strip().lower():
                                            list_id_cpd_kegg.append(local_result_cpd_splt[0])
                                            break
                                else:
                                    if cpd_reactant_sbml.lower() ==
local_result_cpd_splt[1].lower():
```

```
                                list_id_cpd_kegg.append(local_result_cpd_splt[0])
                                break
                    else:
                        len_reactant_sbml = 1
                        result_id_cpd = k.find("compound", reactant_sbml)
                        result_id_cpd_splt = result_id_cpd.split('\n')
                        result_id_cpd_splt = filter(None, result_id_cpd_splt)
                        result_id_cpd_splt_filter = list(result_id_cpd_splt)

                        if len(result_id_cpd_splt_filter) > 0:
                            for result_cpd in result_id_cpd_splt_filter:
                                local_result_cpd_splt = result_cpd.split('\t')
                                if ";" in local_result_cpd_splt[1]:
                                    dsc_cpd_splt = local_result_cpd_splt[1].split(';')
                                    for cpd_splt in dsc_cpd_splt:
                                        if cpd_splt.strip().lower() == reactant_sbml.strip().lower():
                                            list_id_cpd_kegg.append(local_result_cpd_splt[0])
                                            break
                                else:
                                    if reactant_sbml.strip().lower() ==
local_result_cpd_splt[1].strip().lower():
                                        list_id_cpd_kegg.append(local_result_cpd_splt[0])
                                        break

                    # caso nao tenha encontrado todos os compostos, ele nao segue o
fluxo
                    if len(list_id_cpd_kegg) != len_reactant_sbml:

file_comp_not_found_from_reactant_sbml.write("{0}\n".format(compound))
                        continue

                    # PREPARO DA APLICACAO PARA USAR OS IDS DE
COMPOSTOS E BUSCAR PELAS REACOES DE CADA UM
                    reactant_sbml_in_cpd = "+".join(list_id_cpd_kegg)
```

```
                result_link_reactions_cpd = k.link("reaction",
str(reactant_sbml_in_cpd))
                result_link_reactions_cpd_splt = result_link_reactions_cpd.split('\n')
                result_link_reactions_cpd_splt = filter(None,
result_link_reactions_cpd_splt)
                result_link_reactions_cpd_splt_filter =
list(result_link_reactions_cpd_splt)
                df_link_reaction_cpd = pd.DataFrame(columns=['id_cpd', 'id_reaction'])

                index_link_react_cpd = 0
                if len(result_link_reactions_cpd_splt_filter) > 0:
                    for item_result_link_reactions_cpd in
result_link_reactions_cpd_splt_filter:
                        local_item_result_link = item_result_link_reactions_cpd.split('\t')
                        df_link_reaction_cpd.loc[index_link_react_cpd] =
[local_item_result_link[0], local_item_result_link[1]]
                        index_link_react_cpd += 1
                else:

file_reaction_not_found_from_reactant_sbml.write("{0}\n".format(compound))
                    continue

                df_link_reaction_cpd =
df_link_reaction_cpd.groupby("id_reaction").filter(lambda x: len(x) ==
len(reactant_sbml_splt))
                set_id_reaction_kegg = {x.id_reaction for x in
df_link_reaction_cpd.itertuples()}

                if len(set_id_reaction_kegg) > 0:
                    for item_id_react in set_id_reaction_kegg:
                        result_ec_number = k.link("enzyme", item_id_react)
                        result_ec_number_splt = result_ec_number.split('\n')
                        result_ec_number_splt = filter(None, result_ec_number_splt)
                        result_ec_number_filter = list(result_ec_number_splt)
```

```
                    for result_ec in result_ec_number_filter:
                        local_result_ec = result_ec.split('\t')
                        local_result_ec[1] = local_result_ec[1].replace("ec:", "").strip()
                        file_ecs.write("{0}\n".format(local_result_ec[1]))

            file_comp_not_found_from_reactant_sbml.close()
            file_reaction_not_found_from_reactant_sbml.close()

            # AQUI ELE INICIA A BUSCA POR PRODUTO DO QUE NAO FOI
ENCONTRADO POR REAGENTE
            with open(directory+"//idcomp_notfound_kegg.txt", "r") as infile:
                data = infile.read()
            my_list_compound_sbml_not_found_from_reactant = data.splitlines()

            file_comp_not_found_from_product_sbml =
open(directory+"//idcomp_notfound_2_kegg.txt", "w", encoding="ISO-8859-1")
            file_reaction_not_found_from_product_sbml =
open(directory+"//idreaction_notfound_2_kegg.txt", "w", encoding="ISO-8859-1")

            #print("PARTE 2 METODO ALTERNATIVO")
            for compound in my_list_compound_sbml_not_found_from_reactant:
                #print("parte2", compound)
                compound_no_stoich = re.sub("\d+\.\d+", "", compound) # retirada de
todos os valores estequiometricos

                if "- reduced" in compound:
                    compound_no_stoich = compound_no_stoich.replace("- reduced", "
")

                param_splt = "" # preparo do split para separar a composicao quimica
                if "<=>" in compound_no_stoich:
                    param_splt = "<=>"
                else:
                    param_splt = "-->"
```

```
                compound_splt = compound_no_stoich.split(param_splt) # separacao
da composicao quimica pelo reactante(0) e o produto(1)

                # Caso seja uma reacao de insercao ou uma reacao de excrecao, ele
ignora
                if compound_splt[0].strip() == "" or compound_splt[1].strip() == "":
                    continue

                product_sbml = compound_splt[1].strip()

                list_id_cpd_kegg = []
                len_product_sbml = 0

                # verifica os reagentes que possuem mais de um composto envolvido
                if " + " in product_sbml: #(ex.: acetyl-coa + atp + bicarbonate)
                    product_sbml_splt = product_sbml.split(" + ")
                    len_product_sbml = len(product_sbml_splt)

                    # iteracao dentro dos compostos dos reactants (acetyl-coa + atp +
bicarbonate)
                    for cpd_product_sbml in product_sbml_splt:
                        #http://rest.kegg.jp/find/compound/acetyl-coa
                        result_id_cpd = k.find("compound", cpd_product_sbml)
                        result_id_cpd_splt = result_id_cpd.split('\n')
                        result_id_cpd_splt = filter(None, result_id_cpd_splt)
                        result_id_cpd_splt_filter = list(result_id_cpd_splt)

                        # iteracao dentro do resultado encontrado para um dos
compostos do reagente
                        # (cpd:C00024 Acetyl-CoA; Acetyl coenzyme A)
                        if len(result_id_cpd_splt_filter) > 0:
                            for result_cpd in result_id_cpd_splt_filter:
                                local_result_cpd_splt = result_cpd.split('\t')
                                if ";" in local_result_cpd_splt[1]:
                                    dsc_cpd_splt = local_result_cpd_splt[1].split(';')
```

```
                                for cpd_splt in dsc_cpd_splt:
                                    if cpd_splt.strip().lower() ==
cpd_product_sbml.strip().lower():
                                        list_id_cpd_kegg.append(local_result_cpd_splt[0])
                                        break
                            else:
                                if cpd_product_sbml.lower() ==
local_result_cpd_splt[1].lower():
                                    list_id_cpd_kegg.append(local_result_cpd_splt[0])
                                    break
                    else:
                        len_product_sbml = 1
                        result_id_cpd = k.find("compound", product_sbml)
                        result_id_cpd_splt = result_id_cpd.split('\n')
                        result_id_cpd_splt = filter(None, result_id_cpd_splt)
                        result_id_cpd_splt_filter = list(result_id_cpd_splt)

                        if len(result_id_cpd_splt_filter) > 0:
                            for result_cpd in result_id_cpd_splt_filter:
                                local_result_cpd_splt = result_cpd.split('\t')
                                if ";" in local_result_cpd_splt[1]:
                                    dsc_cpd_splt = local_result_cpd_splt[1].split(';')
                                    for cpd_splt in dsc_cpd_splt:
                                        if cpd_splt.strip().lower() == product_sbml.strip().lower():
                                            list_id_cpd_kegg.append(local_result_cpd_splt[0])
                                            break
                                else:
                                    if product_sbml.strip().lower() ==
local_result_cpd_splt[1].strip().lower():
                                        list_id_cpd_kegg.append(local_result_cpd_splt[0])
                                        break

                    # caso nao tenha encontrado todos os compostos, ele nao segue o
fluxo
                    if len(list_id_cpd_kegg) != len_product_sbml:
```

```
file_comp_not_found_from_product_sbml.write("{0}\n".format(compound))
                    continue

                # PREPARO DA APLICACAO PARA USAR OS IDS DE
COMPOSTOS E BUSCAR PELAS REACOES DE CADA UM
                product_sbml_in_cpd = "+".join(list_id_cpd_kegg)
                result_link_reactions_cpd = k.link("reaction",
str(product_sbml_in_cpd))
                result_link_reactions_cpd_splt = result_link_reactions_cpd.split('\n')
                result_link_reactions_cpd_splt = filter(None,
result_link_reactions_cpd_splt)
                result_link_reactions_cpd_splt_filter =
list(result_link_reactions_cpd_splt)
                df_link_reaction_cpd = pd.DataFrame(columns=['id_cpd', 'id_reaction'])

                index_link_react_cpd = 0
                if len(result_link_reactions_cpd_splt_filter) > 0:
                    for item_result_link_reactions_cpd in
result_link_reactions_cpd_splt_filter:
                        local_item_result_link = item_result_link_reactions_cpd.split('\t')
                        df_link_reaction_cpd.loc[index_link_react_cpd] =
[local_item_result_link[0], local_item_result_link[1]]
                        index_link_react_cpd += 1
                else:

file_reaction_not_found_from_product_sbml.write("{0}\n".format(compound))
                    continue

                df_link_reaction_cpd =
df_link_reaction_cpd.groupby("id_reaction").filter(lambda x: len(x) ==
len(product_sbml_splt))
                set_id_reaction_kegg = {x.id_reaction for x in
df_link_reaction_cpd.itertuples()}
```

```
                if len(set_id_reaction_kegg) > 0:
                    for item_id_prod in set_id_reaction_kegg:
                        result_ec_number = k.link("enzyme", item_id_prod)
                        result_ec_number_splt = result_ec_number.split('\n')
                        result_ec_number_splt = filter(None, result_ec_number_splt)
                        result_ec_number_filter = list(result_ec_number_splt)

                        for result_ec in result_ec_number_filter:
                            local_result_ec = result_ec.split('\t')
                            local_result_ec[1] = local_result_ec[1].replace("ec:", "").strip()
                            file_ecs.write("{0}\n".format(local_result_ec[1]))
                            file_ecs_compound.write("{0};{1}\n".format(compound,
local_result_ec[1]))

            file_ecs.close()
            file_ecs_compound.close()
            file_comp_not_found_from_product_sbml.close()
            file_reaction_not_found_from_product_sbml.close()
```

APÊNDICE C - RESULTADOS OBTIDOS – MAIORES INFORMAÇÕES

Aqui nesse anexo apresento todas as tabelas descritas no capítulo de resultados, com o adicional das informações de função e localização de cada proteína mapeada.

Tabela 22 - Lista de potenciais alvos terapêuticos encontrados para os 4 modelos testados. (APÊNDICE C)

EC	Proteína	Gene	E-Value	Via	Função	Localização
1.1.1.25	Shikimate dehydrogenase	aroE PA0025	6E-84	Biossíntese intermediária metabólica; biossíntese de cororateamento; corismato de D-erythrose 4-fosfato e fosfoenolpiruvato: passo 4/7.	Envolvido na biossíntese do chorismo, o que leva à biossíntese de aminoácidos aromáticos. Cataliza a redução reversa do 3-dehidroshikimato (DHSA) ligado ao NADPH reversível para produzir shikimato (SA).	N/A
1.3.1.98	UDP-N-acetylenolpyruvoylglucosamine reductase	murB PA2977	4,7E-79	Biogênese da parede celular; biossíntese de peptidoglicano.	Formação da parede celular.	Citoplasma
1.5.1.3	Dihydrofolate reductase	folA PA0350	2E-38	Biossíntese de cofactores; biossíntese de tetrahidrofolato; 5,6,7,8-tetra-hidrofolato a partir de 7,8-di-hidrofolato: passo 1/1.	Principais enzimas no metabolismo de folato. Cataliza uma reação essencial para síntese de glicina e purina *de novo* e para a síntese de precursores de DNA.	N/A
2.1.1.45	Thymidylate synthase	thyA PA0342	4,3E-141	Metabolismo de pirimidina; biossíntese de TDT.	Cataliza a metilação redutora de 2'-desoxiuridina-5'-monofosfato (dUMP) em 2'-desoxitimidina-5'-monofosfato (dTMP) enquanto utiliza 5,10-metilenetetra-	Citoplasma

					hidrofolato (mTHF) como doador de metilo e redutora na reação, produzindo dihidrofolato (DHF) como subproduto. Esta reação enzimática fornece uma fonte intracelular de novo de dTMP, um precursor essencial para a biossíntese do DNA.	
2.4.1.227	UDP-N-acetylglucosamine--N-acetylmuramyl-(pentapeptide) pyrophosphoryl-undecaprenol N-acetylglucosamine transferase	murG PA4412	2,3E-88	Biogênese da parede celular; biossíntese de peptidoglicano.	Formação da parede celular. Cataliza a transferência de uma subunidade de GlcNAc no pentapéptido de undacaprenil-pirofosforilo-MurNAc (intermediário lipídico I) para formar GlcNAc de undecaprenil-pirofosforilo-MurNAc (pentapeptídeo) (intermediário lipídico II).	Membrana interna celular; Proteína de membrana periférica
2.5.1.15	Dihydropteroate synthase 1	folP PA4750	5,8E-103	Biossíntese de cofactores; biossíntese de tetrahidrofolato; 7,8-di-hidrofolato a partir de difosfato de 2-amino-4-	Cataliza a condensação de para-aminobenzoato (pABA) com difosfato de 6-hidroximetil-7,8-dihidropterina (DHPt-PP) para formar 7,8-	N/A

				hidroxi-6-hidroximetil-7,8-dihidropteridina e 4-aminobenzoato: passo 1/2.	dihidropteroato (H2Pte), o precursor imediato de derivados de folato.	
2.5.1.7	UDP-N-acetylglucosamine 1-carboxyvinyltransferase	murA PA4450	5,1E-171	Biogênese da parede celular; biossíntese de peptidoglycan.	Formação da parede celular. Adiciona enolpiruvil a UDP-N-acetilglucosamina.	Citoplasma
2.6.1.85	Para-aminobenzoate synthase component 1	pabB PA1758	3,2E-119	N/A	N/A	N/A
2.7.4.25	Cytidylate kinase	cmk PA3163	4,9E-86	N/A	N/A	Citoplasma
6.3.2.13	UDP-N-acetylmuramoyl-L-alanyl-D-glutamate--2,6-diaminopimelate ligase	murE PA4417	4,7E-131	Biogênese da parede celular; biossíntese de peptidoglycan.	Cataliza a adição de ácido meso-diaminopimélico ao precursor de nucleótidos UDP-N-acetilmuramoil-L-alanil-D-glutamato (UMAG) na biossíntese de peptidoglicano de parede celular bacteriana.	Citoplasma
6.3.2.9	UDP-N-acetylmuramoylalanine--D-glutamate ligase	murD PA4414	5,8E-123	Biogênese da parede celular; biossíntese de peptidoglycan.	Formação da parede celular. Cataliza a adição de glutamato ao UDP-N-acetilmuramoyl-L-alanina precursor de nucleótidos (UMA).	Citoplasma
1.1.1.100	3-oxoacyl-[acyl-carrier-protein] reductase FabG	fabG PA2967	1,5E-99	Metabolismo lipídico; biossíntese de	Cataliza a redução dependente de NADPH de substratos	N/A

				ácidos graxos.	beta-cetoacilo-ACP em produtos beta-hidroxi-acil-ACP, o primeiro passo redutor no ciclo de alongamento da biossíntese de ácidos graxos.	

Tabela 23 - Lista de potenciais alvos encontrados para os modelos PAO1 2008, PAO1 2017 e PA14 (APÊNDICE C).

EC	Proteína	Gene	E-Value	Via	Função	Localização
1.17.1.8	4-hydroxy-tetrahydrodipicolinate reductase	dapB PA4759	1,5E-107	Biossíntese de aminoácidos; Biossíntese de L-lisina via via DAP; (S) -tetra-hidrodipicolinato de L-aspartato: passo 4/4.	Catalisa a conversão de 4-hidroxi-tetra-hidrodipicolinato (HTPA) em tetra-hidrodipicolinato.	Citoplasma
2.5.1.61	Porphobilinogen deaminase	hemC PA5260	5,8E-137	Metabolismo composto contendo porfirina; biossíntese de protoporfirina-IX; coproporfirinógeno -III a partir de 5-aminolevulinato: passo 2/4.	Tetrapolimerização do monopirrole PBG no pré-uroporfirinogénio hidroximetilbilano em várias etapas discretas.	N/A
2.7.7.23	Bifunctional protein GlmU	glmU PA5552	1,2E-164	Biossíntese de açúcares nucleotídicos; Biossíntese de UDP-N-acetil-alfa-D-glucosamina; N-	Cataliza as duas últimas reações seqüenciais na via biossintética de novo para UDP-N-acetilglucosamina	Citoplasma

				acetil-alfa-D-glucosamina 1-fosfato a partir de 6-fosfato de alfa-D-glucosamina (via II): passo 2/2. Biossíntese de açúcares nucleotídicos; Biossíntese de UDP-N-acetil-alfa-D-glucosamina; UDP-N-acetil-alfa-D-glucosamina a partir de 1-fosfato de N-acetil-alfa-D-glucosamina: passo 1/1. Biogênese da membrana externa bacteriana; Biossíntese LPS lipídica A.	(UDP-GlcNAc). O domínio C-terminal catalisa a transferência do grupo acetilo da acetil coenzima A para o glucosamina-1-fosfato (GlcN-1-P) para produzir N-acetilglucosamina-1-fosfato (GlcNAc-1-P), que é convertido em UDP -GlcNAc pela transferência de 5-monofosfato de uridina (a partir de 5-trifosfato de uridina), uma reacção catalisada pelo domínio N-terminal.	
4.1.3.38	Aminodeoxychorismate lyase	pabC PA2964	1,4E-35	N/A	N/A	N/A
4.2.3.5	Chorismate synthase	aroC PA1681	3,6E-95	Biossíntese intermediária metabólica; biossíntese de cororateamento; corismato de D-erythrose 4-fosfato e fosfoenolpiruvato: passo 7/7.	Cataliza a eliminação anti-1,4 do fosfato C-3 e o hidrogénio Pro-R C-6 a partir de 5-enolpiruvil-shikimato-3-fosfato (EPSP) para produzir o corismato, que é o composto do ponto ramificado que serve como substrato	N/A

					de partida para as três vias terminais da biossíntese de aminoácidos aromáticos. Esta reação introduz uma segunda ligação dupla no sistema de anel aromático.	
5.3.1.1	Triosephosphate isomerase	tpiA PA4748	2E-74	Biossíntese de carboidratos; gluconeogênese. Degradação de carboidratos; glicolise; D-gliceraldeído 3-fosfato a partir de fosfato de glicerona: passo 1/1.	Envolvido na gluconeogênese. Cataliza estereospecificamente a conversão de fosfato de dihidroxiacetona (DHAP) para D-gliceraldeído-3-fosfato (G3P).	Citoplasma
5.3.1.6	Ribose-5-phosphate isomerase A	rpiA PA0330	4,6E-88 / 3,7E-79	Degradação de carboidratos; caminho de pentose fosfato; 5-fosfato de D-ribose a partir de 5-fosfato de D-ribulose (estádio não oxidativo): passo 1/1.	Catalisa a conversão reversível de ribose-5-fosfato em 5-fosfato de ribulose.	N/A

Tabela 24 - Lista de potenciais alvos encontrados para os modelos PAO1 2017, PA14 e CCBH4851 (APÊNDICE C).

EC	Proteína	Gene	E-Value	Via	Função	Localização
2.4.2.10	Orotate phosphoribosyltransferase	pyrE PA5331	1,4E-103	Metabolismo de pirimidina; Biossíntese UMP via via de novo; UMP do orotate: passo 1/2.	Cataliza a transferência de um grupo fosfato de ribosilo de 5-fosforo-1-difosfato para orotato, levando à formação de ortodina monofosfato (OMP).	N/A
4.3.3.7	4-hydroxy-tetrahydrodipicolinate synthase	dapA PA1010	3,6E-79	Biossíntese de aminoácidos; Biossíntese de L-lisina via via DAP; (S) -tetra-hidrodipicolinato de L-aspartato: passo 3/4.	Cataliza a condensação de (S) -aspartato-beta-semialdeído [(S) -ASA] e piruvato para 4-hidroxi-tetra-hidrodipicolinato (HTPA).	Citoplasma

Tabela 25 - Lista de potenciais alvos encontrados para os modelos PAO1 2008 e CCBH4851 (APÊNDICE C).

EC	Proteína	Gene	E-Value	Via	Função	Localização
1.1.1.133	dTDP-4-dehydrorhamnose reductase	rmlD PA5162	1,2E-95	Biossíntese de carboidratos; biossíntese dTDP-L-Rhamnose.	Cataliza a redução de dTDP-6-desoxi-L-liso-4-hexulose para produzir dTDP-L-ramnose.	N/A
1.3.1.9	Enoyl-[acyl-carrier-protein] reductase [NADH] FabI	fabI PA1806	1,1E-121	Metabolismo lipídico; biossíntese de ácidos graxos.	Cataliza a redução de uma ligação dupla carbono-carbono em uma fração enoyl que está ligada covalentemente a uma proteína transportadora de	N/A

					acilo (ACP). Envolvido no ciclo de alongamento do ácido gordo que são utilizados no metabolismo lipídico (por similaridade).	
2.1.2.11	3-methyl-2-oxobutanoate hydroxymethyltransferase	panB2 PA4729	1,1E-85 / 1,1E-93	Biossíntese de cofactores; Biossíntese de pantotenato (R); (R) -panoato de 3-metil-2-oxobutanoato: passo 1/2.	Cataliza a reação reversível em que o grupo hidroximetilo do 5,10-metilenetetra-hidrofolato é transferido para o alfa-cetoisovalerato para formar cetopantoato.	Citoplasma
2.5.1.55	2-dehydro-3-deoxyphosphooctonate aldolase	kdsA PA3636	2,6E-137	Biossíntese de carboidratos; Biossíntese de 3-desoxi-D-manno-octulosonato; 3-desoxi-D-manno-octulosonato de D-ribulose 5-fosfato: passo 2/3. Biogênese da membrana externa bacteriana; biossíntese de lipopolissacarídeos.	N/A	Citoplasma
2.7.1.24	Dephospho-CoA kinase	coaE PA4529	1,9E-56	Biossíntese de cofactores; biossíntese de coenzima A; CoA de (R) -	Cataliza a fosforilação do grupo 3'-hidroxilo da desfosfocoenzima A para formar coenzima A.	Citoplasma

				pantotenato: passo 5/5.		
2.7.1.26	Riboflavin biosynthesis protein RibF	ribF PA4561	5,2E-42	Biossíntese de cofactores; Biossíntese de FAD; FAD da FMN: passo 1/1. Biossíntese de cofactores; Biossíntese FMN; FMN da riboflavina (via ATP): passo 1/1.	N/A	N/A
2.7.4.9	Thymidylate kinase	tmk PA2962	8,6E-46	N/A	Fosforilação de dTMP para formar dTDP em ambos os caminhos de novo e salvamento da síntese de dTTP.	N/A
2.7.7.18	Nicotinate-nucleotide adenylyltransferase	nadD PA4006	1,8E-38	Biossíntese de cofactores; Bensíntese de NAD (+); deamido-NAD (+) do nicotinato D-ribonucleótido: passo 1/1.	Catalisa a adenilação reversível do mononucleótido de nicotinato (NaMN) em dinucleótido de adenina de ácido nicotínico (NaAD).	N/A
2.7.7.3	Phosphopantetheine adenylyltransferase	coaD PA0363	1,4E-48	Biossíntese de cofactores; biossíntese de coenzima A; CoA do (R) - pantotenato: passo 4/5.	Transfere reversivelmente um grupo adenililo de ATP para 4'-fosfopanteteína, produzindo dephospho-CoA (dPCoA) e pirofosfato.	Citoplasma
			2,4E-84	Biossíntese de	Ativa KDO (um açúcar	Citoplasma

2.7.7.38	3-deoxy-manno-octulosonate cytidylyltransferase	kdsB PA2979		açúcares nucleotídicos; Biossíntese de CMP-3-desoxi-D-manno-octulosonato; CMP-3-desoxi-D-mano-octulosonato a partir de 3-desoxi-D-mano-octosonato e CTP: passo 1/1. Biogênese da membrana externa bacteriana; biossíntese de lipopolissacarídeos.	de 8-carbono requerido) para incorporação em lipopolissacarídeo bacteriano em bactérias Gram-negativas.	
3.1.3.45	3-deoxy-D-manno-octulosonate 8-phosphate phosphatase KdsC	PA4458	5,5E-34	N/A	Envolvido na biossíntese de lipopolissacarídeos (LPSs). Cataliza a hidrólise de 8-fosfato de 3-desoxi-D-manno-octulosonato (KDO 8-P) até 3-desoxi-D-mano-octulosonato (KDO) e fosfato inorgânico.	N/A
3.6.1.1	Inorganic pyrophosphatase	ppa PA4031	3,9E-73	N/A	Cataliza a hidrólise do pirofosfato inorgânico (PPi) formando dois íons fosfato.	Citoplasma
4.2.1.46	dTDP-glucose 4,6-dehydratase	rmlB PA5161	2,1E-152	N/A	N/A	N/A

6.3.2.1	Pantothenate synthetase	panC PA4730	7,4E-77 / 5,4E-68	Biossíntese de cofactores; Biossíntese de pantotenato (R); (R) -pantotenato de (R) -pantoato e beta-alanina: passo 1/1.	Cataliza a condensação de pantoato com beta-alanina em uma reação dependente de ATP via intermediário de pantoil-adenilato.	Citoplasma

Tabela 26 - Lista de potenciais alvos encontrados para os modelos PAO1 2017 e PA14 (APÊNDICE C).

EC	Proteína	Gene	E-Value	Via	Função	Localização
2.5.1.10	Farnesyl diphosphate synthase	ispA PA4043	1,9E-81	N/A	N/A	N/A
2.7.1.148	4-diphosphocytidyl-2-C-methyl-D-erythritol kinase	ispE PA4669	1,1E-97	Biossíntese isoprenóidea; biossíntese de isopentenil difosfato via via DXP; difosfato de isopentenilo a partir de 1-desoxi-D-xilulose 5-fosfato: passo 3/6.	Cataliza a fosforilação do grupo hidroxi de posição 2 de 4-difosfocitil-2C-metil-D-eritritol.	N/A
2.7.7.60	2-C-methyl-D-erythritol 4-phosphate cytidylyltransferase	ispD PA3633	9,9E-52	Biossíntese isoprenóidea; biossíntese de isopentenil difosfato via via DXP; difosfato de isopentenilo a partir de 1-desoxi-D-xilulose 5-fosfato: passo 2/6.	Cataliza a formação de 4-difosfocitil-2-C-metil-D-eritritol a partir de CTP e 2-C-metil-D-eritritol 4-fosfato (MEP).	N/A
3.4.16.4	D-alanyl-D-alanine	dacC	3,5E-32 /	N/A	Cataliza a reticulação	Membrana

	carboxypeptidase	PA3999, PA3047, ftsI pbpB PA4418, pbpC PA2272	1,5E-110 / 4,6E-107 / 4,7E-156		da parede celular do peptidoglicano no septo da divisão (por similaridade). Vincula-se à penicilina	interna celular; Proteína de membrana de passagem única
3.5.2.3	Dihydroorotase	pyrC PA3527	5,6E-130	Metabolismo de pirimidina; Biossíntese UMP via via de novo; (S) -dihidroorotato de bicarbonato: passo 3/3.	Cataliza a ciclização reversível de aspartato de carbamoílo para di-hidroorotato.	N/A
4.1.1.23	Orotidine 5'-phosphate decarboxylase	pyrF PA2876	4,2E-85	Metabolismo de pirimidina; Biossíntese UMP via via de novo; UMP do orotate: passo 2/2.	Cataliza a descarboxilação do 5'-monofosfato de orotidina (OMP) para o 5'-monofosfato de uridina (UMP).	N/A
4.2.1.59	3-hydroxydecanoyl-[acyl-carrier-protein] dehydratase	fabA PA1610	2,9E-81	Metabolismo lipídico; biossíntese de ácidos graxos.	Necessário para a introdução da insaturação cis nos ácidos gordos. Cataliza a desidratação de (3R) -3-hidroxidecanoil-ACP para E- (2) -denetil-ACP e depois a sua isomerização em Z- (3) -decenoílo-ACP. Pode catalisar a reação de desidratase para beta-hidroxiacil-ACP com comprimentos de	Citoplasma

					cadeia saturados até 16: 0, sendo mais ativos no comprimento da cadeia intermediária (por similaridade).	
4.6.1.12	2-C-methyl-D-erythritol 2,4-cyclodiphosphate synthase	ispF PA3627	2,9E-69	Biossíntese isoprenóidea; biossíntese de isopentenil difosfato via via DXP; difosfato de isopentenilo a partir de 5-fosfato de 1-desoxi-D-xilulose: passo 4/6.	Envolvido na biossíntese de difosfato de isopentenilo (IPP) e difosfato de dimetilalilo (DMAPP), dois principais blocos de construção de compostos isoprenóides. Catalisa a conversão do 2-fosfato de 4-difosfocitil-2-C-metil-D-eritritol (CDP-ME2P) em 2,4-ciclodifosfato de 2-C-metil-D-eritritol (ME-CPP) com uma libertação correspondente de 5-monofosfato de citidina (CMP).	N/A

Tabela 27 - Lista de potenciais alvos terapêuticos encontrados para o modelo PAO1 2008 (APÊNDICE C).

EC	Proteína	Gene	E-Value	Via	Função	Localização
1.1.1.85	3-isopropylmalate dehydrogenase	leuB PA3118	1,1E-140	Biossíntese de aminoácidos; Biossíntese de L-leucina; L-leucina a partir de 3-metil-2-oxobutanoato: passo 3/4.	Cataliza a oxidação de 3-carboxi-2-hidroxi-4-metilpentanoato (3-isopropilmalato) para 3-carboxi-4-metil-2-oxopentanoato. O produto descarboxila em 4-metil-2 oxopentanoato.	Citoplasma.
2.2.1.6	Acetolactate synthase, catabolic	PA4180	1,8E-95	N/A	N/A	N/A
2.4.2.19	Nicotinate-nucleotide pyrophosphorylase [carboxylating]	nadC PA4524	5,9E-96	Biossíntese de cofactores; Bensíntese de NAD (+); D-ribonucleótido de nicotinato a partir de quinolinato: passo 1/1.	Envolvido no catabolismo do ácido quinolínico (QA).	N/A
4.2.1.20	Tryptophan synthase alpha/beta chain	trpA PA0035 / trpB PA0036	4,5E-53 / 2E-149	Biossíntese de aminoácidos; Biossíntese de L-triptofano; L-triptofano do corisato: passo 5/5.	A subunidade alfa é responsável pela clivagem de aldol do fosfato de indoleglicerol ao indole e gliceraldeído 3-fosfato / A subunidade beta é responsável pela síntese de L-triptofano a partir de indol e L-serina.	N/A
5.1.3.13	dTDP-4-dehydrorhamnose 3,5-epimerase	rmlC PA5164	9,7E-36 / 1,1E-132 / 1,2E-75	Biossíntese de carboidratos; biossíntese dTDP-L-Rhamnose. Biogênese da membrana externa bacteriana; biossíntese de lipopolissacarídeos.	Cataliza a epimerização das posições de C3 'e C5' de dTDP-6-desoxi-D-xilo-4-hexulose, formando dTDP-6-desoxi-L-liso-4-hexulose.	N/A
5.1.3.20	ADP-L-glycero-D-manno-	hldD rfaD	9,1E-103	Biossíntese de	Cataliza a interconversão	N/A

	heptose-6-epimerase	PA3337		açúcares nucleotídicos; Biossíntese de ADP-L-glicero-beta-D-manno-heptose; ADP-L-glicero-beta-D-manno-heptose a partir de D-glicero-beta-D-manno-heptose 7-fosfato: passo 4/4. Biogênese da membrana externa bacteriana; Biossíntese do núcleo LPS.	entre ADP-D-glicero-beta-D-manno-heptose e ADP-L-glicero-beta-D-manno-heptose através de uma epimerização no carbono 6 da heptose.	
5.3.1.24	N-(5'-phosphoribosyl)anthranilate isomerase	trpF PA3113	2,6E-38	Biossíntese de aminoácidos; Biossíntese de L-triptofano; L-triptofano do corismato: passo 3/5.	N/A	N/A
5.3.1.28	Phosphoheptose isomerase	gmhA PA4425	1,2E-132	Biossíntese de carboidratos; Biossíntese de 7-fosfato de D-glicero-D-manno-heptose; D-glicero-alfa-D-manno-heptose 7-fosfato e D-glicero-beta-D-manno-heptose 7-fosfato a partir de 7-fosfato de sedoheptulose: passo 1/1. Biogênese da membrana externa bacteriana; Biossíntese do núcleo LPS.	Cataliza a isomerização de 7-fosfato de sedoheptulose em 7-fosfato de D-glicero-D-manno-heptose.	Citoplasma
5.4.99.5	P-protein / Secreted chorismate mutase	pheA PA3166 /aroQ PA5184	5,6E-46 / 0,000000000	Biossíntese de aminoácidos;	Cataliza o rearranjo Claisen do chorismate para	Citoplasma / Periplasma

			0026	Biossíntese de L-fenilalanina; fenilpiruvato do pré-fenado: passo 1/1. Biossíntese intermediária metabólica; Prevenção da biossíntese; Prefenate from chorismate: passo 1/1.	prefenate. A presença conjunta desta enzima em conjunto com ciclohexadienil desidratase e aminotransferase aromática no compartimento periplásmico compreende uma via completa de corista de três passos para a fenilalanina e explica o chamado caminho de transbordamento escondido para a fenilalanina em P. aeruginosa, em que duas rotas possíveis para ela existe, ou seja, via fenilpiruvato ou L-arogenato.	

Tabela 28 - Lista de potenciais alvos terapêuticos encontrados para o modelo PAO1 2017 (APÊNDICE C).

EC	Proteína	Gene	E-Value	Via	Funcao	Localizacao
1.1.1.267	1-deoxy-D-xylulose 5-phosphate reductoisomerase	dxr PA3650	4,5E-147	Biossíntese isoprenóidea; biossíntese de isopentenil difosfato via via DXP; difosfato de isopentenilo a partir de 1-desoxi-D-xilulose 5-fosfato: passo 1/6.	Cataliza o rearranjo dependente de NADP e a redução de 1-desoxi-D-xilulose-5-fosfato (DXP) para 2-C-metil-D-eritritol 4-fosfato (MEP).	N/A
2.7.4.3	Adenylate kinase	adk PA3686	6,3E-90	Metabolismo de purina; Biossíntese de AMP via caminho de	Cataliza a transferência reversível do grupo fosfato terminal entre ATP e AMP. Reproduz um papel importante na homeostase de energia	Citoplasma

				salvamento; AMP da ADP: passo 1/1.	celular e no metabolismo de nucleótidos de adenina.	
4.1.3.40	Chorismate--pyruvate lyase	ubiC PA5357	1E-12	Biossíntese de cofactores; biossíntese de ubiquinona.	Remove o grupo piruvilo do corismato, com aromatização concomitante do anel, para fornecer 4-hidroxibenzoato (4HB) para a via da ubiquinona.	Citoplasma

Tabela 29 - Lista de potenciais alvos terapêuticos encontrados para o modelo CCBH4851 (APÊNDICE C).

EC	Proteína	Gene	E-Value	Via	Função	Localização
1.2.5.1	Pyruvate dehydrogenase [ubiquinone]	poxB PA5297	1,3E-169	N/A	N/A	N/A
1.4.1.20	L-phenylalanine dehydrogenase	ldh PA3418	5,5E-55	N/A	N/A	N/A
1.8.1.9	Thioredoxin reductase	trxB1 PA2616, trxB2 PA0849	4E-160	N/A	N/A	N/A
2.1.2.2	Phosphoribosylglycinamide formyltransferase	purN PA0944	1,5E-81	Metabolismo de purina; Biossíntese de IMP via via de novo; N (2) -formil-N (1) - (5-fosfo-D-ribosil) glicinamida a partir de N (1) - (5-fosfo-D-ribosil) glicinamida (via 10-formil THF): passo 1/1 .	Catalisa a transferência de um grupo formilo de 10-formiltetra-hidrofolato para 5-fosfo-ribosil-glicinamida (GAR), produzindo 5-fosfo-ribosil-N-formilglicinamida (FGAR) e tetrahidrofolato.	N/A
2.7.6.3	2-amino-4-hydroxy-6-hydroxymethyldihydropteridine pyrophosphokinase	folK PA4728	3,2E-45	Biossíntese de cofactores; biossíntese de tetrahidrofolato; Difosfato de 2-	N/A	N/A

				amino-4-hidroxi-6-hidroximetil-7,8-dihidropteridina a partir de 7,8-dihidroneopterina trifosfato: passo 4/4.		
2.8.1.6	Biotin synthase	bioB PA0500	9E-170	Biossíntese de cofactores; biossíntese de biotina; biotina a partir de 7,8-diaminononanoato: passo 2/2.	Cataliza a conversão de detobiotina (DTB) em biotina pela inserção de um átomo de enxofre em detobiotina através de um mecanismo baseado em radicais.	N/A
3.5.3.23	N-succinylarginine dihydrolase	astB aruB PA0899	6E-174	Degradação de aminoácidos; Degradação de L-arginina via via AST; L-glutamato e succinato da L-arginina: passo 2/5.	Cataliza a hidrólise de N (2) - succinilarginina em N (2) - succinilornitina, amônia e CO (2).	N/A
3.6.1.23	Deoxyuridine 5'-triphosphate nucleotidohydrolase	dut PA5321	7,6E-71	Metabolismo de pirimidina; biossíntese de DUMP; DUMP de dCTP (rota DUTP): etapa 2/2.	Esta enzima está envolvida no metabolismo de nucleótidos: produz DUMP, o precursor imediato de nucleótidos de timidina e diminui a concentração intracelular de	N/A

					dUTP, de modo que o uracilo não pode ser incorporado no DNA..	
3.6.1.55	8-oxo-dGTP diphosphatase	PA4400	5,8E-22	N/A	N/A	N/A
6.3.1.5	NH(3)-dependent NAD(+) synthetase	nadE PA4920	2E-71 / 3,6E-71	Biossíntese de cofactores; Bensíntese de NAD (+); NAD (+) de deamido-NAD (+) (via amoníaca): passo 1/1.	Cataliza a amidação dependente de ATP do deamido-NAD para formar NAD. Usa amônia como fonte de nitrogênio.	N/A