Universidade de Brasília

Faculdade de Economia, Administração,Contabilidade e Gestão de Políticas Públicas

Departamento de Administração

Rafael Levi Anaissi

Análise da viabilidade da utilização do Google Colaboratory e solver gratuito para a resolução de um problema de programação inteira de grande porte

Brasília – DF

2023

Rafael Levi Anaissi

Análise da viabilidade da utilização do Google Colaboratory e solver gratuito para a resolução de um problema de programação inteira de grande porte

Monografia apresentada ao Departamento de Administração como requisito parcial à obtenção do título de Bacharel em Administração.

Professor Orientador: Doutora, Silvia Araújo dos Reis

Brasília – DF

2023

Anaissi, Rafael Levi.
Análise da viabilidade da utilização do Google Colaboratory e solver gratuito para a resolução de um problema de programação inteira de grande porte / Rafael Levi Anaissi – Brasília, 2023.
59 f. : il.

Monografia (bacharelado) – Universidade de Brasília, Departamento de Administração, 2023.
Orientador: Prof. Dra. Silvia Araújo dos Reis, Departamento de Administração.

1. Google Colaboratory. 2.Programação inteira. 3. Python. 4. Open source. 5. Solver. I. Análise da viabilidade da utilização do Google Colaboratory e solver gratuito para a resolução de um problema de programação inteira de grande porte.

RAFAEL LEVI ANAISSI

Análise da viabilidade da utilização do Google Colaboratory e solver gratuito para a resolução de um problema de programação inteira de grande porte

A Comissão Examinadora, abaixo identificada, aprova o Trabalho de Conclusão do Curso de Administração da Universidade de Brasília do (a) aluno (a)

Rafael Levi Anaissi

Doutora, Silvia Araújo dos Reis

Professor-Orientador

Doutor, Victor Rafael Rezende Celestino

Professor-Examinador

Mestre, Olinda Maria Gomes Lesses

Professor-Examinador

Brasília, 11 de dezembro de 2023

Dedico este trabalho aos meus pais, Antonio e Carolina, por todo o apoio, incentivo, cobrança e amor ao longo da minha jornada acadêmica.

AGRADECIMENTOS

Quero agradecer, primeiramente, aos meus pais que sempre fizeram questão de me proporcionar uma educação de qualidade desde a infância e que não mediram esforços para me darem as melhores condições possíveis durante a minha graduação. A vocês, todo o meu amor e gratidão.

À minha namorada Isabela, que me apoiou em todas as minhas escolhas e que sempre esteve ao meu lado. Obrigado por fazer parte da minha tragetória e tornar essa etapa da minha vida mais leve.

À minha orientadora, Doutora Silvia Araújo dos Reis, pela confiança não só durante esta pesquisa, mas também ao me permitir ser monitor em sua disciplina. Pela paciência, atenção e compreensão durante todas as etapas de realização deste trabalho.

RESUMO

Com o avanço da Pesquisa Operacional como ferramenta de apoio à decisão, a utilização de softwares capazes de solucionar problemas matemáticos se faz cada vez mais necessário. Contudo, esses softwares, quase que em sua totalidade, são comerciais ou acadêmicos, tornando seu uso muito restrito. Tendo em vista a importância da utilização destes recursos computacionais na execução desses modelo matemáticos, este trabalho analisou a viabilidade da utilização do Google Colaboratory e de um solver gratuito para solucionar os mesmos problemas de programação de grande porte que antes só eram resolvidos por esses softwares pagos. Essa pesquisa utilizou o projeto *Safety Oversight* como base. Oriundo de uma parceria entre a Agência Nacional de Aviação Civil (ANAC) e a Universidade de Brasília (UNB), se trata de um problema atual e de grande porte e que aborda um problema de Programação Inteira de designação. A pesquisa realizada tem natureza aplicada, com objetivos exploratórios e abordagem mista, quantitativa e qualitativa. O problema foi testado em 6 instâncias com variações entre os principais parâmetros de entrada do modelo e modelado no Colab utilizando a linguaguem Python, com auxílio da linguaguem de modelagem Pyomo e do solver open-source CBC. Ao tentar solucionar as instâncias que possuiam um número maior de variáveis e restrições, o Colab, utilizando o CBC, não foi capaz de resolver o modelo matemático em tempo hábil. Contudo, obteve valores de função objetivo satistafórios. Verificou-se que o Colab é capaz de solucionar a maioria dos cenários, atingindo até mesmo resultados melhores em alguns casos.

Palavras-chave: Google Colaboratory, Programação inteira, Python, Open-source, Solver

ABSTRACT

With the advancement of Operations Research as a decision support tool, the use of software capable of solving mathematical problems is becoming increasingly necessary. However, these software tools are predominantly commercial or academic, limiting their accessibility. Recognizing the importance of utilizing these computational resources in executing mathematical models, this study examined the feasibility of using Google Colaboratory and a free solver to address the same large-scale programming problems that were traditionally tackled by paid software. The research focused on the Safety Oversight project, a collaboration between the National Civil Aviation Agency (ANAC) and the University of Brasília (UNB), addressing a current and large-scale Integer Programming assignment problem. The applied research employed exploratory objectives and a mixed approach, combining quantitative and qualitative methods. The problem was tested on six instances with variations in key input parameters and modeled on Colab using the Python language, aided by the Pyomo modeling language and the open-source CBC solver. When attempting to solve instances with a higher number of variables and constraints, Colab, utilizing CBC, was unable to solve the mathematical model in a timely manner. Nevertheless, it achieved satisfactory objective function values. It was found that Colab is capable of solving the majority of scenarios, even achieving better results in some cases.

Palavras-chave: Google Colaboratory, Integer Programming, Python, Open-source, Solver

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AIMMS – Advanced Integrated Multidimensional Modeling Software
ANAC – Agência Nacional de Aviação Civil
FO – Função Objetivo
LINGO – Language for Intective General Optimizer
MAE – Modelo de Alocação Estendido
MAR – Modelo de Alocação Resumido
PI – Programação Inteira
PIM – Programação Inteira Mista
PL – Programação Linear
PO – Pesquisa Operacional
PYOMO – Python Optimization Modeling Objects
UnB – Universidade de Brasília

SUMÁRIO

INTRODUÇÃO

1.1. Contextualização

Ferramentas de apoio à decisão sempre foram muito cobiçadas pelas organizações, principalmente nos processos decisórios de maior grau de complexidade. A dificuldade em atingir o máximo de eficiência para problemas específicos e a urgência por uma solução culminou no surgimento da Pesquisa Operacional (HILLIER; LIEBERMAN, 2013).

Respaldada pela Programação Matemática, a Pesquisa Operacional atua no campo da ciência que busca solucionar problemas de otimização, sejam eles de maximização, a exemplo do lucro, ou de minimização, como os custos, de um determinado objetivo (função objetivo) e considerando a aplicação de algumas restrições na modelagem para se atingir uma solução ótima (LOESCH; HEIN, 1999).

Contudo, por apresentarem diversas variáveis e restrições, a resolução torna-se inviável de ser realizada manualmente, demandando assim a utilização de computadores e softwares para sua aplicação.

Hoje, a tecnologia permite que esses modelos matemáticos sejam solucionados através de softwares, *solvers* e linguagens de programação, de acordo com o nível de complexidade exigida. Conforme esse nível se eleva, a especificidade computacional mínima para resolução, tanto no que diz respeito ao hardware quanto software, também requer um incremento de performance.

Ainda pouco utilizadas na administração pública, em grande parte devido a burocracias e pela carência de estudos na área, algumas dessas ferramentas apresentam um alto custo para utilização, salvo para fins acadêmicos, além das exigências computacionais para solucionar modelos de médio/grande porte. Os principais softwares para PO são: MPL, CPLEX (HILLIER; LIEBERMAN, 2013), LINGO e AIMMS (BELFIORE; FÁVERO, 2013).

Além dos programas tradicionais para resolução de modelagem amplamente abordados na literatura de Pesquisa Operacional, as linguagens de programação também são utilizadas para a execução desses modelos matemáticos. Pesquisas mostram que o Python — uma linguagem livre, de código aberto, de alto nível e de rápido desenvolvimento — está se tornando a língua franca da programação (The Economist, 2018). A linguagem conta com bibliotecas e *solvers* capazes de resolverem os problemas de Programação Matemática mais complexos e utiliza de ambientes de desenvolvimento integrado para execução dos códigos. Esses ambientes são softwares que propiciam a programação.

A Google disponibilizou gratuitamente, em 2017, o Google Colaboratory, sua web IDE (Integrated development environment), com infraestrutura de sistema de armazenamento em nuvem, ideal para machine learning, educação e pesquisa (T. Carneiro et al, 2018). A tecnologia em nuvem permite a utilização de recursos disponibilizados pela própria Google para execução e processamento dos modelos desenvolvidos dentro do seu IDE, dispensando a necessidade de se ter uma GPU (Graphics processing units) física e suficientemente capaz de resolver os problemas propostos. A disponibilização desses equipamentos para uma equipe de trabalho carrega grandes custos, como manutenção, energia, capital humano, além do próprio preço elevado do equipamento (T. Carneiro et al, 2018).

Posto isto, o presente trabalho apresenta a viabilidade na utilização do Google Colab em um ambiente cujos softwares e *solvers* ideais são pagos e/ou exigem uma capacidade computacional mínima.

1.2. Formulação do problema

Como o próprio nome já diz a Pesquisa Operacional compreende “pesquisa sobre operações” e tem sido largamente utilizada em problemas que buscam encontrar a melhor solução para conduzir e coordenar as operações entre as mais diversas áreas, comportando uma grande gama de aplicações (HILLIER; LIEBERMAN, 2013).

Os problemas de Programação Matemática determinísticos são aqueles em que todas as variáveis envolvidas no processo são fixas, possuem uma única solução e utilizam-se, geralmente, de métodos analíticos para sua solução (BELFIORE; FÁVERO, 2013). Eles são divididos em Programação Linear, Programação Inteira, Programação Inteira Mista e Programação Não Linear.

Um problema é classificado como Programação Inteira 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 (BELFIORE; FÁVERO, 2013).

Os problemas de Programação Inteira podem vir a demandar uma alta capacidade computacional e de processamento de cálculos quando considerados de médio/grande porte. 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 Programação Inteira e Programação Não Linear de grande porte. 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 (HILLIER; LIEBERMAN, 2013).,

Os softwares e os *solvers* tradicionais capazes de solucionar problemas de programação inteira de grande porte são, em sua maioria, pagos. O Google Colab, portanto, surge como uma alternativa econômica. Com base nisso, este trabalho apresenta a seguinte pergunta de pesquisa: É viável a utilização do Google Colab e

um *solver* gratuito para resolver um problema de Programação Inteira de grande porte?

1.3. Objetivo Geral

Verificar a viabilidade de utilização do Google Colaboratory e um *solver* gratuito para resolver um problema de Programação Inteira de grande porte.

1.4. Objetivos Específicos

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

1) Encontrar um modelo de PI de grande porte;
2) Programar o modelo no Google Colab;
3) Testar Solvers Gratuitos;
4) Testar cenários comparando os tempos de resolução entre Colab e Softwares pagos;

1.5. Justificativa

Os softwares capazes de resolverem modelos matemáticos de grande porte são, em sua maioria, inacessíveis à pequenas e médias organizações devido seu alto custo financeiro e tecnológico.

Apesar desses programas serem considerados como *top tier* para resolução dos problemas de otimização, outras linguagens de programação ganharam espaço

e cada vez mais pesquisas estão utilizando o Python. Contudo, o uso do Colab ainda é pouco utilizado e abordado em PO.

Por ser um software gratuito, com tecnologia de armazenamento em nuvem e capacidade para utilização dos equipamentos da Google para execução dos códigos, o Colab tem um grande potencial para substituir os softwares mais tradicionais limitados às grandes organizações.

Esse *gap* de artigos científicos internacionais e nacionais corrobora para o desenvolvimento de uma abordagem com essa temática, apresentando métodos e ferramentas eficientes para resolução de problemas de PI de grande porte.

2. REFERENCIAL TEÓRICO

Este capítulo busca realizar a revisão da literatura expondo os conceitos e estudos dos assuntos abordados neste trabalho, com base em pesquisas realizadas anteriormente por diversos autores, fundamentando e sustentando de forma tangível a pesquisa.

Por conseguinte, os conceitos de Pesquisa Operacional e suas ferramentas serão abordados inicialmente, seguidos pela fundamentação teórica sobre os softwares utilizados a título de comparação.

2.1 Pesquisa Operacional

A Pesquisa Operacional, de acordo com Loesch e Hein (2009), surgiu a partir da problemática militar da segunda guerra mundial como um ramo científico independente. A fim de solucionar os impasses táticos e estratégicos do conflito, mais especificamente o dimensionamento do número de navios, grupos de cientistas considerados especialistas em matemática, estatística e probabilidade formaram unidades especiais de inteligência militar e obtiveram êxito durante a conhecida “Batalha do Atlântico”.

Os resultados provenientes de sua aplicação durante a guerra colaboraram para um aumento significativo nos estudos sobre essa nova técnica e sua aplicabilidade para fins além de bélicos, sendo introduzida em diversas organizações dos setores comercial, industrial e governamental (HILLIER; LIEBERMAN, 2013). Por meio de suas técnicas quantitativas, a PO passou a servir como ferramenta de auxílio à tomada de decisões.

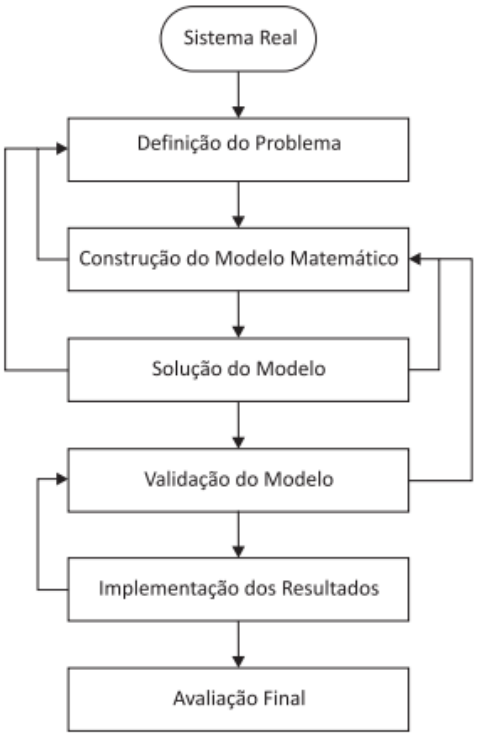
Para Hillier e Lieberman (2013), o crescimento da PO estava atrelado a dois principais fatores. O primeiro, conforme já dito anteriormente, está relacionado com o progresso e a popularização das técnicas da PO, o que culminou no surgimento de novas tecnologias e ferramentas, como o método *Simplex* em 1947.

O segundo fator diz respeito à chamada “revolução computacional”. O desenvolvimento de computadores capazes de realizar cálculos rapidamente e que eram impossíveis de serem feitos à mão fomentou o uso da PO, tornando viável sua aplicação para a resolução de problemas complexos. Consequentemente, com o avanço digital, surgiram diversos pacotes de softwares que estimularam ainda mais o avanço da PO e ampliaram seu alcance.

Em termos gerais, a Pesquisa Operacional se baseia na aplicação de um método científico (modelos matemáticos, estatísticos e algoritmos computacionais) para a tomada de decisão. Assim, pode-se dizer que a PO atua em um campo multidisciplinar, envolvendo áreas de administração, engenharia de produção, matemática aplicada, ciência da computação e gestão de negócios (BELFIORE; FÁVERO, 2013).

De acordo com Belfiore e Fávero (2013), o processo de modelagem é dividido em fases e deve obedecer uma sequência lógica para a elaboração de um estudo em PO. A primeira etapa consiste na definição e construção do modelo propriamente dito, estabelecendo suas variáveis, função objetivo e restrições. Com o modelo estruturado, a próxima fase é a solução do mesmo utilizando técnicas de PO. Por fim, o resultado obtido precisa ser validado e testado, de forma que o objetivo tenha sido atingido e possa ser implementado, conforme exposto na figura 1.

FIGURA 1 – Fases da modelagem.

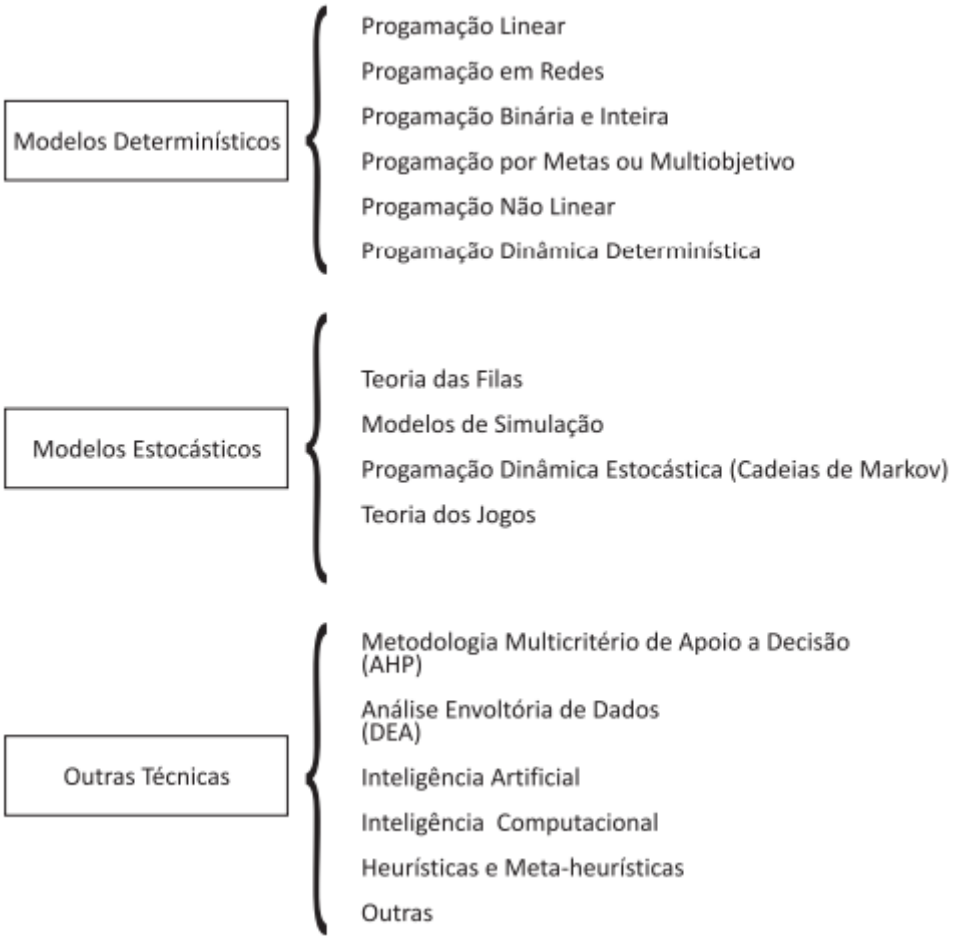


Fonte: Belfiore e Fávero (2013).

Segundo Hillier e Lieberman (2013), deve-se atentar pois existem muitas exceções à regra. Assim, o processo de modelagem descrito acima tem de ser interpretado como uma representação a grosso modo da condução dos estudos em PO. A implementação de cada fase pode variar em função do tipo de problema (BELFIORE; FÁVERO, 2013).

Belfiore e Fávero (2013) classificam, a partir do trabalho de Eom e Kim (2006), as ferramentas da PO em três modelos: determinísticos, estocásticos e outras técnicas, conforme a Figura 2.

FIGURA 2 – Ferramentas da Pesquisa Operacional



Fonte: Belfiore e Fávero (2013).

Este estudo utilizou o ferramental baseado em modelos determinísticos para a estruturação e solução do problema a fim de auxiliar, quantitativamente, o tomador de decisão. Dessa forma, nos aprofundaremos nesse modelo na próxima seção.

2.2 Modelos Determinísticos

Para Belfiore e Fávero (2013), um modelo é caracterizado como determinístico quando todas as variáveis que compõem o problema são conhecidas e constantes. A

solução resultante esperada é exata, e por muitas vezes, ótima, exceto quando fogem a regra, como já alertado por Hillier e Lieberman (2013), especificamente os casos em que múltiplas soluções ótimas são geradas.

Para entendermos mais sobre os diferentes tipos de categorias que fazem parte dos modelos determinísticos, a compreensão acerca da programação matemática faz-se necessária.

Os principais modelos de PO são chamados de Programação Matemática. A programação a que nos referimos expressa ideia de planejamento. Ainda que o termo seja muito associado ao processo computacional, ele é entendido como programação de atividades. Imprescindivelmente, sua aplicação prática implica no uso de computadores capazes de realizarem os cálculos que possuem um número elevado de variáveis (GOLDBARG; LUNA, 2005).

A PM é a área que estuda a otimização de recursos, na qual a quantidade a ser maximizada ou minimizada dos recursos escassos é descrita como uma função matemática (LACHTERMACHER, 2006).

Por ser uma área muito ampla e com diversas peculiaridades, os métodos sofreram especializações e particularizações (GOLDBARG; LUNA, 2005). O Quadro 1 exemplifica as subáreas provenientes dessas variações nas técnicas de solução.

Tipo de modelo	**Tipo de variável**
Programação linear (PL)	Contínua
Programação linear inteira (PLI ou PI)	Discreta
Programação linear inteira mista (PLIM ou PIM)	Contínua e Discreta
Programação linear binária (PLB ou PB)	Binária
Programação linear binária mista (PLBM ou PBM)	Binária e contínua
Programação linear inteira binária (PLIB ou PIB)	Discreta e binária

Fonte: Adaptado de Belfiore e Fávero (2013).

QUADRO 1 – Características dos modelos determinísticos de programação linear

Como já dito, o termo programação refere-se ao planejamento e o adjetivo linear é empregado para indicar que esse modelo é composto apenas por funções lineares. Assim, a PL busca, através do planejamento de atividades, um resultado que atinja o melhor objetivo possível dentro daquilo que foi especificado (HILLIER; LIEBERMAN, 2013).

2.2.1 Programação inteira e binária

Lachtermacher (2006) diz que em uma PM, quando todas as variáveis de decisão forem representadas apenas por números inteiros, caracteriza-se um problema de programação inteira.

A PI ainda se divide em 2 grupos: a pura e a mista. Se, em um problema de PM, apenas algumas variáveis forem inteiras, então trata-se de PI mista. Por outro lado, quando todas adotarem valores inteiros, esta é chamada de PI pura. A PI é muito comum em problemas práticos que só fazem sentido se as variáveis tiverem valores inteiros (HILLIER; LIEBERMAN, 2013).

Alguns exemplos em que há a necessidade de variáveis inteiras são: alocação de pessoal, não sendo possível trabalhar com valores decimais; maquinários e veículos, principalmente quando associados a estoque, no qual uma quantidade mínima ou máxima é estabelecida para otimização do problema e; decisões do tipo “sim” ou não”.

Variáveis de decisão que estão restritas apenas a dois valores, “0” e “1”, são chamadas de variáveis binárias. Consequentemente, nos casos em que todas as variáveis atendem a esse requisito, tem-se um modelo de programação binária (PB). Se o modelo possuir variáveis binárias e *contínuas* ele é chamado de programação binária mista (PBM). Caso seja composto apenas por variáveis *discretas* e *binárias*, o problema é caracterizado como programação inteira binária (PIB) (BELFIORE; FÁVERO, 2013).

Os problemas de PI podem até parecer de fácil resolução por apresentarem números finitos de soluções viáveis. Entretanto, esses mesmos números podem ser exageradamente grandes. Para ilustrar, Hillier e Lieberman (2013) trazem o conceito de crescimento exponencial da dificuldade do problema. Consiste no caso simples dos problemas de PB, nos quais com *n* variáveis existem 2^n soluções a serem consideradas. Assim, para cada 1 *n* acrescido o número de soluções é dobrado.

Devido a esse grande número de cálculos e a alta complexidade computacional exigida para solucionar problemas de PIB, o uso de um auxílio adequado faz-se necessário, como é o caso do Google Colab, LINGO, AIMMS, CPLEX, OSL, entre outros.

2.3 Softwares de Programação Matemática

Os softwares de otimização matemática disponíveis no mercado são essenciais para o cálculo dos problemas de PM. Um que tem sido bastante utilizado devido sua popularidade e simplicidade é o Excel, um software de planilha eletrônica da Microsoft. Ideal para problemas de pequeno porte, o Solver do Excel é capaz de resolver problemas com até 200 variáveis e 100 restrições (BELFIORE; FÁVERO, 2013).

Contudo, os problemas de grande porte e com um número expressivo de variáveis e restrições ainda enfrentam dificuldade em garantir uma solução ótima. Nesse contexto surgem as linguagens de modelagem matemática, software projetado para formular modelos grandes (HILLIER; LIEBERMAN, 2013).

Dentre os vários que existem, alguns softwares e solvers se destacam por serem utilizados para a resolução de problemas de Programação Inteira de grande porte.

2.3.1 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).

Ignácio e Ferreira Filho (2004) descrevem que o software oferece um ambiente de desenvolvimento, possibilitando que pessoas com experiência em modelagem criem aplicações funcionais as quais pessoas leigas possam utilizar.

O AIMMS possui integração com solvers de otimização como CPLEX, XPRESS, Gurobi, BARON e pode vir a incorporar outros disponíveis no mercado atualmente.

2.3.2 Python

O Python é uma linguagem de programação dinâmica e orientada a objetos que começou a ser desenvolvida ao final dos anos 80 por Guido van Rossum. Além de possibilitar a integração com outras linguagens, possui uma ampla biblioteca padrão pronta para uso (COELHO, 2007).

Devido à sua simplicidade e objetividade, a utilização do Python nas mais diversas áreas vem crescendo cada vez mais. Gratuito, pode ser usado para administrar sistemas e desenvolver grandes projetos Menezes (2010).

Por meio de sua extensa biblioteca, temos acesso a um amplo conjunto de recursos e coleção de módulos podendo ser escritos em Python e importados na forma de código. A biblioteca *pandas*, por exemplo, permite realizar a leitura de dados de planilhas eletrônicas e manipulá-los dentro do próprio software, sem ter que utilizar um programa próprio (Excel). Alguns solvers, tais como Gurobi, CBC, GLPK e CPLEX também estão disponíveis para uso a partir de bibliotecas específicas (HART, 2017)

2.3.3 Google Colaboratory

O Google Colaboratory, mais conhecido como Google Colab, é uma ferramenta baseada na tecnologia em nuvem de acesso aberto para aplicação de conhecimentos de programação em Python, adequado para aprendizado de máquina, análise de dados e educação (Carneiro et al., 2018).

Os *notebooks,* como são chamados os documentos que agrupam células de códigos, podem ser compartilhados, editados e salvos de maneira online, pelo navegador, sem a exigência de baixar, instalar ou executar qualquer software.

Por possuir integração com o serviço de armazenamento em nuvem Google Drive, dados, como planilhas eletrônicas, podem facilmente serem analisadas pelo Colab a partir de bibliotecas específicas, sem a necessidade de ter os arquivos previamente salvos no disco rígido do computador.

O Colab permite que o usuário utilize os recursos computacionais do próprio Google por meio da nuvem, disponibilizando GPU e RAM e possibilitando executar cálculos complexos que seriam inviáveis em computadores com hardwares mais simples.

3. MÉTODOS E TÉCNICAS DE PESQUISA

3.1. Tipologia e descrição geral dos métodos de pesquisa

A pesquisa desenvolvida busca estudar a viabilidade da utilização do Google Colab para resolver problemas de PI de grande porte. Do ponto de vista da natureza, trata-se de uma pesquisa aplicada. De acordo com Silva e Menezes (2005), a

pesquisa aplicada é aquela que tem como objetivo gerar conhecimentos para aplicação prática e conduzida à solução de problemas específicos.

Em seus objetivos é caracterizada como exploratória. Para Gil (2008), a pesquisa exploratória tem como objetivo desenvolver, esclarecer e modificar conceitos e ideias, proporcionando uma visão geral acerca de um tema pouco explorado e, a partir dessa primeira investigação, tornar viável a construção de hipóteses mediante procedimentos mais sistematizados.

Da perspectiva da forma de abordagem é classificada tanto como qualitativa como quantitativa, pois ao mesmo tempo em que trabalha acerca de uma modelagem matemática e sua eficiência aplicada ao Colab, busca validar sua facilidade na utilização e praticidade frente aos outros softwares disponíveis no mercado. A pesquisa quantitativa busca traduzir as opiniões e informações em números a fim de classificá-las e analisá-las (SILVA; MENEZES, 2005).

3.2. Procedimentos técnicos

A fim de esclarecer os principais conceitos, técnicas e métodos utilizados durante a pesquisa, e para atingir os objetivos propostos, realizou-se uma revisão da literatura. Segundo Silva e Menezes (2005), trata-se da fundamentação teórica adotada como base para a estruturação conceitual na qual se respaldará o desenvolvimento da pesquisa.

Muito semelhante à bibliográfica, a pesquisa documental se fez presente neste trabalho ao explorar as fontes documentais do Google Colab como o FAQ e os guias disponibilizados oficialmente pela empresa. A diferença entre esses dois tipos de pesquisa reside na natureza das fontes. Enquanto a primeira utiliza das contribuições de diversos autores sobre determinado assunto, a documental conta com materiais que ainda não receberam um tratamento analítico (GIL, 2008).

Ao selecionar um objeto de estudo e definir as variáveis que seriam capazes de influenciá-lo, determinando as formas de controle e observando os efeitos resultantes da variável no objeto, tem-se, em sua essência, a pesquisa experimental

(GIL, 2008). Durante a modelagem do problema matemático no Google Colab, diversas variáveis foram testadas até que o código estivesse funcional, seja na escolha dos *solvers* e bibliotecas, seja ao determinar o quantitativo de nós. Dentro da área de modelagem matemática, os dados foram analisados de acordo com a eficiência da aplicação do Colab e coletados a partir da própria literatura.

3.3. Instrumentos de pesquisa

Considerando estudos anteriores do grupo de pesquisa, foi definido que seria utilizado o modelo matemático de PI de grande porte desenvolvido no projeto *Safety Oversight* (ANAC, 2019), realizado em parceria entre a Universidade de Brasília (UNB) e a Agência Nacional de Aviação Civil (ANAC), o qual tinha como objetivo atender as demandas de fiscalização da agência. Se trata de um problema atual, de grande porte e que ainda não foi solucionado para casos maiores. Além disso, diversas pesquisas já foram desenvolvidas usando esse mesmo projeto como base, como o trabalho de Freitas Júnior (2017), Silva (2018), Reis e Celestino (2018), Couto (2020) e Guth (2021).

Instituída em 2005 e atuando desde 2006, a ANAC é uma das agências reguladoras federais do País, criada para regular e fiscalizar as atividades da aviação civil e a infraestrutura aeronáutica e aeroportuária no Brasil. Através das atividades de vigilância continuada e ações fiscais, a agência realiza a fiscalização da aviação civil no país assegurando níveis aceitáveis de segurança e qualidade para os passageiros (ANAC, 2023).

Guth (2021) realizou estudos comparativos entre diferentes softwares e *solvers* utilizando o mesmo modelo matemático da ANAC. Dentre os trabalhos utilizados por Guth, o desenvolvido por Couto (2020) também será utilizado como ferramenta de comparação dos modelos desenvolvidos.

4. RESULTADOS E DISCUSSÃO

4.1 Descrição do modelo

4.1.1 Caracterização do modelo

Durante as ações de fiscalização, os servidores da agência são designados para qualquer lugar do país onde exista essa necessidade. De forma a seguir o mapa estratégico da autarquia, principalmente no que diz respeito a recursos, a alocação desses profissionais deve ser feita da maneira mais eficiente possível.

Os inspetores responsáveis por realizar tais tarefas podem partir das sedes das agências ou nos Núcleos Regionais de Aviação Civil, denominados como nós de origem. As ações de fiscalização chamam-se missões e são realizadas em qualquer aeródromo do Brasil, ou seja, nós de destino, podendo ser realizadas mais de uma fiscalização no mesmo destino. Contudo, após a conclusão da missão devem retornar ao nó de origem.

Cada missão demanda uma capacidade específica, ou seja, serão alocados apenas os servidores que possuem essa habilidade técnica. Além disso, existe também a limitação de períodos que podem ser alocados de acordo com a disponibilidade, sendo essa mensurada em dias, para cada período de fiscalização.

Cada missão possui um número variado de servidores necessários e um tempo de duração também variado. O modelo considera os arcos de origem-destino formados pelas UF de origem dos servidores e aeródromos de destino das fiscalizações.

O modelo proposto é de programação linear inteira e binária, pois é caracterizado como um problema de designação. Segundo Hillier e Lieberman (2013), o problema da designação é um tipo especial de problema de programação linear e seu modelo matemático usa as seguintes variáveis de decisão binária:

$$x_{ij} = \begin{cases} 1 & \text{se o designado } i \text{ realiza a tarefa } j \\ 0 & \text{caso contrário} \end{cases}$$

Guth (2021), a partir da modelagem apresentada por Reis e Celestino (2018), desenvolveu dois modelos sobre o mesmo problema de designação: modelo de alocação estendido (MAE) e o modelo de alocação resumido (MAR). A principal diferença entre eles é que no MAR são criadas apenas as variáveis que de fato podem vir a ser utilizadas, enquanto no MAE são criadas todas as combinações possíveis de variáveis, limitadas por restrições, possibilitando o transbordo e mais combinações de duplas de inspetores. Um número maior de variáveis, combinações e restrições acaba por impactar significativamente o desempenho dos softwares e inviabilizar sua solução em tempo hábil. Desta forma, o MAR otimiza esse desempenho por se tratar de um modelo com uma quantidade reduzida de dados.

4.2 Modelo de Alocação Estendido (MAE)

4.2.1 Índices

O Modelo de Alocação Estendido é composto pelos seguintes índices:

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
GR	Grupo	Capacitações necessárias

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

4.2.2 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)
$Ofeta_{p,gr,o}$	Oferta do servidor (p) para cada atividade (gr) e origem (o)

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

QUADRO 3 – Parâmetros do modelo MAE

4.2.3 Variáveis

O modelo MAE é composto por quatro variáveis de decisão sendo duas inteiras binárias e duas contínuas, conforme o 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).

4.2.4 Equações

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

Sujeito a:

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

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

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

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

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

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

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

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

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

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

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

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

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

$$Atendida_{t,k,gr,d,p} \in \mathbb{Z}^{+}\{0,1\}$$

4.2.5 Função Objetivo (FO) e Restrições

O modelo matemático busca encontrar uma solução ótima com a otimização dos custos das passagens aéreas por meio da minimização da Função Objetivo (FO). A FO (1) é formada pelo somatório dos custos da alocação geral dos servidores designados multiplicado por dois, pois é considerado a ida e a volta.

A restrição (2) diz que só podem ser alocados nas missões os servidores que tem oferta na sua origem, capacitada no grupo GR.

Na restrição (3) tem-se o balanço dos fluxos, determinando que cada servidor designado que chega em cada destino para realizar uma missão pode sair para realizar outras missões. Essa restrição permite o transbordo com a adição do somatório de alocado, ou seja, a realização de mais uma missão no mesmo destino, ou em outro, a partir do aproveitamento da primeira viagem.

A restrição (4) garante que nenhum fluxo será gerado nos nós de destino, garantindo a possibilidade do transbordo a partir do somatório em O do Alocado sendo menor ou igual ao somatório em D do alocado.

Para formar as equipes que realizarão cada missão, a restrição (5) faz o 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.

As restrições (6) e (7) dizem respeito ao tempo. A primeira indica o tempo utilizado considerando o tempo gasto na missão e o tempo de viagem para atendê-la. A segunda determina que o tempo utilizado sempre será menor ou igual que a disponibilidade de cada servidor que é dada em dias/período.

A restrição (8) limita a cobrança duplicada de um mesmo servidor para um determinado trecho caso ela venha a realizar mais de uma missão.

Por fim, a restrição (9) estabelece o universo dos conjuntos das variáveis, no qual as variáveis Alocado e TempoUtilizado pertencem aos números reais positivos enquanto a variáveis Atendida e AlocadoGeral são binárias, representadas por 1 ou 0.

4.3 Modelo de Alocação Resumido (MAR)

4.3.1 Índices

O Modelo de Alocação Estendido é composto pelos seguintes índices:

QUADRO 5 – Índices do modelo MAR

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

Fonte: Adaptado de Guth (2021)

4.3.2 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 Guth (2021)

4.3.3 Variáveis

O modelo MAR é composto por duas variáveis de decisão binárias, conforme o Quadro 7.

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 Guth (2021)

QUADRO 7 – Variáveis do modelo MAR

4.3.4 Equações

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

Sujeito a:

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

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

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

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

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

4.3.4 Função Objetivo (FO) e Restrições

O modelo matemático busca encontrar uma solução ótima com a otimização dos custos das passagens aéreas por meio da minimização da Função Objetivo (FO). A FO (10) do modelo MAR é formada pelo somatório dos custos da alocação geral dos servidores designados multiplicado por dois, pois é considerado a ida e a volta.

A restrição (11) determina que sejam alocados para as missões apenas os agentes que são capazes de realizar a atividade, considerando a equipe necessária para realizá-la.

As restrições (12) e (13) dizem sobre a disponibilidade dos servidores para realizar as fiscalizações e forçam o modelo a alocar o servidor para realizá-la, respectivamente.

Já as restrições (14) e (15) determinam que ambas as variáveis do MAR são binárias, assumindo valores de 1 ou 0.

4.4 Implementação do modelo

Ambos os modelos matemáticos descritos nas seções anteriores foram executados no Google Colab, utilizando o modelo elaborado por Guth (2021) escrito na linguagem de programação Python e os modelos desenvolvidos por Couto (2020) utilizando os softwares LINGO/AIMMS. Algumas adaptações no código foram necessárias para o bom funcionamento no Colab.

Para a leitura dos dados organizados no software Excel, utilizou-se a biblioteca “Pandas”. Já para a estruturação das equações e tratamento dos dados, foi

utilizada a biblioteca “Pyomo” (*Python Optimization Modeling Objects*), que possui suporte à maioria dos principais solvers.

Foi utilizado para a resolução do problema o CBC (*Coin-or branch and cut*), um solver de programação linear inteira, de código aberto, muito recomendado para modelos de médio e grande porte. O algoritmo do CBC funciona como uma combinação de outros dois métodos: *branch-and-bound* e *cutting-planes*.

O código foi executado em um computador com Windows 10 Pro 64bits, processador Intel(R) Core(TM) i3-4160 CPU @ 3.60GHz e 8 GB de memória RAM.

4.5 Instâncias e resultados gerais

Guth (2021) criou 6 diferentes instâncias com variações entre os principais parâmetros de entrada do modelo. O objetivo era indicar as diferenças na capacidade de resolução dos modelos MAE e MAR. As seguintes classificações foram utilizadas para identificar e explicitar as diferentes instâncias:

• 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 1 – Características das instâncias implementadas

	Instâncias					
Classificações	A1	A2	A3	B1	B2	B3
Servidores	151	151	96	124	125	76
Disponibilidade	20	12	15	5	5	5
Origens	3	3	3	16	16	15
Destinos	22	22	21	30	74	66
Missões	118	118	80	50	120	100
Atividades	23	23	23	21	41	30

Fonte: Adaptado de Guth (2021)

A partir dos modelos MAE e MAR, e da criação das instâncias por Guth (2021), os cenários foram testados no Google Colab, utilizando o solver CBC, e o quantitativo referente às restrições e variáveis de cada instância estão representados na Tabela 2. Na tabela também estão descritos os números das variáveis e restrições encontradas por Guth (2021) ao implementar o modelo no LINGO, AIMMS e Python com os solvers LINGO, CPLEX e Gurobi, respectivamente.

Tabela 2 – Quantitativo das variáveis e restrições referentes a cada instância em cada modelo

		A1		A2		A3		B1		B2		B3	
Software	Modelo	V	R	V	R	V	R	V	R	V	R	V	R
Colab solver CBC	MAE	14446	9981	14446	9981	7216	4786	7619	4121	17349	8229	10115	5091
Colab solver CBC	MAR	13947	9829	13947	9829	6904	4689	6418	4020	13775	8229	8408	5091

LINGO solver LINGO	MAE	1.023.144	1.429.497	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
LINGO solver LINGO	MAR	14243	11612	14243	11612	10841	8.329	6.464	4.089	13858	8459	13196	8060
AIMMS solver CPLEX	MAE	1.022.878	1.667.589	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
AIMMS solver CPLEX	MAR	-	-	-	-	-	-	-	-	-	-	-	-
Python solver Gurobi	MAE	264.340	263.141	264.340	263.141	119.021	118.400	125.355	122.924	620.849	613.615	342.441	338.565
Python solver Gurobi	MAR	14.244	11.612	14.244	11.612	10.841	8.329	6.464	4.089	13.858	8.459	13.196	8.060

Fonte: Adaptado de Guth (2021)

Percebe-se que o objetivo do modelo de alocação resumido (MAR) de criar apenas as variáveis que realmente seriam utilizadas é atingido. Ao comparar o número de variáveis e restrições de cada instância aplicadas em cada modelo, percebe-se que, em 100% dos casos válidos de comparação, a quantidade em MAE é maior.

O modelo de alocação resumido não foi executado no AIMMS, por esse motivo seus resultados estão representados por “-”.

Percebe-se que o problema executado no Google Colab obteve um quantitativo menor de variáveis e restrições, tanto no modelo de alocação estendido quanto no resumido.

Para a instância A1, em comparação com o LINGO, o MAE executado no Colab apresentou uma redução de cerca de 98,6% no número de variáveis e de 99% no número de restrições. Essa redução significativa ocorreu pois, na linguagem Python, o MAE cria somente as combinações que poderiam ser realizadas,

considerando a origem do servidor, grupo e missão. Além disso, não considera as variáveis zero, trabalhando apenas com as variáveis inteiras.

Ao analisarmos os modelos testados nas mesmas linguagens de programação, essa diferença cai consideravelmente, mas apenas no MAR, visto que no modelo estendido ela continua significativa em todas as instâncias. Enquanto a B2 seguindo o modelo MAR no Colab obteve 6418 variáveis e 4020 restrições, a mesma instância quando executada no software Python resultou em 6464 variáveis e 4089 restrições, uma diferença de apenas 0,07% e 1,7% respectivamente.

Consequentemente, por ter menos combinações, essas diminuições podem ser observadas nos tempos de solução de cada um dos modelos e instâncias, conforme seções a seguir.

4.5.1 Resultados da Instância A

Os resultados obtidos pela resolução de ambos os modelos (estendido e resumido), considerando Função Objetivo, tempo de solver e tempo total, estão consolidados na Tabela 3, compilando as instâncias A1, A2 e A3 para cada software e modelo testado.

Tabela 3 – Resultados para Instância A

Software	Modelo	Instância	Função Objetivo	Tempo de Construção (s)	Tempo Solver (s)	Tempo Total (s)
Google Colab solver CBC	MAE	A1 A2 A3	27100 28300* 24953	- - -	- - -	9.83 >1800s 15,72
LINGO solver LINGO	MAE	A1 A2 A3	27100 28286* 24953	- - -	- - -	49 >1800 25
AIMMS solver CPLEX	MAE	A1 A2 A3	27100 28286 24953	47,55 38,78 46,58	9,84 130,61 6,22	57,39 169,39 52,8
Python solver Gurobi	MAE	A1 A2 A3	27100 28286 24953	173,09 169,42 76,63	7,82 113,14 4,39	180,91 282,56 81,02
Google Colab solver CBC	MAR	A1 A2 A3	27100 28300* 24953	- - -	- - -	8,8 >1800s 13,17
LINGO solver LINGO	MAR	A1 A2 A3	27100 28300* 24953	- - -	- - -	9 >1800s 6,56
AIMMS solver CPLEX	MAR	A1 A2 A3	- - -	- - -	- - -	- - -
Python solver Gurobi	MAR	A1 A2 A3	27100 28286 24953	7,07 6,97 5,03	2,17 68,52 1,46	9,24 75,49 6,49

Fonte: Adaptado de Guth (2021)

Assim como o LINGO, o Colab, ao utilizar o CBC como solver, não faz a distinção entre os tempos de construção e de solver, resultando apenas no tempo total gasto para a resolução do modelo matemático. Os resultados do campo “Função Objetivo” representados por “*” indicam que aquela solução foi encontrada e pode ser considerada válida, ainda que o modelo tenha excedido o tempo estipulado de 1800 segundos para encontrar uma solução ótima. Valores de tempo representados por “>1800s” remetem a essa situação.

Ao analisar os cenários do MAE, observa-se que o Google Colab apresentou o melhor tempo total de solução em todas as instâncias, exceto a A2 em que o tempo limite de solução foi excedido, assim como no LINGO. O AIMMS teve um desempenho 40% melhor do que o modelo implementado no Python. O destaque fica para o Colab na instância A1, no qual o solver conseguiu solucionar o problema em 9,83 segundos, um desempenho 80% melhor do que o LINGO que resolveu em 49 segundos.

No MAR, o Google Colab encontra a solução mais rapidamente que os outros softwares apenas na instância A1, resolvendo em 8,8s, enquanto o LINGO resolveu em 9s.

O resultado da instância A2 implementada no Python foi a única que conseguiu solucionar o problema matemático sem que o tempo limite de 1800 segundos fosse ultrapassado. Com um tempo total de 75,49 segundos, o Python inclusive apresentou uma Função Objetivo ótima de 28.286, enquanto o LINGO e o Colab só conseguiram alcançar o melhor valor possível de 28.300.

Já para a instância A3, o Colab apresentou o pior resultado dentro os tempos totais, utilizando o dobro do tempo para solucionar o problema, 13,17 segundos. Apesar disso, todos os *solvers* atingiram o mesmo valor para a FO.

4.5.2 Resultados da Instância B

Os resultados obtidos pela resolução de ambos os modelos (estendido e resumido), considerando Função Objetivo, tempo de solver e tempo total, estão consolidados na Tabela 4, compilando as instâncias B1, B2 e B3 para cada software e modelo testado.

Tabela 4 – Resultados para Instância B

Software	Modelo	Instância	Função Objetivo	Tempo de Construção (s)	Tempo Solver (s)	Tempo Total (s)
Google Colab solver CBC	MAE	B1 B2 B3	29058* 87653* 61622*	- - -	- - -	>1800s >1800s >1800s
LINGO solver LINGO	MAE	B1 B2 B3	29058* 86693* 61223*	- - -	- - -	>1800s >1800s >1800s
AIMMS solver CPLEX	MAE	B1 B2 B3	29058 85800 60350	215,28 334,36 297,45	67,02 151,72 129,91	282,3 486,08 427,36
Python solver Gurobi	MAE	B1 B2 B3	29058 85800 60350	103,17 514,77 252,4	24,67 69,16 28,47	127,84 583,93 280,87
Google Colab solver CBC	MAR	B1 B2 B3	29058* 85806* 60769*	- - -	- - -	>1800s >1800s >1800s
LINGO solver LINGO	MAR	B1 B2 B3	29058* 85998* 78412*	- - -	- - -	>1800s >1800s >1800s
AIMMS solver CPLEX	MAR	B1 B2 B3	- - -	- - -	- - -	- - -
Python solver Gurobi	MAR	B1 B2 B3	29058 85800 60350	5,75 11,57 13,1	21,07 48,46 29,53	26,82 60,03 42,63

Fonte: Adaptado de Guth (2021)

Como explicado na análise dos resultados das instâncias “A”, os resultados do campo “Função Objetivo” representados por “*” indicam que aquela solução foi encontrada e pode ser considerada válida, ainda que o modelo tenha excedido o tempo estipulado de 1800 segundos para encontrar uma solução ótima. Valores de tempo representados por “>1800s” remetem a essa situação.

O Google Colab, tanto no MAE quanto no MAR, não conseguiu encontrar nenhuma solução sem que ultrapassasse o tempo limite de 1800 segundos. Contudo,

esse tempo excedido diz respeito à tentativa de reduzir o gap, pois alcançou os mesmos resultados do LINGO, AIMMS e Python para a FO na instância B1 e apresentou uma solução ligeiramente pior nas outras instâncias.

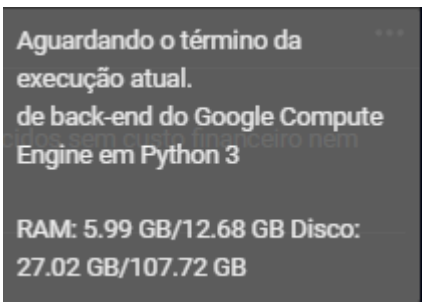
O gap refere-se a diferença entre o valor da melhor solução encontrada até o momento e a solução ótima, fornecendo uma medida em porcentagem da otimalidade da solução encontrada em relação à ótima. Reduzir esse gap significa zerar essa diferença, alcançando um gap de 0% e indicando que a solução encontrada é ótima.

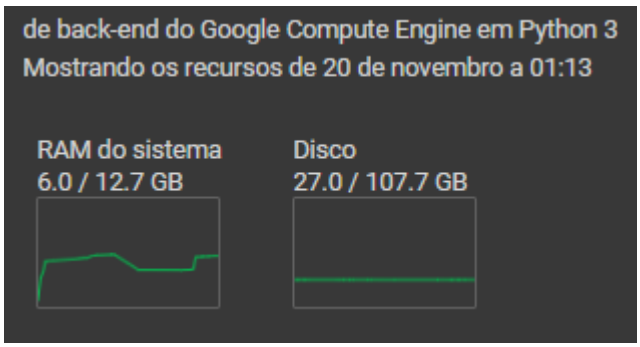
O Python se destacou ao solucionar a instância B1 do MAE em 127,84 segundos, um desempenho 55% melhor do que o AIMMS, o qual resolveu em 282.3 segundos.

Guth (2021) constatou que a exigência computacional que cada software exigia para resolver a instância B2 do MAE era consideravelmente alta. No Python, utilizou-se 7GB de memória RAM e 14% de CPU. No LINGO, 12GB de RAM e 13% de CPU. E no AIMMS, 22GB de RAM e 99% de CPU. Guth utilizou uma máquina com 32 GB de RAM, ou seja, quase 76% da memória total foi exigida para que o modelo fosse implementado.

Por outro lado, pelo fato do Colab ser hospedado em nuvem e utilizar recursos da própria Google, verificou-se que o pico exigido de RAM não ultrapassou 6GB e tampouco impactou no uso da máquina, permitindo o uso contínuo enquanto o solver resolvia o modelo.

Figura 3 – Exigência computacional da instância B2 implementada no Google Colab





Fonte: Autor

4.6 Análise dos modelos e instâncias

Quando analisados os modelos implementados e os softwares em que foram testados, destacam-se aqueles que possuem uma linguagem de programação aberta, ou seja, o Python e o Colab. Com desempenhos próximos a 99% na redução do quantitativo de variáveis e restrições, o Colab, através do solver CBC, simplificou os modelos e solucionou as instâncias A1 e A3 do MAE 40% mais rápido que os outros softwares.

Importante destacar que os *solvers* Gurobi, utilizado pelo Python, e o CPLEX, utilizado pelo AIMMS, são comerciais. Por outro lado, o CBC, utilizado pelo Google Colab, é de código aberto, gratuito, e foi capaz de solucionar as instâncias comentadas de maneira mais eficiente que os *solvers* pagos.

Ainda que nas instâncias B1, B2 e B3 não tenha conseguido solucionar em um tempo viável, alcançou ou os mesmos valores ou valores muito próximos para a Função Objetivo enquanto continuava rodando com o intuito de diminuir o gap.

5. CONCLUSÕES E RECOMENDAÇÕES

Perante a quantidade escassa de estudos em relação à pequena variedade de softwares utilizados na resolução de problemas de programação linear inteira de grande porte, e o uso de *solvers* comerciais, observou-se a necessidade de desenvolver essa pesquisa para averiguar a viabilidade do uso do Google Colaboratory e, principalmente, de um *solver* gratuito que desempenhe as mesmas funções de maneira satisfatória.

Na seção 1.4 deste estudo, foram listados os objetivos específicos os quais foram cumpridos durante o processo de pesquisa. O primeiro consistia em encontrar um modelo de PI de grande porte, e a decisão de utilizar o projeto *Safety Oversight*, oriundo da parceria entre ANAC e UNB, se mostrou adequado, uma vez que outros estudos utilizaram esse mesmo projeto para atestar a viabilidade de outros softwares. Com a escolha desse modelo matemático, o objetivo de testar cenários e comparar os tempos de resolução entre o Colab e os softwares/solvers pagos foi facilitado.

Ao compararmos esses tempos totais de resolução, foi constatado que os modelos quando implementados no Colab, juntamente com o CBC, atingiram resultados consideráveis por se tratar de ferramentas gratuitas, performando bem na maioria dos cenários e até mesmo melhor do que as opções comerciais já conhecidas.

Durante esta pesquisa, algumas limitações impossibilitaram uma maior riqueza nas análises, principalmente na questão do comparativo entre os tempos de solução de cada modelo. Por não dispor de estudos científicos sobre o tema, muitas conclusões foram feitas através das hipóteses sobre o funcionamento do Colab. A falta de respaldo técnico, inclusive o disponibilizado pela própria Google e/ou fóruns dedicados, dificultou o processo de modelagem e de entendimento das linguagens de programação, principalmente na escolha das bibliotecas do Colab. Outro desafio ocasionado pela falta de informação para cenários específicos foi a interpretação dos resultados, impossibilitando a realização de uma análise sobre o processo de solução do problema matemático que antecede o resultado final encontrado pelo *solver*, ou seja, o processo de branch-and-bound e cut feito pelo *solver.*

Todos os modelos foram testados e validados por um único computador, modelo e *solver*, inviabilizando que decisões genéricas fossem tiradas.

A fim de contribuir para esse estudo e enriquecer o conhecimento científico acerca da viabilidade do Colab frente os outros softwares de otimização, sugere-se a resolução do modelo utilizado nessa pesquisa e de outros modelos de PI de grande porte utilizando outros *solvers* de código aberto e bibliotecas do Colab. A comparação do CBC com outros *solvers* como o GLPK ou o próprio Gurobi pode atestar que a redução das variáveis e restrições é consequência do algoritmo utilizado pelo CBC.

Sugere-se a implementação do modelo em diferentes computadores para verificar se o hardware tem algum impacto na resolução do modelo hospedado em nuvem ou se de fato os recursos computacionais utilizados são apenas os disponibilizados pela Google. Outra sugestão seria determinar que o solver parasse de tentar solucionar o problema assim que encontrasse os mesmos valores de FO encontrados nos softwares comparados.

A expansão da pesquisa acerca desse tema, e o consequente aumento do número de comparações válidas entre os modelos matemáticos de programação inteira de grande complexidade, irá contribuir com a disseminação do conhecimento acerca desses softwares e sua utilidade a níveis estratégicos para as organizações.

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.

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

BISSCHOP, J. & Entriken, R. (2002). AIMMS the Modeling System. Paragon Decision Technology. .

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

CARNEIRO, T.; MEDEIROS DA NÓBREGA, R. V.; NEPOMUCENO, T.; BIAN, G.; DE ALBUQUERQUE, V. H. C.; FILHO, P. P. R. Performance analysis of Google Colaboratory as a tool for accelerating deep learning applications. IEEE Access, v. 6, p. 61677–61685, 2018.

CBC, COIN-OR. Disponível em: < https://github.com/coin-or/Cbc >. Acesso em: 15 out. 2023

COELHO, Flávio Codeço. Computação Científica com Python: Uma introdução à programação para cientistas. 1. ed. Petrópolis, RJ: [Editora], 2007.

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.

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

FERREIRA FILHO, V. J. M.; IGNÁCIO, A. A. V. "O uso de software de modelagem AIMMS na solução de problemas de programação matemática". Pesquisa Operacional. v. 24, n. 1, pp. 197-210, 2004

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.

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

GOLDBARG, M. C.; LUNA, H. P. L. Programação linear e otimização combinatória: modelos e algoritmos. 2.ed. – Rio de Janeiro: Elsevier, 2005

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. In: 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.

J. E. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. In P. M. Pardalos and M. G. C. Resende, editors, Handbook of Applied Optimization, pages 65–77. Oxford University Press, 2002.

KENWORTHY, T. and Balakrishnan, J. (2016), "Theory usage in empirical operations management research: a review and discussion", Management Decision, Vol. 54 No. 10, pp. 2413-2432.

LACHTERMACHER, G.. Pesquisa Operacional na Tomada de Decisões: modelagem em EXCEL. 5ª edição. Rio de Janeiro: LTC, 2016.

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.

NUMPY. Disponível em: <http://www.numpy.org\>. Acesso em: 10 set. 2022.

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.

PYTHON, Software Foundation. Python documentation. Disponível em: http://docs.python.org/. Acesso em: 20 out. 2023.

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

TOWHIDI, M., Orban, D. Customizing the solution process of COIN-OR’s linear solvers with Python, 2016.

WINSTON, W. L.; GOLDBERG, J. B. Operations research: applications and algorithms. 4ed. Thomson Brooks/Cole, 2004.

APÊNDICES

Apêndice A – Código do modelo MAE

```
---------- INSTALAÇÃO E IMPORTAÇÃO DAS BIBLIOTECAS E SOLVER--------
!pip install pyomo
!pip install xlrd
!pip install -U -q PyDrive
!apt-get install -y -qq coinor-cbc

from pyomo.opt import SolverFactory
import pandas as pd
import time
from google.colab import drive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

---------- UTILIZANDO O GOOGLE DRIVE PARA IMPORTAR OS DADOS ----------

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

file_id = 'CÓDIGO_DO_ARQUIVO_EXCEL_AQUI'
downloaded = drive.CreateFile({'id': file_id})

downloaded.GetContentFile('exported.xlsx')

------------------------------ LEITURA DOS DADOS DO EXCEL  ------------------------------

!ls -lha exported.xlsx

ef = pd.read_excel('exported.xlsx')
ef

disponibilidade = pd.read_excel('exported.xlsx', 'Disponibilidade -
PREENCHER', header=1, usecols=['NOME', 'UF',  'DISPONIBILIDADE'])
arcos           = pd.read_excel('exported.xlsx', 'Arcos',
header=1, usecols=['ORIGEM_ARCO', 'DESTINO_ARCO', 'CUSTO', 'TEMPO'])
```

```
atividade       = pd.read_excel('exported.xlsx', 'Atividades',
header=1, usecols=['GRUPO_A', 'DURACAO', 'EQUIPE'])
oferta          = pd.read_excel('exported.xlsx', 'Atividades',
header=1, usecols=['SERVIDOR',  'GRUPO_P'])
nos             = pd.read_excel('exported.xlsx', 'Arcos',
header=1, usecols=['NOS', 'ORIGEM_NO', 'DESTINO_NO'])
demanda         = pd.read_excel('exported.xlsx', 'Demanda - PREENCHER',
header=1, usecols=['PERIODO', 'MISSAO', 'ATIVIDADE', 'AEROPORTO' ])

-------- LIMPEZA DOS DADOS QUE NÃO SERÃO UTILIZADOS NO MODELO --------


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()

---------- 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))

#print(len(pessoa))
#print(len(grupo))
#print(len(origem))
#print(len(destino))
```

```
#print(len(missao))



---------- TABELA DE DEMANDAS EM 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)

-----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')

----------DICIONÁRIO DE PESSOA/GRUPO ATIVO ----------

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

----------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

----------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')



---------------------------- DICIONÁRIO DAS ATIVIDADES ATIVAS --------------------------

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

------------------------------- DICIONÁRIO DOS ARCOS ---------------------------------------

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

---------------------- DICIONÁRIO DE DISPONIBILIDADE DE CADA PESSOA ----------

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

---------------------------------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)
```

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

Apêndice B – Código do modelo MAR

```
!pip install pyomo
!pip install xlrd
!pip install -U -q PyDrive

from pyomo.opt import SolverFactory
import pandas as pd
import time
from google.colab import drive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
```

```
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

file_id = '1QVI40sDLP3XqmQEaUbeKSDTSg9IPA6w6'
downloaded = drive.CreateFile({'id': file_id})

downloaded.GetContentFile('exported.xlsx')

!ls -lha exported.xlsx

ef = pd.read_excel('exported.xlsx')
ef

#Leitura dos dados
DEM = pd.read_excel('exported.xlsx', 'DADOS', header = 0, usecols =
['MISSAO1', 'ATIVIDADE', 'DESTINO1', 'DURACAO', 'EQUIPE'], converters =
{'MISSAO1': int}).dropna()
PD = pd.read_excel('exported.xlsx', 'DADOS', header = 0, usecols =
['PESSOA1', 'DISPONIBILIDADE']).dropna()
ORIGENS = pd.read_excel('exported.xlsx', 'DADOS', header = 0, usecols =
['ORIGENS']).dropna()
DESTINOS = pd.read_excel('exported.xlsx', 'DADOS', header = 0, usecols
= ['DESTINOS']).dropna()
ARCOS = pd.read_excel('exported.xlsx', 'DADOS', header = 0, usecols =
['ORIGEM2', 'DESTINO2', 'CUSTO', 'TEMPO']).dropna()
ALOC = pd.read_excel('exported.xlsx', 'DADOS', header = 0, usecols =
['PESSOA3', 'ORIGEM3', 'DESTINO3']).dropna()
ALOC_ATIV = pd.read_excel('exported.xlsx', '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')

opt = pyo.SolverFactory('gurobi')

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)

#R1
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)

#R2
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)
```

```
#R3
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)

!apt-get install -y -qq coinor-cbc

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