Universidade de Brasília

Faculdade de Tecnologia

Departamento de Engenharia de Produção

Análise dos métodos de resolução de problemas de programação inteira: Um estudo de caso no problema de alocação de servidores na ANAC

Por,

GABRIEL DURÃES GUTH

Brasília – DF

Novembro, 2021

GABRIEL DURÃES GUTH

Análise dos métodos de resolução de problemas de programação inteira: Um estudo de caso no problema de alocação de servidores na ANAC

Monografia apresentada ao Departamento de Engenharia de Produção como requisito parcial à obtenção do título de Engenheiro de Produção.

Professor Orientador: Doutor, Reinaldo Crispiniano Garcia

Professor Coorientador: Doutora, Silvia Araújo dos Reis

Brasília – DF

2021

Guth, Gabriel Durães.
Análise dos métodos de resolução de problemas de programação inteira: Um estudo de caso no problema de alocação de servidores na ANAC / Gabriel Durães Guth – Brasília, 2021.
116 f. : il.

Monografia (bacharelado) – Universidade de Brasília, Departamento de Engenharia de Produção, 2021.
Orientador: Prof. Dr. Reinaldo Crispiniano Garcia, Departamento de Engenharia de Produção.
Corientadora: Prof. Dra. Silvia Araújo dos Reis, Departamento de Administração.

1. Programação Inteira. 2.Python. 3. ANAC. 4. Modelo de Apoio à Decisão. 5. Revisão Sistemática de Literatura. I. Análise dos métodos de resolução de problemas de programação inteira: Um estudo de caso no problema de alocação de servidores na ANAC.

GABRIEL DURÃES GUTH

Análise dos métodos de resolução de problemas de programação inteira: Um estudo de caso no problema de alocação de servidores na ANAC

A Comissão Examinadora, abaixo identificada, aprova o Trabalho de Conclusão do Curso de Engenharia de Produção da Universidade de Brasília do aluno

GABRIEL DURÃES GUTH

Doutor, Reinaldo Crispiniano Garcia,
Professor-Orientador

Doutora, Silvia Araújo dos Reis
Professor-Coorientador

Doutor, Victor Rafael Rezende Celestino
Professor-Orientador

Brasília, 04 de Novembro de 2021

Dedico este trabalho aos meus pais, Avelange e Guilherme, que não mediram esforços em garantir uma educação de qualidade e me permitirem concretizar meus sonhos. Dedico especialmente ao meu pai, que hoje não está mais presente conosco, mas que sentiria grande orgulho em formar um filho engenheiro.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais que sempre me incentivaram a estudar, proporcionando um ensino de qualidade durante toda minha trajetória e me apoiando na realização da graduação em engenharia de produção. Obrigado por acreditarem em mim e me permitirem ser a pessoa que sou hoje, todo o caráter e conquista que tive foi graças a vocês. Aos meus familiares que me acompanharam durante toda a vida e graduação e que de alguma forma me apoiaram e torceram para o meu sucesso. Obrigado por todo apoio nesse processo.

Ao meu orientador Reinaldo Crispiniano, à minha coorientadora Silvia Araújo por me apoiarem no desenvolvimento desta pesquisa, tendo confiança, dedicação e paciência no processo de elevar a qualidade do trabalho.

Aos professores Reinaldo Crispiniano, Silvia Araújo e Victor Celestino por todos os ensinamentos e conhecimentos repassados durante a graduação, principalmente por me mostrarem o mundo da Pesquisa Operacional e me fazerem apaixonar por essa área. Vocês são responsáveis por hoje eu ter o sonho de um dia me tornar professor.

À Agência Nacional de Aviação Civil (ANAC) pelos anos de trabalho e possibilidade de desenvolvimento do projeto Safety Oversight (ANAC) junto aos meus amigos Beatriz Simões, Camila Mayerhofer, Hélio Santana, Henrique Daminelli, Hugo Frota, Ingrid Silva, Olivia Ziller e Thiago Dias, e professores Silvia Araújo e Victor Celestino, cujo trabalho permitiu o desenvolvimento deste projeto de graduação.

A todos os professores que tive durante o meu ensino fundamental, médio e faculdade, que me fizeram aprender os diversos conteúdos e aprendizados para vida. Em especial, a professora Iracema, que me fez gostar e amar o mundo dos números e me incentivou ao caminho das exatas.

Ao meu amigo mais antigo, André. Aos meus amigos do ensino fundamental, Bernardo, Couto e Matheus. Aos meus amigos do ensino médio, Danna, Victor, Vitor, Marco, Gabriel, Rafael. Aos meus colegas e amigos de curso, Alexandre, Camila, Eduardo, Gabriel, Guilherme, Igor, Isabela, Rafaela, Vanessa, Yasser. Aos meus amigos de jogos. À minha namorada, Letícia. Obrigado por acompanharem o meu desenvolvimento e por fazerem parte da minha vida, vocês são especiais para mim. Por fim, agradeço imensamente a todos que tiveram uma contribuição na minha vida.

RESUMO

A evolução das tecnologias e sistemas computacionais tem permitido maior utilização de modelos de apoio à decisão formulados através da programação matemática. Dentre eles, destacam-se os que utilizam da Programação Inteira (PI), que por terem seu processo de solução mais complexo, demandam maior esforço computacional e de tempo para problemas de grande porte. Tendo em vista a importância desses tipos de modelos e a diversidade de métodos e ferramentas utilizadas na resolução destes, essa pesquisa buscou identificar os métodos e técnicas adotados na solução de modelos de Programação Inteira, por meio de uma revisão sistemática da literatura, e realizar uma comparação entre a utilização de alguns métodos e ferramentas para solução de um modelo de grande porte. A pesquisa utilizou o estudo desenvolvido no projeto *Safety Oversight* na Agência Nacional de Aviação Civil (ANAC), que consiste em um problema de alocação dos servidores lotados em diferentes estados do Brasil e que realizam fiscalizações nos diversos aeroportos, levando em consideração o tempo, custo e atividade. Para isso, foram utilizados dois modelos implementados na linguagem Python com auxílio da biblioteca “Pyomo” e dos *solvers* CPLEX e Gurobi, e testadas 6 instâncias. Os resultados dos testes foram consolidados em uma tabela apresentando o tempo de execução e a Função Objetivo (FO) encontrados para cada instância, comparando o algoritmo implementado em Python com os modelos desenvolvidos na literatura nos *softwares* LINGO, AIMMS, Julia e heurísticas (*Tabu Search, Simulated Annealing e Hibrid*). A partir disso, verificou-se que ambos os softwares testados são viáveis e capazes de solucionar a maioria dos cenários no problema de programação inteira da alocação de servidores da ANAC, minimizando o custo com passagem aérea nas fiscalizações. Por fim, destaca-se o desempenho da linguagem de código aberto Python com o *solver* Gurobi por ter sido capaz de solucionar todas as instâncias testadas, obtendo o melhor desempenho em duas delas comparado aos outros softwares

Palavras-chave: Programação Inteira, Python, ANAC, Modelo de Apoio a Decisão, Revisão Sistemática de Literatura

ABSTRACT

The evolution of technologies and computational systems has allowed greater use of decision support models formulated through mathematical programming. Among them, those that use Integer Programming (IP) stand out, as their solution process is more complex, requiring greater computational effort and time for large problems. Considering the importance of these types of models and the diversity of methods and tools used to solve them, this research sought to identify the methods and techniques adopted to solve Integer Programming models, through a systematic review of the literature, and to carry out a comparison between the use of some methods and tools to implement a large model. Thus, to compare available softwares taking into account the solution time, the research used the study developed in the Safety Oversight Project at the National Civil Aviation Agency (ANAC), which consists of a problem of allocation of servants located in different Brazilian States carrying out inspections at the various airports, considering time, cost and activity. Two models are then implemented in Python language with the aid of the “Pyomo” library and CPLEX and Gurobi solvers, and 6 scenarios were tested. The test results were consolidated in a table showing the execution time and the Objective Function (OF) found for each instance, comparing the algorithm implemented in Python with the models developed in the literature in LINGO, AIMMS, Julia and heuristics (Tabu Search, Simulated Annealing and Hybrid). It is then observed that both tested softwares are viable and capable of solving most scenarios in the entire programming problem of ANAC's servants allocation, minimizing the cost of airfare in inspections. Finally, the performance of the open source language Python with the Gurobi solver stands out for being able to solve all implemented scenarios, obtaining the best performance in two of them compared to the other software.

Keywords: Integer Programming, Python, ANAC, Decision Support Model, Systematic Literature Review

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ACO – *Colony Optimisation*

AIMMS – *Advanced Integrated Multidimensional Modeling Software*

ANAC – Agência Nacional de Aviação Civil

B&P – *Branch-and-Price*

CD – *Chronological Decomposition*

EA – *Evolutionary Algorithm*

EUA – Estados Unidos da América

F&O – *Fix-and-Optimize*

FO – Função Objetivo

GA – *Genectic Algorithm*

GAMS – *General Algebraic Modeling System*

GWO – *Grey Wolf Optimizer*

HCEA – *Hybrid Cross Entropy Algorithm*

IA – Inteligência Artificial

ICA – *Imperialist Competitive Algorithm*

IFRH – *Incremental Fix and Release Heuristic*

IWO – *Invasive Weed Optimization*

JuMP – *Julia for Mathematical Programming*

LINGO – *Language for Intective General Optimizer*

LNS – L*arge Nerighborhood Search*

LSA – *Local Search Algorithm*

MA – *Memetic Algorithm*

MAE – Modelo de Alocação Estendido

MAR – Modelo de Alocação Resumido

MILP – *Mixed Integer Linear Programming*

NURAC – Núcleos Regionais de Aviação Civil

PI – Programação Inteira

PIM – Programação Inteira Mista

PL – Programação Linear

PNL – Programação Não Linear

PO – Pesquisa Operacional

PSO – *Particle Swarm Optimization*

PYOMO – *Python Optimization Modeling Objects*

R&F – *Relax-and-Fix*

RSA – *Restricted Simulated Annealing*

SA – *Simulated Annealing*

SAICA – *Self-Adaptive Imperialist Competitive Algorithm*

SAR – Superintendência de Aeronavegabilidade

SFI – Superintendência de Ação Fiscal

SIA – Superintendência de Infraestrutura Aeroportuária

SPO – Superintendência de Padrões Operacionais

TS – *Tabu Search*

UnB – Universidade de Brasília

VBA – *Visual Basic for Applications*

VNS – *Variable Neighborhood Search*

WCA – *Water Cycle Algorithm*

WSA – *Weighted Superposition Attraction*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

Problemas de otimização sempre existiram na humanidade, seja para utilização de recursos disponíveis ou economia de tempo, distribuição de produtos, alocação de pessoas, entre outros. Com o avanço dos estudos para encontrar soluções desses problemas, surgiu a Pesquisa Operacional (PO) nos primórdios da Segunda Guerra Mundial, sendo utilizada pelos militares para alocar recursos escassos de forma eficiente. Após esse período, houve um crescimento industrial mundial e a PO começou a ser utilizada por diversas organizações nos setores comerciais, industrial e governamental (HILLIER; LIEBERMAN, 2013).

A solução de um problema de Programação Matemática, abordada na Pesquisa Operacional, consiste na elaboração de um modelo matemático capaz de encontrar o valor ótimo para uma função-objetivo, sendo ela de maximização ou minimização, considerando todas as variáveis e restrições aplicadas ao modelo. Dessa forma, as organizações começaram a enxergar uma grande economia de recurso, principalmente financeiro, com a utilização de modelos de Programação Matemática no processo de apoio à decisão da gestão.

Além disso, ao mesmo passo do aumento do número e complexidade de problemas envolvendo otimização, ocorreu também uma expansão da capacidade computacional e de métodos para resolução de modelos complexos. Essa expansão permitiu o desenvolvimento de softwares especializados para modelar e solucionar problemas de otimização contendo diferentes algoritmos de otimização. Sendo o CPLEX um pacote de software de última geração amplamente utilizado em sistemas como MPL e Excel para solucionar problemas de PO. Já alguns outros sistemas como o LINGO apresentam solvers próprios (HILLIER; LIEBERMAN, 2013).

Por outro lado, observa-se também linguagens de programação utilizados na resolução desses tipos de problema. Dentre eles destacam-se linguagens de código aberto (*open source*) que permitem modificações e estudo de seus códigos por

qualquer usuário, tornando-se acessíveis para serem utilizadas na resolução de diversos modelos. Segundo a *Open Source Initiative* (2021), um software de código aberto deve apresentar redistribuição livre, código fonte, trabalhos relacionados, integridade do código fonte do autor, não possuir discriminação contra pessoas, grupos ou campos de empreendimento e licenças livres.

Assim, esses softwares permitem uma grande vantagem competitiva em relação a implementação de modelos de programação matemática, devido à quantidade de bibliotecas e comunidades de discussão sobre o assunto, além de serem gratuitos para utilização. Leno (2014) destaca que o Python está sob a licença *Python Software Foundation License*, sendo uma linguagem livre e de código aberto. Além de ser de alto nível, acessível e de desenvolvimento rápido, ele possui diversas bibliotecas de auxílio disponibilizadas gratuitamente. Outras linguagens de programação de código aberto utilizadas para desenvolvimento de modelos de PO são C++, julia, C#, C e java (GUTH; REIS, 2021)

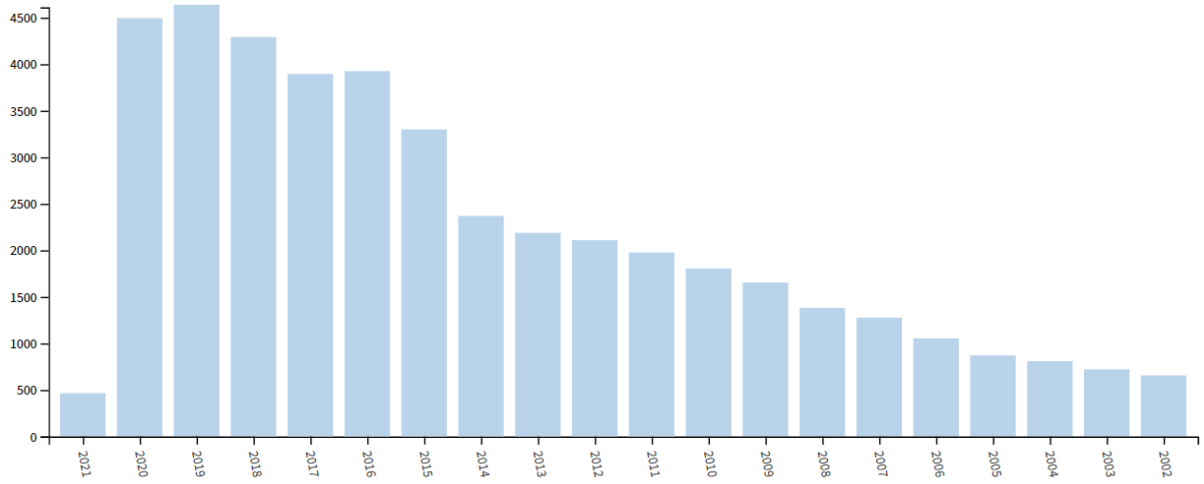
Segundo Sperandio e Coelho (2010), os algoritmos para solução de modelos mais complexos de Programação Inteira (PI) não apresentam a solução em um tempo polinomial determinado, ou seja, não é possível determinar o tempo necessário para encontrar a solução. Isso acontece porque problemas de programação inteira são modelos em que as variáveis assumem totalmente ou parcialmente valores inteiros, geralmente utilizados quando é necessário encontrar a solução exata dentro de um conjunto finito de soluções ou quando se trata de decisões binárias. Dessa forma, para resolução desses problemas em tempos hábeis, alguns algoritmos desenvolvidos com base em meta-heurística e heurísticas baseada em modelos (*matheurístics)* têm alcançado resultados notórios, frequentemente apresentando soluções razoáveis em curto espaço de tempo, mas não garantindo a otimização (FONSECA et al., 2017).

A partir do exposto, esse trabalho apresenta um levantamento dos métodos de solução de problemas de Programação Inteira (PI) de grande complexidade e aplica uma linguagem de programação de código aberto para a resolução de um problema envolvendo especificamente o ambiente de trabalho relacionado à administração pública, mediante um estudo de caso em um modelo de PI. Dessa forma, busca-se realizar uma comparação entre o desempenho e resultados apresentados pelos diferentes softwares e técnicas no estudo.

1.2 Formulação do problema

O crescimento da Pesquisa Operacional ocorre em conjunto com a evolução da capacidade tecnológica dos computadores, sendo cada vez mais viáveis o desenvolvimento de problemas complexos envolvendo otimização. Dessa forma, a continuidade de pesquisas acadêmicas nessa área permite uma constante atualização dos problemas abordados e soluções encontradas para resolvê-los, a partir da aplicação de algoritmos e métodos já conhecidos ou surgimento de novos. O que pode ser observado a partir da evolução do número de publicações relacionadas a PO ao passar dos anos (Figura 1), alcançando o impressionante número de 4466 publicações em 2020.

Figura 1 - Evolução das publicações de Pesquisa Operacional (PO)



Fonte: *Web of Science*

Porém, esse tópico ainda é pouco abordado quando se considera a produção científica nacional e internacional sobre PO relacionadas à administração pública. Santos, Simonetto e Ferreira (2017) realizaram um estudo sobre a produção científica internacional em PO atrelada ao serviço público e constataram que somente 0,46% dos autores que produziram conteúdo publicado de PO eram brasileiros. Além disso, menos de 2% das publicações internacionais publicadas entre 1993 e 2013 estão relacionadas à área de Gestão Pública.

Grande parte dos estudos em PO são para o desenvolvimento de problemas de Programação Matemática determinísticos, que podem ser divididos em

Programação Linear (PL), Programação Inteira (PI), Programação Inteira Mista (PIM), Programação Não Linear (PNL) e suas combinações. Contudo, Longo (2004) aponta que a maioria dos softwares disponíveis até o momento, apresenta dificuldades para encontrar, em tempos computacionais aceitáveis, a solução ótima para problemas de PI e PNL de grande porte.

Um problema é classificado como Programação Inteira (PI) quando todas as variáveis de decisão são discretas (os valores representam um conjunto finito ou enumerável de números). Quando parte das variáveis de decisão é discreta e as demais são contínuas (assumem valores em um intervalo de números reais), o modelo é chamado de Programação Inteira Mista (PIM) (BELFIORE; FÁVERO, 2013).

Embora nos problemas de Programação Inteira exista um número bem menor de soluções a serem consideradas quando comparados aos de Programação Linear, isso não garante que o problema seja prontamente solucionável. Considerando por exemplo o caso de Programação Inteira Binária, com n variáveis, há 2^n soluções a serem consideradas, cada vez que n for incrementado em 1, o número de soluções dobra. Dessa maneira, até mesmo os computadores mais rápidos com maior capacidade computacional podem ser incapazes de realizar uma enumeração exaustiva para verificar se cada uma das soluções é viável ou não e, em caso positivo calcular o valor da solução objetivo (HILLIER; LIEBERMAN, 2013).

A resolução dos modelos de Programação Inteira é mais complexa, já que as soluções do modelo podem estar em qualquer parte do espaço factível, não somente nos vértices, como nos modelos de Programação Linear. Existem bons algoritmos desenvolvidos para a busca de solução ótima, entre eles o Branch and Bound e Branch and Cut (HILLIER; LIEBERMAN, 2013).

Para obtenção de solução ótima, também são importantes métodos de relaxamento, que são utilizados para obter informação na resposta final do problema inteiro, onde eles alteram as restrições ou a função objetivo para aumentar o conjunto de soluções e identificar algumas propriedades. Esses métodos de relaxamento comumente categorizados como heurística variam de acordo com o tipo de problema que está sendo analisado, dentre os quais estão o relaxamento lagrangeano, busca tabu, algoritmos de intercâmbio, recozimento simulado.

Além disso, diversos problemas só podem ser tratados com Programação Inteira ou Programação Inteira Mista devido à necessidade de encontrar um valor inteiro para as variáveis de decisão. Alguns exemplos são: problemas com a utilização de equipamentos ou pessoas, no qual é necessário que a variável X assuma um valor inteiro, não podendo ser ele decimal; questões sobre tamanhos de lote, no qual utiliza-se geralmente uma quantidade mínima de produtos produzidos e busca-se a quantidade exata para compor o lote; decisões de “sim ou não”, com as variáveis assumindo valores binários; entre outros.

Porém, devido a quantidade de métodos de solução e a complexidade dos modelos atuais exigindo a utilização da PI, é difícil encontrar qual algoritmo e sistema são os mais eficientes na busca da solução ótima ou próxima do ótimo em um tempo razoável a ser utilizado para resolução de problemas de Programação Inteira de grande porte. Por outro lado, verifica-se o crescimento do número de sistemas disponíveis, sejam de código aberto ou licenciados, para resolução desses tipos de problema.

Tendo em vista o exposto acima, com o aumento do uso de modelos de apoio à decisão, ainda há uma carência aos estudos desenvolvidos por brasileiros e aplicados ao setor público. Entretanto, o mercado está cada vez mais competitivo, exigindo a necessidade de resolução de modelos de PI com certa eficiência e com menor custo. Dada quantidade e complexidade dos diversos métodos para solução de problemas envolvendo Programação Inteira, esse trabalho apresenta, assim, o seguinte problema de pesquisa: É viável a utilização de linguagem de programação de código aberto para resolução de problemas de Programação Inteira de grande porte?

1.3 Objetivo Geral

Analisar a viabilidade da aplicação de linguagem de programação de código aberto na resolução de um problema de Programação Inteira (PI) de grande porte aplicado ao contexto da Administração Pública.

1.4 Objetivos Específicos

A fim de alcançar o objetivo geral, foram definidos os seguintes objetivos específicos:

1) Buscar os principais métodos, algoritmos e tecnologias adotados na resolução de problemas de Programação Inteira de grande porte, por meio de uma revisão sistemática de literatura;
2) Programar e resolver o problema apresentado no estudo de caso em órgão público no Brasil com uma técnica selecionada;
3) Testar e gerar as soluções para os modelos já desenvolvidos para o estudo de caso;
4) Realizar uma comparação da tecnologia escolhida com os modelos já implementados;
5) Elaborar uma tabela apresentando os resultados obtidos;

1.5 Justificativa

Dado o crescimento industrial e das organizações, o surgimento de novas empresas e ramos da tecnologia são uma oportunidade de viabilizar a implantação de Pesquisa Operacional em diversos novos ambientes. Assim, pesquisas acadêmicas com enfoque em criação de modelos matemáticos de otimização para solução de problemas reais de grande porte apresentam grande viabilidade na gestão das organizações.

Além disso, projetos na administração pública e privada com temáticas de Pesquisa Operacional são bem visados, pois apresentam uma abordagem quantitativa de resultados e mostra na prática a eficiência de dinheiro, tempo e pessoas obtida com a utilização desses modelos na tomada de decisão das empresas. Dessa forma, os gestores conseguem tomar melhores decisões de acordo com o tipo de necessidade da empresa, otimizando a distribuição de recursos, principalmente financeiros.

O crescimento de publicações em PO colabora para o desenvolvimento dessa pesquisa, pois muitos dos problemas de grande complexidade envolvem a necessidade de utilização de PI para resolução. Nesse sentido, ainda há bastante dúvida na escolha de um método eficiente na solução desses problemas, pois existe uma lacuna na literatura científica com a falta de artigos e pesquisas que compilem todos os algoritmos para resolução de modelos de PI de grande complexidade.

De outra forma, no estudo realizado por Dias (2019) constata-se que o serviço público brasileiro carece de utilização e publicação de artigos científicos abordando a temática de Pesquisa Operacional. No trabalho verificou-se a quantidade ínfima de publicações realizadas por brasileiros aplicadas à Administração Pública. Assim encontra-se uma grande oportunidade de abordagem nessa pesquisa com a apresentação de métodos eficientes para resolução de problemas de PI com grande complexidade na Administração Pública.

Contudo, para testagem do método e definição de eficiência é necessária uma comparação relacionando o número de variáveis, tempo de execução, exigência computacional, entre outros fatores para os diversos sistemas disponíveis para solução de problemas de PO. Portanto, será realizada uma comparação no modelo apresentado por um estudo de caso na Administração Pública com uma linguagem de programação de código aberto apresentada nesse trabalho.

2 REFERENCIAL TEÓRICO

A revisão de literatura é uma das partes mais importantes de uma pesquisa científica, sendo vital para o processo de investigação que envolve localizar, analisar, sintetizar e interpretar os trabalhos realizados anteriormente na área de estudo (BENTO, 2012). A partir dela se apresenta todo o conteúdo teórico necessário para contribuir com o entendimento dessa pesquisa, trazendo referências de trabalhos já publicados em livros, resumos, revistas e artigos científicos.

A revisão da literatura possibilita uma visão atual de informações importantes sobre a situação da pesquisa abordada e o conhecimento prévio do que já foi desenvolvido sobre o tema. Além disso, denomina-se referencial teórico o capítulo do trabalho com finalidade de realizar a revisão da literatura e deve estar em consonância com o tema do trabalho, uma vez que se desenvolve a narrativa de conhecimento sobre a pesquisa desde o passado até o momento presente (SILVA; MENEZES, 2005).

Dessa forma, a fundamentação teórica apresentada nesta seção aborda os conceitos elencados na problemática de pesquisa de modo a fundamentar o trabalho de pesquisa. Nesse sentido, serão abordados os tópicos de Pesquisa Operacional e abordagens das ferramentas de PO com foco nos modelos determinísticos. Em seguida, será aprofundado o tema de Programação Inteira com os principais métodos e algoritmos para resolução de problemas de grande porte com PI. Foram utilizadas as bases de dados do *Web of Science, ScientDirect, Scielo, Scopus, Google Scholar* como fontes de artigos, livros e publicações para construção dessa etapa.

2.1 Pesquisa Operacional

O surgimento do termo Pesquisa Operacional (PO) é atribuído às ações militares desenvolvidas nos primórdios da Segunda Guerra Mundial com a necessidade de alocar os escassos recursos de forma eficiente para diferentes operações militares. Diante dessa necessidade, os exércitos convocaram um grande número de cientistas para lidar com problemas táticos e estratégicos na realização de

pesquisas sobre operações. Após o sucesso do PO contribuindo com a vitória na guerra, despertou-se um interesse de aplicação fora do ambiente militar e que foi impulsionado pelo crescimento industrial nesse período, sendo assim, a PO foi introduzida em diversas organizações dos setores industriais, comercial e governamentais nesse período (HILLIER; LIEBERMAN, 2013).

Segundo Johnes (2015), os grupos de cientistas que atuaram com PO continuaram a receber apoio após a guerra, tendo alterado o foco para logística, planejamento e modelagem. Tornou-se aparente que a Pesquisa Operacional ganhou espaço em ambientes não-militares com a publicação de Gass (1994), que traz a aplicação de PO em ambientes públicos e compara com o setor privado, comprovando a aplicação de técnicas de PO aos negócios pós-guerra. Por outro lado, Horvarth (1995) comenta sobre a vantagem competitiva e o crescimento nos lucros nas empresas que aplicaram com sucesso as novas abordagens de PO nas operações incentivaram a utilização de PO como abordagem aceitável na resolução de problemas no ambiente de negócio.

A partir dessa época, a PO vem sendo utilizada em uma série de problemas envolvendo organizações públicas e privadas. Esse crescimento proporcionado pela aplicação de PO nas organizações conta atualmente com a atuação de muitos softwares e algoritmos que auxiliam na evolução das técnicas desenvolvidas, com resultados cada vez mais precisos e em menor tempo. Pesquisas realizadas em empresas que utilizam estas ferramentas mostram uma redução de custos em uma faixa de 1% e 5%, com casos em que essa melhoria pode chegar até a 15% (HILLIER; LIEBERMAN, 2013).

Em termos gerais, configura-se como Pesquisa Operacional a utilização de um método científico (modelagem matemática, estatística e algoritmos computacionais) que auxiliam o processo de tomada de decisão nas organizações. Sendo assim, a PO atua em diferentes contextos com uma aplicação multidisciplinar, envolvendo áreas da engenharia de produção, ciência da computação, matemática aplicada e gestão de negócios (BELFIORE; FÁVERO, 2013).

De acordo com Hillier e Lierberman (2013), Belfiore e Fávero (2013) e Araújo et. Al. (2018), para analisar e implementar as ferramentas de Pesquisa Operacional deve-se seguir algumas etapas para organização:

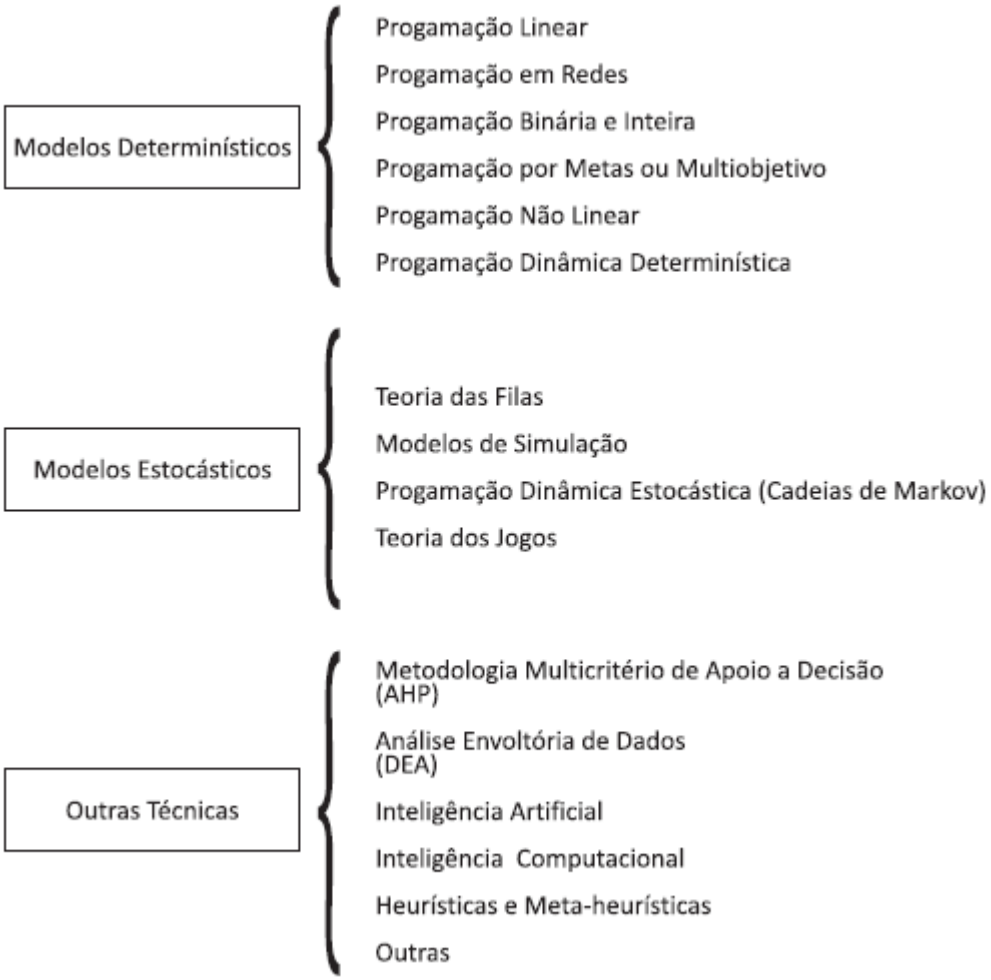
1) Definição do problema: a maior parte dos problemas é inicialmente descrita de forma vaga e insuficiente, dessa forma faz-se necessário desenvolver um enunciado bem definido do problema. A obtenção futura de resultados depende da formulação estruturada do problema, determinar os objetivos apropriados e restrições ao sistema, relacionar a área a ser estudada com os demais setores da organização, definição de planos alternativos e limites de tempo para tomada de decisão. Este processo é vital e essencial de ser feito com atenção, pois afeta criticamente a relevância das conclusões do estudo, podendo inviabilizar o projeto caso feito de forma incompleta;
2) Construção do modelo: consiste em formular um modelo matemático para representar o problema real. Um modelo é composto por três elementos principais: a) variáveis de decisão e parâmetros; b) função objetivo; c) restrições;
    a) Variáveis de decisão são valores desconhecidos ou incógnitas na equação, que serão determinados com a solução da modelagem. Elas são classificadas em contínuas, quando podem assumir qualquer valor em um intervalo de números reais; em discretas, quando podem assumir valores dentro de um conjunto finito ou quantidade enumerável, sendo aquelas proveniente de determinadas contagens; e em binárias, quando assumem apenas dois valores: 1 (quando está presente na variável a característica de interesse) ou 0 (caso contrário). Os parâmetros são valores fixos já conhecidos pelo problema;
    b) Função objetivo é uma função matemática que define o valor-alvo que se quer alcançar ou a qualidade da solução, de acordo com as variáveis de decisão e parâmetros. Ela pode ser de maximização (receita, lucro, nível de serviço, satisfação, entre outros) ou de minimização (custo, risco, erro, entre outros);
    c) Restrições são definidas como um conjunto de equações matemáticas de igualdade ou desigualdade (equações ou inequações, respectivamente) que as variáveis de decisão do modelo devem satisfazer. Elas são colocadas no modelo a fim de considerar as limitações físicas presentes no sistema, afetando diretamente os valores das variáveis de decisão;

3) Solução do modelo: essa fase objetiva implementar ferramentas matemáticas utilizando diversas técnicas de Pesquisa Operacional para buscar o algoritmo adequado para solução precisa, rápida e econômica do modelo proposto em que a rapidez se relaciona com a prática do algoritmo e a economia vincula-se às limitações de recursos humanos e computacionais;

4) Implementação do modelo: este procedimento é a fase crítica do estudo, pois é nela que surgem os resultados práticos e deve ser controlada e acompanhada por uma equipe responsável junto com o desenvolvedor de forma a detectar e corrigir possíveis mudanças no modelo, o que pode fazer com que partes do modelo seja reformulada. Dessa forma, certifica-se que as soluções estão sendo implementadas de acordo com o objetivo e prontas para serem entregues.

Com base no exposto, verifica-se a necessidade de planejamento, organização, levantamento e confiabilidade de dados, participação das áreas envolvidas e interessadas na criação de um modelo de PO. Além disso, para oferecer um modelo científico robusto que atenda as necessidades das organizações, a PO apresenta várias ferramentas que auxiliam as empresas no processo de tomada de decisão. Belfiore e Fávero (2013) a partir do trabalho realizado por Eom e Kim (2006) classifica essas ferramentas de Pesquisa Operacional em três grupos: modelos determinísticos, modelos estocásticos e outras técnicas (Figura 2).

Figura 2 - Ferramentas da Pesquisa Operacional



Fonte: Belfiore e Fávero (2013)

Os modelos estocásticos utilizam variáveis aleatórias que possuem ao menos uma das características de suas variáveis definida por meio de funções de probabilidade. Por tratar de previsões probabilísticas, os modelos estocásticos geram mais de uma solução e busca analisar diferentes cenários, não tendo a garantia de encontrar uma solução ótima. Eles geralmente são resolvidos por meio do levantamento de dados histórico e inserção em distribuições estatísticas com a utilização de métodos numéricos (programas de computador). Os principais modelos estocásticos são: teoria de filas, simulação, teoria dos jogos e programação dinâmica estocástica (BELFIORE; FÁVERO, 2013).

Os modelos determinísticos serão abordados na próxima seção desse trabalho, com foco nos métodos utilizados para resolução de Programação Inteira (PI) e suas variantes. Além disso, novas técnicas surgem na Pesquisa Operacional com o avanço do desenvolvimento computacional, incluindo a Inteligência Artificial (IA), a

metodologia multicritério de apoio à decisão, a inteligência computacional, as heurísticas, meta-heurísticas e matheurísticas, entre outras. Algumas dessas técnicas são complementares e podem auxiliar também na resolução de problemas determinísticos e estocásticos (BELFIORE; FÁVERO, 2013).

2.1.1 Modelos Determinísticos

Os modelos determinísticos são referentes àqueles em que todos os componentes, variáveis e restrições, envolvidos na sua formulação são conhecidos e constantes. Dessa forma, possui uma única solução exata, que geralmente é a solução ótima, excetuando-se os casos em que o problema gera múltiplas soluções de otimização. Os modelos determinísticos são resolvidos frequentemente por métodos analíticos, com a utilização de sistemas de equações que geram a solução ótima (BELFIORE; FÁVERO, 2013).

As principais categorias de problemas que envolvem programação matemática com modelos determinísticos são: programação linear, não linear, por metas ou multiobjetivo, em redes, dinâmica determinística, inteira e inteira mista. Essas ferramentas são utilizadas para resolução de problemas de diferentes assuntos, sendo crucial a definição do tipo de modelo para desenvolver adequadamente uma solução.

Os problemas de Programação Linear (PL) são definidos por esse nome, pois todas as funções matemáticas que envolvem o problema (função objetivo, restrições, variáveis) são representadas por funções lineares. Sendo considerada uma função linear aquele que envolve apenas constantes e variáveis de primeira ordem. Além disso, em um modelo de PL, as variáveis de decisão são contínuas, podendo assumir qualquer valor em um intervalo real (BELFIORE; FÁVERO, 2013). Para solução de problemas de PL, um procedimento altamente eficiente é o método simplex, capaz de solucionar até problemas de PL de enormes dimensões, buscando a solução ótima por meio de interações nos vértices do poliedro (HILLIER; LIEBERMAN, 2013).

Nos problemas de Programação Não Linear (PNL), ao menos uma de suas funções matemáticas (função objetivo, restrições, variáveis) do modelo descrito é

representada por uma função não linear. Geralmente são representadas por multiplicação ou divisão entre variáveis e variáveis exponenciais. Alguns exemplos de problemas que envolvem PNL são mix de produtos com elasticidade no preço, seleção de carteiras de ações com alto risco, problema de transporte com descontos por volume transportado, entre outros (BELFIORE; FÁVERO, 2013).

Na programação por metas ou multi-objetivo (*goal programming)* os problemas envolvem a solução de múltiplos objetivos ou metas. Assim, procura-se a minimização de desvios entre os multiobjectivos especificados conforme as variáveis de desvio. Já a programação dinâmica determinística é utilizada quando é possível decompor o problema principal em subproblemas, sendo útil para tomar uma sequência de decisões inter-relacionadas (BELFIORE; FÁVERO, 2013).

Os problemas de programação em redes são modelados em estruturas que se assemelham a grafos ou redes que consiste em diversos pontos (nós) que se conectam entre si, formando arcos (ligação entre nós). Alguns exemplos comuns de modelos de programação em redes são os problemas de transportes, de transbordo, do caminho mais curto, da árvore de expansão mínima, do fluxo máximo, entre outros (HILLIER; LIEBERMAN, 2013).

Dentro dos modelos determinísticos estão também os problemas de Programação Inteira (PI), que possuem todas as variáveis de decisão discretas. O modelo pode ser também de Programação Inteira Mista (PIM) quando apenas algumas variáveis assumem valores inteiros e as demais são contínuas. Dentre as diversas aplicações de problemas envolvendo PI, encontra-se alguns cuja decisão está entre somente duas escolhas possíveis, sim ou não. Nesse caso, as variáveis são chamadas de binárias e os problemas classificados em Programação Inteira Binária (PIB) (BELFIORE; FÁVERO, 2013).

Devido ao foco desse trabalho ser a busca de métodos e algoritmos eficientes para resolução de problemas de Programação Inteira de grande porte, na próxima seção serão detalhados os principais métodos encontrados na literatura para resolução de problemas de Programação Inteira de grande porte, sendo considerado os algoritmos e softwares utilizados, a complexidade do problema e o tempo para encontrar a solução ótima.

2.2 Revisão Sistemática

Este trabalho utilizou-se de uma revisão sistemática de literatura baseado no procedimento metodológico de Cronin, Ryan e Coughlan (2008) para encontrar o estado da arte de algoritmos, métodos e softwares utilizados na resolução de problemas de Programação Inteira em modelos de grande complexidade. Além disso, desejou-se identificar nos últimos 5 anos, os principais problemas modelados, os tipos de algoritmos, solvers e softwares utilizados.

A abordagem da revisão sistemática difere-se das revisões narrativas, pois exigem uma sequência de atividades com metodologia claramente explicitada, técnicas padronizadas e passíveis de reprodução. As principais características dessa revisão são fontes de busca abrangente, seleção de estudos primários com critérios uniformemente aplicados e avaliação rigorosa da amostra (LOPES; FRACOLLI, 2008).

Segundo o protocolo da revisão sistemática definido por Cronin, Ryan e Coughlan (2008), as etapas podem ser assim definidas: (a) formulação da questão de pesquisa; (b) estratégia de pesquisa e estabelecimento de critérios de inclusão e exclusão; (c) seleção e acesso de literatura; (d) avaliação da qualidade da literatura incluída na revisão; e, (e) análise, síntese e disseminação dos resultados.

Seguem como foram abordadas as etapas do protocolo da revisão sistemática da literatura para atingir o objetivo deste trabalho.

2.2.1 Etapas da Revisão Sistemática de Literatura

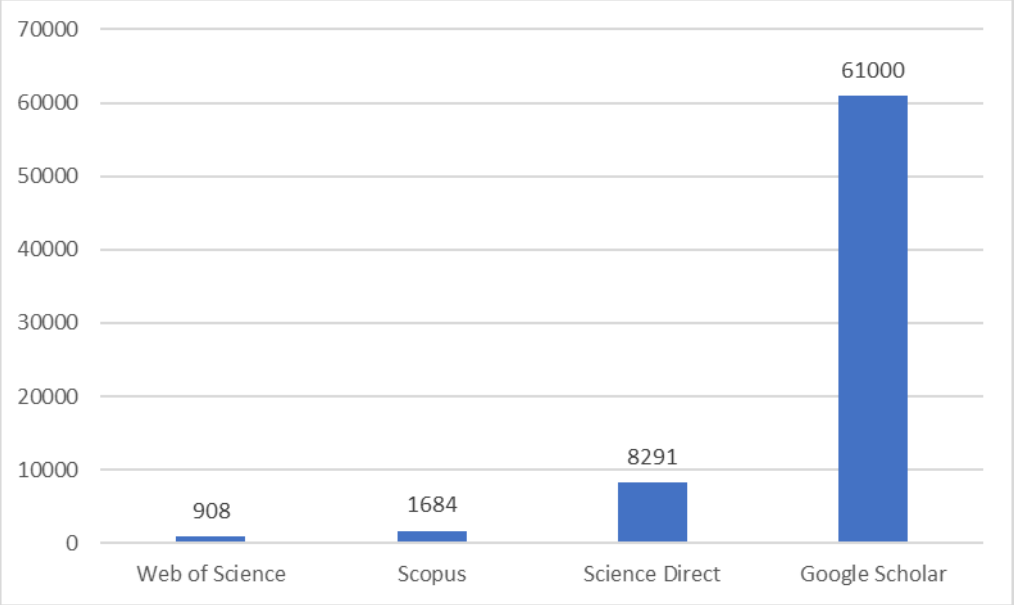
(a) Formulação da questão de pesquisa: Dada a complexidade dos modelos de PI e quantidade de métodos para solução, a questão a ser respondida é “Quais métodos têm sido utilizados na resolução de problemas de programação linear inteira determinístico de grande porte?”.

(b) Estratégia de pesquisa e estabelecimento de critérios de inclusão e exclusão: Para uma melhor abrangência acerca do tema e busca de métodos para

solução de problemas de PI de grande porte, foram estabelecidas as palavras-chave integer programming (programação inteira), case study (estudo de caso) e large, big, complex (grande complexidade). Como estratégia de pesquisa, foram utilizadas as palavras-chave em conjunto com os operadores lógicos AND (E) e OR (OU) da seguinte forma: (integer programming) AND ("case study") AND ("large" OR "big" OR "complex"). Considerou-se também o período de 2016 a 2021 para a análise.

(c) Seleção e acesso de literatura: Como o tema Problemas de Programação Inteira é multidisciplinar e pode ser encontrado em periódicos de diversas áreas como em Análise de Decisões, Administração, Computação, Engenharias, Matemática, entre outras; A quantidade de trabalhos verificados ao pesquisar os termos de pesquisa em todo o período histórico de publicação pode ser verificada na Figura 3.

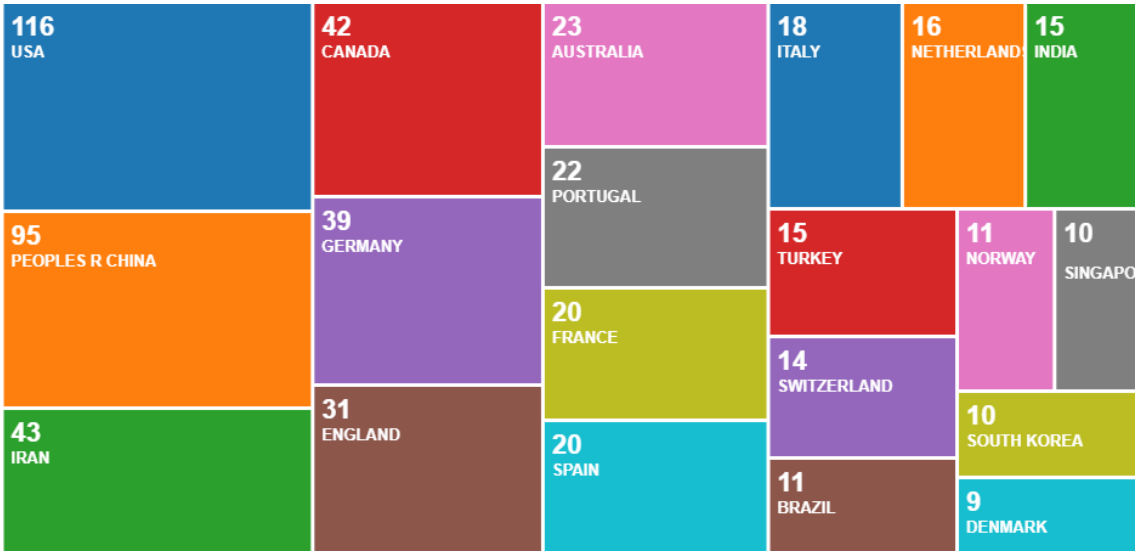
Figura 3 - Resultado da pesquisa por base



Fonte: Autor

Apesar da quantidade de artigos encontrados ser menor, definiu-se a pesquisa com referência na base científica Web of Science, da Clarivate Analytics, que é multidisciplinar, constituída por periódicos nacionais e internacionais com alto fator de impacto, além de possuir um mecanismo de busca e filtros avançados. Todos os artigos aceitos nessa base passam por uma avaliação editorial de qualidade, na qual se verifica a validade das declarações, relevância do conteúdo, citações apropriadas, revisão por pares, entre outras. Para o período selecionado, 2016 a 2021, obteve-se 526 resultados com a distribuição de publicações por país ilustrado na Figura 4.

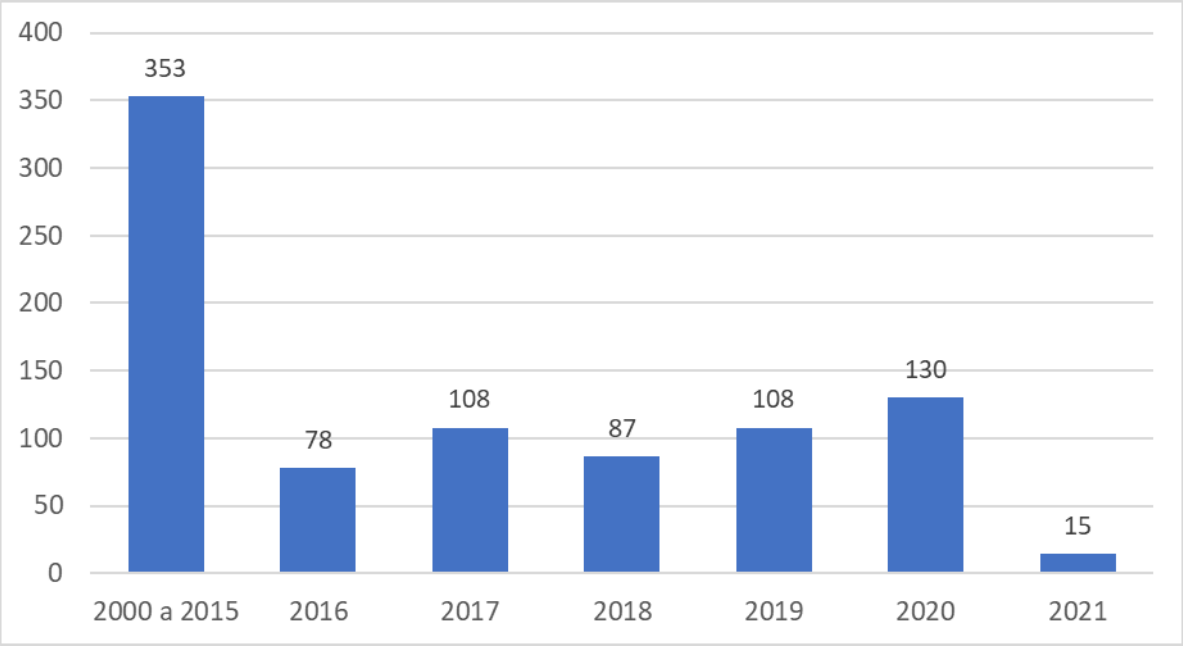
Figura 4 - Quantidade de publicações por país



Fonte: *Web of Science*

Os Estados Unidos da América (EUA - USA) é o país no qual mais se desenvolveram pesquisas relacionadas a modelos de programação inteira de grande porte nos últimos 5 anos, contando com 116 publicações, seguido da China com 95. O Brasil está na 16ª posição em nível de publicações, com apenas 11 pesquisas, o que aponta grande oportunidade de desenvolvimento do tema no Brasil. De outra forma, pode-se observar a evolução de resultados a partir dos anos 2000 (Figura 5).

Figura 5 - Quantidade de publicações por ano

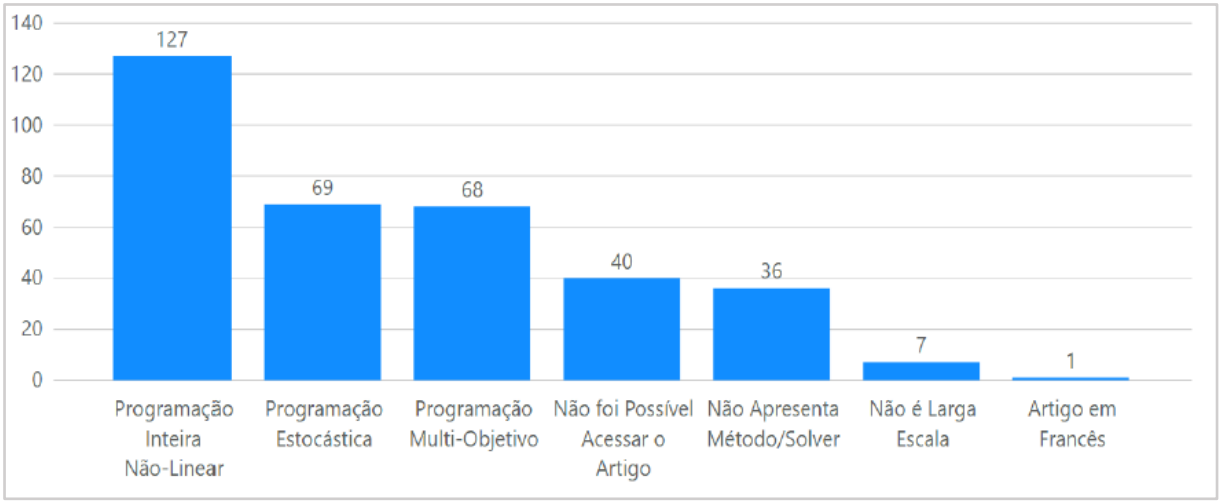


Fonte: Autor por meio dos dados obtidos da base *Web of Science*

No gráfico observa-se que nos últimos cinco anos, o número de publicações foi superior ao período de 2000 a 2015, evidenciando crescimento de artigos sobre o tema. Além disso, entre 2016 e 2017, houve um crescimento de aproximadamente 38% no número de publicações. Em 2018 aconteceu uma diminuição de resultados para 87 trabalhos, porém retornando ao patamar de 108 publicações em 2019. O mesmo padrão de aumento foi mantido entre 2019 e 2020, representando um aumento de aproximadamente 20% na quantidade de pesquisas. O ano de 2021 apresentou poucos resultados, devido à pesquisa ter sido realizada no mês de abril de 2021.

(d) Avaliação da qualidade da literatura incluída na revisão: Para separar os artigos que serão utilizados como base para essa pesquisa, foi realizada uma análise dos resumos (*abstracts*) e complementadas com informações no corpo texto dos artigos. Após a análise realizada, foram selecionados 178 artigos para a revisão sistemática e excluídos 348, separados em diferentes critérios de exclusão listados na Figura 6.

Figura 6 - Número de artigos excluídos da revisão



Fonte: Autor

O principal motivo para exclusão dos artigos foi a utilização de programação inteira não-linear, removendo-se 127 artigos, pois tratavam da resolução de modelos de PI com a função objetivo ou restrições não-lineares.

Outro fator de exclusão foi sobre 69 artigos que utilizavam programação estocástica, ou seja, possuem variáveis ou funções aleatórias que seguem alguma distribuição probabilística. Esses problemas podem exigir um maior tempo de processamento de máquina quando comparados com um mesmo problema

determinístico, devido ao maior número de variáveis a serem consideradas (BIRGE; LOUVEAUX, 2013).

Exclui-se também 68 pesquisas pela utilização de programação multi-objetivo, que interfere diretamente no tempo de processamento de máquina. Além disso, outros fatores de rejeição considerados foram: a impossibilidade de acessar o artigo, falta de apresentação do método e/ou solver utilizado para resolver o problema, não se tratar de um modelo de larga escala e estar em uma língua de baixa fluência dos autores dessa pesquisa.

(e) análise, síntese e disseminação dos resultados: Realizou-se uma análise com as pesquisas dos últimos 5 anos, sendo apresentados os principais problemas modelados, os tipos de algoritmos, solvers e softwares utilizados. Todas as análises e resultados serão apresentados na próxima seção.

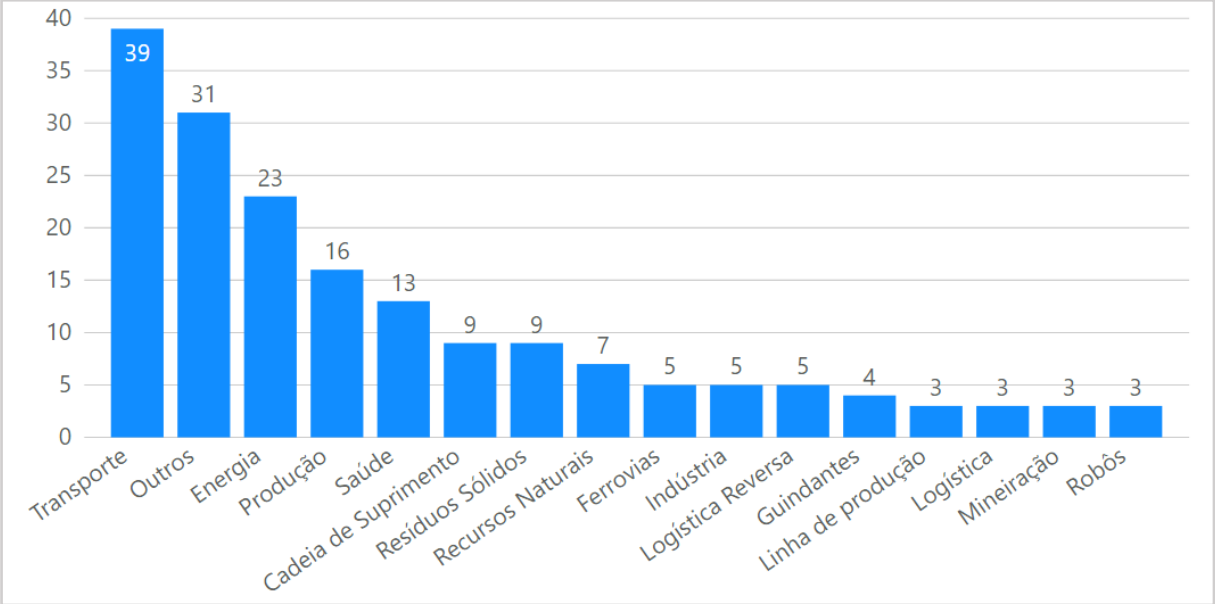
2.2.2 Análise e Resultados da Revisão Sistemática

A partir da leitura dos 178 artigos selecionados, os seguintes critérios foram utilizados para classificá-los:

- Área de aplicação do problema;
- Tipo de problema de pesquisa operacional;
- Método utilizado para resolução do problema;
- Solver empregado na solução;
- Linguagem de programação ou software adotados.

A primeira classificação realizada se refere a área de aplicação do problema, que pode ser verificada de acordo com a realidade e situação proposta de solução. Essa categoria auxilia a busca de modelos específicos para algum nicho de problemas. As áreas de aplicação foram separadas em quinze categorias principais e uma de “Outros” para trabalhos que não se enquadravam em nenhuma outra categoria (Figura 7).

Figura 7 - Número de artigos por área



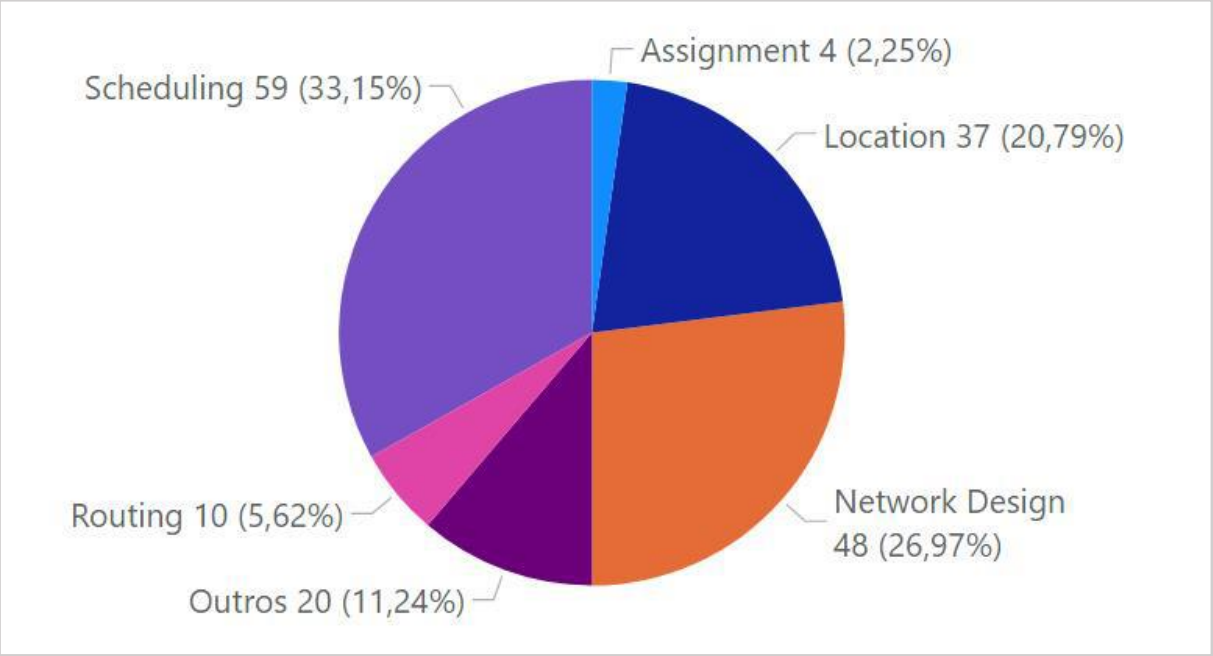
Fonte: Autor

A partir do gráfico é possível extrair que aproximadamente 22% dos problemas selecionados tratam de soluções na área de transporte, são 39 artigos com soluções para modelos que envolvem transporte terrestre, aéreo, marítimo, estações de carregamento de veículos, entre outros. Além dessa, outra categoria de problemas recentes envolve a área de energia, com 23 trabalhos selecionados na pesquisa que, em sua maioria, tratam questões de otimização de energia com redução de tarifas elétricas, instalação de centros elétricos, distribuição da rede elétrica e afins.

De outra forma, verifica-se uma abordagem de artigos na área de produção (16), apresentando problemas sobre produção de biomassa, ferro, animais, alimentos. Na área de Saúde, são apresentadas 13 pesquisas com situações de abastecimento de hospitais, escala de pacientes, utilização de remédios, entre outros. Outras categorias listadas foram: cadeia de suprimento (9), resíduos sólidos (9), recursos naturais (7), ferrovias (5), logística reversa (5), guindastes de construção civil (4), linhas de produção (3), logística (3), mineração (3) e robôs (3). Além disso, 31 artigos não se enquadraram em nenhuma área, sendo classificados como “Outros”.

Seguindo a classificação realizada, foram tratados também os tipos de modelos mais convencionais utilizados na pesquisa operacional. Esses tipos refletem o intuito do modelo adotado e são apresentados na Figura 8, sendo separados em cinco categorias conhecidas em inglês por *Scheduling, Network Design, Location, Routing, Assignment* e na categoria “Outros” caso não se encaixem em uma dessas listadas.

Figura 8 - Número de artigos por tipo



Fonte: Autor

O principal tipo de problema identificado nos artigos é de *Scheduling*, que representa questões sobre otimização de tarefas, sequenciamento de atividades, organização de layout, alocação de recursos (pessoas, tempo, materiais), entre outros (Hillier; Lieberman, 2013). Ao todo foram 59 pesquisas sobre esse tema, representando 33,15% dos resultados, o que mostra a utilidade desse tipo de problemas em áreas como transporte, produção e saúde.

Por outro lado, existem 48 problemas relacionados com Network Design, que significa construção de rede, ou seja, os trabalhos apresentam situações em que geralmente é necessário elaborar e otimizar uma rede ou malha entre dois ou mais pontos, muito utilizado em problemas de distribuição de um centro produtor para um mercado demandante, envolvendo áreas principalmente de transporte, cadeia de suprimento e energia. Além desses, outros tipos de problema foram levantados, como: *Location* (37), *Routing* (10) e *Assignment* (4). Sendo que em 20 artigos não foi possível classificar nos tipos definidos e, portanto, foram incluídos em “Outros”.

Dessa forma, após a análise das áreas e tipos de problemas que os artigos estavam resolvendo, buscou-se identificar os métodos adotados para a solução dos problemas determinísticos de Programação Inteira, cujos dados foram consolidados no Quadro 1.

Quadro 1 - Quantidade de artigos publicados por método/algoritmo

Método	Algoritmo	Artigos
MILP	*Mixed Integer Linear Programming* (MILP)	110
Heurística	Não identificado	25
	Genetic Algorithm (GA)	8
	Greedy	6
	Large Neighborhood Search (LNS)	3
	Tabu Search (TS)	3
	Particle Swarm Optimization (PSO)	2
	Local Search Algorithm (LSA)	2
	Cluster-First	1
	Route-Second	1
	Chronological Decomposition (CD)	1
	Branch-and-Price (B&P)	1
	CWIGALNS	1
	Simulated Annealing (SA)	1
	Gamma-robust	1
	Grey Wolf Optimizer (GWO)	1
	Memetic Algorithm (MA)	1
	YAG	1
	Colony Optimisation (ACO)	1
	Restricted Simulated Annealing (RSA)	1
	Shortest-Path Based	1
	Incremental Fix and Release Heuristic (IFRH)	1
	Relaxed Model	1
Meta-Heurística	*Simulated Annealing (SA)*	6
	Genetic Algorithm (GA)	3
	Large Neighborhood Search (LNS)	2
	Tabu Search (TS)	2
	Particle Swarm Optimization Algorithm (PSO)	1
	Grey Wolf Optimizer (GWO)	1
	Weighted Superposition Attraction (WSA)	1
	Não identificado	1
	Water Cycle Algorithm (WCA)	1
	Evolutionary Algorithm (EA)	1
	Variable Neighborhood Search (VNS)	1
	Constructive	1
	Imperialist Competitive Algorithm (ICA)	1
	Invasive Weed Optimization (IWO)	1
	Self-Adaptive Imperialist Competitive Algorithm (SAICA)	1
	GRASP	1
	Hybrid Cross Entropy Algorithm (HCEA)	1
Matheurística	*Fix-and-Optimize (F&O)*	2
	Relax-and-Fix (R&F)	1

	Hill-Climbing	1
	Aggregation Technique	1
Benders	*Benders*	7
Lagrangian Relaxation	*Lagrangian Relaxation*	4
	Total	**216**

Fonte: Autor

Ao todo foram identificados 216 métodos utilizados nos artigos para resolução de problemas de Programação Inteira de grande porte, número maior que os 178 artigos selecionados para análise, pois alguns deles apresentam mais de um método e realizam comparação entre os desenvolvidos. Sendo assim, observa-se pelo Quadro 1, que 110 pesquisas apresentam um modelo ILP/MILP (Programação Linear Inteira ou Programação Linear Inteira Mista), os quais geralmente são resolvidos por meio de solvers comercias como CPLEX, Gurobi, LINGO, entre outros.

Além disso, é possível identificar o método de heurística sendo desenvolvido em 64 trabalhos. Ele é utilizado quando a complexidade do problema é muito alta e o algoritmo MILP não é capaz de resolvê-lo em tempos aceitáveis. Esse método consiste em encontrar uma solução viável, mas não necessariamente ótima por meio de um algoritmo iterativo que a cada iteração busca uma nova solução para o problema e quando o algoritmo para de rodar, fornece a melhor solução entre as iterações em um tempo hábil (BABAEI et.al., 2015).

Porém, métodos heurísticos precisam ser cuidadosamente adaptados para atender um problema de interesse específico. Dessa forma cada algoritmo é desenvolvido para solucionar algum tipo desse problema particular em vez de uma variedade de aplicações (HILLIER; LIEBERMAN, 2013). Sendo assim, pode-se observar a quantidade de algoritmos diferentes utilizados para resolução desses problemas, sendo os principais *Genetic Algorithm (GA), Greedy, Large Neighborhood Search (LNS), Tabu Search (TS)*, entre outros.

Por outro lado, por muito tempo foi necessário começar do zero para desenvolver um método heurístico que se adequasse ao problema específico sempre que não estivesse disponível algum algoritmo para encontrar a solução ótima. Porém, isso foi alterado recentemente com o desenvolvimento da meta-heurística, método de resolução geral que disponibiliza uma estrutura e diretrizes de estratégias gerais para desenvolver um método de heurística específico (HILLIER; LIEBERMAN, 2013).

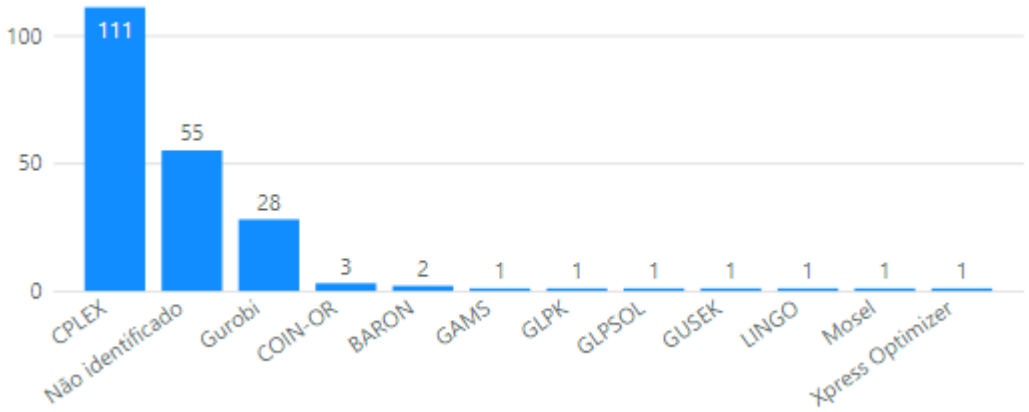
Nos trabalhos analisados, observa-se a utilização de meta-heurística em 26 artigos com proposição de diferentes algoritmos, como: *Simulated Annealing (SA), Genetic Algorithm (GA), Large Neighborhood Search (LNS), Tabu Search (TS), Particle Swarm Optimization Algorithm (PSO), Grey Wolf Optimizer (GWO*), etc.

Outros 5 artigos utilizaram o recente método classificado por matheurística, que, segundo Ribeiro et.al. (2020), consiste na exploração de técnicas matemáticas em estruturas de heurística e meta-heurística garantindo a robustez e eficácia do tempo nos problemas formulados de programação matemática.

Além desses métodos, a técnica de decomposição de Benders foi aplicada em 7 pesquisas. A formulação dela consiste em dividir as variáveis do problema em subconjuntos e resolver o problema “mestre” considerando só um conjunto de variáveis. Os valores das variáveis do segundo conjunto são definidos em um próximo estágio, a partir do resultado encontrado (fixado) das variáveis do primeiro conjunto (BENDERS, 1962). A utilização do método de relaxamento lagrangiano está presente em 4 artigos e propõe relaxar algumas restrições com base em multiplicadores lagrangianos aplicados na solução dual do problema (LEMARÉCHAL, 2001).

A partir da relação dos métodos utilizados para resolução dos problemas de programação inteira de grande porte, é necessário também conhecer quais são os solvers e linguagens de programação ou softwares utilizados para programar esses modelos. Sendo assim, a próxima análise foi realizada reunindo os principais solvers utilizados pelos artigos para resolução desses problemas (Figura 9).

Figura 9 - Número de artigos por solver



Fonte: Autor

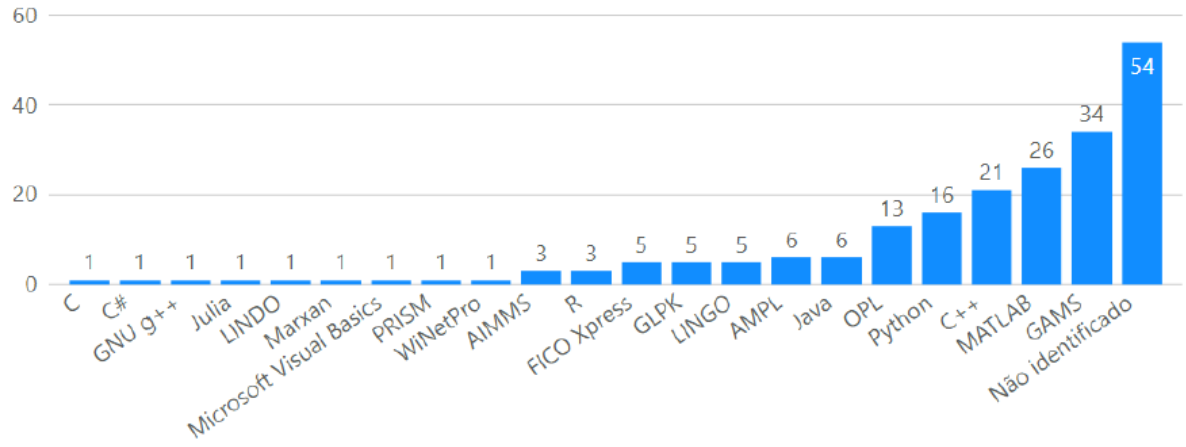
Foram encontrados 206 resultados de *solvers*, sendo a maior parte deles o CPLEX, com 111 resultados e tendo cerca de, aproximadamente, 54% de utilização para resolução dos problemas. O CPLEX, ou IBM ILOG CPLEX, desde 1988 ajudou com o desenvolvimento da solução de problemas cada vez maiores de programação linear, contém um pacote de software de otimização de ponta que aplica geralmente o método simplex e suas variantes para resolução dos problemas de programação matemática (HILLIER; LIEBERMAN, 2013).

Desde a sua criação o CPLEX já passou por diversas atualizações, possibilitando a resolução de problemas de programação inteira, quadrática, entre outros. Ele geralmente é utilizado em conjunto com uma linguagem de modelagem ou software (C, C++, Java, Python, MATLAB, AIMMS, AMPL, GAMS) ou com o software desenvolvido pela própria IBM, o IBM ILOG CPLEX Optimization Studio - OPL (IBM ILOG, 2021).

Apesar do CPLEX ser o solver com maior utilização na atualidade, observa-se o crescimento nos últimos tempos do Gurobi, tendo 28 trabalhos utilizando-o. Esse solver é aplicado na resolução de problemas de programação linear, programação quadrática, programação linear inteira e tem suporte com diferentes linguagens de programação e softwares, como: C++, Java, .NET, Python, MATLAB, R, AIMMS, GAMS, entre outros (GUROBI, 2021). Além desses solvers mais comuns, existem outros utilizados como COIN-OR, BARON, GLPK, LINGO, Xpress Optimizer, etc. Porém, em 54 trabalhos não foi relatado qual o solver utilizado e tiveram a classificação de “Não identificado”.

Por outro lado, para utilização de alguns desses solver é necessário o apoio de alguma linguagem de programação ou software como citado nos exemplos do CPLEX e Gurobi, sendo assim, analisou-se também nos artigos quais eram os mais acionados para resolver os problemas de programação inteira (Figura 10).

Figura 10 - Número de artigos por linguagem de programação/software



Fonte: Autor

Dessa análise encontrou-se uma amostra de 206 artigos, resultado maior do que as 178 pesquisas selecionadas para essa etapa, pois em algumas são utilizadas mais de uma linguagem ou software. Pelo gráfico observa-se que em 54 artigos não foi possível identificar qual a linguagem de programação ou solver utilizado, muitas vezes porque os autores não citam essa informação.

Porém, extrai-se do gráfico que o principal software utilizado é o *General Algebraic Modeling System* (GAMS) com 34 aparições, cerca de 16,5% dos trabalhos, além desse, em segundo lugar vem o MATLAB com 26 aplicações (12,6%). Ambos softwares contém uma interface de desenvolvimento para o usuário com linguagem de programação própria e são capazes de conectar com diversos solvers de otimização, como CPLEX, Gurobi, BARON, COIN-OR, entre outros.

Por outro lado, a linguagem de programação mais utilizada é a C++ com 21 artigos aplicando-a na resolução dos problemas. Em seguida, destaca-se a linguagem Python, com 16 artigos e Java, com 6. Ambas linguagens podem ser escritas e rodadas em diferentes ambientes integrados de desenvolvimento (IDE) gratuitos e são capazes de conectar com os diversos solvers, assim como os softwares citados acima. Alguns outros exemplos de linguagens de programação e softwares utilizados são: IBM ILOG CPLEX Optimization Studio (OPL), AMPL, LINGO, GLPK, FICO Xpress, AIMMS, R Studio, Julia, C, C#, entre outros.

2.3 Softwares de Programação Matemática

De acordo com a revisão sistemática realizada, verifica-se a quantidade de diferentes métodos e *softwares* utilizados para resolução de problemas de programação inteira de grande complexidade. Porém, não é possível identificar a eficácia e eficiência dos diferentes softwares para os tipos de modelo abordados na literatura. Porém, destacam-se alguns *softwares* e *solvers* adotados nesse tipo de problema, que posteriormente serão utilizados nesse trabalho. Eles estão listados nas próximas seções.

2.3.1 LINGO

O LINGO (Language for Intective General Optimizer) é uma ferramenta abrangente projetada para a construção e a resolução de modelos de otimização de Programação Linear, Linear Inteira Mista, Não-Linear, entre outras. A linguagem de modelagem do LINGO permite expressar modelos de maneira intuitiva e direta, usando somatórios e variáveis subscritas (LINDO, 2020).

Ele possui conectividade com a leitura e exportação de dados em planilhas do *Excel*, além de uma linguagem simples de programação para construção dos diversos modelos de pesquisa operacional. Sendo assim, apesar de ser um software robusto, exige um conhecimento intermediário sobre lógica de programação e desenvolvimento de modelos de programação matemática.

O LINGO é amplamente divulgado na comunidade acadêmica devido à liberação de licenças para estudantes, professores e pesquisadores, além de ser referência na explicação e desenvolvimento de modelos apresentados no livro “Introdução à Pesquisa Operacional” (HILLIER; LIEBERMAN, 2013).

Além disso, o LINGO possui solver próprios para resolução dos problemas de programação matemática e não permite a alteração para outros *solvers* comerciais ou de código aberto disponíveis.

2.3.2 AIMMS

O software AIMMS (Advanced Integrated Multidimensional Modeling Software), desenvolvido pela empresa Paragon Decision Technology, é uma linguagem de modelagem algébrica de alto nível que resolve problemas complexos de programação linear, não linear e inteira (BELFIORE; FÁVERO, 2013).

Ele é um software com uma interface intuitiva para construção dos modelos de programação matemática, onde a declaração de variáveis, restrições, parâmetros e índices do modelo podem ser feitos diretamente no menu do AIMMS. Sendo assim, é o *software* ideal quando se busca uma forma mais simples de aprender a construir um modelo de otimização. Assim como o LINGO, ele possui conectividade com a leitura e exportação de dados em planilhas do *Excel*, porém o próprio *software* permite a visualização dos resultados na própria interface, não se fazendo necessário exportar os dados do resultado final para um Excel para ser visualizado.

O AIMMS possui integração com diversos *solvers*, como CPLEX, Gurobi, CBC, utilizados na resolução dos problemas de PO na própria ferramenta, sendo possível alternar entre eles nas configurações de execução do modelo. O AIMMS também disponibiliza licenças acadêmicas para estudantes, professores e pesquisadores.

2.3.3 Julia

A linguagem de programação Julia é voltada principalmente para aplicação na computação científica, sendo uma linguagem de programação dinâmica de alto nível e código aberto (BEZANSON et al., 2012). Essa linguagem é dinamicamente compilada pelo compilador *just-in-time* (JIT) baseado em LLVM e combinado com seu design, apresenta desempenhos semelhantes a linguagem C (JULIA, 2021).

O Julia é uma linguagem recente, disponibilizada a partir de 2012, que começou a ser utilizada também para resolução de modelos de programação matemática complexos, devido à forma de escrita da linguagem se aproximar do Python e a eficiência do código à linguagem C. Dessa forma, bibliotecas foram desenvolvidas para resolução dos problemas de PO, com destaque ao pacote JuMP

(*Julia for Mathematical Programming*), que permite a modelagem de modelos de Programação Linear, Linear Inteira, quadrática, entre outros. Além disso, o pacote possui suporte aos diversos solvers comercias e de código aberto, sendo utilizados no modelo o CPLEX, Gurobi e CBC.

2.3.4 Python

A linguagem de programação Python foi escolhida para replicação do modelo por se tratar de uma linguagem extremamente poderosa de código aberto e que o interesse por ela tem aumentado bastante nos últimos anos, principalmente por possuir estruturas lógicas e diversas bibliotecas disponíveis para os usuários (BORGES, 2014). O Python é extremamente versátil, sendo aplicado em uma enorme variedade de áreas, resolvendo diferentes problemas. De acordo com uma pesquisa realizada pelo site *stackoverflow* em 2020, o python aparece como a terceira linguagem de programação mais amada pelos seus usuários, com 66,7% dos participantes e aparece em primeiro na lista de desejos para aprendizado, com 30%. Destacando assim, o crescimento por amantes e novos programadores em busca da linguagem.

O Python possui integração com a biblioteca “pandas” para leitura dos dados nas planilhas do *Excel* e conversão dos mesmos para formatos suportados pela biblioteca “pyomo” (*Python Optimization Modeling Objects*). Essa biblioteca permite a formulação e análise de diversas classes de modelos matemáticos de otimização, como Programação Linear, Linear Inteira Mista, Não-Linear, entre outras. Além disso, a biblioteca possui suporte à maioria dos principais solvers de código aberto e comerciais, como CBC, GLPK, CPLEX e Gurobi (HART, 2017).

3 MÉTODOS E TÉCNICAS DE PESQUISA

Com base no referencial apresentado no capítulo 2, verifica-se uma dificuldade em identificar e comparar os diferentes métodos e softwares utilizados para resolução de problemas de programação inteira de grande complexidade na questão de eficiência e eficácia. Dessa forma, pesquisas que realizam a comparação entre softwares permitem embasar os gestores e técnicos na melhor escolha de software e solver para o problema enfrentado. Sendo assim, busca-se com essa pesquisa realizar uma comparação entre os diferentes softwares/solvers e algoritmos utilizados na resolução de problemas de programação inteira de grande complexidade.

Essa comparação será realizada a partir de um estudo de caso com base no modelo desenvolvido no projeto “Safety Oversight” (ANAC, 2019) realizado em parceria da Agência Nacional de Aviação Civil (ANAC) e a Universidade de Brasília (UnB) em 2019. O atual capítulo apresenta os métodos e técnicas empregados para a elaboração da pesquisa, bem como as suas classificações de acordo com a literatura analisada. Pretende-se, portanto, explicitar a metodologia utilizada para o alcance dos objetivos, envolvendo a contextualização da Agência e do modelo desenvolvido, que serão detalhados nas próximas seções.

3.1 Tipo e descrição geral da pesquisa

O presente trabalho tem por objetivo desenvolver e explorar um modelo aplicado de grande complexidade a fim de procurar hipóteses e realizar comparações entre os diferentes métodos de resolução do problema. Para isso, utiliza -se o tipo de pesquisa de cunho exploratória, pois busca-se aprimorar o entendimento do problema, a partir do levantamento bibliográfico da questão e aplicação prática do observado. Tendo como finalidade desenvolver e esclarecer questões mais precisas e hipóteses em torno de estudos pouco explorados (COLLIS; HUSSEY, 2005).

Segundo Santos (2000), a pesquisa qualitativa funciona como fator de coleta e análise dos dados e a pesquisa quantitativa sendo de extrema importância a análise de um objeto de estudo para quantificar os seus dados e mensurá-los de acordo com

o nível de importância. Dessa forma, considerou-se para a pesquisa uma abordagem do problema qualitativa e quantitativa, por meio de interpretação e mensuração de fenômenos, coleta e análise de dados numéricos e aplicação de testes comparativos entre os resultados.

Para cumprir com todas as etapas e alcançar os objetivos propostos na pesquisa, realizou-se uma revisão sistemática da literatura para entender os principais métodos e técnicas utilizados na resolução de problemas de programação inteira de grande complexidade. Essa revisão baseou a pesquisa na escolha do software e técnica a serem utilizadas para um estudo comparativo no contexto real do projeto *Safety Oversight* (ANAC, 2019), realizado em parceria entre a Universidade de Brasília (UnB) e a Agência Nacional de Aviação Civil (ANAC), no qual foi desenvolvido um modelo de programação matemática inteira na ANAC para atender as demandas de fiscalização da agência.

3.2 Caracterização da organização ANAC

A Agência Nacional de Aviação Civil (ANAC) é uma das agências reguladoras federais do Brasil, instituída em 2005 e criada para regular e fiscalizar as atividades da aviação civil e infraestrutura aeronáutica e aeroportuária no Brasil. De acordo com a ANAC (2021), dentre as suas principais competências destacam-se:

- Representar o Brasil junto a organismos internacionais de aviação e negociar acordos e tratados sobre transporte aéreo internacional.
- Emitir regras sobre segurança em área aeroportuária e a bordo de aeronaves civis.
- Conceder, permitir ou autorizar a exploração de serviços aéreos e de infraestrutura aeroportuária.
- Estabelecer o regime tarifário da exploração da infraestrutura aeroportuária.
- Administrar o Registro Aeronáutico Brasileiro (RAB).
- Homologar, registrar e cadastrar os aeródromos.
- Emitir certificados de aeronavegabilidade atestando aeronaves, produtos e processos aeronáuticos e oficinas de manutenção.
- Fiscalizar serviços aéreos e aeronaves civis.
- Certificar licenças e habilitações dos profissionais de aviação civil.
- Autorizar, regular e fiscalizar atividades de aeroclubes e escolas e cursos de aviação civil.
- Reprimir infrações às normas do setor, inclusive quanto aos direitos dos usuários, aplicando as sanções cabíveis.

Além das competências, a ANAC segue um plano de objetivos estratégicos elaborado para o período 2016-2020, dentre as quais destacam-se alguns relacionados a melhoria na alocação de recursos de forma estratégica e efetiva, contribuir para o desenvolvimento sustentável da aviação civil, garantir a segurança da aviação civil, garantir a regulação efetiva para a aviação civil, entre outros (Figura 11).

Figura 11 - Mapa Estratégico 2020-2026

Fonte: Adaptado de ANAC (2021)

Tendo em vista os objetivos estratégicos da Agência, diversas pesquisas foram elaboradas no sentido de encontrar boas formas de otimizar a alocação de recursos da ANAC no que tange à alocação de seus servidores para realizarem as fiscalizações finalísticas da agência e serão apresentadas na próxima seção.

3.3 Trabalhos relacionados

Algumas pesquisas já foram desenvolvidas na ANAC entre os anos de 2017 e 2021 para colaborar com a otimização na alocação dos servidores da Agência para as diferentes fiscalizações. O primeiro trabalho foi desenvolvido por Freitas Júnior (2017) que aplicou um modelo de designação clássico na alocação dos servidores da ANAC por meio do *Open Solver* do *Excel*, considerando de forma macro a quantidade de servidores em cada origem e a demanda de atividades em cada destino, não sendo analisada a carga horária e a capacitação para realizar a atividade.

Outro trabalho foi desenvolvido por Silva (2018), no qual foram elaborados dois modelos matemáticos para alocação dos inspetores sendo que o primeiro contém as atividades planejadas e o outro as atividades demandadas (não-planejadas). O modelo 1 leva em consideração o Plano de Trabalho Anual (PTA) da Superintendência de Padrões Operacionais (SPO) para definir as atividades planejadas para serem realizadas no ano pela superintendência. O outro conta com demandas urgentes e pode ser rodado diariamente ou periodicamente de acordo com a necessidade desse tipo de atividade.

Silva (2018) elaborou esses modelos matemáticos com base na construção de heurística por meio do software *Excel* e do *Visual Basic for Applications (VBA)* devido à complexidade do modelo e da quantidade de variáveis binárias necessárias para designação dos servidores. Nesses modelos são considerados além das capacitações para realizar cada atividade, o nível de satisfação de cada servidor em estar realizando a atividade de fiscalização no destino indicado e com a equipe escalada.

O próximo modelo foi proposto por Reis e Celestino (2018) e apresenta um modelo com demanda anual e agrupa a oferta das fiscalizações por grupo de servidores habilitados para realizar a atividade. Sendo assim, cada demanda deve ser realizada por servidores pertencentes a um grupo específico capaz de realizar a atividade. Os servidores foram divididos em 5 grupos e alocados de forma macro de suas origens para o destino e o software utilizado para desenvolver o modelo foi o LINGO 17.1 (licença acadêmica).

Pinheiro (2018) apresenta um modelo matemático para minimização do custo de translado semelhante ao desenvolvido por Reis e Celestino (2018), porém conta

com novos grupos de atividades ofertada pelos servidores da Superintendência de Padrões Operacionais (SPO) e diferentes fiscalizações realizadas ao longo do ano. O modelo foi implementado no *software LINGO 17.0* e os resultados foram exportados para o *Excel.*

O projeto *Safety Oversight* (ANAC, 2019) atuou, em uma de suas frentes, na elaboração de um modelo de programação matemática inteira para alocar os servidores da agência para as diversas atividades de fiscalização realizadas por 4 superintendências finalísticas da ANAC (SAR, SIA, SFI e SPO). O modelo considera a capacitação e disponibilidade dos servidores, o tempo e duração para fazer cada fiscalização nos diversos aeródromos do Brasil com intuito de otimizar os custos de deslocamento. Uma das evoluções no modelo foi considerar a alocação individual de cada servidor, além de considerar a possibilidade de transbordo e a utilização de um ou dois períodos.

O trabalho de Lopes (2019) realiza a análise de sensibilidade em alguns dos parâmetros apresentados por Pinheiro (2018), considerando principalmente os casos de solução degenerada, quando uma ou mais variáveis básicas do modelo possuem o valor igual a 0. No estudo, a autora realiza as análises em uma instância simplificada.

Couto (2020) realiza uma comparação do modelo desenvolvido pela ANAC (2019) com a utilização do *software* AIMMS com o *solver* CPLEX. Na comparação são utilizadas como base três instâncias, alterando-se a quantidade de dias disponíveis para os servidores. Nas duas soluções com maior disponibilidade os dois softwares, LINGO e AIMMS, apresentaram desempenho semelhantes. Porém, na terceira instância o AIMMS obteve um resultado com menor GAP em relação ao LINGO.

Frota (2020) desenvolveu um modelo adaptado de ANAC (2019) na linguagem de programação Julia e resolveu o problema com os *solvers* CBC, CPLEX, GLPK e Gurobi. Realizou também um estudo comparativo desses modelos com o software LINGO em relação ao tempo de execução e entre computadores com diferentes configurações de *hardware*.

Por outro lado, Gamermman (2020) desenvolveu um modelo de alocação voltado para os Centros de Instrução de Aviação Civil (CIAC) localizados em mais de 180 cidades. No modelo desenvolvido na linguagem de programação Julia, o autor

implementou um método de Geração de Colunas, bem como um algoritmo heurístico para resolução do problema.

Silva Junior (2021) desenvolve o modelo de alocação dos servidores da agência com a utilização de diferentes heurísticas. No trabalho são apresentadas ao todo cinco algoritmos de heurística: Algoritmo Genético, Busca Tabu, *Simulated Annealing* e dois métodos de hibridização entre os algoritmos. Todos os resultados obtidos são comparados com a solução ótima encontrada com base no modelo adaptado apresentado por ANAC (2019).

Lim-Apo (2021) propõe a construção de dois modelos de programação matemática com base no já desenvolvido pela ANAC (2019). O primeiro método traz uma reformulação do modelo diminuindo o número de variáveis e restrições, permitindo assim a solução em tempos entre 2 e 25 vezes menores para a maioria dos cenários. O Segundo leva em consideração a alocação em multi-períodos, para o planejamento operacional quinzenal, por meio da implementação de programação dinâmica, com o método de geração de colunas, sendo capaz de reduzir em até 95% o número de variáveis de decisão. Porém, ambos os modelos não permitem a realização de transbordo e formação de duplas com não-especialistas, contemplados no Projeto *Safety Oversight.*

Todos os trabalhos desenvolvidos na ANAC são apresentados na Tabela 1, considerando o software, solver, a consideração de carga horária, número de períodos, se a demanda exige habilitação na atividade para ser realizada, se a alocação é individual ou em grupos e o número de instâncias testadas.

Tabela 1 - Trabalhos realizados na ANAC

Autores	Sofware	Solver	Carga horária?	Períodos	Habilitação?	Individual /Grupos	Instâncias
Freitas Júnior (2017)	Excel	OpenSolver	Não	1	Não	Grupos	1
Silva (2018)	Excel	Heurística	Sim	1	Sim	Grupos	1
Reis e Celestino (2018)	Lingo	Lingo	Sim	1	Sim	Grupos	1
Pinheiro (2018)	Lingo	Lingo	Não	1	Sim	Grupos	1
ANAC (2019)	Lingo	Lingo	Sim	1/2	Sim	Ambos	58/80
Lopes (2019)	Excel	Solver	Não	1	Não	Individual	1
Couto (2020)	Lingo/AIM MS	Lingo/CPLEX, Cbc	Sim	1	Sim	Ambos	3
Frota (2020)	Lingo/Julia	Lingo/CPLEX, Gurobi, Cbc	Sim	1	Sim	Ambos	1
Gamerrmman (2020)	Julia	Gurobi	Sim	N	Sim	Ambos	2
Silva Junior (2021)	Lingo/-	Lingo/Heurística	Sim	1	Sim	Individual	20
Lim-Apo (2021)	Lingo/Julia	Lingo/CPLEX, Gurobi	Sim	N	Sim	Ambos	9

Fonte: Adaptado de Lim-Apo (2021)

Tendo em vista os trabalhos desenvolvidos até o momento na ANAC e a revisão sistemática de literatura sobre métodos e técnicas para resolução de problemas de programação inteira de grande complexidade, desenvolveu-se nessa pesquisa dois modelos, um estendido e outro resumido, sobre o mesmo problema de designação de servidores na Agência e realizou-se uma comparação com os *softwares/solvers* já utilizados em termos de desempenho no tempo de execução e no encontro da solução ótima do problema.

3.4 Caracterização do modelo

A partir do modelo existente, considerou-se premissas para o desenvolvimento tais como os dados utilizados nos modelos, disponibilizados no projeto Safety Oversight (ANAC, 2019), são reais e relativos à quatro superintendências – Superintendência de Aeronavegabilidade (SAR), Superintendência de Infraestrutura Aeroportuária (SIA), Superintendência de Ação Fiscal (SFI) e Superintendência de Padrões Operacionais (SPO).

A alocação considera atividades que foram planejadas para serem executadas em um período, seja ele semanal, mensal ou anual. Sendo que os modelos são ferramentas de apoio à decisão aos gerentes de cada superintendência, indicando uma ideia do possível cenário de alocação dos servidores, mas que pode ser alterado se necessário.

Nas atividades de fiscalização da ANAC, os servidores podem ser alocados a qualquer localidade do Brasil para cumprir a demanda de fiscalizar os diferentes regulados ou não da Agência. Para isso, os servidores podem estar lotados nas sedes da Agência (DF, RJ e SP) ou nos diferentes Núcleos Regionais de Aviação Civil (NURAC) espalhados pelo Brasil, o que no modelo é chamado de nó de origem. E realizam a fiscalização em um nó de destino (aeródromos com atividade no Brasil), podendo eles estarem localizados em qualquer Unidade Federativa (UF), sendo possível realizar mais de uma fiscalização no mesmo destino. Porém, eles devem retornar a sua origem após a missão.

Além disso, os servidores possuem uma lista de atividades que podem desempenhar, sendo somente alocados naquelas missões que possuem capacitação para realizá-las. Eles estão limitados também à quantidade de períodos que podem ser alocados de acordo com a disponibilidade, em dias, para cada período de fiscalização.

Cada demanda de fiscalização ocorre em um destino e com a atividade pré-estabelecidos, sendo que cada atividade possui o número de servidores necessários e o tempo de duração (em dias) para realizá-la. Todas as demandas devem ser cumpridas no período indicado no modelo. O modelo considera os arcos de origem-destino formados pelas UF de origem dos servidores e aeródromos de destino das fiscalizações, tendo os custos dos arcos de origem-destino calculados em 2019 por meio de um *Web Crawler* desenvolvido na linguagem de programação *Python* com utilização da biblioteca *Selenium*. Os modelos utilizados nessa pesquisa serão apresentados a seguir, no capítulo 4.

Utilizou-se o *Excel* para o tratamento dos dados consolidados para o modelo, pois ambos os softwares aplicados nessa pesquisa possuem formas de leitura e escrita em planilhas eletrônicas. Assim, os dados foram computados e agrupados de acordo com as necessidades específicas de modelagem para cada sistema. A implementação e software empregados está listada na seção 4.3 dessa pesquisa.

4 DESCRIÇÃO DO MODELO

4.1 Modelo 1 - Modelo de Alocação Estendido (MAE)

O Modelo de Alocação Estendido (MAE) tem como referência o algoritmo apresentado por Reis e Celestino (2018) e no projeto *Safety Oversight* (ANAC, 2019). O algoritmo considera os servidores alocados para realizar as atividades demandadas em um determinado período e faz alguns cortes de parâmetros, como:

- Considerar somente as origens, destinos e ativos. Sendo as origens ativas filtradas pelos servidores aptos para realizar fiscalizações nas demandas e os destinos ativos com base nas demandas de fiscalização. Os arcos ativos são a combinação de origem e destino;
- Utilizar só as atividades que foram demandadas para o período específico, podendo esse período ser diário, semanal, mensal de acordo com a quantidade de demandas;
- Definir os servidores que estão aptos para realizar fiscalizações no período de acordo com a capacitação necessária e disponibilidade;

4.1.1 Índices

O Quadro 2 apresenta os índices utilizados no modelo MAE.

Quadro 2 - Índices do modelo MAE

Índice	Significado	Descrição
N	Nós	Conjunto de origens e destinos
I	Subconjunto de N	Origens em N
J	Subconjunto de N	Destinos em N
O	Origem	Unidades Federativas
D	Destino	Aeroportos
K	Missão	Números das demandas
P	Pessoa	Servidores da ANAC

T	Período	Período de realização das demandas (diário, semanal, mensal)
GR	Grupo	Capacitações necessárias

Fonte: Adaptado do Projeto *Safety Oversight* (ANAC, 2019)

4.1.2 Parâmetros

O Quadro 3 mostra os parâmetros do modelo MAE.

Quadro 3 - Parâmetros do modelo MAE

Parâmetro	Descrição
$Custo_{i,j}$	Custo da passagem aérea da origem (i) para o destino (j)
$Tempo_{i,j}$	Tempo de viagem da origem (i) para o destino (j)
$Duração_{k}$	Duração da missão (k)
$Equipe_{k}$	Equipe necessária para a missão (k)
$Disponibilidade_{p,t}$	Disponibilidade em dias do servidor (p) para o período (t)
$Demanda_{t,k,gr,d}$	Demanda da missão (k) para o período (t) e destino (d) com a atividade (gr)
$Oferta_{p,gr,o}$	Oferta do servidor (p) para cada atividade (gr) e origem (o)

Fonte: Adaptado do Projeto *Safety Oversight* (ANAC, 2019)

4.1.3 Variáveis

O modelo MAE é composto por quatro variáveis de decisão alternando entre contínuas e binárias, apresentadas no Quadro 4.

Quadro 4 - Variáveis do modelo MAE

Variável	Tipo	Descrição
$Atendida_{t,k,gr,d,p}$	Binária	Indica a pessoa (p) que atendeu a missão (k) do grupo (gr) no destino (d) no período (t)
$Alocado_{t,i,j,p,gr,k}$	Contínua	Indica o fluxo da pessoa (p) de uma origem (i) para um destino (j) que atendeu a missão (k) com grupo (gr) no período (t)
$AlocadoGeral_{t,i,j,p}$	Binária	Indica o fluxo da pessoa (p) de uma origem (i) para um destino (j) no período (t)
$TempoUtilizado_{p,t}$	Contínua	Indica o tempo utilizado pela pessoa (p) no período (t)

Fonte: Adaptado do Projeto *Safety Oversight* (ANAC, 2019)

4.1.4 Equações

$$Minimizar \sum_{t,i,j,p} AlocadoGeral_{t,i,j,p} * 2 * Custo_{i,j} \quad \textbf{(1)}$$

Sujeito a:

$$\sum_{k} Alocado_{t,o,d,p,gr,k} \leq Oferta_{p,gr,o} \; \forall \, t \in T, o \in O, d \in D, p \in P, gr \in GR, k \in K \quad \textbf{(2)}$$

$$\sum_{i} Alocado_{t,i,d,p,gr,k} = Atendida_{t,k,gr,d,p} + \sum_{j} Alocado_{t,d,j,p,gr,k} \quad \textbf{(3)}$$

$$\forall \, t \in T, i \in O, d \in D, p \in P, gr \in GR, k \in K$$

$$\sum_{i} Alocado_{t,i,d,p,gr,k} \geq \sum_{j} Alocado_{t,d,j,p,gr,k} \quad \textbf{(4)}$$

$$\forall \, t \in T, o \in O, d \in D, p \in P, gr \in GR, k \in K$$

$$\sum_{p} Atendida_{t,k,gr,d,p} = Equipe_{k} * Demanda_{t,k,gr,d} \quad \textbf{(5)}$$

$$\forall \, t \in T, d \in D, p \in P, gr \in GR, k \in K$$

$$\sum_{k,gr,d} Atendida_{t,k,gr,d,p} * Duração_{k} + \sum_{i,j} AlocadoGeral_{t,i,j,p} * 2 * \quad \textbf{(6)}$$

$$Tempo_{i,j} = TempoUtilizado_{p,t} \qquad \forall \, t \in T, p \in P$$

$$TempoUtilizado_{p,t} \leq Disponibilidade_{p,t} \; \forall \, t \in T, p \in P \quad \textbf{(7)}$$

$$AlocadoGeral_{t,i,j,p} \geq Alocado_{t,i,j,p,gr,k} \quad \textbf{(8)}$$

$$\forall \, t \in T, i \in N, j \in N, p \in P, gr \in GR, k \in K$$

$$AlocadoGeral_{t,i,j,p} \in \mathbb{Z}^{+}\{0,1\}, \quad Alocado_{t,i,j,p,gr,k} \in \mathbb{R}^{+}, \quad TempoUtilizado_{p,t} \in \mathbb{R}^{+}, \quad Atendida_{t,k,gr,d,p} \in \mathbb{Z}^{+}\{0,1\} \tag{9}$$

4.1.5 Função Objetivo (FO) e Restrições

A resolução do problema de alocação dos servidores da ANAC deve encontrar a solução ótima com ênfase em otimizar os custos com passagem aérea com a minimização da FO. A Função Objetivo (1) do modelo MAE é um somatório da alocação geral dos servidores vezes o custo com passagem aérea para levar o servidor de sua origem ao destino da demanda. O custo total é considerado como duas vezes o custo do arco, considerando a ida e volta do servidor.

Na restrição (2) tem-se que o somatório para cada missão do Alocado, que indica a pessoa P que atende a missão K do grupo GR, no destino D, no período T deve ser menor ou igual a Oferta de pessoas P do grupo GR na origem O. Essa restrição se faz necessária para que possamos alocar para as missões apenas os servidores que tem oferta na sua origem e realizavam a atividade do grupo.

Na restrição (3) realiza-se o somatório em O do Alocado para determinado período, origem, destino, pessoa, grupo e missão sendo igual ao somatório em D do Alocado mais a variável Atendida. Ela representa o balanço dos fluxos, garantindo que cada pessoa que chega em cada destino atende uma missão com possibilidade de sair para realizar mais missões. Essa restrição é de fundamental importância para possibilitar a realização de missões de transbordo com a adição do somatório de alocado no lado direito da equação.

Já na restrição (4) calcula-se o somatório em O do Alocado sendo menor ou igual ao somatório em D do alocado, sendo necessária para garantir que nenhum fluxo será gerado nos nós de destino. Essa restrição visa garantir a possibilidade de transbordo.

A restrição (5) se dá pelo somatório em P da variável Atendida sendo igual a equipe da missão K vezes a demanda em um período T, de missão K, grupo GR no destino D. Ela é necessária para formar as equipes de cada missão.

A restrição (6) se dá pela multiplicação do somatório em K, GR e D da variável Atendida pela Duração de K mais a multiplicação do somatório em I e J do AlocadoGeral com o Tempo de viagem de I para J vezes 2, sendo igual a variável Tempo Utilizado, que indica quanto tempo do servidor foi utilizado, considerando o tempo para realizar a missão mais tempo de viagem. Essa restrição faz com que o modelo considere o tempo total utilizado de viagem e missão para cada alocação. Sendo que a restrição (7) representa que Tempo Utilizado será sempre menor ou igual que a Disponibilidade de cada servidor que é dada em dias/período.

A restrição (8) impõe que AlocadoGeral deve ser menor ou igual que o Alocado, com intenção de não cobrar mais de uma vez por passagem aérea do servidor para um determinado trecho. Enquanto a variável Alocado é gerada para cada pessoa, grupo e missão que percorrer a rede, uma mesma pessoa pode gerar várias variáveis Alocado, porém, isso iria gerar custos repetidos se fosse colocada na função objetiva, por isso, a determinação da variável Alocado Geral.

As restrições representadas em (9) definem o universo dos conjuntos de variáveis, onde as variáveis Alocado e TempoUtilizado pertencem aos números reais positivos enquanto a variáveis Atendida e AlocadoGeral são binárias, representadas de 1 ou 0.

4.2 Modelo 2 - Modelo de Alocação Resumido (MAR)

O Modelo de Alocação Resumido (MAR) tem como referência o algoritmo apresentado por Reis e Celestino (2018) e no projeto *Safety Oversight* (ANAC, 2019), elucidado no Modelo de Alocação Estendido (MAE) na seção 4.1. No MAR não é determinado o período em que as atividades devem ser realizadas, porém ainda são mantidas as inspeções que cada servidor pode realizar e também a disponibilidade de cada colaborador.

Esse modelo foi desenvolvido no trabalho de Lim-Apo (2021), sendo que a principal diferença entre o MAE e o MAR é que neste são criadas apenas as variáveis de alocação que realmente podem ser utilizadas para solução do problema. No modelo do MAE são criadas todas as combinações possíveis de variáveis e, no modelo, elas são limitadas por meio de restrições. Porém, um maior número de variáveis e restrições prejudica o desempenho dos softwares quando se aumenta a quantidade de demandas, tornando algumas combinações inviáveis de serem resolvidas em tempos hábeis. Com essa alteração, o MAR permite resolver o mesmo problema com um menor número de variáveis e restrições.

4.2.1 Índices

O Quadro 5 apresenta os índices utilizados no modelo MAR.

Quadro 5 - Índices do modelo MAR

Índice	Significado	Descrição
M	Missões	Números das missões
O	Origem	Nós de Origem
D	Destino	Nós de Destino
I	Servidores	Servidores da ANAC
MI_i	Missão_Servidor	Missões *m* que do servidor *i* pode realizar
DI_i	Destino_Servidor	Destinos *d* que do servidor *i* pode ir
OI_i	Origem_Servidor	Origem *o* do servidor *i*

Fonte: Adaptado de Lim-Apo (2021)

4.2.2 Parâmetros

O Quadro 6 mostra os parâmetros do modelo MAR.

Quadro 6 - Parâmetros do modelo MAR

Parâmetro	Descrição
DM_m	Duração da missão *m*
E_m	Tamanho da equipe de inspetores necessários da missão *m*
DP_i	Disponibilidade de trabalho do servidor *i*
$C_{o,d}$	Custo da viagem da origem o para o destino *d*

$T_{o,d}$	Tempo da viagem da origem o para o destino *d*

Fonte: Adaptado de Lim-Apo (2021)

4.2.3 Variáveis

O modelo MAR é composto por duas variáveis de decisão binárias, apresentadas no Quadro 4.

Quadro 7 - Variáveis do modelo MAR

Variável	Tipo	Descrição
$AM_{i,m}$	Binária	Define se o servidor *i* será alocado na missão *m*
$AG_{i,d}$	Binária	Define se o inspetor i será alocado no arco (OI_i, *d*)

Fonte: Adaptado de Lim-Apo (2021)

4.2.4 Equações

$$Minimizar \sum_{i,d|d\in DI_i} 2 * C_{OI_i,d} * AG_{i,d} \quad \textbf{(10)}$$

Sujeito a:

$$\sum_{i|m\in MI_i} AM_{i,m} = E_m \qquad \forall\, m \in M \quad \textbf{(11)}$$

$$\sum_{m|m\in MI_i} DM_m * AM_{i,m} + \sum_{d|d\in DI_i} 2 * T_{OI_i,d} * AG_{i,d} \leq DP_i \qquad \forall\, i \in I \quad \textbf{(12)}$$

$$AG_{i,d} \geq AM_{i,m} \qquad \forall\, i \in I, d \in DI_i, m \in M|m \in MI_i \quad \textbf{(13)}$$

$$AM_{i,m}\ Bin \qquad \forall\, i \in I, m \in M|m \in MI_i \quad \textbf{(14)}$$

$$AG_{i,d}\ Bin \qquad \forall\ i \in I, d \in D | d \in DI_i \tag{15}$$

4.2.5 Função Objetivo (FO) e Restrições

A resolução do problema de alocação dos servidores da ANAC deve encontrar a solução ótima com ênfase em otimizar os custos com passagem aérea com a minimização da FO. A Função Objetivo (10) do modelo MAR é um somatório da alocação geral dos servidores vezes o custo com passagem aérea para levar o servidor de sua origem ao destino da demanda. O custo total é considerado como duas vezes o custo do arco, considerando a ida e volta do servidor.

Na restrição (11) tem-se que o somatório para cada missão do servidor capaz de realizar aquela missão ($AM_{i,m}$) deve ser igual a equipe necessária para realizar a missão. Essa restrição se faz necessária para que possamos alocar para as missões apenas os servidores que podem realizar a atividade do grupo, levando em consideração a equipe necessária para realiza-la.

Na restrição (12) se dá pelo somatório para cada missão M do alocado para missão $AM_{i,m}$ vezes a duração da missão DM_m mais o somatório para cada destino D da missão de 2 vezes o tempo para a viagem de OI_i para d vezes o alocado para o destino D deve ser menor ou igual a disponibilidade do servidor. Cumprindo assim a disponibilidade dos servidores para realizar as fiscalizações.

Já na restrição (13) realiza-se que para cada servidor, missão que pode realizar e destino da missão, a variável $AG_{i,d}$ deve ser maior ou igual a variável $AM_{i,m}$. Essa restrição força o modelo a alocar o servidor para realizar a missão.

As restrições representadas por (14) e (15) impões que ambas as variáveis $AG_{i,d}$ e $AM_{i,m}$ são binárias, representadas de 1 ou 0.

4.3 Implementação do modelo

Após a formulação dos modelos definidos nas seções 4.1 e 4.2 acima, buscou-se implementá-los em diferentes *softwares* para realizar as comparações de desempenho entre eles. Sendo assim, aproveitou-se o modelo desenvolvido no projeto *Safety Oversight* (ANAC, 2019) para rodar o MAE no *software* LINGO 18.0. Também foi utilizado o modelo desenvolvido por Couto (2020) no sistema AIMMS para comparação e as heurísticas apresentadas por Silva Junior (2021).

Além disso, essa pesquisa implementou o MAE na linguagem de código aberta Python, que estava dentre as mais utilizadas para resolução de problema de Programação Inteira de grande complexidade apresentado na revisão sistemática (seção 2.2) e ser a terceira linguagem de programação mais amada pelos seus usuários. Para construção do modelo no Python, foram utilizadas as bibliotecas “Pyomo” e “Pandas”.

Por outro lado, os testes com o modelo MAR foram construídos no *software* LINGO 18.0 e na linguagem de programação Julia no trabalho desenvolvido por Lim-Apo (2021). Sendo assim, essa pesquisa também realizou a elaboração do modelo MAR na linguagem de programação Python.

Todos os experimentos com os modelos foram executados em um servidor *Windows* com processador Intel® Xeon® CPU E5-2650 @2.00 GHz (8 CPUs), 32 GB de memória RAM e sistema operacional *Windows Server 2016 Standard 64-bit*. Os *softwares* utilizados na pesquisa, bem como suas especificações, estão listados na seção 2.3.

5 RESULTADOS E DISCUSSÃO

Este capítulo aborda as análises e resultados obtidos por meio dos experimentos computacionais nos modelos MAE e MAR nos diferentes softwares e solver escolhidos para comparação. O Quadro 8 apresenta um resumo dos modelos/sistemas que serão abordados nas análises, contendo os softwares, versões, solvers e modelos rodados. Os códigos de implementação dos modelos nos softwares estão disponíveis no capítulo de apêndices.

Quadro 8 - Modelos implementados

Sigla	Software	Versão	Solver	Modelo
lingo_extend	LINGO	18.0	LINGO	MAE
aimms	AIMMS	4.82	CPLEX 20.1	MAE
py_extend_gurob	Python	3.9.7	Gurobi 9.1	MAE
py_extend_cplex	Python	3.9.7	CPLEX 20.1	MAE
jl_resum_gurob	Julia	1.6	Gurobi 9.1	MAR
jl_resum_cplex	Julia	1.6	CPLEX 20.1	MAR
jl_resum_cbc	Julia	1.6	CBC 2.10	MAR
py_resum_gurob	Python	3.9.7	Gurobi 9.1	MAR
py_resum_cplex	Python	3.9.7	CPLEX 20.1	MAR
lingo_resum	LINGO	18.0	LINGO	MAR
ts_heurist	Python	3.9.7	Heurítisca (Busca Tabu)	-
sa_heurist	Python	3.9.7	Heurística (*Simulated Annealing)*	-
hibrid_heurist	Python	3.9.7	Heurística (*Genetic Algorithm* e *Simulated Annealing*)	

Fonte: Autor

Todos os experimentos com os modelos foram executados em um servidor *Windows* com processador Intel® Xeon® CPU E5-2650 @2.00 GHz (8 CPUs), 32 GB de memória RAM e sistema operacional *Windows Server 2016 Standard 64-bit*. Todos os modelos foram implementados e executados com *softwares/solvers* com licença acadêmica ou de código aberto, sendo mantidas as configurações *default* de cada sistema. Ao todo foram criadas 6 instâncias, apresentadas na seção 5.1, com diferentes parâmetros para rodar todos os modelos implementados.

5.1 Instâncias

A fim de identificar as diferenças na eficiência de resolução dos modelos propostos, criaram-se 6 instâncias com diferenciações entre os principais parâmetros de entrada do modelo. A Tabela 2 serve para identificar e explicitar as diferentes instâncias de acordo com as classificações:

- Servidores - |S| - Quantidade de servidores aptos para realizar ao menos uma fiscalização (considerando a capacitação e disponibilidade);
- Disponibilidade - |T| - Disponibilidade, em dias, que cada um dos servidores |S| possui para realizar as fiscalizações;
- Origens - |O| - Quantidade de origens ativas;
- Destinos - |D| - Quantidade de destinos ativos;
- Missões - |M| - Quantidade de missões para o determinado período;
- Atividades - |G| - Quantidade de atividades únicas habilitadas nas missões;

Tabela 2 - Características das instâncias implementadas

Instância	|S|	|T|	|O|	|D|	|M|	|G|
A1	151	20	3	22	118	23
A2	151	12	3	22	118	23
A3	96	15	3	21	80	23
B1	124	5	16	30	50	21
B2	125	5	16	74	120	41
B3	76	5	15	66	100	30

Fonte: Autor

A instância “A” é formada com referência nas demandas da SPO elaborada no projeto *Safety Oversight* (ANAC, 2019) e conta com o cenário base “A1” e variações dele, considerando uma menor disponibilidade dos servidores no cenário “A2” e com a diminuição do número de servidores disponíveis e de missões para realizar no cenário “A3”.

Por outro lado, a instância “B” compõe-se pelas demandas da SFI, composto por atividades das 4 superintendências finalísticas da ANAC (SAR, SIA, SFI e SPO) e com servidores lotados nas sedes da agência e nos diferentes NURAC espalhados pelo Brasil. Dessa forma, o cenário base “B1” conta com 16 origens, 124 servidores

disponíveis para realizar as 50 missões. Uma variação do cenário “B1” foi uma tentativa de estressar o modelo, considerando as 16 origens base, mas aumentando o número de missões para 120, o que ocasionou um aumento de destinos (D) ativos e a quantidade de atividades a serem realizadas no cenário “B2”. Por último, a instância “B3” considerou diminuir o número de servidores e a quantidade de missões.

5.2 Resultados gerais

As instâncias criadas e apresentadas na Tabela 2 foram organizadas por meio de planilhas no *software Excel*, que possui integração com os sistemas utilizados nessa pesquisa. Dessa forma, todos os cenários foram testados nos modelos (Quadro 8) implementados e, pela relação direta entre o número de variáveis e restrições com o desempenho do software em termos de tempo de solução, construiu-se a Tabela 3, que apresenta o número de variáveis (V) e restrições (R) para cada modelo em cada instância.

Tabela 3 - Número de variáveis (V) e restrições (R) para cada modelo

Sigla	Modelo	A1 e A2		A3		B1		B2		B3	
		(V)	(R)	(V)	(R)	(V)	(R)	(V)	(R)	(V)	(R)
lingo_extend	MAE	1.023.144	1.429.497	701.335	1.021.047	1.958.568	2.551.485	9.795.944	12.364.217	8.453.772	10.594.463
aimms	MAE	1.022.878	1.667.589	701.081	1.182.901	1.934.869	2.657.946	9.747.726	12.927.053	8.401.473	11.089.400
py_extend_gurob py_extend_cplex	MAE	264.340	263.141	119.021	118.400	125.355	122.924	620.849	613.615	342.441	338.565
jl_resum_gurob jl_resum_cplex jl_resum_cbc	MAR	14.243	11.611	10.840	8.328	6.463	4.088	13.857	8.458	13.195	8.059
py_resum_gurob py_resum_cplex	MAR	14.244	11.612	10.841	8.329	6.464	4.089	13.858	8.459	13.196	8.060
lingo_resum	MAR	14.243	11.612	10.840	8.329	6.463	4.089	13.857	8.459	13.195	8.060
ts_heurist sa_heurist hibrid_heurit	-	-	-	-	-	-	-	-	-	-	-

Fonte: Autor

Nos modelos com base no MAE, observa-se a primeira redução do número de variáveis e restrições entres o “lingo_extend” e “aimms” para o “py_extend_gurob e cplex”. Observa-se que para as instâncias A1 e A2, o número de variáveis reduziu de 1023144 para 264340, cerca de 74% e o de restrições 81%. Para a instância B2, a diminuição foi ainda maior, partindo de 9795944 variáveis e 12364217 de restrições para 620849 e 613615, respectivamente, representando uma redução próxima a 94%. Essa redução ocorreu devido a formulação do modelo MAE no Python criar somente as variáveis e restrições considerando a origem do servidor, o grupo e missão que cada um poderia realizar, sendo assim, não eram criadas combinações que não poderiam ser utilizadas no modelo.

Por outro lado, considerando que ambos os modelos “jl_resum_gurob, cplex e cbc”, “py_resum_gurob e cplex” e “lingo_resum” possuem o mesmo número de restrições e variáveis, alterando somente 1 unidade devido a forma de entendimento de cada linguagem em relação à construção da Função Objetivo (FO), observa-se que a diferença para os modelos MAR é ainda maior. Neles, ocorreu a redução de aproximadamente 98,5% e 99,8% para as variáveis das instâncias “A” e “B”, respectivamente, e cerca de 99,2% e 99,9% para as restrições dessas instâncias. Por exemplo, na instância B2 uma diminuição de 12364217 restrições para 8459 do modelo MAE no LINGO para o modelo MAR no Python.

A consequência dessa diminuição pode ser observada no tempo de solução para cada um dos modelos. Devido a quantidade de combinações entre Modelos versus Instâncias, os dados foram consolidados e os resultados apresentados na seção 5.2.1 para a instância “A” e na seção 5.2.2 para a “B”.

5.2.1 Resultados da Instância A

A Tabela 4 - Resultados dos modelos para Instância "A"Tabela 4 apresenta os resultados obtidos nos testes realizados para todos os modelos desenvolvidos e apresentados no Quadro 8, considerando a Função Objetivo (FO) encontrada, o tempo de construção, tempo de solver e tempo total, em segundos, para cada modelo. Nessa tabela, estão consolidados os dados para as instâncias A1, A2 e A3, separadas em cada linha para cada modelo.

Tabela 4 - Resultados dos modelos para Instância "A"

Sigla	Modelo	Instância	Função Objetivo (FO)	Tempo de Construção (s)	Tempo Solver (s)	Tempo Total (s)
lingo_extend	MAE	A1	27.100	-	-	49
		A2	28.286*	-	-	>1800
		A3	24.953	-	-	25
aimms	MAE	A1	27.100	47,55	9,84	57,39
		A2	28.286	38,78	130,61	169,39
		A3	24.953	46,58	6,22	52,8
py_extend_gurob	MAE	A1	27.100	173,09	7,82	180,91
		A2	28.286	169,42	113,14	282,56
		A3	24.953	76,63	4,39	81,02
py_extend_cplex	MAE	A1	27.100	177,37	5,5	182,87
		A2	28.286*	175,73	>1800	>1800
		A3	24.953	69,52	2,58	72,1
jl_resum_gurob	MAR	A1	27.100	3,37	1,62	4,99
		A2	28.286	3,43	172,92	176,35
		A3	24.953	3,1	0,67	3,77
jl_resum_cplex	MAR	A1	27.100	3,82	5,44	9,26
		A2	28.286	3,31	177,22	180,53
		A3	24.953	2,9	0,69	3,59
jl_resum_cbc	MAR	A1	27.100	3,42	13,31	16,73
		A2	28.286*	3,5	>1800	>1800
		A3	24.953	2,93	14,77	17,7
py_resum_gurob	MAR	A1	27.100	7,07	2,17	9,24
		A2	28.286	6,97	68,52	75,49
		A3	24.953	5,03	1,46	6,49
py_resum_cplex	MAR	A1	27.100	7,09	1,67	8,76
		A2	28.286*	9,6	>1800	>1800
		A3	24.953	5,3	1,26	6,56
lingo_resum	MAR	A1	27.100	-	-	9
		A2	28.300*	-	-	>1800
		A3	24.953	-	-	60
ts_heurist	-	A1	88.824	224,59	971,49	1196,08
		A2	89.126	180,93	871,52	1052,45
		A3	62.986	140,04	417,24	557,28
sa_heurist	-	A1	101.050	210,39	20,06	230,45
		A2	95.198	185,55	23,75	209,3
		A3	65.328	139,88	14,42	154,3
hibrid_heurist	-	A1	35.697	224,59	459	683,59
		A2	36.599	213,72	521,44	735,16
		A3	25.853	85,32	203,23	288,55

Fonte: Autor

Observa-se nos resultados que o LINGO não faz distinção entre o tempo gasto para construção do modelo e o tempo do solver para resolução do modelo, dessa forma somente o tempo total é considerado na análise. Além disso, os campos de “Tempo Total (s)” representados com o valor “>1800” informam que os modelos excederam o tempo limite para solução, porém encontraram uma solução factível representada pelo caractere “*” no campo “Função Objetivo (FO)”.

Tendo em vista os modelos implementados, observa-se que dentre os modelos MAE, o LINGO apresenta o melhor tempo total de solução nas instâncias A1 e A3, com 49 e 25 segundos, respectivamente, de resolução. Considerando, somente o tempo do solver, destacam-se os modelos implementados no Python, com ambos os *solvers* CPLEX e Gurobi, todos próximos de 5 segundos de solução. Identificando-se assim, um gap no tempo de construção do modelo nesse sistema.

Porém, na instância A2 identifica-se que os modelos “lingo_extend” e “py_extend_cplex” não conseguiram encontrar a solução até o tempo limite de execução. Apesar disso, a solução viável encontrada pelos modelos nesse tempo limite foi a ótima “27100” e ambos os modelos a alcançaram em alguns minutos, mas continuaram a rodar com intuito de diminuir o gap e não finalizaram a execução. O melhor tento de solução foi apresentado pelo AIMMS, com 169,39 segundos no total, seguido pelo Python com *solver* Gurobi, com 282,56 segundos, sendo que o tempo de execução do solver foi mais rápido no segundo do que no primeiro modelo.

Por outro lado, os modelos com base no MAR apresentaram resultados em tempos computacionais, na maioria dos casos, melhores do que os do MAE. Na instância A1, o principal destaque é para a linguagem Julia com o *solver* Gurobi, que apresentou uma solução do modelo “jl_resum_gurob” em 4,99 segundos, o melhor resultado verificado para o cenário. Outros modelos obtiveram soluções bem próximas como o “jl_resum_cplex” “jl_resum_cbc”, “py_resum_gurob”, “py_resum_cplex” e “lingo_resum” com 9,26;16,73; 9,24; 8,76 e 9 segundos, respectivamente.

Em relação ao cenário A2, o modelo “py_resum_gurob” obteve o melhor resultado com resolução do modelo em 75,49 segundos, cerca de 2 vezes mais rápido do que o desempenhado pelos modelos “jl_resum_cplex” e “jl_resum_gurob”, com tempo de 180,53 e 176,35, respectivamente. Apesar disso, os modelos “jl_resum_cbc”, “py_resum_cplex” e “lingo_resum” excederam o tempo limite de 1800

segundos, sendo que os dois primeiros chegaram na solução viável ótima, 28286, porém o LINGO só conseguiu alcançar o valor 28300.

Para a instância A3, ambos os modelos MAR conseguiram encontrar uma solução em poucos segundos, sendo destaque o modelo “jl_resum_cplex” e “jl_resum_gurob” com resolução em 3,59 e 3,77 segundos, respectivamente. Os resultados no Python foram parecidos com os tempos de 6,56 e 6,49 para os modelos “py_resum_cplex” e “py_resum_gurob”. O LINGO resolveu o modelo em 60 segundos e destaca-se também a solução do modelo “jl_resum_cbc”, com 17,7 segundos para encontrar o ponto ótimo.

Os modelos de heurística “ts_heuristic” e “as_heuristic” não obtiveram um bom desempenho em relação aos modelos MILP apresentados. Devido à quantidade de variáveis e restrições verificados nas instâncias, o modelo apresentou uma diferença de mais de duas vezes o valor das funções objetivos. Porém, para o modelo “hibrid_heuristic”, o resultado encontrado para FO chegou próximo a 30% do ótimo. Apesar disso, todos os cenários foram rodados antes do tempo limite de 1800 segundos.

5.2.2 Resultados da Instância B

A Tabela 5 apresenta os resultados obtidos nos testes realizados para todos os modelos desenvolvidos e apresentados no Quadro 8, considerando a Função Objetivo (FO) encontrada, o tempo de construção, tempo de solver e tempo total, em segundos, para cada modelo. Nessa tabela, estão consolidados os dados para as instâncias B1, B2 e B3, separadas em cada linha para cada modelo.

Tabela 5 - Resultados dos modelos para Instância "B"

Sigla	Modelo	Instância	Função Objetivo (FO)	Tempo de Construção (s)	Tempo Solver (s)	Tempo Total (s)
lingo_extend	MAE	B1	29.058*	-	-	>1800
		B2	86.693*	-	-	>1800
		B3	61.233*	-	-	>1800
aimms	MAE	B1	29.058	215,28	67,02	282,3
		B2	85.800	334,36	151,72	486,08
		B3	60.350	297,45	129,91	427,36
py_extend_gurob	MAE	B1	29.058	103,17	24,67	127,84
		B2	85.800	514,77	69,16	583,93

		B3	60.350	252,4	28,47	280,87
py_extend_cplex	MAE	B1	29.058	92,55	63,73	156,28
		B2	85.800	580,71	36,18	616,89
		B3	60.350	283,23	24,17	307,4
jl_resum_gurob	MAR	B1	29.058	2,36	13,33	15,69
		B2	85.800	3,96	28,09	32,05
		B3	60.350	3,6	20,16	23,76
jl_resum_cplex	MAR	B1	29.058	2,1	16,1	18,2
		B2	85.800	4,03	54,86	58,89
		B3	60.350	3,74	22,11	25,85
jl_resum_cbc	MAR	B1	29.058	2,16	599,96	602,12
		B2	86.616*	3,29	>1800	>1800
		B3	61.571*	3,8	>1800	>1800
py_resum_gurob	MAR	B1	29.058	5,75	21,07	26,82
		B2	85.800	11,57	48,46	60,03
		B3	60.350	13,1	29,53	42,63
py_resum_cplex	MAR	B1	29.058	6,57	9,12	15,69
		B2	85.800	12,22	27,21	39,43
		B3	60.350	11,75	18,88	30,63
lingo_resum	MAR	B1	29.058*	-	-	>1800
		B2	85.998*	-	-	>1800
		B3	78.412*	-	-	>1800
ts_heurist	-	B1	65.085	86,28	201,73	288,01
		B2	-	-	-	-
		B3	-	-	-	-
sa_heurist	-	B1	76.260	80,76	21,8	102,56
		B2	-	-	-	-
		B3	-	-	-	-
hibrid_heurist	-	B1	52.619	67,05	175,11	242,16
		B2	-	-	-	-
		B3	-	-	-	-

Fonte: Autor

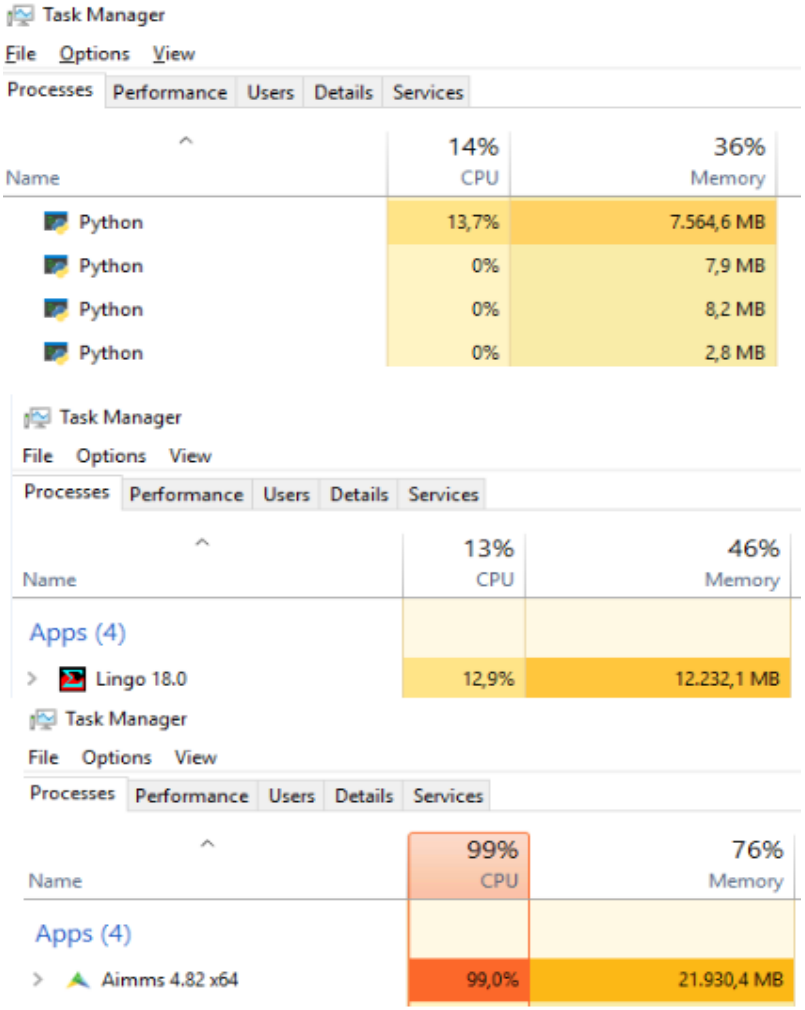
Considera-se que o LINGO não faz distinção entre o tempo gasto para construção do modelo e o tempo do solver para resolução do modelo, dessa forma somente o tempo total é considerado na análise. Além disso, os campos de “Tempo Total (s)” representados com o valor “>1800”, informa que os modelos excederam o tempo limite para solução e a melhor solução factível foi representada com o caractere “*” no campo “Função Objetivo (FO)”. Quando não foi possível resolver o modelo, a FO e o tempo total aparecem com o símbolo ‘-‘.

De acordo com os modelos implementados para a instância B, os que tiveram como base a modelagem MAE, destaca-se o tempo de solução para o cenário B1 do

modelo “py_extend_gurob” e “py_extend_cplex” com resolução em 127,84 e 156,26 segundos, respectivamente. Eles foram capazes de resolver o modelo cerca de duas vezes mais rápido do que o AIMMS. Por outro lado, na instância B2 o modelo no AIMMS apresentou o melhor tempo de solução com 486,08 segundos, contra 583,93 e 616,89 referentes aos modelos “py_extend_gurob” e “py_extend_cplex”.

Porém, o principal gargalo para essa instância no Python foi o tempo de construção do modelo, representando cerca de 90% do tempo de solução, isso ocorreu devido à complexidade e quantidade de variáveis, restrições criadas para resolver esse cenário. A exigência computacional necessária para resolver a instância B2 pode ser verificada na Figura 12.

Figura 12 - Exigência computacional instância B2



Fonte: Autor

Observa-se que em ambos os *softwares* foi necessário utilizar mais de 7GB de memória RAM para leitura e resolução do modelo. Sendo que o desempenho mais

custoso foi no AIMMS, exigindo cerca de 22 GB de RAM, correspondendo à 76% do uso da memória total do servidor.

Em relação a instância B3, o modelo “py_extend_gurobi” obteve o melhor resultado com 280,87 segundos, seguido pelo “py_extend_cplex” com 307,4. Já para o AIMMS, o tempo de solução foi um pouco maior com 427,36 segundos. O LINGO excedeu o tempo limite de 1800 segundos em todos os testes da instância B, chegando ao resultado viável ótimo no B1 e resultados viáveis no B2 e B3 com diferença de 2% em relação ao ótimo.

De outra forma, os modelos com base no MAR apresentaram resultados em tempos computacionais melhores do que os do MAE. Na instância B1, os dois modelos “jl_resum_gurob” e “py_resum_cplex” apresentaram o mesmo tempo de solução com 15,69 segundos para o resultado ótimo. Seguidos pelo “jl_resum_cplex” e “py_resum_gurob” com 18,2 e 26,82 segundos de solução, respectivamente.

Em relação ao cenário B2, o modelo “jl_resum_gurob” e “py_resum_cplex” desempenharam os melhores resultados com resolução do modelo em 32,05 e 39,43 segundos, respectivamente, cerca de 2 vezes mais rápido do que o desempenhado pelos modelos “jl_resum_cplex” e “py_resum_gurob”, com tempo de 58,89 e 60,03, respectivamente. Porém, é válido ressaltar que ambos modelos tiverem desempenho na ordem de 10 vezes mais rápidos se comparados aos modelos MAE.

Para a instância B3, destacam-se os modelos desenvolvidos no Julia, “jl_resum_gurob” e “jl_resum_cplex”, com resolução em 23,76 e 25,85 segundos, respectivamente. Os resultados no Python foram um pouco maiores com os tempos de 30,63 e 42,63 para os modelos “py_resum_cplex” e “py_resum_gurob.

O modelo “lingo_resum” não terminou de rodar as instâncias B dentro do tempo limite de 1800 segundos, encontrando assim o resultado viável ótimo no B1 e resultados viáveis no B2 com 2% e no B3 com 30% de diferença em relação ao ótimo, respectivamente. As heurísticas “ts_heurist”, “sa_heurist” e “hibrid_heurist” foram capazes de solucionar apenas a instância B1 devido à quantidade de variáveis e restrições envolvendo os outros cenários ser maior. Porém, o custo obtido após execução da heurística apresentou um valor maior que duas vezes o ótimo encontrado pela solução MILP. Por outro lado, o modelo “jl_resum_cbc” também atingiu o tempo limite, porém para as instâncias B2 e B3, a diferença foi de 0,1% para o valor ótimo.

5.2.3 Análise dos modelos e instâncias

Dentre os principais modelos apresentados, destacam-se os que possuem uma linguagem de programação de código aberto, Python e Julia, na solução dos modelos. O Julia se destacou com os menores tempos de solução dos modelos MAR nas instâncias “A1”, “A3”, “B1”, “B2” e “B3”, isso ocorre devido ao seu mecanismo de pré-processamento dos dados a partir da segunda execução do modelo, além de sua linguagem ser desenvolvida com base na linguagem de programação C, para um melhor desempenho computacional. Por outro lado, o Python desempenhou os melhores tempos na solução dos modelos MAR para as instâncias “A2” e “B1” e dos modelos MAE nas instâncias “B1” e “B3”, apesar de possuir um gargalo no tempo para construção do modelo.

Ainda assim, os softwares AIMMS e LINGO foram capazes de resolver a maioria das instâncias em tempos computacionais aceitáveis, com destaque para o AIMMS, que nos modelos MAE desempenhou a melhor solução para as instâncias “A1”, “A2”, “A3” e “B2”. Além disso, as heurísticas foram capazes de resolver a maioria dos cenários, exceto os “B2” e “B3” devido à sua complexidade, porém não apresentou valores próximos das funções objetivo nas heurísticas de Busca Tabu e no *Simulated Annealing*, porém obteve um resultado mais aproximado da FO com o algoritmo de heurística híbrido.

Destacam-se também os *solvers* Gurobi e CPLEX na resolução dos problemas, principalmente o Gurobi por ser capaz de encontrar o melhor resultado em menos tempo na maioria das instâncias, porém esses solver são comerciais e no modelo foram utilizadas as licenças acadêmicas ilimitadas. Por outro lado, o *solver* CBC, de código aberto, foi capaz de solucionar a instância “A” e chegar em resultados viáveis nas instâncias “B”.

6 CONCLUSÕES E RECOMENDAÇÕES

Diante da variedade de softwares e sistemas utilizados na resolução de problemas de programação linear inteira de grande complexidade e a falta de estudos científicos comparando a viabilidade e o desempenho dos mesmos, verificou-se a necessidade de desenvolver essa pesquisa para comparar a aplicação de um mesmo modelo em diferentes configurações de *softwares* e *solvers*. Esse tipo de pesquisa gera conhecimento para a escolha do *software* mais indicado para organização no processo de apoio à tomada de decisão, além de contribuir com a literatura científica sobre comparações de diferentes sistemas no mesmo modelo de programação matemática.

Dessa forma, essa pesquisa cumpriu com os objetivos específicos definidos na seção 1.4. Sendo o primeiro de buscar os principais métodos, algoritmos e tecnologias adotados na resolução de problemas de Programação Inteira de grande complexidade que foi apresentado na Revisão Sistemática apresentada no trabalho, na qual os resultados obtidos, estruturados em forma de gráficos e tabelas, permitiu a identificação das áreas de aplicação, o tipo de problema de pesquisa operacional, o método utilizado para resolvê-lo, o solver empregado na solução e a linguagem de programação ou software adotados. Assim, foi possível entender a evolução das ferramentas e técnicas utilizadas nos últimos anos na resolução de programação inteira de grande complexidade.

Após esse levantamento, foi possível verificar a lacuna de pesquisa e o destaque da linguagem de programação Python e dos *solvers* CPLEX e Gurobi na resolução desse tipo de problema. Sendo assim, a pesquisa implementou e resolveu o problema apresentado no projeto *Safety Oversight* da ANAC e por outros trabalhos correlatos no Python, linguagem de programação de código aberto. Depois, os testes e soluções foram realizados para 6 instâncias nos modelos desenvolvidos no LINGO, AIMMS, Julia e Python, utilizando os diferentes *solvers* disponíveis, tais como CPLEX, Gurobi e alguns algoritmos de heurística (*Tabu Search, Simulated Annealing e Hibrid)*.

Posteriormente, realizou-se uma comparação entre todos os modelos implementados nas linguagens Python, Julia, LINGO e AIMMS para as 6 instâncias para identificar o desempenho em tempos computacionais e encontro da solução

ótima ou viável de cada modelo para os diversos cenários. Todos os dados foram consolidados nas Tabelas Tabela 4 e Tabela 5, cumprindo-se assim os objetivos específicos apontados na pesquisa.

Em relação ao objetivo geral sobre a viabilidade da aplicação de uma linguagem de programação de código aberto, no caso específico o Python, na resolução de um problema de Programação Inteira de grande complexidade, observou-se que o Python desempenhou ser uma boa linguagem em conjunto com as bibliotecas “pyomo” e “pandas”, e dos *solvers* CPLEX e Gurobi na solução dos modelos apresentados. Os modelos no Python foram capazes de resolver as instâncias mais simples “A”, com um menor número de origens, destinos e arcos disponíveis para fiscalização, e as mais complexas “B”, na qual os servidores estavam lotados em várias localidades no Brasil e realizavam um rol maior de atividades.

Além disso, verificou-se o ganho em relação ao tempo de solução do problema ao analisar o Modelo 1 - Modelo de Alocação Estendido (MAE) e o Modelo 2 - Modelo de Alocação Resumido (MAR), ambos possuem a mesma solução para os cenários testados, mas variam a estrutura de construção e a quantidade de variáveis e restrições. Com eles, foi possível observar um ganho na resolução do cenário mais complexo “B2” de cerca de 10 vezes em média mais rápido, sendo o melhor modelo MAR executado em 32,05 segundos e o melhor modelo MAE em 486,08. Porém, ressalta-se que o modelo MAE desenvolvido no Projeto *Safety Oversight* permite a realização de transbordo e formação de duplas com não-especialistas, o que não é verificado no modelo MAR.

De acordo com os resultados apresentados no capítulo 5, verificou-se que ambos os sistemas LINGO, AIMMS, Julia e Python são viáveis e capazes de solucionar a maioria dos cenários no problema de programação inteira da alocação de servidores da ANAC, minimizando o custo com passagem aérea nas fiscalizações, com destaque ao AIMMS e às linguagens de programação Julia e Python e do *solver* Gurobi, que foram capazes de solucionar todas as instâncias.

Por fim, é possível verificar que a pesquisa alcançou todos os objetivos propostos e contribuiu com a literatura acadêmica, tendo em vista que uma revisão sistemática da literatura e aplicação prática da comparação entre os diferentes métodos e tecnologias para resolução de problemas de programação inteira de grande complexidade foram apresentados na pesquisa. Sendo assim, essa pesquisa

pode auxiliar e incentivar ensaios futuros sobre a escolha da ferramenta adequada na resolução desses problemas.

6.1 Limitações da pesquisa

Todos os modelos foram testados em um único servidor *Windows* com processador Intel® Xeon® CPU E5-2650 @2.00 GHz (8 CPUs), 32 GB de memória RAM e sistema operacional *Windows Server 2016 Standard 64-bit*, havendo a possibilidade de não ter sido o cenário ideal para rodar todos os modelos e cenários de acordo com a complexidade do problema. Sendo assim, a performance dos modelos e instâncias talvez possam ser melhorados com a utilização de um servidor ou uma máquina mais potente.

Além disso, foram utilizadas as configurações padrão de cada *software* e *solver* utilizados, podendo uma alteração nas opções aumentar a eficiência na resolução dos modelos. Outras limitações se dão a pesquisa ter sido desenvolvida considerando somente o cenário de alocação de servidores da ANAC, dessa forma, não é possível concluir e apontar um melhor *software e solver* para resolução de todos os tipos de problema, dada a complexidade e realidade do universo de modelos que envolvem programação inteira.

6.2 Sugestões de pesquisa futuras

Todos os resultados analisados na Revisão Sistemática foram tratados com referência na escolha da base do Web of Science, sendo assim, existe a possibilidade de expandir essa pesquisa considerando outros bancos de artigos e pesquisas científicas como ScienceDirect, Scopus, Google Scholar, entre outras. Além disso, considerou-se para na revisão o período de 2016 até 2021 para conhecer os trabalhos mais recentes, porém a pesquisa pode ser expandida para um horizonte de tempo maior, considerando os métodos utilizados no período anterior a 2015, inclusive, para resolução desses problemas.

Outra oportunidade para a continuidade da revisão é considerar não somente os problemas determinísticos de PLI de grande porte com uma função objetivo, mas também os critérios utilizados como corte nessa pesquisa. Dessa forma, é possível expandir a pesquisa para modelos de programação inteira não-linear, modelos estocásticos e programação multi-objetivo.

Sugere-se que o mesmo teste seja realizado para outros modelos além da alocação de servidores da ANAC, para que outros estudos analisem a comparação entre as tecnologias utilizadas na solução de problemas de PO. Além disso, o mesmo teste pode ser realizado utilizando diferentes configurações computacionais, a fim de testar o impacto do *hardware* na solução desses tipos de problema.

Finalmente, outros tipos de heurística podem ser explorados no problema de alocação de servidores da ANAC, quando se deseja expandir o número de servidores, fiscalizações, origens e destinos do modelo. Sendo assim, um dos temas de pesquisa futura pode ser verificar a abordagem dos novos tipos de heurística para o problema.

REFERÊNCIAS

ANAC. Agência Nacional de Aviação Civil. Disponível em: <https://www.gov.br/anac/pt-br >. Acesso em: 17 out. 2021.

ANAC. Projeto Safety Oversight, A04 – Relatório Técnico de Resultados do Modelo Inicial, 2019.

BABAEI, Hamed; KARIMPOUR, Jaber; HADIDI, Amin. A survey of approaches for university course timetabling problem. **Computers & Industrial Engineering**, v. 86, p. 43-59, 2015.

BEZANSON, J. et al. Julia: A Fast Dynamic Language for Technical Computing. 2012.

BELFIORE, Patrícia; FÁVERO, Luiz Paulo. **Pesquisa operacional para cursos de engenharia**. Elsevier Brasil, 2013.

BENDERS, J. F. Partitioning Procedures for Solving Mixed-Variable Program-ming Problems, **Numerische Matkematic** 4. SS8, 1962.

BENTO, A. Como fazer uma revisão da literatura: Considerações teóricas e práticas. **Revista JA (Associação Académica da Universidade da Madeira)**, nº 65, ano VII (pp. 42-44), 2012. ISSN: 1647-8975.

BORGES, Luiz Eduardo. **Python para desenvolvedores: aborda Python 3.3**. Novatec Editora, 2014.

CBC, COIN-OR. Disponível em: < https://github.com/coin-or/Cbc >. Acesso em: 17 out. 2021

COLLIS, Jill; HUSSEY, Roger. **Pesquisa em administração: um guia prático para alunos de graduação e pós-graduação**. Bookman, 2005.

COUTO, Lorranna Gabriele Gonçalves. **Viabilidade da utilização de softwares em Programação Matemática Inteira: Análise de um problema de grande porte nos softwares AIMMS e LINGO**. Departamento de Administração, Universidade de Brasília. Brasília, 2020.

CRONIN, Patricia; RYAN, Frances; COUGHLAN, Michael. Undertaking a literature review: a step-by-step approach. **British journal of nursing**, v. 17, n. 1, p. 38-43, 2008.

DIAS, Mateus Carvalho Barros. **PESQUISA OPERACIONAL EM ORGANIZAÇÕES PÚBLICAS BRASILEIRAS: Uma análise qualitativa. Departamento de Administração**, Universidade de Brasília. Brasília, 2019.

DUNNING, I.; HUCHETTE, J.; LUBIN, M. JuMP: A Modeling Language for Mathematical Optimization. **SIAM Review**, v. 59, n. 2, p. 295–320, jan 2017. ISSN 0036-1445.

FACHIN, Odília. **Fundamentos de metodologias**. Saraiva Educação SA, 2001.

FONSECA, George HG et al. Integer programming techniques for educational timetabling. **European Journal of Operational Research**, v. 262, n. 1, p. 28-39, 2017.

FREITAS JÚNIOR, L. S. P. d. **Modelo de transbordo para problemas de designação de inspetores: um estudo de caso na Agência Nacional de Aviação Civil**. 2017.

FROTA, Hugo Faria. **Otimização na alocação de inspetores no setor de aviação civil**. Departamento de Engenharia de Produção, Universidade de Brasília. 2020

GAMERMANN, Ronaldo Wajnberg. **Uma aplicação de programação inteira para otimização do planejamento da fiscalização no âmbito da Agência Nacional de Aviação Civil-ANAC**. 2020.

GASS, Saul I. Public sector analysis and operations research/management science. **Handbooks in operations research and management science**, v. 6, p. 23-46, 1994.

Gurobi Optimizer. Disponível em: < https://www.gurobi.com/ >. Acesso em: 17 out. 2021

Guth, Gabriel Durães; Reis, Silvia Araújo dos. Métodos e Técnicas para Resolução de Modelos de Programação Inteira de Grande Complexidade: Uma Revisão Sistemática da Literatura. **XLI ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO**, 2021.

HART, William E. et al. **Pyomo-optimization modeling in python**. Berlin: Springer, 2017.

HILLIER, F. S.; LIEBERMAN, G. J. **Introdução à Pesquisa Operacional**. 9 ed. Porto Alegre: AMGH, 2013.

HORVATH, William J. Letter to the Editor—Some Thoughts on Operations Research on Municipal Operations. **Journal of the Operations Research Society of America**, v. 3, n. 3, p. 339-340, 1955.

IBM ILOG CPLEX Optimization Studio. Disponível em: < https://www.ibm.com/br-pt/products/ilog-cplex-optimization-studio >. Acesso em: 17 out. 2021.

JULIA. Disponível em: < https://julialang.org/ >. Acesso em: 17 out. 2021.

JOHNES, Jill. Operational research in education. **European Journal of Operational Research**, v. 243, n. 3, p. 683-696, 2015.

LEMARÉCHAL, Claude. Lagrangian relaxation. In: **Computational combinatorial optimization**. Springer, Berlin, Heidelberg, 2001. p. 112-156.

LENO, Magnun. A História do Python. 2014. Diponível em:< http://mindbending.org/pt/ahistoria-do-python >. Acesso em: 17 out. 2021.

LIM-APO, Flávio Augusto. **Alocação de colaboradores qualificados em missões de fiscalização de uma agência reguladora**. Departamento de Engenharia Industrial – PUC-Rio, Rio de Janeiro – RJ, Brasil. 2021.

LONGO, H. J. **Técnicas para Programação Inteira e Aplicações em Problemas de Roteamento de Veículos**. Departamento de Informática, PUC-Rio, Rio de Janeiro-RJ, Brasil. 2004.

LOPES, Ana Lúcia Mendes; FRACOLLI, Lislaine Aparecida. Revisão sistemática de literatura e metassíntese qualitativa: considerações sobre sua aplicação na pesquisa em enfermagem. **Texto & Contexto-Enfermagem**, v. 17, n. 4, p. 771-778, 2008.

LOPES, T. F. **ANÁLISE DE SENSIBILIDADE EM MODELOS DE PROGRAMAÇÃO LINEAR COM SOLUÇÃO DEGENERADA: Revisão sistemática e estudo aplicado na Agência de Aviação Civil**. Departamento de Administração, Universidade de Brasília. 2019.

Open Source Initiative. Disponível em: < https://opensource.org/osd >. Acesso em: 17 out. 2021

PINHEIRO, Raylene dos Santos. Modelo matemático para apoio a decisão no processo de designação de inspetores em missões de fiscalização na ANAC. 2018.

REIS, S. A. d.; CELESTINO, V. R. R. Um modelo matemático para alocação de pessoas na gestão da capacidade em serviços públicos. 2018.

RIBEIRO, Celso C. et al. Preface to the special issue on matheuristics and metaheuristics. **International Transactions in Operational Research**, v. 27, n. 1, p. 5-8, 2020.

SANTOS, Antonio Raimundo dos. Metodologia científica: a construção do conhecimento. 3. ed. **Rio de Janeiro: DP&A editora**, 2000.

SANTOS, Wellington Furtado; SIMONETTO, Eugenio de Oliveira; FERREIRA, David Luiz Silva. Pesquisa Operacional aplicada à Administração: um estudo sobre os artigos internacionais publicados entre 1993 e 2013. **Revista de Administração da Universidade Federal de Santa Maria**, v. 10, n. 5, p. 844-853, 2017.

SILVA JUNIOR, H. S. d. **Estudo Comparativo de Meta-Heurísticas: Otimização do Modelo de Alocação na Agência Nacional de Aviação Civil**. Departamento de Ciência da Computação, Universidade de Brasília. 2021.

SILVA, M. A. D. **Escalamento de Inspetores de Aviação Civil no Brasil: Formalização do Problema e Metodologia de Resolução**. 2018. f. 110.Tese (Doutorado).

SPERANDIO, Mauricio; COELHO, Jorge. Métodos de programação inteira aplicados ao planejamento da automação de sistemas de manobra em redes de distribuição. **Sba: Controle & Automação Sociedade Brasileira de Automatica**, v. 21, n. 5, p. 463-476, 2010.

APÊNDICES

Apêndice A – Códigos do modelo MAE

LINGO – Modelo Estendido (MAE):

```
Sets:
        nos/@ole()/;
        origem(nos)/@ole()/:OrigemAtiva;
        destino(nos)/@ole()/:DestinoAtivo;
        Arco(nos,nos)/@ole()/:Custo,tempo;

        Periodo/1/;
        missao/@ole()/;
        Pessoas/@ole()/;
        grupo/@ole()/:duracao,equipe,GrupoAtivo;
        PessoaAtiva(Pessoas);

        DATIVO(destino)|DestinoAtivo(&1)#GE#1;
        OATIVA(origem)|OrigemAtiva(&1)#GE#1;
        NOSATIVOS(nos)| @in(OATIVA, nos(&1)) #OR# @in(DATIVO, nos(&1)):;
        ARCOA(nosativos,nosativos);
        ARCOATIVO(nosativos,nosativos);

        TPe(pessoas,periodo):tempoutilizado,disponibilidade;
        pessoaquebrada(pessoas,grupo)/@ole()/;
        PO(pessoas,origem)/@ole()/;
        NEPO(pessoas,origem)|#not#@in(PO,&1,&2);
        PGM(pessoas,grupo,missao);

        ExisteMissao(missao,grupo)/@ole()/;
        GRATIVO(grupo)|GrupoAtivo(&1)#GE#1;
        PGATIVO(pessoas,grupo);

        moradia(pgativo,oativa):oferta;
        dem(periodo,missao, grupo, destino)/@ole()/:demanda;

    ArcoMi(periodo,arcoativo,PGM):Alocado;
        NoPeGr(periodo,missao,grupo,destino,pessoas):Atendida;
        VarInt(periodo,arcoativo,pessoas):AlocadoGeral;


Endsets

Data:

        disponibilidade,duracao,equipe,custo,tempo,grupoativo,destinoativo,origemativa = @ole();

Enddata

Submodel Alocacao:
[FO]

        MIN = @SUM(VarInt(t,i,j,p): AlocadoGeral(t,i,j,p)*2*Custo(i,j));
```

```
	@for(dem(t,k,gr,d): [demmissao] demanda(t,k,gr,d) = 1);

	@for(PO(p,o):@for(moradia(p,gr,o)|p #LE# @index(pessoas, Dummy3): [ofertaexist]
oferta(p,gr,o) = 100));
	@for(PO(p,o):@for(moradia(p,gr,o)|p #GT# @index(pessoas, Dummy3): [ofertaexist1]
oferta(p,gr,o) = 10));
	@for(NEPO(p,o):@for(moradia(p,gr,o): [ofertanaoexist] oferta(p,gr,o) = 0));

	@for(PGATIVO(p,gr): @for(oativa(oa): @for(dativo(e):
@for(periodo(t):[oft]@sum(missao(k)|@in(ArcoMi,t,oa,e,p,gr,k):alocado(t,oa,e,p,gr,k)) <=
oferta(p,gr,oa)))));

	@for(nopegr(t,k,gr,d,p):[balanco] @sum(nosativos(h)|@in(ArcoMi,t,h,d,p,gr,k):
alocado(t,h,d,p,gr,k)) = atendida(t,k,gr,d,p) +
@sum(nosativos(h)|@in(ArcoMi,t,d,h,p,gr,k):Alocado(t,d,h,p,gr,k)));


@for(PGM(p,gr,k):@for(periodo(t):@for(dativo(e):[fixa]@sum(nosativos(h)|@in(ArcoMi,t,h,e,p,gr,k):
alocado(t,h,e,p,gr,k))>= @sum(nosativos(h)|@in(ArcoMi,t,e,h,p,gr,k):Alocado(t,e,h,p,gr,k)))));

	@for(dem(t,k,gr,d):[equi]@sum(nopegr(t,k,gr,d,p): atendida(t,k,gr,d,p))=
Equipe(gr)*demanda(t,k,gr,d));

	@for(periodo(t):@for(pessoas(p):[temp]
@sum(nopegr(t,k,gr,d,p):atendida(t,k,gr,d,p)*duracao(gr)) + @sum(VarInt(t,i,j,p):
Alocadogeral(t,i,j,p)*2*tempo(i,j)) = TempoUtilizado(p,t)));

	@for(TPE(p,t): [dispontempo] TempoUtilizado(p,t) <= disponibilidade(p,t));

	@for(ArcoMi(t,i,j,p,gr,k):[aloc] alocadogeral(t,i,j,p) >= alocado(t,i,j,p,gr,k));


	@for(NoPeGr(t,k,gr,d,p):@bin(atendida(t,k,gr,d,p)));!Restrição de binario;
	@for(VarInt(t,i,j,p):@bin(alocadogeral(t,i,j,p)));!Restrição de binario;

Endsubmodel

calc:

	@set('terseo', 1); !Reduce output level;
	@set('loopop', 1); !Loop optimization;

	@for(pessoaquebrada(p,gr): @for(existemissao(k,gr): @insert(PGM, p, gr, k)));

	@for(pessoaquebrada(p,gr): @for(dem(t,k,gr,d): @insert(nopegr,t,k,gr,d,p)));

	@for(pessoaquebrada(p,gr): @for(grativo(gr): @insert(pgativo,p,gr)));

	@for(arco(i,j): @for(arcoa(i,j): @insert(arcoativo,i,j)));

	@DIVERT(file.txt');
	@solve(Alocacao);
	@write('Periodo origem destino pessoa alocadogeral',@NEWLINE(1));
	@writefor (VarInt(t,i,j,p) | alocadogeral(t,i,j,p) #GT# 0: Varint(t,i,j,p), ' ', Alocadogeral(t,i,j,p),
@NEWLINE(1)) ;
	@write('Periodo missao atividade destino pessoas Atendida',@NEWLINE(1));
	@writefor (NoPeGr(t,k,gr,j,p) | atendida(t,k,gr,j,p) #GT# 0: NoPeGr(t,k,gr,j,p), ' ',
atendida(t,k,gr,j,p), @NEWLINE(1)) ;
	@write('Pessoa Periodo Tempo Utilizado',@NEWLINE(1));
```

```
        @writefor (TPe(p,t) | TempoUtilizado(p,t) #GT# 0: TPe(p,t), ' ', TempoUtilizado(p,t),
@NEWLINE(1)) ;
        @DIVERT();

endcalc
```

AIMMS – Modelo Estendido (MAE):

```
Set NOS {Index: I, J;}

Set ORIGEM {SubsetOf: NOS;Index: O;}

Set DESTINO { SubsetOf: NOS; Index: D; }

Set NosAtivos { SubsetOf: NOS; Index: n, n1; }

Set OrigemAtiva { SubsetOf: NosAtivos; Index: oa; }

Set DestinoAtivo { SubsetOf: NosAtivos; Index: da; }

Set PESSOAS { Index: P; }

Set GRUPOS { Index: GR; }

Set MISSAO { Index: K; }

Set PERIODO { SubsetOf: Integers; Index: T; Definition: DATA{1}; }

Parameter ARCO { IndexDomain: (I,J); }

Parameter DISPONIBILIDADE { IndexDomain: (T,P); }

Parameter EQUIPE { IndexDomain: (GR); }

Parameter DURACAO { IndexDomain: (GR); }

Parameter DEMANDA { IndexDomain: (T,D,GR,K); }

Parameter PessoaGrupo { IndexDomain: (P,GR); }

Parameter PessGRupoK { IndexDomain: (P,GR,K); Definition: sum((t,d),DEMANDA(T, D, GR,
K))$PessoaGrupo(P, GR); }

Parameter OFERTA {IndexDomain: (T,O,P,GR); }

Parameter CUSTO {IndexDomain: (I,J); }

Parameter TEMPO {IndexDomain: (I,J); }

Variable ALOCADOGERAL {IndexDomain:T,n,da,P)$ARCO(n,da)$sum((gr,k),PessGRupoK(P, GR,
K));
Range: binary;}

Variable ALOCADO {IndexDomain: (T,n,da,P,GR,K)$ ARCO(n, da)$ PessGRupoK(P, GR, K);
Range: nonnegative;}

Variable TEMPOUTILIZADO {IndexDomain: (T,P);
Range: nonnegative;}

Variable ATENDIDA {IndexDomain: (T,P,da,GR,K)$DEMANDA(T, da, GR, K)$PessoaGrupo(P, GR);
Range: binary;}

Variable FUNCAOOBJETIVO {Range: free;
Definition: SUM((T,n,da,P),ALOCADOGERAL(T, n, da, P)*2*CUSTO(n, da)); }
```

MathematicalProgram FO {Objective: FUNCAOOBJETIVO;Direction: minimize;Constraints: AllConstraints;Variables: AllVariables;Type:Automatic; }

Constraint R1 {IndexDomain: (T,OA,DA,P,GR)$sum((k),PessGRupoK(P, GR,K)); Definition: SUM((K),ALOCADO(T, OA, DA, P, GR, K))<=OFERTA(T, OA, P, GR); }

Constraint R2 {IndexDomain: (T,P,Da,GR,K)$PessGRupoK(P, GR, K);Definition: SUM((n),ALOCADO(T, n, Da, P, GR, K))=(ATENDIDA(T, P, Da, GR, K)+SUM((n1),ALOCADO(T, Da, n1, P, GR, K))); }

Constraint R3 {IndexDomain: (T,Da,P,GR,K)$PessGRupoK(P, GR, K);Definition: SUM((n1),ALOCADO(T, Da, n1, P, GR, K))<=SUM((n),ALOCADO(T, n, Da, P, GR, K)); }

Constraint R4 {IndexDomain: (T,Da,GR,K)$DEMANDA(T, Da, GR, K);Definition: SUM((P),ATENDIDA(T, P, Da, GR, K))=(EQUIPE(GR)*DEMANDA(T, Da, GR, K)); }

Constraint R5 {IndexDomain: (T,P)$sum((gr,k),PessGRupoK(P, GR, K));Definition: SUM((Da,GR,K),ATENDIDA(T, P, Da, GR, K)*DURACAO(GR))+ SUM((N,DA),(ALOCADOGERAL(T,N,DA,P)*2*TEMPO(N,DA)))=TEMPOUTILIZADO(T, P); }

Constraint R6 {IndexDomain: (T,P)$sum((gr,k),PessGRupoK(P, GR, K));Definition: TEMPOUTILIZADO(T, P)<=DISPONIBILIDADE(T, P); }

Constraint R7 { IndexDomain: (T,n,Da,P,GR,K)$ ARCO(n, Da)$PessGRupoK(P, GR, K); Definition: ALOCADOGERAL(T, n, Da, P)>=ALOCADO(T, n, Da, P, GR, K); }

PYTHON – Modelo Estendido (MAE):

```
from pyomo.core.base import misc
import pyomo.environ as pyo
from pyomo.environ import *
from pyomo.opt import SolverFactory
from numpy.lib.function_base import select
import pandas as pd
import time

#---------------- LEITURA DOS DADOS DO EXCEL (DISPONIBILIDADE, ARCOS, ATIVIDADE, OFERTA, DEMANDA) ----------
inic_cm = time.time()
ef = pd.ExcelFile('file.xlsx')
disponibilidade = pd.read_excel(ef, 'Disponibilidade - PREENCHER', header=1, usecols=['NOME', 'UF',  'DISPONIBILIDADE'])
arcos           = pd.read_excel(ef, 'Arcos',                       header=1, usecols=['ORIGEM_ARCO', 'DESTINO_ARCO', 'CUSTO', 'TEMPO'])
atividade       = pd.read_excel(ef, 'Atividades',                  header=1, usecols=['GRUPO_A', 'DURACAO', 'EQUIPE'])
oferta          = pd.read_excel(ef, 'Atividades',                  header=1, usecols=['SERVIDOR',  'GRUPO_P'])
nos             = pd.read_excel(ef, 'Arcos',                       header=1, usecols=['NOS', 'ORIGEM_NO', 'DESTINO_NO'])
demanda         = pd.read_excel(ef, 'Demanda - PREENCHER',         header=1, usecols=['PERIODO', 'MISSAO', 'ATIVIDADE', 'AEROPORTO' ])

#---------------- FAZENDO A LIMPEZA DOS DADOS QUE NÃO SERÃO UTILIZADOS NO MODELO POR NÃO APARECER NA DEMANDA ----------

disponibilidade  = disponibilidade[(disponibilidade['DISPONIBILIDADE'] > 0)]
oferta           = oferta.where((oferta['GRUPO_P'].isin(demanda['ATIVIDADE'].values) == True)).dropna()
oferta           = oferta.where((oferta['SERVIDOR'].isin(disponibilidade['NOME'].values) == True)).dropna()
disponibilidade  = disponibilidade.where((disponibilidade['NOME'].isin(oferta['SERVIDOR'].values) == True)).dropna()
arcos            = arcos.where((arcos['ORIGEM_ARCO'].isin(disponibilidade['UF'].values) == True )).dropna()
arcos            = arcos.where((arcos['DESTINO_ARCO'].isin(demanda['AEROPORTO'].values) == True )).dropna()
atividade        = atividade.where(atividade['GRUPO_A'].isin(demanda['ATIVIDADE'].values) == True).dropna()

#---------------- CRIANDO AS LISTAS DE PESSOA, GRUPO, ORIGEM, DESTINO ATIVOS ----------

pessoa = disponibilidade['NOME'].unique()
grupo = atividade['GRUPO_A'].dropna().values
```

```
origem = disponibilidade['UF'].unique()
destino = demanda['AEROPORTO'].unique()
periodo = [1]
missao = list(range(1,len(demanda)+1))

#---------------- TRANSFORMANDO A TABELA DE DEMANDAS EM UM DICIONÁRIO ----------
--

existe_missao = demanda[['MISSAO', 'ATIVIDADE']]
nopegr = demanda[['PERIODO', 'MISSAO', 'ATIVIDADE', 'AEROPORTO']]
demanda = demanda.set_index(['PERIODO', 'MISSAO', 'ATIVIDADE', 'AEROPORTO'])
demanda = demanda.to_dict('index')
demanda = dict.fromkeys(demanda,1)

#---------------- CRIANDO O DICIONÁRIO DE PERIODO/MISSAO/GRUPO/DESTINO/PESSOA
ATIVOS ----------

nopegr = pd.merge(nopegr, oferta[['SERVIDOR','GRUPO_P']], how = 'inner', left_on =
'ATIVIDADE', right_on = 'GRUPO_P').drop(columns= 'GRUPO_P')
nopegr = nopegr.set_index(['PERIODO', 'MISSAO', 'ATIVIDADE', 'AEROPORTO', 'SERVIDOR'])
nopegr = nopegr.to_dict('index')

#---------------- CRIANDO O DICIONÁRIO DE PESSOA/GRUPO ATIVO ----------

pg_ativo = oferta.set_index(['SERVIDOR','GRUPO_P'])
pg_ativo = pg_ativo.to_dict('index')

#---------------- CRIANDO O DICIONÁRIO DE PESSOA/GRUPO/UF ATIVO ----------

pgo_ativo = oferta[['SERVIDOR','GRUPO_P']]
pgo_ativo = pd.merge(pgo_ativo, disponibilidade[['NOME','UF']], how = 'inner', left_on =
'SERVIDOR', right_on ='NOME', suffixes = ('UF'))
pgo_ativo = pgo_ativo[['NOME', 'GRUPO_P', 'UF']]
pgo_ativo = pgo_ativo.set_index(['NOME', 'GRUPO_P', 'UF'])
pgo_ativo = pgo_ativo.to_dict('index')
pgo_ativo = dict.fromkeys(pgo_ativo,10)

for nome, value in pgo_ativo.items():
    if 'Dummy' in nome[0]:
        pgo_ativo[nome] = 100

#---------------- CRIANDO O DICIONÁRIO DE PESSOA/GRUPO/MISSAO ATIVO ----------

pgm = oferta[['SERVIDOR','GRUPO_P']]
pgm = pd.merge(pgm, existe_missao, how = 'inner', left_on = 'GRUPO_P', right_on =
'ATIVIDADE').drop(columns = ['ATIVIDADE'])
pgm = pgm.set_index(['SERVIDOR', 'GRUPO_P', 'MISSAO'])
pgm = pgm.to_dict('index')
```

```
#---------------- CRIANDO O DICIONÁRIO DAS ATIVIDADES (DURAÇÃO E EQUIPE) ATIVAS ----------

dado_grupo = atividade.set_index(['GRUPO_A'])
dado_grupo = dado_grupo.to_dict('index')

#---------------- CRIANDO O DICIONÁRIO DOS ARCOS (CUSTO E TEMPO) ----------------

arcos = arcos.set_index(['ORIGEM_ARCO','DESTINO_ARCO'])
arcos = arcos.to_dict('index')

#---------------- CRIANDO O DICIONÁRIO DE DISPONIBILIDADE DE CADA PESSOA ----------
--

disponibilidade = disponibilidade.set_index(['NOME'])
disponibilidade = disponibilidade.to_dict('index')

#---------------- CRIANDO O MODELO NA BIBLIOTECA PYOMO -----------------

model = ConcreteModel()

model.oferta = Param(pessoa, grupo, origem, initialize = pgo_ativo, default = 0)

model.alocado_geral = Var(periodo, arcos, pessoa, domain = NonNegativeReals, bounds = (0,1))
model.alocado = Var(periodo, arcos, pgm, domain = PositiveReals)
model.atendida = Var(nopegr, domain = Binary)
model.tempo_utilizado = Var(periodo, pessoa, domain = PositiveReals)

def FuncaoObjetivo(model):
    return sum(model.alocado_geral[t,i,j,p] * arcos[i,j]['CUSTO'] * 2
        for t in periodo
        for i,j in arcos
        for p in pessoa
        if i == disponibilidade[p]['UF'])

model.obj = Objective(rule = FuncaoObjetivo, sense = minimize)

def constr_oferta(model, p, g, i):
    return sum(model.alocado[t,o,d,a,gr,m] for t in periodo for o,d in arcos if i == o for a,gr,m in
pgm if p == a and g == gr) <= model.oferta[p,g,i]

model.constr_o = Constraint(pgo_ativo, rule = constr_oferta)

def constr_atendida(model, t, k, g, j, p):
    return sum(model.alocado[t,o,d,p,g,k] for o,d in arcos if o == disponibilidade[p]['UF'] if d == j)
== model.atendida[t,k,g,j,p]

model.constr_at = Constraint(nopegr, rule = constr_atendida)
```

```
def constr_demanda(model, t, k, g, j):
    return sum(model.atendida[t,m,gr,d,p] for t, m, gr, d, p in nopegr if m == k and gr == g and d ==
j) == dado_grupo[g]['EQUIPE']*demanda[t,k,g,j]

model.constr_d = Constraint(demanda, rule = constr_demanda)

def constr_tempo(model, t, p):
    return sum(model.atendida[t,k,g,j,a] * dado_grupo[g]['DURACAO'] for t,k,g,j,a in nopegr if a ==
p) \
        + sum(model.alocado_geral[t,i,j,p] * arcos[i,j]['TEMPO'] * 2 for i,j in arcos if i ==
disponibilidade[p]['UF']) \
        == model.tempo_utilizado[t,p]

model.constr_t = Constraint(periodo, pessoa, rule = constr_tempo)

def constr_disponibilidade (model, t, p):
    return model.tempo_utilizado[t,p] <= disponibilidade[p]['DISPONIBILIDADE']

model.constr_dis = Constraint(periodo, pessoa, rule = constr_disponibilidade)

def constr_alocacao (model, t,i,j,p,g,k):
    if i == disponibilidade[p]['UF']:
        return model.alocado_geral[t,i,j,p] >= model.alocado[t,i,j,p,g,k]
    else:
        return Constraint.Skip

model.constr_al = Constraint(periodo, arcos, pgm, rule = constr_alocacao)

term_cm = time.time()

solver = SolverFactory('cplex').solve(model, timelimit = 1800).write()
```

Apêndice B – Códigos do modelo MAR

LINGO – Modelo Resumido (MAR):

```
sets:
pessoas:disponibilidade;
origens;
missao:duracao, equipe;
destinos;
arcos(origens,destinos):tempo_viagem, custo;
aloc(pessoas, origens, destinos):alocado;
auxiliar(pessoas, origens, missao, destinos);
aloc_ativ(pessoas, origens, missao):alocado_atividade;
endsets

data:
pessoas, origens, missao, destinos, arcos, tempo_viagem, aloc, aloc_ativ, disponibilidade, duracao,
equipe, custo, auxiliar = @ole();
!BigM = 1000;
enddata

Submodel Alocacao:
        min = @sum(aloc(p,o,d):2*custo(o,d)*alocado(p,o,d));

        @for(missao(m):@sum(aloc_ativ(p,o,m):alocado_atividade(p,o,m)) >= equipe(m));

        @for(pessoas(p):@sum(aloc(p,o,d):@roundup(2*tempo_viagem(o,d),0)*alocado(p,o,d)) +
@sum(aloc_ativ(p,o,m):@roundup(duracao(m),0)*alocado_atividade(p,o,m)) <= disponibilidade(p));

        @for(auxiliar(p,o,m,d):alocado(p,o,d) >= alocado_atividade(p,o,m));

        @for(aloc(p,o,d):@bnd(0,alocado(p,o,d),1));
        @for(aloc_ativ(p,o,m):@bin(alocado_atividade(p,o,m)));
Endsubmodel

calc:
        @set('loopop', 1);
        @solve(Alocacao);

endcalc
```

Julia – Modelo Resumido (MAR):

```
#import Pkg
#Pkg.add("XLSX")
#Pkg.add("DataFrames")
#Pkg.add("JuMP")
#Pkg.add("Gurobi")
#Pkg.add("CPLEX")
#Pkg.add("Cbc")

using XLSX, DataFrames, JuMP, CPLEX, Gurobi, Cbc

@time begin
    arquivo = XLSX.readxlsx("file.xlsx")

    DEM = arquivo["DADOS!A2:E51"]
    PD = arquivo["DADOS!G2:H128"]
    ORIGENS = arquivo["DADOS!J2:J17"]
    DESTINOS = arquivo["DADOS!K2:K31"]
    ARCOS = arquivo["DADOS!M2:P474"]
    ALOC = arquivo["DADOS!R2:T2553"]
    ALOC_ATIV = arquivo["DADOS!V2:Y3912"]

    struct ALOCADO # Variavel de decisão
        p::String # Pessoa
        o::String # Origem
        d::String # Destino
    end

    struct ALOCADO_ATIVIDADE # Variavel de decisão
        p::String # Pessoa
        o::String # Origem
        d::String # Destino
        m::Int64 # Missao
    end

    struct TRANSPORTE #
        o::String # Origem
        d::String # Destino
    end

    ALOCADO_GERAL = [ALOCADO(ALOC[i,1], ALOC[i,2], ALOC[i,3]) for i in 1:size(ALOC,1)];

    ALOCADO_MISSAO = [ALOCADO_ATIVIDADE(ALOC_ATIV[i,1], ALOC_ATIV[i,2],
ALOC_ATIV[i,3], parse(Int64,ALOC_ATIV[i,4])) for i in 1:size(ALOC_ATIV,1)];

    ARCO_TRANSPORTE = [TRANSPORTE(ARCOS[i,1], ARCOS[i,2]) for i in 1:size(ARCOS,1)];
```

```
    CUSTO_TRANSPORTE = Dict{TRANSPORTE, Float64}(TRANSPORTE(ARCOS[i,1],
ARCOS[i,2]) => ARCOS[i,3] for i in 1:size(ARCOS,1));
    TEMPO_TRANSPORTE = Dict{TRANSPORTE, Float64}(TRANSPORTE(ARCOS[i,1],
ARCOS[i,2]) => ARCOS[i,4] for i in 1:size(ARCOS,1));
    DISPONIBILIDADE = Dict{String, Float64}(PD[i,1] => PD[i,2] for i in 1:size(PD,1));

    TEMPO_MISSAO = Dict{Int, Float64}(DEM[i,1] => DEM[i,4] for i in 1:size(DEM,1));

    m = Model(Cbc.Optimizer);

    set_time_limit_sec(m, 600);

    @variable(m, X[ALOCADO_GERAL], binary=true);
    @variable(m, Y[ALOCADO_MISSAO], binary=true);

    @objective(m, Min, sum(2*X[i]*CUSTO_TRANSPORTE[TRANSPORTE(i.o,i.d)] for i in
ALOCADO_GERAL));

    @constraint(m, [d in 1:size(DEM,1)], sum(Y[j] for j in ALOCADO_MISSAO if j.m == DEM[d,1])
== DEM[d,5]);

    @constraint(m, TEMPO_RESTRICAO[pp in PD[:,1]], sum(Y[j]*TEMPO_MISSAO[j.m] for j in
ALOCADO_MISSAO if j.p == pp) +
            sum(2 * X[i] * TEMPO_TRANSPORTE[TRANSPORTE(i.o,i.d)] for i in ALOCADO_GERAL
if i.p == pp) <= DISPONIBILIDADE[pp]);

    @constraint(m, [j in ALOCADO_MISSAO], X[ALOCADO(j.p, j.o, j.d)] >= Y[j]);

    optimize!(m)
end

open("file.txt" , "w") do f
    println(f, solution_summary(m))
    println(f, "RESULTADO INTEIRO = ", objective_value(m))
    println(f,"\t")
    println(f, "VARIABLES = ", num_variables(m))
    println(f,"\t")
    println(f, "CONSTRAINTS = ", num_constraints(m, AffExpr, MOI.EqualTo{Float64}) +
num_constraints(m, AffExpr, MOI.GreaterThan{Float64}) + num_constraints(m, AffExpr,
MOI.LessThan{Float64}))
end

println("RESULTADO INTEIRO = ", objective_value(m))
println("VARIABLES = ", num_variables(m))
println("CONSTRAINTS = ", num_constraints(m, AffExpr, MOI.EqualTo{Float64}) +
num_constraints(m, AffExpr, MOI.GreaterThan{Float64}) + num_constraints(m, AffExpr,
MOI.LessThan{Float64}))
```

PYTHON – Modelo Resumido (MAR):

```
from pyomo.core.base import misc
from pyomo.core.base.piecewise import Bound
import pyomo.environ as pyo
from pyomo.environ import *
from pyomo.opt import SolverFactory
from numpy.lib.function_base import select
import pandas as pd
import numpy as np
import time

inic_cm = time.time()

ef = pd.ExcelFile('file.xlsx')

DEM = pd.read_excel(ef, 'DADOS', header = 0, usecols = ['MISSAO1', 'ATIVIDADE',
'DESTINO1', 'DURACAO', 'EQUIPE'], converters = {'MISSAO1': int}).dropna()
PD = pd.read_excel(ef, 'DADOS', header = 0, usecols = ['PESSOA1',
'DISPONIBILIDADE']).dropna()
ORIGENS = pd.read_excel(ef, 'DADOS', header = 0, usecols = ['ORIGENS']).dropna()
DESTINOS = pd.read_excel(ef, 'DADOS', header = 0, usecols = ['DESTINOS']).dropna()
ARCOS = pd.read_excel(ef, 'DADOS', header = 0, usecols = ['ORIGEM2', 'DESTINO2', 'CUSTO',
'TEMPO']).dropna()
ALOC = pd.read_excel(ef, 'DADOS', header = 0, usecols = ['PESSOA3', 'ORIGEM3',
'DESTINO3']).dropna()
ALOC_ATIV = pd.read_excel(ef, 'DADOS', header = 0, usecols = ['PESSOA4', 'ORIGEM4',
'DESTINO4', 'MISSAO4'], converters = {'MISSAO4': int}).dropna()

MISSAO = DEM['MISSAO1'].unique()
PESSOA = PD['PESSOA1'].unique()

ARCOS = ARCOS.set_index(['ORIGEM2', 'DESTINO2'])
ARCOS = ARCOS.to_dict('index')

DISPONIBILIDADE = PD.set_index(['PESSOA1'])
DISPONIBILIDADE = DISPONIBILIDADE.to_dict('index')

DEMANDA_MISSAO = DEM[['MISSAO1', 'DURACAO','EQUIPE']]
DEMANDA_MISSAO = DEMANDA_MISSAO.set_index(['MISSAO1'])
DEMANDA_MISSAO = DEMANDA_MISSAO.to_dict('index')

ALOC = ALOC.set_index(['PESSOA3', 'ORIGEM3', 'DESTINO3'])
ALOC = ALOC.to_dict('index')

AUXILIAR = ALOC_ATIV.set_index(['PESSOA4', 'ORIGEM4', 'DESTINO4', 'MISSAO4'])
AUXILIAR = AUXILIAR.to_dict('index')

ALOC_ATIV = ALOC_ATIV[['PESSOA4', 'ORIGEM4', 'MISSAO4']]
```

```
ALOC_ATIV = ALOC_ATIV.set_index(['PESSOA4', 'ORIGEM4', 'MISSAO4'])
ALOC_ATIV = ALOC_ATIV.to_dict('index')

model = ConcreteModel()

model.alocado_geral = Var(ALOC, domain = Reals, bounds = (0,1))
model.alocado_missao = Var(ALOC_ATIV, domain = Binary)

def FuncaoObjetivo(model):
    return sum(model.alocado_geral[p,o,d] * ARCOS[o,d]['CUSTO'] * 2
        for p,o,d in ALOC)

model.obj = Objective(rule = FuncaoObjetivo, sense = minimize)

def constr_demanda(model, m):
    return sum(model.alocado_missao[p,o,k] for p, o, k in ALOC_ATIV if k == m) ==
DEMANDA_MISSAO[m]['EQUIPE']

model.constr_d = Constraint(MISSAO, rule = constr_demanda)

def constr_tempo(model, p):
    return sum(model.alocado_missao[a,o,m]*DEMANDA_MISSAO[m]['DURACAO'] for a, o, m in
ALOC_ATIV if a == p) \
        + sum(2*model.alocado_geral[a,o,d] * ARCOS[o,d]['TEMPO'] for a,o,d in ALOC if a == p) \
            <= DISPONIBILIDADE[p]['DISPONIBILIDADE']

model.constr_t = Constraint(PESSOA, rule = constr_tempo)

def constr_alocacao (model, p,o,d,m):
    return model.alocado_geral[p,o,d] >= model.alocado_missao[p,o,m]

model.constr_al = Constraint(AUXILIAR, rule = constr_alocacao)

solver = SolverFactory('gurobi').solve(model, tee = True, timelimit = 1800).write()
```

Apêndice C – Códigos do modelo Heurística

Busca Tabu:

```
import pandas as pd
import csv
from numpy import array
import numpy as np
import math
import random
import warnings
from random import randint
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")


def validaFO(a0, funcao_objetivo):
    y,x = np.where(a0 == 1)
    if(sum(funcao_objetivo[y,x]) == 5):
        return True
    else:
        return False


def valida(a0):
    if (len(np.where(np.sum(a0, axis=0) > 1)[0]) > 1):
        return False
    else:
        return True


def isInList(a0, tabuList):
    r = False
    for i in tabuList:
        if(np.array_equal(i, a0) == True):
            r = True
            break
        else:
            r = False
    return r

def getNbhd (a0,funcao_objetivo,passo):
    x = len(a0[0,:])
    aList = []
    p = -1
    a1 = a0.copy()
    if(int(x/passo) == 0):
        k = 1
```

```
    else:
        k = int(x/passo)
    for i in range (0, k):
        fator = random.randint(1, 5)
        for j in range (0,fator):
            a0 = salta(a0,1,funcao_objetivo,(passo*i), passo)
            a1 = salta(a1,1,funcao_objetivo,(passo*i), passo)
        aList.append(salta(a0,1,funcao_objetivo,(passo*i), passo))
        aList.append(salta(a1,p,funcao_objetivo,(passo*i), passo))

    return aList


def salta(a0, direcao,FO,posicao_passo,tamanho_passo):
    a_aux = a0.copy()
    if(tamanho_passo+posicao_passo < len(FO[0,:] )):
        final = tamanho_passo+posicao_passo
    else:
        final = len(FO[0,:] )

    for i in range (posicao_passo, final):
        #fazer o recorte do salto em cima dos servidores_aptos tambem
        servidores_aptos = np.where(FO[:,i] == 1)[0]
        try:
            indice_atual = np.where(a0[:,i] == 1)[0][0]
            servidor_salto = np.where(servidores_aptos == indice_atual)[0][0]
            if((servidor_salto - direcao) == len(servidores_aptos)):
                novo_indice = servidores_aptos[0]
            else:
                novo_indice = servidores_aptos[servidor_salto - direcao]

            a_aux[indice_atual,i] = 0
            a_aux[novo_indice,i] = 1
        except:
            l = True
    return  a_aux


def geraIndividuo(funcao_objetivo, disponibilidade,disp_aux,inspecoes, demanda_aux,arcos):
    funcao_objetivo2 =  np.zeros((len(disponibilidade),len(demanda_aux)), dtype=int)
    custo = 0
    disponibilidade_copia = disponibilidade.copy()
    for i in range (len(demanda)):
        indice_aux  =  random.choice(np.where(funcao_objetivo[:,i] == 1)[0])
        origem      =
disponibilidade_copia.where(disponibilidade_copia['NOME'].isin([disp_aux[indice_aux]]) ==
True).dropna()
        origem      = origem['UF'].values[0]
        destino     = demanda['AEROPORTO'].values[i]
```

```
        disponibilidade_atual =
disponibilidade_copia['DISPONIBILIDADE'].loc[disponibilidade_copia['NOME'] ==
disp_aux[indice_aux]].values[0]
        tempo_total          = inspecoes['DURACAO'].loc[inspecoes['GRUPO_A'] ==
demanda_aux[i]].values[0]
        tempo_total          = tempo_total + (2*(arcos['TEMPO'].loc[(arcos['ORIGEM_ARCO'] ==
origem) & (arcos['DESTINO_ARCO'] == destino)].values[0]))

        while(tempo_total > disponibilidade_atual):
            indice_aux  =  random.choice(np.where(funcao_objetivo[:,i] == 1)[0])
            origem      =
disponibilidade_copia.where(disponibilidade['NOME'].isin([disp_aux[indice_aux]]) ==
True).dropna()
            origem      = origem['UF'].values[0]
            destino     = demanda['AEROPORTO'].values[i]

            disponibilidade_atual =
disponibilidade_copia['DISPONIBILIDADE'].loc[disponibilidade_copia['NOME'] ==
disp_aux[indice_aux]].values[0]
            tempo_total          = inspecoes['DURACAO'].loc[inspecoes['GRUPO_A'] ==
demanda_aux[i]].values[0]
            tempo_total          = tempo_total + (2*(arcos['TEMPO'].loc[(arcos['ORIGEM_ARCO'] ==
origem) & (arcos['DESTINO_ARCO'] == destino)].values[0]))

        disponibilidade_copia['DISPONIBILIDADE'].loc[disponibilidade_copia['NOME'] ==
disp_aux[indice_aux]] = disponibilidade_atual - tempo_total # aqui tá dando rpoblema, eu preciso
fazer uma copia

        origem   =
disponibilidade_copia.where(disponibilidade_copia['NOME'].isin([disp_aux[indice_aux]]) ==
True).dropna()
        origem   = origem['UF'].values[0]
        destino  = demanda['AEROPORTO'].values[i]
        valor    = arcos.where(arcos['ORIGEM_ARCO'].isin([origem]) == True).dropna()
        valor    = valor.where(valor['DESTINO_ARCO'].isin([destino]) == True).dropna()
        valor    = valor['CUSTO'].values[0]
        custo    = custo + valor

        funcao_objetivo2[indice_aux, i] = 1


    return funcao_objetivo2.copy()

def respeitaDisponibilidade(a0, matriz_tempo_total):
    a0rd = a0.copy()

    for i in range (0, len(a0rd[:,0])):
        indice_aux = np.where(a0rd[i,:] == 1)[0]
```

```
        if(len(indice_aux > 0)):
            soma = matriz_tempo_total[i, indice_aux].sum()
            if(soma > 5.0):
                return False
    return True


#calcula custo da matriz
def calculaCusto(a0, matriz_custo):
    i,j = np.where(a0 == 1)
    custo = matriz_custo[i,j].sum() * 2
    return custo


ef = pd.ExcelFile('SFI_ALOCACAO.xlsx')

x_graph = []
y_graph = []
h = []

#************************************************
#*********LEITURA DE DADOS***************
#************************************************
#************************************************

disponibilidade = pd.read_excel(ef, 'Disponibilidade - PREENCHER', 1, usecols=['NOME',
'UF',  'DISPONIBILIDADE'])
arcos           = pd.read_excel(ef, 'Arcos',                    1, usecols=['ORIGEM_ARCO',
'DESTINO_ARCO', 'CUSTO', 'TEMPO'])
inspecoes       = pd.read_excel(ef, 'Atividades',                1, usecols=['GRUPO_A', 'DURACAO',
'EQUIPE'])
oferta          = pd.read_excel(ef, 'Atividades',                1,usecols=['SERVIDOR',  'GRUPO_P'])
demanda         = pd.read_csv('demanda1.csv', sep=';')

destinos_distintos   = pd.read_excel(ef, 'Arcos', 1, usecols=['NOS', 'ORIGEM_NO',
'DESTINO_NO'])
atividades_oferta    = oferta['GRUPO_P'].dropna().values
atividades_restritas = inspecoes['GRUPO_A'].dropna().values
destinos_distintos   = destinos_distintos['DESTINO_NO'].dropna().values

oferta          = oferta.where((oferta['GRUPO_P'].isin(demanda['ATIVIDADE'].values) ==
True)).dropna()
disponibilidade  =  disponibilidade.where((disponibilidade['NOME'].isin(oferta['SERVIDOR'].values
) == True)).dropna()
arcos           = arcos.where((arcos['ORIGEM_ARCO'].isin(disponibilidade['UF'].values) == True
)).dropna()
arcos           = arcos.where((arcos['DESTINO_ARCO'].isin(demanda['AEROPORTO'].values) ==
True )).dropna()
```

```
inspecoes       =  inspecoes.where(inspecoes['GRUPO_A'].isin(demanda['ATIVIDADE'].values)
== True).dropna()

matriz_alocado = np.zeros((len(disponibilidade),len(inspecoes)), dtype=int)

disponibilidade_aux = disponibilidade.copy()
insp_aux            = inspecoes['GRUPO_A'].values
disp_aux            = disponibilidade['NOME'].values
demanda_aux         = demanda['ATIVIDADE'].values


for i in range (0, len(disponibilidade)):
    for j in range (0, len(inspecoes)):
        if(len(oferta[oferta['SERVIDOR'].str.match(disp_aux[i]) == True].values) > 0):
            aux1 = oferta[(oferta['SERVIDOR'].str.match(disp_aux[i]) == True)]
            if(len(aux1[(oferta['GRUPO_P'].str.match(insp_aux[j]) == True)]) > 0):
                matriz_alocado[i][j] = 1


funcao_objetivo     =  np.zeros((len(disponibilidade),len(demanda)), dtype=int)
matriz_tempo_viagem =  np.zeros((len(disponibilidade),len(demanda)), dtype=float)
matriz_tempo_total  =  np.zeros((len(disponibilidade),len(demanda)), dtype=float)
matriz_custo        =  np.zeros((len(disponibilidade),len(demanda)), dtype=float)
matriz_tempo_missao =  []

for i in demanda_aux:
    d = inspecoes['DURACAO'].loc[(inspecoes['GRUPO_A'] == i)].values[0]
    matriz_tempo_missao.append(d)

i_aux = []
j_aux = []
for i in range (0,len(disponibilidade) ):
    for j in range (0, len(demanda)):
        origem  = disponibilidade.where(disponibilidade['NOME'].isin([disp_aux[i]]) == True).dropna()
        origem  = origem['UF'].values[0]
        destino = demanda['AEROPORTO'].values[j]

        tempo   = arcos.where(arcos['ORIGEM_ARCO'].isin([origem]) == True).dropna()
        tempo   = tempo.where(tempo['DESTINO_ARCO'].isin([destino]) == True).dropna()
        if(len(tempo) > 0):
            custo   = tempo['CUSTO'].values[0]
            tempo   = tempo['TEMPO'].values[0]
        else:
            i_aux.append(i)
            j_aux.append(j)
            custo   = 10000000
            tempo   = 10000000

        matriz_custo[i][j] = custo
```

```
        matriz_tempo_viagem[i][j] = tempo

for i in range (0,len(disponibilidade) ):
    matriz_tempo_total[i,:] = matriz_tempo_viagem[i,:] + matriz_tempo_missao

#************************************************
#*********INICIA FUNCAO OBJETIVO********
#************************************************
#************************************************
i = 0
for atividade in demanda['ATIVIDADE'].values:
    indice_missao       =   np.where(insp_aux == atividade)[0][0]
    funcao_objetivo[:,i] = matriz_alocado[:,indice_missao]
    i = i + 1

for i in i_aux:
    for j in j_aux:
        funcao_objetivo[i,j] = 0

#************************************************
#*********INICIA TABU*************************
#************************************************
#************************************************

s0 = geraIndividuo(funcao_objetivo, disponibilidade,disp_aux,inspecoes, demanda_aux,arcos)
sBest = s0
bestCandidate = s0
tabuList = []
tabuList.append(s0)
print("COMECOU")

for i in range (0,5000):
    passoAleatorio = random.randint(1, 7)
    sNbhd = getNbhd(bestCandidate,funcao_objetivo,passoAleatorio)
    #for j in sNbhd:
        #print(respeitaDisponibilidade(j, disponibilidade, demanda, arcos,inspecoes))
    bestCandidate = sNbhd[0]

    for sCandidate in sNbhd:

        custoCandidate = calculaCusto(sCandidate, matriz_custo)
        custoBest      = calculaCusto(bestCandidate, matriz_custo)
        print(sNbhd, custoCandidate, custoBest)
        #não tá na lista?
        #custo do canditado é menor que o custo do best?
        #candidato respeita disponibilidade?
        if((isInList(sCandidate, tabuList) == False) and (custoCandidate < custoBest) and
(respeitaDisponibilidade(sCandidate, matriz_tempo_total) == True) and (valida(sCandidate) ==
True)):
```

```
            bestCandidate = sCandidate

    custoBest = calculaCusto(bestCandidate, matriz_custo)
    custosBest = calculaCusto(sBest, matriz_custo)
    tabuList.append(bestCandidate)

    x_graph.append(i)
    y_graph.append(custosBest)
    h.append(19017)

    #print(i,int(custosBest), int(custoBest),"TABU")
    if(custoBest < custosBest):
        sBest = bestCandidate

    if(len(tabuList) > 60):
        del tabuList[0]

print(i,int(custosBest), int(custoBest),"TABU")
```

Simulated Annealing:

```
import pandas as pd
import csv
from numpy import array
import numpy as np
import math
import random
import warnings
from random import randint

import time

warnings.filterwarnings("ignore")

def validaFO(a0, funcao_objetivo):
    tamanho = len(funcao_objetivo[0,:])
    y,x = np.where(a0 == 1)
    if(sum(funcao_objetivo[y,x]) == tamanho):
        return True
    else:
        return False


def valida(a0):
    a0v = a0.copy()
    if (len(np.where(a0v == 1)[0]) == 5):
        return True
    else:
        return False

def respeitaDisponibilidade(a0, matriz_tempo_total):
    a0rd = a0.copy()

    for i in range (0, len(a0rd[:,0])):
        indice_aux = np.where(a0rd[i,:] == 1)[0]
        if(len(indice_aux > 0)):
            soma = matriz_tempo_total[i, indice_aux].sum()
            if(soma > 5.0):
                return False
    return True


#calcula custo da matriz
def calculaCusto(a0, matriz_custo):
    a0cc = a0.copy()
    i,j = np.where(a0cc == 1)
    custo = matriz_custo[i,j].sum() * 2
    return custo
```

```
#direcao: -1 desce, 1 sobe
#a0:       matriz com a solução possivel
#FO:       matriz gabarito
def salta(a0, direcao,FO,posicao_passo,tamanho_passo):
    a_aux = a0.copy()
    if(tamanho_passo+posicao_passo < len(FO[0,:] )):
        final = tamanho_passo+posicao_passo
    else:
        final = len(FO[0,:] )

    for i in range (posicao_passo, final):
        #fazer o recorte do salto em cima dos servidores_aptos tambem
        servidores_aptos = np.where(FO[:,i] == 1)[0]
        try:
            indice_atual = np.where(a0[:,i] == 1)[0][0]
            servidor_salto = np.where(servidores_aptos == indice_atual)[0][0]
            if((servidor_salto - direcao) == len(servidores_aptos)):
                novo_indice = servidores_aptos[0]
            else:
                novo_indice = servidores_aptos[servidor_salto - direcao]

            a_aux[indice_atual,i] = 0
            a_aux[novo_indice,i] = 1
        except:
            l = True
    return  a_aux



#função que conserva da parte deslocada e adiciona a parte
def mascara(a0, a1, y_inicio, y_fim):
    a0mf = a0.copy()
    a1mf = a1.copy()

    a_aux = np.zeros((len(a0mf[:,0]),len(a0mf[0,:])), dtype=int)
    a_aux[y_inicio:y_fim,:] = np.absolute(~a_aux[y_inicio:y_fim,:])
    a_aux = np.logical_and(a_aux,a0mf).astype(int)

    a_aux2 = np.ones((len(a0mf[:,0]),len(a0mf[0,:])), dtype=int)
    a_aux2[y_inicio:y_fim,:] = np.logical_not(a_aux2[y_inicio:y_fim,:]).astype(int)
    a_aux2 = np.logical_and(a_aux2,a1mf).astype(int)
    a_final = np.logical_or(a_aux2,a_aux).astype(int)

    return a_final.copy()

def geraIndividuo(funcao_objetivo, disponibilidade,disp_aux,inspecoes, demanda_aux,arcos):
    funcao_objetivo2 =  np.zeros((len(disponibilidade),len(demanda_aux)), dtype=int)
```

```
    custo = 0
    disponibilidade_copia = disponibilidade.copy()
    for i in range (len(demanda)):
        indice_aux  =  random.choice(np.where(funcao_objetivo[:,i] == 1)[0])
        origem      =
disponibilidade_copia.where(disponibilidade_copia['NOME'].isin([disp_aux[indice_aux]]) ==
True).dropna()
        origem      = origem['UF'].values[0]
        destino     = demanda['AEROPORTO'].values[i]

        disponibilidade_atual =
disponibilidade_copia['DISPONIBILIDADE'].loc[disponibilidade_copia['NOME'] ==
disp_aux[indice_aux]].values[0]
        tempo_total          = inspecoes['DURACAO'].loc[inspecoes['GRUPO_A'] ==
demanda_aux[i]].values[0]
        tempo_total          = tempo_total + (2*(arcos['TEMPO'].loc[(arcos['ORIGEM_ARCO'] ==
origem) & (arcos['DESTINO_ARCO'] == destino)].values[0]))

        while(tempo_total > disponibilidade_atual):
            indice_aux  =  random.choice(np.where(funcao_objetivo[:,i] == 1)[0])
            origem      =
disponibilidade_copia.where(disponibilidade['NOME'].isin([disp_aux[indice_aux]]) ==
True).dropna()
            origem      = origem['UF'].values[0]
            destino     = demanda['AEROPORTO'].values[i]

            disponibilidade_atual =
disponibilidade_copia['DISPONIBILIDADE'].loc[disponibilidade_copia['NOME'] ==
disp_aux[indice_aux]].values[0]
            tempo_total          = inspecoes['DURACAO'].loc[inspecoes['GRUPO_A'] ==
demanda_aux[i]].values[0]
            tempo_total          = tempo_total + (2*(arcos['TEMPO'].loc[(arcos['ORIGEM_ARCO'] ==
origem) & (arcos['DESTINO_ARCO'] == destino)].values[0]))

        disponibilidade_copia['DISPONIBILIDADE'].loc[disponibilidade_copia['NOME'] ==
disp_aux[indice_aux]] = disponibilidade_atual - tempo_total # aqui tá dando rpoblema, eu preciso
fazer uma copia

        origem   =
disponibilidade_copia.where(disponibilidade_copia['NOME'].isin([disp_aux[indice_aux]]) ==
True).dropna()
        origem   = origem['UF'].values[0]
        destino  = demanda['AEROPORTO'].values[i]
        valor    = arcos.where(arcos['ORIGEM_ARCO'].isin([origem]) == True).dropna()
        valor    = valor.where(valor['DESTINO_ARCO'].isin([destino]) == True).dropna()
        valor    = valor['CUSTO'].values[0]
        custo    = custo + valor

        funcao_objetivo2[indice_aux, i] = 1
```

```
    return funcao_objetivo2.copy()

inicioLeitura = time.time()
ef = pd.ExcelFile('InstanciaB3.xlsx')

#************************************************
#*********LEITURA DE DADOS***********************
#************************************************
#************************************************

# Leitura do excel de entrada do LINGO
# Recorte do dominio


disponibilidade = pd.read_excel(ef, 'Disponibilidade - PREENCHER', 1, usecols=['NOME',
'UF',  'DISPONIBILIDADE'])
arcos           = pd.read_excel(ef, 'Arcos',                       1, usecols=['ORIGEM_ARCO',
'DESTINO_ARCO', 'CUSTO', 'TEMPO'])
inspecoes       = pd.read_excel(ef, 'Atividades',                  1, usecols=['GRUPO_A', 'DURACAO',
'EQUIPE'])
oferta          = pd.read_excel(ef, 'Atividades',                  1 ,usecols=['SERVIDOR',  'GRUPO_P'])
#demanda         = pd.read_csv('demandaAleatoria/10/demanda40.csv', sep=';')
demanda         = pd.read_excel(ef, 'Demanda - PREENCHER', 1,usecols=[ 'MISSAO',
'ATIVIDADE', 'AEROPORTO'])

destinos_distintos   = pd.read_excel(ef, 'Arcos', 1, usecols=['NOS', 'ORIGEM_NO',
'DESTINO_NO'])
atividades_oferta    = oferta['GRUPO_P'].dropna().values
atividades_restritas = inspecoes['GRUPO_A'].dropna().values
destinos_distintos   = destinos_distintos['DESTINO_NO'].dropna().values

oferta          = oferta.where((oferta['GRUPO_P'].isin(demanda['ATIVIDADE'].values) ==
True)).dropna()
disponibilidade  =  disponibilidade.where((disponibilidade['NOME'].isin(oferta['SERVIDOR'].values
) == True)).dropna()
arcos           = arcos.where((arcos['ORIGEM_ARCO'].isin(disponibilidade['UF'].values) == True
)).dropna()
arcos           = arcos.where((arcos['DESTINO_ARCO'].isin(demanda['AEROPORTO'].values) ==
True )).dropna()
inspecoes       =  inspecoes.where(inspecoes['GRUPO_A'].isin(demanda['ATIVIDADE'].values)
== True).dropna()

matriz_alocado = np.zeros((len(disponibilidade),len(inspecoes)), dtype=int)

disponibilidade_aux = disponibilidade.copy()
insp_aux            = inspecoes['GRUPO_A'].values
disp_aux            = disponibilidade['NOME'].values
```

```
demanda_aux         = demanda['ATIVIDADE'].values


for i in range (0, len(disponibilidade)):
    for j in range (0, len(inspecoes)):
        if(len(oferta[oferta['SERVIDOR'].str.match(disp_aux[i]) == True].values) > 0):
            aux1 = oferta[(oferta['SERVIDOR'].str.match(disp_aux[i]) == True)]
            if(len(aux1[(oferta['GRUPO_P'].str.match(insp_aux[j]) == True)]) > 0):
                matriz_alocado[i][j] = 1


funcao_objetivo     =  np.zeros((len(disponibilidade),len(demanda)), dtype=int)
matriz_tempo_viagem =  np.zeros((len(disponibilidade),len(demanda)), dtype=float)
matriz_tempo_total  =  np.zeros((len(disponibilidade),len(demanda)), dtype=float)
matriz_custo        =  np.zeros((len(disponibilidade),len(demanda)), dtype=float)
matriz_tempo_missao =  []

for i in demanda_aux:
    d = inspecoes['DURACAO'].loc[(inspecoes['GRUPO_A'] == i)].values[0]
    matriz_tempo_missao.append(d)

i_aux = []
j_aux = []
for i in range (0,len(disponibilidade) ):
    for j in range (0, len(demanda)):
        origem  = disponibilidade.where(disponibilidade['NOME'].isin([disp_aux[i]]) == True).dropna()
        origem  = origem['UF'].values[0]
        destino = demanda['AEROPORTO'].values[j]

        tempo   = arcos.where(arcos['ORIGEM_ARCO'].isin([origem]) == True).dropna()
        tempo   = tempo.where(tempo['DESTINO_ARCO'].isin([destino]) == True).dropna()
        if(len(tempo) > 0):
            custo   = tempo['CUSTO'].values[0]
            tempo   = tempo['TEMPO'].values[0] * 2.0
        else:
            i_aux.append(i)
            j_aux.append(j)
            custo   = 10000000
            tempo   = 10000000

        matriz_custo[i][j] = custo
        matriz_tempo_viagem[i][j] = tempo

for i in range (0,len(disponibilidade) ):
    matriz_tempo_total[i,:] = matriz_tempo_viagem[i,:] + matriz_tempo_missao

#************************************************
#*********INICIA FUNCAO OBJETIVO********
#************************************************
```

```
#************************************************
i = 0
for atividade in demanda['ATIVIDADE'].values:
    indice_missao        =   np.where(insp_aux == atividade)[0][0]
    funcao_objetivo[:,i] = matriz_alocado[:,indice_missao]
    i = i + 1

for i in i_aux:
    for j in j_aux:
        funcao_objetivo[i,j] = 0


funcao_objetivo2 = geraIndividuo(funcao_objetivo, disponibilidade,disp_aux,inspecoes,
demanda_aux,arcos)
print("começou")
#************************************************
#*********INICIO DO ALGORITMO************
#************************************************
#************************************************

# a0 - original
# a1 - novo

# custos[0] - cima_desce
# custos[1] - cima_sobe
# custos[2] - baixo_sobe
# custos[3] - baixo_desce

# n = corte inicial
# k = corte iterativo e randomico
# iteracoes = quantidades de iteracoes que o algoritmo irá rodar

iteracoes = 5000
p = -1

inicio = 0
fim = len(funcao_objetivo2[:,0])
meio = int(fim/3)

a0 = funcao_objetivo2.copy()
a1 = funcao_objetivo2.copy()
n = 2.7
k = 0
custos = np.zeros(4, dtype=int)

fimLeitura = time.time()
inicioHeuristica = time.time()

for x in range (0,iteracoes):
```

```
#DESCE CIMA
a1      = salta(a0, 1,funcao_objetivo,inicio, meio)
a1      = mascara(a1, a0, inicio, meio)
disp_check = respeitaDisponibilidade(a1, matriz_tempo_total)
if(disp_check):
    custos[0] = calculaCusto(a1, matriz_custo)
    a_aux1 = a1.copy()
else:
    custos[0] = 1000000000
    a_aux1 = a0.copy()

#SOBE CIMA
a1     = salta(a0, p,funcao_objetivo,inicio, meio)
a1     = mascara(a1, a0, inicio, meio)
disp_check = respeitaDisponibilidade(a1, matriz_tempo_total)
if(disp_check):
    a_aux2 = a1.copy()
    custos[1] = calculaCusto(a1, matriz_custo)
else:
    a_aux2 = a0.copy()
    custos[1] = 1000000000


#SOBE BAIXO
a1     = salta(a0, p,funcao_objetivo,meio, fim)
a1     = mascara(a1, a0, meio, fim)
disp_check = respeitaDisponibilidade(a1, matriz_tempo_total)
if(disp_check):
    a_aux3 = a1.copy()
    custos[2] = calculaCusto(a1, matriz_custo)
else:
    a_aux3 = a0.copy()
    custos[2] = 1000000000

#DESCE BAIXO
a1     = salta(a0, 1,funcao_objetivo,meio, fim)
a1     = mascara(a1, a0, meio, fim)
disp_check = respeitaDisponibilidade(a1, matriz_tempo_total)
if(disp_check):
    a_aux4 = a1.copy()
    custos[3] =  calculaCusto(a1, matriz_custo)
else:
    a_aux4 = a0.copy()
    custos[3] = 1000000000


result = all(elem == custos[0] for elem in custos)
```

```
menor = np.argmin(custos)

if((fim - inicio) < 2):
    n      = randint(2, 10)
    inicio = 0
    fim    = len(funcao_objetivo2[:,0])
    meio   = int(fim/n)
else:
    if((custos[menor] < 1000000000) and (result == False)):

        if(menor == 0):
            if(validaFO(a_aux1, funcao_objetivo) == True):
                a0   = a_aux1
            else:
                a0 = a0
            fim  = meio
            meio = int(((fim - inicio)/n)+inicio)

        elif(menor == 1):
            if(validaFO(a_aux2, funcao_objetivo) == True):
                a0   = a_aux2
            else:
                a0 = a0
            fim  = meio
            meio = int(((fim - inicio)/n)+inicio)

        elif(menor == 2):
            if(validaFO(a_aux3, funcao_objetivo) == True):
                a0   = a_aux3
            else:
                a0 = a0
            inicio = meio
            meio   = int(((fim - inicio)/n)+inicio)

        else:
            if(validaFO(a_aux4, funcao_objetivo) == True):
                a0   = a_aux4
            else:
                a0 = a0
            inicio = meio
            meio   = int(((fim - inicio)/n)+inicio)


    elif((custos[menor] >= 1000000000) and (result == True) ):
        n      = randint(2, 10)
        inicio = 0
```

```
            fim    = len(funcao_objetivo2[:,0])
            meio = int(fim/n)
fimHeuristica = time.time()
print(fimLeitura - inicioLeitura)
print(fimHeuristica - inicioHeuristica)
print((calculaCusto(a0, matriz_custo)), custos,  "SA")
```

Híbrido (Genetic Algorithm e Simulated Annealing):

```
import pandas as pd
import csv
from numpy import array
import numpy as np
import math
import random
import warnings
from random import randint
import time

pd.set_option("display.max_rows", None, "display.max_columns", None)
warnings.filterwarnings("ignore")


def salta(a0, direcao,FO,posicao_passo,tamanho_passo):
    a_aux = a0.copy()
    if(tamanho_passo+posicao_passo < len(FO[0,:] )):
        final = tamanho_passo+posicao_passo
    else:
        final = len(FO[0,:] )

    for i in range (posicao_passo, final):
        servidores_aptos = np.where(FO[:,i] == 1)[0]
        try:
            indice_atual = np.where(a0[:,i] == 1)[0][0]
            servidor_salto = np.where(servidores_aptos == indice_atual)[0][0]
            if((servidor_salto - direcao) == len(servidores_aptos)):
                novo_indice = servidores_aptos[0]
            else:
                novo_indice = servidores_aptos[servidor_salto - direcao]

            a_aux[indice_atual,i] = 0
            a_aux[novo_indice,i] = 1
        except:
            l = True
    return  a_aux

def getNbhd (a0,funcao_objetivo,passo):
    x = len(a0[0,:])
    aList = []
    p = -1
    a1 = a0.copy()
    if(int(x/passo) == 0):
        k = 1
    else:
        k = int(x/passo)
    for i in range (0, k):
```

```
        fator = random.randint(1, 5)
        for j in range (0,fator):
            a0 = salta(a0,1,funcao_objetivo,(passo*i), passo)
            a1 = salta(a1,1,funcao_objetivo,(passo*i), passo)
        aList.append(salta(a0,1,funcao_objetivo,(passo*i), passo))
        aList.append(salta(a1,p,funcao_objetivo,(passo*i), passo))

    return aList

def isInList(a0, tabuList):
    r = any((np.array_equal(a0, dict_arr) for dict_arr in tabuList))
    return r

def validaFO(a0, funcao_objetivo):
    tamanho = len(funcao_objetivo[0,:])
    y,x = np.where(a0 == 1)
    if(sum(funcao_objetivo[y,x]) == tamanho):
        return True
    else:
        return False

def valida(a0):
    if (len(np.where(np.sum(a0, axis=0) != 1)[0]) > 1):
        return False
    else:
        return True
# pega metade de cada pai
# caso apareca mais de uma pessoa por missao, escolhe aleatoriamente
def crossover(a0, a1,funcao_objetivo):
    a0f = a0.copy()
    a1f = a1.copy()

    b =  mascara(a0f, a1f, 0, int(len(a0[:,0])/2))
    for i in range (0, len(b[0,:])):
        indice_aux = np.where(b[:,i] == 1)
        if(len(indice_aux[0]) == 2):
            indice_random  =  random.choice(indice_aux[0])
            b[indice_random,i] = 0
        elif(len(indice_aux[0]) == 0):
            indice_aux = np.where(funcao_objetivo[:,i] == 1)
            indice_random  =  random.choice(indice_aux[0])
            b[indice_random,i] = 1

    return b.copy()

def geraIndividuo2(funcao_objetivo, matriz_tempo_total, disponibilidade, demanda, disp_aux):
    histograma =  np.zeros((2,len(demanda)), dtype=int)
    for i in range (len(demanda)):
        histograma[0,i] = np.sum(funcao_objetivo[:,i])
```

```
        histograma[1,i] = i

    histograma = histograma[:, histograma[0].argsort()]
    disponibilidade_copia = disponibilidade.copy()
    funcao_objetivo2 =  np.zeros((len(disponibilidade),len(demanda)), dtype=int)

    for i in range (len(demanda)):
        j = histograma[1,i]
        indices_fo = np.where(funcao_objetivo[:,j] == 1)[0]
        indices_tempo = indices_fo
        escolhido = random.choice(indices_tempo)
        disp_escolhida =
int(disponibilidade_copia['DISPONIBILIDADE'].loc[disponibilidade_copia['NOME'] ==
disp_aux[escolhido]])

        while((float(matriz_tempo_total[escolhido,j]) > float(disp_escolhida)) and
(matriz_tempo_total[escolhido,j] < 1000)):
            escolhido = random.choice(indices_tempo)
            disp_escolhida =
float(disponibilidade_copia['DISPONIBILIDADE'].loc[disponibilidade_copia['NOME'] ==
disp_aux[escolhido]])

        funcao_objetivo2[escolhido, j] = 1
        disponibilidade_copia['DISPONIBILIDADE'].loc[disponibilidade_copia['NOME'] ==
disp_aux[escolhido]] = disp_escolhida - float(matriz_tempo_total[escolhido,j])
    return funcao_objetivo2

def respeitaDisponibilidade(a0, matriz_tempo_total, disp_oferta):
    a0rd = a0.copy()
    for i in range (0, len(a0rd[:,0])):
        indice_aux = np.where(a0rd[i,:] == 1)[0]
        if(len(indice_aux > 0)):
            soma = matriz_tempo_total[i, indice_aux].sum()
            if(float(soma) > float(disp_oferta[i])):
                return False
    return True

def calculaCusto(a0, matriz_custo):
    i,j = np.where(a0 == 1)
    custo = matriz_custo[i,j].sum() * 2
    return custo

#função que conserva da parte deslocada e adiciona a parte
def mascara(a0, a1, y_inicio, y_fim):
    a0mf = a0.copy()
    a1mf = a1.copy()

    a_aux = np.zeros((len(a0mf[:,0]),len(a0mf[0,:])), dtype=int)
    a_aux[y_inicio:y_fim,:] = np.absolute(~a_aux[y_inicio:y_fim,:])
```

```
    a_aux = np.logical_and(a_aux,a0mf).astype(int)

    a_aux2 = np.ones((len(a0mf[:,0]),len(a0mf[0,:])), dtype=int)
    a_aux2[y_inicio:y_fim,:] = np.logical_not(a_aux2[y_inicio:y_fim,:]).astype(int)
    a_aux2 = np.logical_and(a_aux2,a1mf).astype(int)
    a_final = np.logical_or(a_aux2,a_aux).astype(int)

    return a_final.copy()

def mutacao(a0, funcao_objetivo):
    a0fm = a0.copy()
    for i in range (0, 2):
        missao = randint(0, np.size(a0fm,1) - 1)

        indice = np.where(a0fm[:,missao] == 1)
        indice_aux  =  random.choice(np.where(funcao_objetivo[:,missao] == 1)[0])

        a0fm[indice_aux,missao] = 1
        a0fm[indice,missao] = 0

    return a0fm.copy()

start = time.perf_counter()

inicio = time.time()

ef = pd.ExcelFile('InstanciaB3.xlsx')

#************************************************
#*********LEITURA DE DADOS***********************
#************************************************
#************************************************

disponibilidade = pd.read_excel(ef, 'Disponibilidade - PREENCHER', 1, usecols=['NOME',
'UF',  'DISPONIBILIDADE'])
demanda         = pd.read_excel(ef, 'Demanda - PREENCHER', 1,usecols=[ 'MISSAO',
'ATIVIDADE', 'AEROPORTO'])
arcos           = pd.read_excel(ef, 'Arcos',                    1, usecols=['ORIGEM_ARCO',
'DESTINO_ARCO', 'CUSTO', 'TEMPO'])
inspecoes       = pd.read_excel(ef, 'Atividades',                1, usecols=['GRUPO_A', 'DURACAO',
'EQUIPE'])
oferta          = pd.read_excel(ef, 'Atividades',                1 ,usecols=['SERVIDOR',  'GRUPO_P'])

disponibilidade = disponibilidade.where(disponibilidade['DISPONIBILIDADE'] != 0).dropna()
disp_aux        = disponibilidade['NOME'].values
oferta          = oferta.where((oferta['SERVIDOR'].isin(disp_aux)) == True).dropna()

destinos_distintos   = pd.read_excel(ef, 'Arcos', 1, usecols=['NOS', 'ORIGEM_NO',
'DESTINO_NO'])
```

```
atividades_oferta    = oferta['GRUPO_P'].dropna().values
atividades_restritas = inspecoes['GRUPO_A'].dropna().values
destinos_distintos   = destinos_distintos['DESTINO_NO'].dropna().values

oferta          = oferta.where((oferta['GRUPO_P'].isin(demanda['ATIVIDADE'].values) ==
True)).dropna()
disponibilidade  =  disponibilidade.where((disponibilidade['NOME'].isin(oferta['SERVIDOR'].values
) == True)).dropna()
arcos           = arcos.where((arcos['ORIGEM_ARCO'].isin(disponibilidade['UF'].values) == True
)).dropna()
arcos           = arcos.where((arcos['DESTINO_ARCO'].isin(demanda['AEROPORTO'].values) ==
True )).dropna()
inspecoes       =  inspecoes.where(inspecoes['GRUPO_A'].isin(demanda['ATIVIDADE'].values)
== True).dropna()

matriz_alocado = np.zeros((len(disponibilidade),len(inspecoes)), dtype=int)

disponibilidade_aux = disponibilidade.copy()
insp_aux            = inspecoes['GRUPO_A'].values
disp_aux            = disponibilidade['NOME'].values
demanda_aux         = demanda['ATIVIDADE'].values
disp_oferta         = disponibilidade['DISPONIBILIDADE'].values

#0,42 s

for i in range (0, len(disponibilidade)):
    for j in range (0, len(inspecoes)):
        if(len(oferta[oferta['SERVIDOR'].str.match(disp_aux[i]) == True].values) > 0):
            aux1 = oferta[(oferta['SERVIDOR'].str.match(disp_aux[i]) == True)]
            if(len(aux1[(oferta['GRUPO_P'].str.match(insp_aux[j]) == True)]) > 0):
                matriz_alocado[i][j] = 1

#7 segundos


funcao_objetivo     =  np.zeros((len(disponibilidade),len(demanda)), dtype=int)
matriz_tempo_viagem =  np.zeros((len(disponibilidade),len(demanda)), dtype=float)
matriz_tempo_total  =  np.zeros((len(disponibilidade),len(demanda)), dtype=float)
matriz_custo        =  np.zeros((len(disponibilidade),len(demanda)), dtype=float)
matriz_tempo_missao =  []

for i in demanda_aux:
    d = inspecoes['DURACAO'].loc[(inspecoes['GRUPO_A'] == i)].values[0]
    matriz_tempo_missao.append(d)

i_aux = []
j_aux = []
for i in range (0,len(disponibilidade) ):
    for j in range (0, len(demanda)):
```

```
        origem  = disponibilidade.where(disponibilidade['NOME'].isin([disp_aux[i]]) == True).dropna()
        origem  = origem['UF'].values[0]
        destino = demanda['AEROPORTO'].values[j]

        tempo   = arcos.where(arcos['ORIGEM_ARCO'].isin([origem]) == True).dropna()
        tempo   = tempo.where(tempo['DESTINO_ARCO'].isin([destino]) == True).dropna()
        if(len(tempo) > 0):
            custo   = tempo['CUSTO'].values[0]
            tempo   = tempo['TEMPO'].values[0] * 2.0
        else:
            i_aux.append(i)
            j_aux.append(j)
            custo   = 1000000
            tempo   = 1000000

        matriz_custo[i][j] = custo
        matriz_tempo_viagem[i][j] = tempo

for i in range (0,len(disponibilidade) ):
    matriz_tempo_total[i,:] = matriz_tempo_viagem[i,:] + matriz_tempo_missao

#50 segundos


#************************************************
#*********INICIA FUNCAO OBJETIVO********
#************************************************
#************************************************
i = 0
for atividade in demanda['ATIVIDADE'].values:
    indice_missao       =   np.where(insp_aux == atividade)[0][0]
    funcao_objetivo[:,i] = matriz_alocado[:,indice_missao]
    i = i + 1

for i in i_aux:
    for j in j_aux:
        funcao_objetivo[i,j] = 0



# Inicialização da funçao objetivo com as restrições devidas
# Escolhe aleatoriamente um servidor
# Pega origem e destino do servidor escolhido
# Calcula a disponibilidade do servidor e o tempo total pra realizar a missão
# Testa se o tempo pra fazer a missão cabe na disponibilidade
# Caso não caiba, sorteia outro servidor
# Sorteio de novo servidor (dica: retirar o primeiro servidor)
# Pega origem e destino
# Calcula novos tempos para possivel teste
```

```
# Atualiza a disponibilidade do servidor escolhido
# Pega origem e destino (dica: talvez não precise desse pass)

a = []

a.append(geraIndividuo2(funcao_objetivo, matriz_tempo_total, disponibilidade, demanda,
disp_aux))
a.append(geraIndividuo2(funcao_objetivo, matriz_tempo_total, disponibilidade, demanda,
disp_aux))
a.append(geraIndividuo2(funcao_objetivo, matriz_tempo_total, disponibilidade, demanda,
disp_aux))
a.append(geraIndividuo2(funcao_objetivo, matriz_tempo_total, disponibilidade, demanda,
disp_aux))

custos_pais = np.zeros(4, dtype=int)
for i in range (0,4):
    custos_pais[i] = calculaCusto(a[i], matriz_custo)

#**********************************************
#*********INICIA GENETIC*********************
#**********************************************
#**********************************************


fim = time.time()
print(fim-inicio)

print("COMEÇA")
for j in range (0,3000):
    # 6 cruzamentos possiveis
    b = []
    b.append(crossover(a[0], a[1],funcao_objetivo))
    b.append(crossover(a[1], a[2],funcao_objetivo))
    b.append(crossover(a[2], a[3],funcao_objetivo))
    b.append(crossover(a[1], a[3],funcao_objetivo))
    b.append(crossover(a[0], a[2],funcao_objetivo))
    b.append(crossover(a[0], a[3],funcao_objetivo))

    #Calcula custo apenas dos validos
    custos_filhos = np.zeros(6, dtype=int)
    for i in range (0,6):
        v = valida(b[i])
        d = respeitaDisponibilidade(b[i], matriz_tempo_total,disp_oferta)
        if((d == True) and (v == True)):
            custos_filhos[i] = calculaCusto(b[i], matriz_custo)
        else:
            custos_filhos[i] = 1000000

    #pega o menor valor dos filhos e insere nos pais
```

```
    quantidades_validos  = np.where(custos_filhos < 1000000)[0]

    if (len(quantidades_validos) > 0):
        for i in range (0,len(quantidades_validos)):
            if(custos_pais[np.argmax(custos_pais)] >  custos_filhos[quantidades_validos[i]]):
                a[np.argmax(custos_pais)] = b[quantidades_validos[i]]

    for i in range (0,4):
        if((valida(a[i]) == True)  and (respeitaDisponibilidade(a[i],matriz_tempo_total,disp_oferta) ==
True) and (validaFO(a[i], funcao_objetivo) == True) ):
            custos_pais[i] = calculaCusto(a[i], matriz_custo)
        else:
            custos_pais[i] = 1000000

    # a cada 5 iterações muta os pais
    if ((j % 7) == 0):
        a[np.argmin(custos_pais) - 1] = mutacao(a[np.argmin(custos_pais) - 1],funcao_objetivo)
        a[np.argmin(custos_pais) - 2] = mutacao(a[np.argmin(custos_pais) - 2],funcao_objetivo)
        a[np.argmin(custos_pais) - 3] = mutacao(a[np.argmin(custos_pais) - 3],funcao_objetivo)

    menor = np.argmin(custos_pais)

s0 = a[menor]
valor = int(calculaCusto(s0, matriz_custo))
print(valor)

sBest = s0
bestCandidate = s0
tabuList = []
tabuList.append(s0)
print("COMECOU")

for i in range (0,3000):
    passoAleatorio = random.randint(1, 7)
    sNbhd = getNbhd(bestCandidate,funcao_objetivo,passoAleatorio)
    bestCandidate = sNbhd[0]
    for sCandidate in sNbhd:
        custoCandidate = calculaCusto(sCandidate, matriz_custo)
        custoBest      = calculaCusto(bestCandidate, matriz_custo)
        if((isInList(sCandidate.flatten(), tabuList) == False) and (custoCandidate < custoBest) and
(respeitaDisponibilidade(sCandidate, matriz_tempo_total,disp_oferta) == True) and
(valida(sCandidate) == True)):
            bestCandidate = sCandidate
    custoBest  = calculaCusto(bestCandidate, matriz_custo)
    custosBest = calculaCusto(sBest, matriz_custo)
    tabuList.append(bestCandidate.flatten())
    if(custoBest < custosBest):
        sBest = bestCandidate
    if(len(tabuList) > 20):
```

```
        del tabuList[0]

print(i,int(custosBest),"TABU")

#print(j, custos_pais[menor], custos_pais,"GA",  respeitaDisponibilidade(a[menor],
matriz_tempo_total,disp_oferta))
finish = time.perf_counter()
print(f'Finished in {round(finish-start, 2)} second(s)')
```