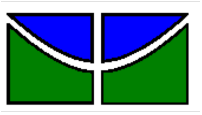


Universidade de Brasília

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

Departamento de Administração

ANDRÉ CREMA BORGES MARQUES

Estudo comparativo entre AIMMS e Python no Google Colab para um modelo de programação inteira de minimização de custos agrícola

Brasília – DF

2023

ANDRÉ CREMA BORGES MARQUES

Estudo comparativo entre AIMMS e Python no Google Colab para um modelo de programação inteira de minimização de custos agrícola

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

Professora Orientadora: Profª Drª Silvia Araújo dos Reis

Brasília – DF

2023

Crema Borges Marques, André.
Estudo comparativo entre AIMMS e Python no Google Colab para um modelo de programação inteira de minimização de custos agrícola / André Crema Borges Marques. - Brasília, 2023
66 f. : il

Monografia (bacharelado) - Universidade de Brasília, Departamento de Administração, 2023.
Orientador: Prof.ª Dr.ª Silvia Araújo dos Reis, Departamento de Administração.

1. Pesquisa operacional. 2. Programação binária. 3. Modelagem matemática. 4. Python. 5. Agricultura orgânica. I. Título.

ANDRÉ CREMA BORGES MARQUES

Estudo comparativo entre AIMMS e Python no Google Colab para um modelo de programação inteira de minimização de custos agrícola

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

André Crema Borges Marques

Profª Drª Silvia Araújo dos Reis

Professor-Orientador

Profª. Drª. Olinda Maria Gomes Lessa,

Professor-Examinador

Prof. Dr. Victor Rafael R. Celestino

Professor-Examinador

Brasília, 25 de julho de 2023

A Alícia, Ana Lúcia e João Batista por todo o apoio prestado ao longo dos anos para que eu pudesse alcançar uma educação superior.

RESUMO

Este trabalho apresenta um estudo comparativo entre dois métodos de solução de um mesmo problema de otimização. A tomada de decisão e solução de problemas, implica em conhecer e desenvolver técnicas que permitam pesquisar soluções satisfatórias em tempo hábil para o problema em questão. Ferramentas e métodos de apoio à tomada de decisão, dessa maneira, devem buscar um caminho mais rápido para alcançar um resultado. O objetivo deste trabalho é comparar uma solução de modelo matemático de otimização em cadeia de suprimentos de alimentos orgânicos resolvido com a ferramenta paga *AIMMS Developer* e aplicar o mesmo modelo - com as devidas traduções e adaptações - a uma técnica de solução usando programação computacional com em linguagem *Python* 3.10.12 e compartilhar a solução via caderno virtual gratuito, valendo de computação em nuvem, chamado Google Colab, usando, *solvers* nativos gratuitos como GLPK e CBC e integrados como HiGHS. Foi possível concluir que o *AIMMS Develope*r é preferível para solução de problemas de grande porte, mas é possível realizar as mesmas operações com ferramentas gratuitas.

Palavras-chave: Pesquisa operacional; Programação binária; Modelagem matemática; Python.

LISTA DE TABELAS

LISTA DE IMAGENS

SUMÁRIO

1. INTRODUÇÃO

Um estudo apresentado em 2022 pela organização Rehagro aponta o Agronegócio como um dos segmentos econômicos com evolução mais veloz e mais capaz de gerar riqueza, trazendo cerca de 26% de todo o Produto Interno Bruto (PIB) do Brasil, gerando receitas que superam a grandeza dos U$ 100 bilhões de dólares no ano de 2020, segundo estudo CEPEA realizado em 2021. Uma das causas desse aumento e destaque do Brasil se dá pelo protagonismo da soja nas demandas de países que buscam importar o produto, como Estados Unidos da América e China.

Apoiado de desenvolvimentos tecnológicos tanto no âmbito técnico da agricultura em si, com maquinário e semente (ZAMBALDE, 2011), mas também com uma maior possibilidade de processamento e análise de dados para tomada de decisões gerenciais (RINALDI, 2014) o Brasil vem apresentando esse crescimento e destaque no agronegócio mundial.

Uma ferramenta para análise gerencial de dados que tem ganhado considerável força é a linguagem de programação computacional Python. Isso se dá devido a uma facilidade de uso e vasta galeria de possíveis repositórios de ferramentas de código para viabilizar diferentes modos de processamento de dados e, dessa maneira, diferentes modos de visualização de resultados e pontos de vista para auxílio na tomada de decisão gerencial.

Azevedo (2022) discutiu um problema de programação inteira com o objetivo de buscar uma otimização rota de tratores e maquinário disponível, sob as condições de tempo e limitação de espaço e recursos humanos a fim de reduzir custos com combustível e de acionamento de maquinário. Para isso foi usada a programação inteira com o auxílio da ferramenta *AIMMS* com auxílio de *Microsoft Excel*. Devido a uma dificuldade no volume de variáveis estudadas, o programa não apresentou uma solução em tempo ideal.

Baseado nesse trabalho, será discutida uma comparação da solução buscada com a ferramenta *AIMMS* e uma solução alternativa, obedecendo a mesma lógica do modelo de Azevedo (2022), contudo em uma abordagem de programação

computacional utilizando linguagem *Python* com o apoio de plataforma online colaborativa chamada *Google Colab*.

1.1. Contextualização

A função da teoria administrativa é interpretar objetivos e transformá-los em ação (CHIAVENATO, 2004). Se o objetivo é a redução de custos, aumento de alcance, de percepção, de entrega, etc é função do administrador entender o que são os passos que viabilizam esse objetivo. Muitas vezes, a gestão de recursos significa saber aproveitar da melhor maneira possível os recursos finitos disponíveis a fim de tirar o maior resultado. Assim, é justo assumir que o estudo da otimização de recursos é parte essencial do estudo da gestão.

Em uma ótica de otimização, é possível observar o custeio do plantio e colheita de produtos orgânicos - para além das sementes, adubos, insumos agrícolas, consideração de rotação de cultura, etc (RECH; GONÇALVES; VIEIRA, 2019) - passa também por custos de maquinário.

Em um mundo em constante evolução tecnológica, é necessário que, para manter vantagem competitiva e buscar aumento de resultados, sejam buscadas novas técnicas que lancem mão dos avanços tecnológicos para o auxílio na tomada de decisão. Azevedo (2022) em sua monografia explorou os custos operacionais de roteirização ligados justamente ao maquinário, considerando que o gasto de combustível pode ser largamente reduzido se a rota do maquinário de colheita e plantio for otimizada. Assim, Azevedo (2022) realizou o estudo se valendo da Programação Inteira, auxiliado pelo programa AIMMS com o método conhecido como “Problema do Caixeiro Viajante” (*Traveling Salesman Problem,* TSP) para encontrar a melhor maneira de roteirizar a colheita e otimizar custos. Não encontrando solução em tempo viável para problemas com muitas variáveis, validou o modelo com problemas menores.

Há, contudo, métodos alternativos para alcançar resultados semelhantes, talvez de maneira mais rápida e gratuita. Um desses possíveis métodos é a programação computacional por meio de linguagem de código, como por exemplo a

linguagem *Python* que se propõe a ser uma linguagem de aprendizado acessível com o foco em análise de dados utilizado na ferramenta *Google Colab* que se propõe a ser uma ferramenta colaborativa de desenvolvimento *Python* armazenada em nuvem.

1.2. Formulação do problema

A linguagem de programação utilizada prioritariamente pela ferramenta *AIMMS*, apesar de fácil compreensão para pessoas habituadas, traz consigo uma dificuldade de manutenção, sendo a linguagem *Python* muito difundida e de documentações de fácil acesso para eventuais dúvidas além de grande possibilidade de integração com outras ferramentas, como *plotagem* de gráficos, exportações dinâmicas e, de maneira geral, grande personalização das funções.

Se o objetivo é a economia de recursos a fim de otimizar o custo de produção de alimentos orgânicos, em um caso em que o maquinário pode variar, assim como a mão de obra disponível, não é viável que se decorra tempos elevados diariamente na busca de uma otimização.

Outro aspecto da otimização de custos é o custo da ferramenta *AIMMS Developer* que exige uma licença paga (ou acadêmica) para ser utilizada com toda sua capacidade. O *Google Colab*, como uma plataforma colaborativa de desenvolvimento, é gratuita e não necessita do uso de máquinas físicas. É um caderno de desenvolvimento virtual que mantém seus dados com integração ao armazenamento de nuvem *Google Drive*, que também é gratuito. O limite dessa gratuidade é o limite de armazenamento das ferramentas.

Tendo a Pesquisa Operacional grande importância na tomada de decisão gerencial é possível que existam soluções gratuitas tão eficazes quanto soluções proprietárias com custo elevado?

1.3. Objetivo Geral

O presente trabalho tem como objetivo estudar e comparar a eficiência dos softwares *AIMMS Developer* e código *Python* armazenado no Google Colab, em relação à resolução de um modelo de programação inteira para minimização de custos agrícolas.

1.4. Objetivos Específicos

Para alcançar o objetivo geral, foram definidos os objetivos específicos que seguem:

Analisar o modelo matemático usado na ferramenta AIMMS por Azevedo (2022)

Reformular o mesmo problema de roteirização com código *Python*;

Disponibilizar código em “caderno” *Google Colab*;

Comparar as duas soluções em alguns critérios:

- Tempo de solução;
- Facilidade de alteração de dados de entrada;
- Compreensão da informação apresentada;

1.5. Justificativa

Com a presença ubíqua da tecnologia nas vidas cotidianas, na indústria e no processo decisório das gerências, é importante que se entenda a melhor maneira de alcançar uma sintetização de dados crus para análise. Assim buscando não só uma otimização de resultados, mas também uma otimização de processos, auxiliando no processo de escolha.

De um ponto de vista gerencial, é necessário comparar e questionar os benefícios da implementação de novas tecnologias (assim como seus respectivos

técnicos) com os métodos tradicionais. Comparando seus custos, tempos de operação e reflexo nos resultados.

2. REFERENCIAL TEÓRICO

2.1. Modelagem de problema em Pesquisa Operacional

De acordo com Longaray (2013), Pesquisa Operacional (PO) se refere a um compilado de técnicas que, com o uso do método científico, guiam a tomada de decisão dos atores competentes. Longaray apresenta a PO como uma disciplina que tem ganhado espaço em âmbitos práticos, além de acadêmicos.

Em livro destinado ao primeiro contato com a PO (Longaray, 2013), são enumerados passos do processo de definição e modelagem de um problema de pesquisa operacional, evidenciando que a definição correta dos parâmetros de um problema define o êxito da solução mais adequada. A modelagem se refere ao método matemático que vai resolver o problema. Sendo uma representação simbólica e signológica das interações entre os itens e parâmetros identificados no reconhecimento do problema, de maneira que seja possível uma resolver fazendo uso de equações algébricas para a solução. A modelagem matemática busca uma maneira genérica de representar um problema de PO.

Azevedo (2022), por meio de entrevista e pesquisa prática identificou alta de custos na cadeia de suprimentos de produtos orgânicos no que tange o uso de maquinário. Assim, deveria ser modelado um problema que buscasse minimizar as rotas que o maquinário deve fazer para o plantio, colheita e atividades de manutenção, restringidas pelas quantidades de mão de obra (homem x hora), disponibilidade de tratores, suas respectivas manutenções.

O processo de construção do modelo matemático consiste em alguns passos:

I. Definir as variáveis de decisão (quais aspectos estão em competição para chegar a um resultado ótimo);
II. Definir a Função Objetivo (função busca maximizar ou minimizar a relação entre as variáveis);
III. Definir as restrições (quais restrições regem e limitam o problema)

2.1.1. Programação Linear, Inteira e Mista

Programação Inteira se define como uma expressão de otimização de uma função linear sujeita a restrições de variáveis inteiras. Prado (2016) afirma que a Programação Linear (PL) é uma técnica da Ciência da Pesquisa Operacional para a otimização. É uma ferramenta frequentemente usada para encontrar o lucro máximo ou o custo mínimo em situações em que há várias opções de escolha sujeitas a algum tipo de restrição ou regulamentação. Bradley (1992) fala de um tipo misto de programação. Em que as restrições apresentam variáveis tanto inteiras quanto lineares.

2.1.2. Problema do Caixeiro Viajante

O Problema do Caixeiro viajante (ou *Travelling Salesman Problem*, TSP) se refere a um problema clássico de pesquisa operacional como um problema de roteirização. O objetivo é cumprir as demandas estabelecidas em nós em certa malha espacial, encontrando o menor caminho possível, podendo otimizar tempo e distância. Em 1978 Gavish e Graves publicaram um trabalho em nome da universidade MIT (*Massachusetts Institute of Technology)* chamado “Problema do Caixeiro Viajante e Problemas Relacionados” onde delimitam a teoria da solução do problema e sua utilidade para o estudo de otimização.

Bock (1963) referenciava a origem do problema para embasar seu trabalho sobre seu uso em um modelo estocástico para apoio na análise de decisão de cadeia de suprimentos:

> *“Designamos como o Problema do Mensageiro (uma vez que este problema é encontrado por todos os mensageiros postais, bem como por muitos viajantes) a tarefa de encontrar, para um número finito de pontos cujas distâncias pareadas são conhecidas, o caminho mais curto conectando os pontos. Este problema é naturalmente sempre solucionável fazendo um número finito de tentativas. Não são conhecidas as regras que reduziriam o número de tentativas de permutações dos pontos dados. A regra que uma pessoa deve partir é que ir do ponto de partida para o ponto mais próximo, depois para o*

ponto mais próximo disso, etc., não resulta em geral no menor caminho." (Karl Menger, 1930)

Mas Dantzig, Fulkerson e Johnson (1954) formalizaram como é conhecido hoje o TSP em um trabalho intitulado "Solução de um Grande Problema do Caixeiro Viajante" onde delimitaram a menor distância de estrada entre 49 cidades na massa contígua dos Estados Unidos da América em 48 estados e o distrito de Columbia.

Em sua forma mais resumida, o problema TSP busca saber a quantidade de nós em uma malha espacial a ser percorrida, a distância entre nós e a disposição dos nós no espaço. No caso a ser estudado, contudo, foram implantadas demandas específicas em cada nó, restrições de cumprimento da demanda da parte do maquinário e a necessidade de voltar à origem para realizar troca de maquinário, a fim de suprir a demanda em novo nó. Também difere na presença de vários atores "viajantes", representados por tratores que podem realizar diferentes demandas com maquinários específicos em nós distintos.

2.2. AIMMS no apoio à decisão.

No ano 1993 um software de análise prescritiva é lançado com o nome *Advanced Interactive Multidimensional Modeling System,* comumente abreviado como *AIMMS*. O software é uma solução que oferece um ambiente de desenvolvimento avançado, no qual usuários experientes em modelagem matemática podem projetar aplicações prontas para serem facilmente utilizadas por usuários leigos ou usuários finais. Como plataforma de suporte analítico à decisão, o AIMMS proporciona uma combinação única de capacidades, desde métodos matemáticos até ferramentas de desenvolvimento, tais como: um ambiente gráfico para construção e manutenção de aplicações de modelagem avançada; métodos de solução exata; além de procedimentos que permitem a interação com linguagens de programação como C, C++, Fortran, *Python* e ferramentas de interface com banco de dado. O *AIMMS* também permite aos usuários criar e explorar modelos de forma intuitiva, usando o Explorador de Modelos, que oferece recursos gráficos para facilitar o entendimento, visualização e manipulação de modelos, possibilitando que usuários menos experientes possam compreender e interagir com os modelos.

O *AIMMS* possui uma notação de índice intuitiva e poderosa que captura a complexidade dos problemas de forma eficiente. O Explorador de Modelos Gráfico torna os dados visíveis como nós em uma árvore, enquanto sua linguagem de modelagem permite expressar cálculos complexos de maneira compacta, sem necessidade de gerenciamento de memória e armazenamento de dados. Além disso, existe a conectividade com o Excel para importação e exportação de dados (AIMMS, 2020). Ignácio e Ferreira Filho (2004) acrescentam que a notação de índice do *AIMMS* possibilita ao usuário capturar a complexidade de problemas reais por meio de uma linguagem que utiliza o Solver como instrução. Por fim, o *AIMMS* oferece um conjunto de interfaces gráficas que proporcionam ao usuário final uma visualização abrangente do problema. *AIMMS* possui licenças gratuitas com *solvers* limitados e licenças completas pagas (dispondo de solvers CPLEX, BARON, etc). Possui também licenças completas gratuitas para estudantes.

Em trabalho para conclusão de curso de bacharel em Administração de Lorrana Couto (2020) na Universidade de Brasília, discutia a viabilidade de uso de softwares *AIMMS* e LINGO para a solução de problemas de grande porte. No trabalho, Couto (2020) discorre sobre a importância cotidiana da resolução de problemas de otimização para organizações, que por vezes, apresentam grande volume de dados e variáveis.

2.3. Python

Python é uma linguagem de programação computacional. Assim como as línguas humanas, existem muitas linguagens de computador diferentes, como Java, LISP, PHP, C e outras. A maioria das linguagens tem proficiência em pelo menos um aspecto - por exemplo, escrever programas facilmente portáteis é um ponto forte do Java e acessar bancos de dados e dividi-los em páginas da web é a especialidade do PHP. Contudo, todas as linguagens, em seu cerne, são muito semelhantes em termos de conceitos básicos - a maioria usa definição de variáveis e funções (procedimentos, métodos) para operar os dados..

Algumas linguagens até combinam dados e funções em pacotes chamados objetos, e outras como o LISP permitem tratar funções como variáveis e vice-versa. O *Python* é uma linguagem de programação poderosa, elegante e fácil de ler e entender. Ele demonstra a maioria dessas características comuns a muitas outras

linguagens e é útil para aplicações do mundo real. É também um software livre, tem uma implementação padrão e uma grande e amigável comunidade de hackers ao seu redor. Uma vez que você aprende *Python*, todas as outras linguagens que você quiser aprender devem parecer bastante familiares.

2.3.1. Python 3.10.12

A versão 3.10.12 do Python é a versão que é suportada no momento da finalização da pesquisa pela plataforma *Google Colab* (disponibilizada em junho de 2023). É possível acompanhar as mudanças implementadas entre as versões pelo *site* do próprio *Python* (python.org) em que até o momento da pesquisa tem como versão mais recente Python 3.11.4 (também lançado em junho de 2023), que não possui suporte na plataforma usada.

As principais diferenças entre as versões 3.10.12 e 3.11.4 estão em pequenas revisões internas de processamento de dados que traduzem para cálculos microscópicamente mais rápidos e revisões de funções especializadas que não interferem nos cálculos algébricos simples utilizados pela biblioteca. Assim sendo, usar o mesmo código com uma versão mais atualizada do *Python*, não deve haver quebra nos cálculos. Contudo, recomenda-se sempre a revisão cuidadosa do programa antes que se recalcule assim e o uso da documentação como referência para eventuais ajustes.

Portanto, apesar de ser possível a atualização da versão, para o uso no caso, não se faz necessária, nem mais benéfica que use uma versão mais atualizada do que a versão padrão, 3.10.12.

2.3.2. Pyomo

Pyomo é uma biblioteca -também chamado de pacote- de funções integradas ao *Python* que permite simplificar passos do código. Pyomo é especificamente preocupada em Programação Matemática e traz em si *solvers* integrados. Pyomo não permite o uso do CPLEX em sua versão completa - apenas uma versão com limite de variáveis -, mas permite uso do solver GLPK, CBC e HiGHS que foi usado para o cálculo. Graças à biblioteca, os passos do solver GLPK não precisam ser programados individualmente, pode ser chamado usando comandos específicos do pacote, resultando em um código mais limpo e conciso.

Pyomo é compatível com a versão 3.10.12 do *Python*, e assim é possível ser usada na plataforma *Google Colab*.

2.3.3. Pandas

Pandas é uma outra biblioteca *Python* usada para análise e manipulação de dados. No trabalho ela é usada para exportar informações de planilhas em formato de dados separados por vírgula (*comma-separated values),* comumente e doravante mencionado como formato *csv* e deixar disponível para cálculos por meio do Pyomo. É especialmente útil para que o *input* de dados seja mais acessível, não necessitando *inputs* de código. Apenas a manutenção das planilhas e arquivos em formato *csv.*

2.4. Google Colab

O *Google Colaboratory*, conhecido como *Google Colab*, é uma ferramenta usada para escrever e executar códigos de programação na linguagem *Python* e permite seu compartilhamento por meio de tecnologia em nuvem. Ferramentas assim, são conhecidas como *notebooks,* ou cadernos, e traz como outro exemplo a ferramenta Jupyter que tem como maior diferença uma dificuldade de configuração e integração com o ecossistema *Google*.

Por sua natureza e manutenção realizada por uma empresa única, a integração do *Google Colab* com o armazenamento em nuvem *Google (Google Drive)* é mais rápida e simples, gerando uma pasta com o arquivo do caderno Colab, permitindo usar (no caso do trabalho) pacote Pandas para ler arquivos diretamente de pastas de nuvem. No caso do Jupyter, o padrão é que a informação (tanto o caderno, quanto os dados a serem analisados) fiquem armazenados no disco rígido (*hard drive)* do usuário. Dessa maneira, pode se dizer que o *Google Colab* é mais portátil e compartilhável.

3. MÉTODOS E TÉCNICAS DE PESQUISA

3.1. Tipologia e descrição geral dos métodos de pesquisa

A pesquisa é classificada, de acordo com proposto por Silva e Menezes (2005), como aplicada, exploratória e quantitativa. Em sua natureza, é aplicada pois busca informações com aplicação prática com o fim de solucionar um problema. Em seus objetivos é exploratório, pois busca clarificar o problema e torná-lo explícito. Em sua forma é quantitativa, pois compara numericamente resultados matemáticos, analisando o menor valor.

Os procedimentos técnicos, de acordo com Silva e Menezes (2005), se classificam como uma pesquisa experimental, uma vez que um objeto de estudo foi selecionado junto com as variáveis de influência e a forma de controle. A coleta de dados e variáveis foi feita com leitura de um trabalho específico: **CADEIA DE SUPRIMENTOS DE ALIMENTOS ORGÂNICOS: Uma abordagem matemática para o planejamento das rotas do maquinário** de Pedro Henrique Ferreira Azevedo (2022).

O experimento foi realizado de forma comparativa com o modelo gerado por Azevedo (2022) utilizando a plataforma *AIMMS.* As fórmulas geradas para compor o modelo foram traduzidas e adaptadas para cumprir o mesmo papel de acordo com a linguagem de programação *Python* buscando manter iguais as entradas que podem ser mantidas iguais e adaptando as que necessitam de um modo de leitura diferente entre os dois métodos.

Para essas correspondências e adaptações foram usadas as documentações (sendo “documentação” um material que fornece informações que servem como registro) da linguagem *Python* e do pacote Pyomo para entender a sintaxe de escrita dos termos e funções apropriadas. Enquanto a documentação Python guia o desenvolvimento em si, como as entradas de dados, as operações; a documentação Pyomo guia as declarações pertinentes à Pesquisa Operacional.

3.2. Caracterização da organização objeto do estudo

A pesquisa busca comparar dois modelos de apoio à decisão para o planejamento de um processo de cultivo de alimentos orgânicos. O estudo de caso será realizado comparando o trabalho escrito por Azevedo (2022) caracterizando o modelo pela ferramenta de cálculo de Pesquisa Operacional *AIMMS*.

O modelo foi definido corretamente e caracteriza em suas restrições um caminho ideal para roteamento de tratores pelos lotes de um determinado terreno. O modelo utiliza o método do Problema do Caixeiro Viajante que é um problema de roteamento conhecido e utilizado largamente dentro da Pesquisa Operacional.

3.3. População e amostra ou Participantes da pesquisa

Este trabalho foi construído analisando o modelo fornecido por Pedro Azevedo que o desenvolveu. O modelo e o trabalho escrito serviram de base para o desenvolvimento da escrita em Python assim como a planilha Excel utilizada para a entrada de dados no modelo AIMMS. Com o auxílio da planilha, foi possível entender as diferenças nas metodologias de entrada de dados e adaptar adequadamente.

3.4. Caracterização e descrição dos instrumentos de pesquisa

Os procedimentos técnicos se dão da seguinte forma:

a) **Revisão narrativa da literatura** nas bibliografias de referência da área, para buscar uma melhor compreensão da área de pesquisa e apoiar o desenvolvimento do modelo em outra linguagem.

b) **Estudo de caso**, para comparar o modelo criado por Azevedo na ferramenta de apoio à decisão *AIMMS* com o código *Python* desenvolvido por meio da plataforma *Google Colaboratory (Colab*).

3.5. Procedimentos de coleta e de análise de dados

Foi estudado o modelo desenvolvido por Azevedo (2022) na ferramenta *AIMMS* com o apoio do trabalho escrito a fim de compreender profundamente as entradas de dados via *Microsoft Excel* e declarações de conjuntos, índices, parâmetros, restrições e função objetivo realizados diretamente no *AIMMS Developer* em sua versão 4.84.3.4 64- bit, em conjunto com o solver IBM CPLEX 20.1. Softwares pagos que possuem licença acadêmica gratuita.

A escrita do modelo na linguagem *Python* (versão 3.8.10) por meio da plataforma gratuita *Google Colaboratory (Google Colab)* com o uso do pacote especializado de Pesquisa Operacional chamado Pyomo e os solvers integrados GLPK, CBC (ambos gratuitos), um solver externo HiGHS (gratuito).

4. RESULTADOS E DISCUSSÃO

4.1. Entrada de dados AIMMS e Python

Por serem métodos distintos de modelagem, a entrada de dados deve ser específica para cada um. No AIMMS é possível utilizar um arquivo Microsoft Excel para alimentar todas as declarações do problema. Contudo, a leitura dos dados no Python é mais efetiva se feita em uma matriz de dois eixos.

Utilizando uma entrada de dados exemplo será mostrado a seguir na Tabela 1 as diferenças entre métodos assim como a escrita utilizada em Python (Tabela 2) para as demandas do problema. Os conjuntos dos índices (Tabela 3) e suas correspondência (Tabela 4).

Tabela 1 - Dados de entrada demanda no AIMMS (Azevedo 2022)

LOTES	TAREFAS	DEMANDA
1	Limpar	1
1	Subsolar	1
1	Incorporar	1
2	Levantar Canteiro	1
3	Limpar	1
4	Subsolar	1
5	Incorporar	1
6	Pulverizacao	1
7	Limpar	1
7	Subsolar	1
7	Incorporar	1
8	Levantar Canteiro	1

8	Adubar	1
8	Incorporar	1

Significa que em um lote **i** (coluna Lotes) há uma tarefa (também chamada de “atividade”) **a** (coluna Tarefas). Demanda 1 é binário para indicar que a tabela existe e deve ser cumprida.

Tabela 2 - Dados de entrada demanda no Python (autoria própria)

	Limpar	Subsolar	Incorporar	Levantar Canteiro	Pulverização	Adubar
Depósito	0	0	0	0	0	0
1	1	1	1	0	0	0
2	0	0	0	1	0	0
3	1	1	1	0	0	0
4	0	1	0	0	0	0
5	0	0	1	0	0	0
6	0	0	0	0	1	0
7	1	0	1	0	0	0
8	1	0	0	1	0	1

Significa que em um lote **l** (de 1 a 8, excluindo depósito) há ou não a demanda pela atividade **a**. O código para leitura dessa tabela se dá da seguinte forma

```
pathdem = "/content/drive/MyDrive/Colab Notebooks/Dados - Demandas.csv"

df = pd.read_csv(pathdem)

df.describe

Demanda_la = np.genfromtxt(pathdem, delimiter =",")




tabdemand = pd.DataFrame(np.array(Demanda_la,list), columns = [Atividades],
index = [Lotes])

print(tabdemand)
```

Assim é possível ler o arquivo .csv disponibilizado na pasta onde é executado o caderno Google Colab em nuvem e retorna:

Imagem 1 - Retorno do código de entrada de demanda

```
• Demandas em um Lote (l) de uma Atividade (a)
   Limpar Subsolar Levantar Canteiro Adubar Incorporar Pulverização
1    1.0      1.0              1.0    0.0        0.0           0.0
2    0.0      0.0              0.0    1.0        0.0           0.0
3    1.0      1.0              1.0    0.0        0.0           0.0
4    0.0      1.0              0.0    0.0        0.0           0.0
5    0.0      0.0              1.0    0.0        0.0           0.0
6    0.0      0.0              0.0    0.0        1.0           0.0
7    1.0      0.0              1.0    0.0        0.0           0.0
8    1.0      0.0              0.0    1.0        0.0           1.0
```

Tabela 3 - Entrada dos conjuntos dos índices para o AIMMS (Azevedo 2022)

NÓS	ATIVIDADES	DURAÇÃO	TRATORES	CUSTO	MAQUINAS
Deposito	Limpar	20	TT02	10	Triton

1	Subsolar	20	TT03	15	Subsolador
2	Levantar Canteiro	20	TT07	20	Rotoencanteirador
3	Adubar	20	TT08	30	Adubadeira
4	Incorporar	20			Carreta Bio
5	Pulverizacao	20			
6					
7					
8					

Tabela 4 - Entrada dos conjuntos dos índices para python (autoria própria)

Conjunto	**Código**

Nós	`Nós = ["Deposito", "1", "2", "3", "4", "5", "6", "7", "8"]`
	
	`Lotes = Nós[1:9]`
	`l = len(Lotes)`
	
	`Deposito = Nós[0:1]`
	`o = len(Deposito)`

Atividades	`Atividades = ["Limpar", "Subsolar", "Levantar Canteiro", "Adubar", "Incorporar", "Pulverização"]` `a = len(Atividades)`
Duração	`duracao_a = [20, 20, 20, 20, 20, 20]`

Tratores	`Tratores = ["TT02", "TT03", "TT07", "TT08"]` `t = len(Tratores)`
Custo	`CustoLigar_t = [10, 15, 20, 30]`
Máquinas	`Máquinas = ["Triton", "Subsolador", "Rotoencanteirador", "Adubadeira", "Carreta Bio"]` `m = len(Máquinas)`

A matriz de distância (tabelas 5 e 6) e a matriz que relaciona a atividade com sua respectiva máquina (tabela 7 e 8) foram inseridas em formato .csv por meio do pacote Pandas. Contudo, enquanto o AIMMS permite a operação em matrizes para declarar um novo conjunto, o Pyomo apresenta dificuldades no cálculo se feito dessa maneira. Assim, a matriz que representa o tempo entre um nó e outro foi inserida também via Pandas (tabela 9), enquanto no AIMMS foi declarada como `DISTANCIA/4` para indicar que para cada minuto andaria 4 metros da matriz de distância.

Tabela 5 - Entrada das distâncias entre os nós (Azevedo 2022)

	Deposito	1	2	3	4	5	6	7	8
Deposito	X	408	416	424	432	440	448	456	464
1	408	X	8	16	24	32	40	48	56
2	416	8	X	8	16	24	32	40	48
3	424	16	8	X	8	16	24	32	40

4	432	24	16	8	X	8	16	24	32
5	440	32	24	16	8	X	8	16	24
6	448	40	32	24	16	8	X	8	16
7	456	48	40	32	24	16	8	X	8
8	464	56	48	40	32	24	16	8	X

Tabela 6 - Entrada das distâncias entre os nós em Python (Autoria própria)

Depósito	1	2	3	4	5	6	7	8

Depósito	0	408	416	424	432	440	448	456	464
1	408	0	8	16	24	32	40	48	56
2	416	8	0	8	16	24	32	40	48
3	424	16	8	0	8	16	24	32	40
4	432	24	16	8	0	8	16	24	32
5	440	32	24	16	8	0	8	16	24
6	448	40	32	24	16	8	0	8	16
7	456	48	40	32	24	16	8	0	8
8	464	56	48	40	32	24	16	8	0

Significa que entre um lote **i** e **j** (de 0 a 8) há a distância **i,j** em metros.

```
pathdist ="/content/drive/MyDrive/Colab Notebooks/Dados - Distancia.csv"

df = pd.read_csv(pathdist)

df.describe

cost_matrix = np.genfromtxt(pathdist, delimiter =",")



tabcost = pd.DataFrame(np.array(cost_matrix,list), columns =
[np.array(Nós,list)], index=[np.array(Nós,list)])

print(tabcost)
```

Assim é possível ler o arquivo .csv disponibilizado na pasta onde é executado o caderno Google Colab em nuvem e retorna:

Imagem 2 - Retorno do código de entrada de distância

```
• Distância entre os nós:
             Deposito           1           2           3           4           5           6           7           8
Deposito  1000000000.0       408.0       416.0       424.0       432.0       440.0       448.0       456.0       464.0
1             408.0  1000000000.0         8.0        16.0        24.0        32.0        40.0        48.0        56.0
2             416.0         8.0  1000000000.0         8.0        16.0        24.0        32.0        40.0        48.0
3             424.0        16.0         8.0  1000000000.0         8.0        16.0        24.0        32.0        40.0
4             432.0        24.0        16.0         8.0  1000000000.0         8.0        16.0        24.0        32.0
5             440.0        32.0        24.0        16.0         8.0  1000000000.0         8.0        16.0        24.0
6             448.0        40.0        32.0        24.0        16.0         8.0  1000000000.0         8.0        16.0
7             456.0        48.0        40.0        32.0        24.0        16.0         8.0  1000000000.0         8.0
8             464.0        56.0        48.0        40.0        32.0        24.0        16.0         8.0  1000000000.0
```

Tabela 7 - Entrada das atividades relacionadas com cada uma das máquinas no AIMMS (Azevedo 2022)

	Triton	Subsolador	Rotoencanteirador	Adubadeira	Carreta Bio
Limpar	1				
Subsolar		1			
Levantar Canteiro			1		
Adubar				1	

Incorporar			1		
Pulverizacao					1

Tabela 8 - Entrada das atividades relacionadas com cada uma das máquinas no Python (autoria própria)

	Triton	Subsolador	Rotoencanteirador	Adubadeira	Carreta Bio
Limpar	1	0	0	0	0
Subsolar	0	1	0	0	0
Levantar Canteiro	0	0	1	0	0
Adubar	0	0	0	1	0
Incorporar	0	0	1	0	0
Pulverizacao	0	0	0	0	1

```
pathatv = "/content/drive/MyDrive/Colab Notebooks/Dados - AtvMaq.csv"

df = pd.read_csv(pathatv)

df.describe

atvmaq = np.genfromtxt(pathatv, delimiter =",")
```

```
tabatvmaq = pd.DataFrame(np.array(atvmaq,list), columns = [Máquinas],
index=[Atividades])

print(tabatvmaq)
```

Assim é possível ler o arquivo .csv disponibilizado na pasta onde é executado o caderno Google Colab em nuvem e retorna:

Imagem 3 - Retorno do código de entrada de Atividade de Maquinário

```
• Atividade (a) de Maquinário (m):
                Triton Subsolador Rotoencanteirador Adubadeira Carreta Bio
Limpar             1.0        0.0                0.0        0.0        0.0
Subsolar           0.0        1.0                0.0        0.0        0.0
Levantar Canteiro  0.0        0.0                1.0        0.0        0.0
Adubar             0.0        0.0                0.0        1.0        0.0
Incorporar         0.0        0.0                1.0        0.0        0.0
Pulverização       0.0        0.0                0.0        0.0        1.0
```

Tabela 9 - Entrada de distâncias temporais entre nós (autoria própria)

Depósito	1	2	3	4	5	6	7	8

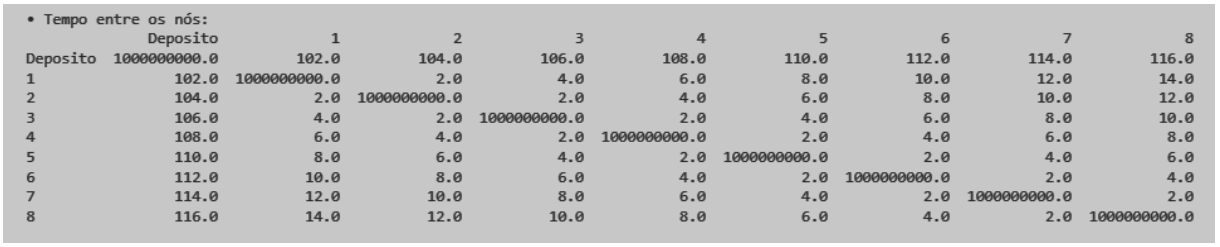
Depósito	0	102	104	106	108	110	112	114	116
1	102	0	2	4	6	8	10	12	14
2	104	2	0	2	4	6	8	10	12
3	106	4	2	0	2	4	6	8	10
4	108	6	4	2	0	2	4	6	8
5	110	8	6	4	2	0	2	4	6

Depósito	1	2	3	4	5	6	7	8
6	112	10	8	6	4	2	0	2	4
7	114	12	10	8	6	4	2	0	2
8	116	14	12	10	8	6	4	2	0

```
pathdisttemp ="/content/drive/MyDrive/Colab Notebooks/Dados -
DistanciaTempo.csv"

df = pd.read_csv(pathdisttemp)

df.describe

disttemp = np.genfromtxt(pathdisttemp, delimiter =",")



tabdist = pd.DataFrame(np.array(disttemp,list), columns =
[np.array(Nós,list)], index=[np.array(Nós,list)])

print(tabdist)
```

Assim é possível ler o arquivo .csv disponibilizado na pasta onde é executado o caderno Google Colab em nuvem e retorna:

Imagem 4 - Relação de distância com o tempo.



• Tempo entre os nós:

	Deposito	1	2	3	4	5	6	7	8
Deposito	1000000000.0	102.0	104.0	106.0	108.0	110.0	112.0	114.0	116.0
1	102.0	1000000000.0	2.0	4.0	6.0	8.0	10.0	12.0	14.0
2	104.0	2.0	1000000000.0	2.0	4.0	6.0	8.0	10.0	12.0
3	106.0	4.0	2.0	1000000000.0	2.0	4.0	6.0	8.0	10.0
4	108.0	6.0	4.0	2.0	1000000000.0	2.0	4.0	6.0	8.0
5	110.0	8.0	6.0	4.0	2.0	1000000000.0	2.0	4.0	6.0
6	112.0	10.0	8.0	6.0	4.0	2.0	1000000000.0	2.0	4.0
7	114.0	12.0	10.0	8.0	6.0	4.0	2.0	1000000000.0	2.0
8	116.0	14.0	12.0	10.0	8.0	6.0	4.0	2.0	1000000000.0

4.2. Modelagem Matemática no AIMMS

Azevedo (2022) desenvolveu o modelo usando AIMMS Developer fazendo as declarações seguindo os passos da modelagem de problema como descrito anteriormente. Declarou as variáveis de decisão e seus índices, a Função Objetivo e as restrições às quais o problema fica sujeito.

Segue a Tabela 10 com as variáveis usadas e sua descrição de acordo com descrito por Azevedo (2022).

Tabela 10 - Variáveis de cálculo do problema (Azevedo 2022)

Variável	Descrição
$Inclui_{ptij}$	Variável binária que indica se no período p, um trator t percorreu o trecho do nó i ao j
$Serviu_{ptla}$	Variável binária que indica se no período p, um trator t atendeu à demanda no lote l da atividade a
$LigaTrator_{ptj}$	Variável binária que indica se no período p, o trator t será utilizado
$Conectado_{ptm}$	Variável binária que indica se no período p, o trator t será utilizado
$Demanda_{ia}$	Parâmetro binário que indica se em um lote i existe demanda para uma atividade **a**
$AtividadeMaq_{am}$	Parâmetro binário que indica se, para atender à demanda de uma atividade a, a máquina m pode ser utilizada
$Duração_{a}$	Parâmetro numérico não negativo que indica, em minutos, o tempo que cada tipo de atividade **a** leva para ser completada.
$Distância_{ij}$	Parâmetro numérico não negativo que indica a distância, em metros, entre dois nós i e j

$DistanciaTempo_{ij}$	Parâmetro numérico não negativo que indica, em minutos, o tempo necessário para percorrer a distância entre dois nós i e j
$Turno$	Parâmetro numérico não negativo que indica, em minutos, a duração do turno de trabalho dos tratoristas
big_M	Parâmetro numérico não negativo utilizado nas restrições quando há a necessidade de utilizar um valor grande
$CustoLigar_t$	Parâmetro numérico não negativo que indica o custo para ligar cada um dos tratores

Em que os índices se explicam como segue a Tabela 11:

Tabela 11 - Descrição dos índices do problema. (Azevedo 2022)

Índices	**Descrição**
t	Conjunto de tratores disponíveis
m	Conjunto das máquinas disponíveis
a	Conjunto das diversas atividades
p	Conjunto dos períodos
i, j	Conjunto dos nós pelos quais passam os tratores
o	Subconjunto de nós que contém apenas o depósito das máquinas
l	Subconjunto dos nós que contém todos os nós menos o depósito

De acordo com a modelagem do problema, Azevedo (2022) delimitou a Função Objetivo como a expressão que resolve o problema como se mostra a seguir:

$$min\left(\sum_{p}\sum_{t}\sum_{i}\sum_{j}(Inclui_{ptij} * Distância_{ij}) + \sum_{p}\sum_{t}(ligaTrator_{pt} * CustoLigar_{t})\right)$$

Sujeito às seguintes restrições com suas devidas descrições (Tabela 12):

Tabela 12 - Restrições da Função Objetivo (Azevedo, 2022)

Restrição	Equação	Descrição
$RestEquilibrio_{pti}$	$\sum_{j}(Inclui_{ptij}) = \sum_{j}(Inclui_{ptij}) \ \forall \ p,t,i$	Certifica que, no período **p**, toda vez que um trator **t** entra em um nó **i**, também irá sair dele. A soma de todos os trechos que entram em um nó é igual à soma de todos os trechos que saem do nó, para todo período, trator e nó.
$RestIncluiDemanda_{l}$	$\sum_{p}\sum_{t}\sum_{i}(Inclui_{ptil}) = \sum_{i}(Demanda_{la}) \ \forall \ l$	Certifica que, para todos os lotes **l**, a soma dos trechos que entram no lote, considerando todos os períodos **p**, todos os tratores **t** e todas os nós de início **i**, seja igual à soma da demanda de todas as atividades daquele lote.
$RestServiuDemanda_{l}$	$\sum_{p}\sum_{t}(Serviu_{ptla}) = Demanda_{la} \quad \forall \ l,a$	Certifica que, para um dado lote **l** e atividade **a**, a soma de todas as vezes que o lote foi servido, considerando

		todos os períodos **p** e todos os tratores **t**, seja igual à demanda do lote por aquela atividade. Ou seja, se a demanda é 1, em algum período, algum trator irá atender àquela demanda.
$RestNoFinal_{tp}$	$\sum_{l}\sum_{o}(Inclui_{ptlo}) = LigaTrator_{pt} \quad \forall\ p,t$	Certifica que, caso um trator **t** seja ligado em um período **p**, algum trecho que tenha como ponto de destino a origem **o**, que representa o Depósito, seja incluído para aquele trator no período em questão. A partir desta restrição, a restrição de equilíbrio se encarrega de garantir que exista também um trecho que saia do Depósito
$RestTurno_{pt}$	$\sum_{l}\sum_{a}(Serviu_{ptla} * Duração_{a}) +$ $\sum_{i}\sum_{j}(Inclui_{ptij} * DistânciaTempo_{ij}) \leq turno \forall\ p,t$	Certifica que, a cada período **p**, **a** soma do tempo de todas as tarefas realizadas por um trator **t** adicionada à soma do tempo de deslocamento entre os lotes das atividades realizadas seja menor ou igual a duração do turno.
$RestLink_{mltp}$	$\sum_{i} Inclui_{ptil} \geq Serviu_{ptla} \quad \forall\ p,t,l,a$	Restrição que conecta as variáveis “Inclui” e “Serviu”.

$RestLigaTrator_{pt}$	$\sum_{l}\sum_{a}(Serviu_{ptla}) \leq big_{M^*} LigaTrator_{pt} \quad \forall\ p,t$	Esta é a restrição responsável por manipular a variável LigaTrator. Caso, no período **p**, o trator **t** tenha servido a alguma demanda, o valor da variável LigaTrator deverá ser 1, ou então, a variável só terá valor 1 se for necessário utilizar aquele trator.
$RestMaqPorLote_{lp}$	$\sum_{t}\sum_{a}(Serviu_{ptla}) \leq 1 \quad \forall\ p,t$	Certifica que, no período **p**, o lote **l** seja servido por apenas um trator **t**. Portanto, fica implícito, a partir dessa restrição, que não haverá mais de uma atividade **a** sendo atendida em um mesmo período **p** num lote **l**
$RestMaq_{pm}$	$\sum_{t}(Conectado_{ptm}) \leq 1 \quad \forall\ p,m$	Certifica que, no período **p**, a máquina **m** esteja conectada a apenas um trator **t**.
$RestTrator_{pt}$	$\sum_{m}(Conectado_{ptm}) \leq 1 \quad \forall\ p,t$	Certifica que, no período **p**, o trator **t** esteja conectada a apenas uma máquina **m**.
$RestMaqConectada_{ptm}$	$\sum_{m}(Conectado_{ptm}) \leq 1 \quad \forall\ p,t,\mathrm{m}$	Certifica que, no período **p**, para que o trator **t** atenda à demanda da atividade **a** em um lote **l**, esse mesmo trator esteja conectado à máquina **m** necessária para o

		desenvolvimento da atividade. Caso o trator não esteja conectado à máquina, a demanda não será atendida.
$RestSEC_{pt}$	$\sum_i \sum_j (Inclui_{ptij} * Inclui_{ptji}) \leq 2 \quad \forall\ p, t, m$ $sendo\ que\ i \subset subtour\ e\ j \subset fora\ do\ subtour$	Certifica que, no período **p**, para a rota realizada pelo trator **t**, caso haja um **subtour**, ele possua um trecho com origem em um nó fora do subtour e possua um trecho com destino à um nó também fora do subtour.

Essas foram as declarações do modelo usadas para o desenvolvimento sob duas premissas:

a) Ao início de cada rota em cada período, os tratores que forem acionados partem do mesmo ponto, o depósito das máquinas. Posteriormente, no mesmo período, o trator deve retornar para o ponto de origem.

b) Mesmo que exista a demanda por mais de uma atividade, um mesmo lote só é atendido uma vez por período. A soma do tempo necessário para finalizar as atividades desenvolvidas com a ajuda de um trator não pode ultrapassar a duração máxima de um turno

4.3. Modelagem em Python utilizando Pyomo

4.3.1. Inicialização em Python

Os primeiros passos da modelagem em Python são as instalações e importações necessárias para o modelo. Primeiramente a instalação do pacote Pyomo com a abreviação *pyo*; a instalação dos solvers GLPK e CBC; a instalação do pacote *numpy* que ajudará a manipular números; a montagem do espaço Google

Drive para acessar os dados de planilhas e arquivos; importação do pacote Pandas para manipular estes dados em tabelas.

```
#Pyomo
!pip install -q pyomo
from pyomo.environ import *
import pyomo.environ as pyo


#Solvers
!apt-get install -y -qq glpk-utils #GLPK
!apt-get install -y -qq coinor-cbc #CBC
!pip install highspy #HiGHS
import highspy
h = highspy.Highs()


#Operadores de Número
import numpy as np
from numpy.lib.shape_base import array_split


import pandas as pd
pd.set_option('display.max_rows', 1000)
pd.set_option('display.max_columns', 1000)
pd.set_option('display.width', 1000)


#Google Drive
from re import A
from google.colab import drive
drive.mount('/content/drive')
```

4.3.2. Índices

Tabela 13 - Índices em Python (autoria própria)

Índices do modelo	Índices para Pyomo
t	`model.t`
m	`model.m`
a	`model.a`
p	`model.p`
i, j	`model.i model.j`
o	`model.o`
l	`model.l`

A modelagem em linguagem de programação Python seguiu à risca a mesma lógica utilizada no AIMMS, inclusive utilizando os mesmos termos para os índices das variáveis, priorizando a tradução *ipsis litteris* com apenas as alterações necessárias, quando as fossem exigidas.

4.3.3. Variáveis de decisão

Para fazer a declaração de variável utilizando o pacote Pyomo deve ser indicado da seguinte maneira

```
model.[Nome da Variável]=pyo.Var(model.[índice 1],model.[índice 2], within=pyo.[Domínio]
```

Exemplo: Variável inteira não negativa chamada X_{ab}

```
model.X=pyo.Var(model.a, model.b, within=NonNegativeIntegers)
```

Na tabela 14 podemos ver como foram definidas as variáveis de decisão do problema utilizando python.

Tabela 14 - Comparação da declaração da variável no AIMMS e Python

Variável no modelo	Variável em Python
$Inclui_{ptij}$	`model.Inclui=pyo.Var(model.p,model.t,model.i,model.j, within=pyo.Binary)`
$Serviu_{ptla}$	`model.Serviu=pyo.Var(model.p,model.t,model.l,model.a, within=pyo.Binary)`
$LigaTrator_{ptj}$	`model.LigarTrator=pyo.Var(model.p,model.t, within=pyo.Binary)`
$Conectado_{ptm}$	`model.Conectado=pyo.Var(model.p,model.t,model.m, within=pyo.Binary)`

4.3.4. Parâmetros

Para fazer a declaração de parâmetros a serem usados utilizando o pacote Pyomo deve ser indicado da seguinte maneira

```
model.[Nome do Parâmetro]=pyo.Param(model.[índice 1],model.[índice 2], initialize=lambda model.[índice 1],[índice 2]: {Nome da Matriz}[[índice.1]-1][[índice.2]-1]
```

Exemplo: Parâmetro binário X_{ab} na matriz "matriz_de_custo"

```
model.X=pyo.Param(model.a, model.b, initialize=lamba model.a,b: matriz_de_custo[a-1][b-1]
```

Os índices na função lambda são subtraídos de 1, pois a contagem da entrada vai de 0 a x, contendo x índices. Caso tente inicializar a função com índice de valor x, não considerará que o início da contagem é 0 e retornará fora dos limites, pois x é na realidade o x-ésimo+1 termo. Dessa forma se subtrai 1 do índice para inicializar como 0 a x em vez de 1 a x+1.

Na tabela 15 podemos ver como foram definidos os parâmetros do problema utilizando python.

Tabela 15 - Comparação da declaração dos parâmetros no AIMMS e Python

Parâmetro no modelo	Parâmetro em Python
$Demanda_{ia}$	`model.Demanda = pyo.Param(model.i, model.a, initialize = lambda model, i, a: Demanda_la[i-1][a-1])`

$AtividadeMaq_{am}$	`model.AtvMaq = pyo.Param(model.a, model.m, initialize = lambda model, a, m: atvmaq[a-1][m-1])`
$Duração_{a}$	`model.Duracao = pyo.Param(model.a, initialize = lambda model, a: duracao_a[a-1])`
$Distância_{ij}$	`model.Distancia = pyo.Param(model.i, model.j,initialize=lambda model, i, j: cost_matrix[i-1][j-1])`
$DistanciaTempo_{ij}$	`model.DistanciaTempo = pyo.Param(model.i, model.j,initialize=lambda model, i, j: disttemp[i-1][j-1])`
$Turno$	`model.turno = pyo.Param(initialize=480)`
big_{M}	`model.bigM = pyo.Param(initialize=10**9)`
$CustoLigar_{t}$	`model.CustoLigar = pyo.Param(model.t, initialize = lambda model, t: CustoLigar_t[t-1])`

4.3.5. Função Objetivo

A Função Objetivo foi definida utilizando a função objetivo do problema original. Minimizar a função em que a soma de todos os índices p, t, i, j para a multiplicação da variável Inclui com o parâmetro Distância somado à soma de todos os índices p, t da multiplicação entre a variável LigarTrator com o parâmetro CustoLigar que define o valor a ser pago para acionar o trator.

$$min\left(\sum_{p}\sum_{t}\sum_{i}\sum_{j}(Inclui_{ptij} * Distância_{ij}) + \sum_{p}\sum_{t}(ligaTrator_{pt} * CustoLigar_{t})\right)$$

```
def func_obj(model):

  return sum(model.Inclui[p,t,i,j]*model.Distancia[i,j] for p in
model.p for t in model.t for i in model.i for j in model.j) +
sum(model.LigarTrator[p,t]*model.CustoLigar[t] for p in model.p
for t in model.t)

model.objective = pyo.Objective(rule=func_obj,sense=pyo.minimize)
```

4.3.6. Restrições

Para fazer a declaração de restrições a serem usados utilizando o pacote Pyomo deve ser indicado da seguinte maneira:

```
def rule_[Nome do Restrição](model,[Índice 1],[Índice 2]):

    return [operação] model.[Nome de uma Variável][[Índice 1],[Índice 2]]
for [Índice 1] in model.[Índice 1] [sinal de relação (igualdade,
desigualdade)] [operação] model.[Nome de uma Variável][[Índice 1],[Índice
2]] for [Índice 1] in model.[Índice 1]



model.[Nome do Restrição] = pyo.Constraint(model.[Índice 1],model.[Índice
2], rule = rule_[Nome do Restrição]
```

Exemplo: Restrição r1: $\sum_{a} X_{ab} * 2 \leq 5 \, \forall \, b$

```
def rule_r1(model,b) #para todo índice b

    return sum(model.X[a,b] for a in model.a)*2 <= 5 #soma de todos os
índices a



model.r1 = pyo.Constraint(model.b, rule=rule_r1)  #para todo índice b
```

Dessa forma, as variáveis definidas em Python com o uso do pacote Pyomo foram:

Tabela 16 - Comparação da declaração dos restrição no AIMMS e Python

Restrição	Equação	Python

$RestEquilibrio_{pti}$	$\sum_{j}(Inclui_{ptji}) = \sum_{j}(Inclui_{ptij}) \ \forall \ p,t,i$	`def rule_r1(model,p,t,i): #RestEquilibrio_pti`
		`return sum(model.Inclui[p,t,j,i] for j in model.j) == sum(model.Inclui[p,t,i,j] for j in model.j)`
		
		`model.r1 = pyo.Constraint(model.p, model.t, model.i,rule=rule_r1)`

$RestIncluiDemanda_{l}$	$\sum_{p}\sum_{t}\sum_{i}(Inclui_{ptil}) = \sum_{i}(Demanda_{la}) \ \forall \ l$	`def rule_r2(model,l): #RestIncluiDemanda`
		`return sum(model.Inclui[p,t,i,l] for p in model.p for t in model.t for i in model.i) == sum(model.Demanda[l,a] for a in model.a)`
		
		`model.r2 = pyo.Constraint(model.l,rule=rule_r2)`

$RestServiuDemanda_{la}$	$\sum_{p}\sum_{t}(Serviu_{ptla}) = Demanda_{la} \quad \forall\ l, a$	def rule_r3(model,l,a): #RestServiuDemanda return sum(model.Serviu[p,t,l,a] for p in model.p for t in model.t) == model.Demanda[l,a] model.r3 = pyo.Constraint(model.l, model.a,rule=rule_r3)
$RestNoFinal_{tp}$	$\sum_{l}\sum_{o}(Inclui_{ptlo}) = LigaTrator_{pt} \quad \forall\ p, t$	def rule_r4(model,p,t): #RestNoFinal return sum(model.Inclui[p,t,l,o] for l in model.l for o in model.o) == model.LigarTrator[p,t] model.r4 = pyo.Constraint(model.p, model.t, rule=rule_r4)
$RestTurno_{pt}$	$\sum_{l}\sum_{a}(Serviu_{ptla} * Duração_{a}) +$ $\sum_{i}\sum_{j}(Inclui_{ptij} * DistânciaTempo_{ij}) \leq turno \forall\ p, t$	def rule_r5(model,p,t): #RestTurno return sum(model.Serviu[p,t,l,a]*model.Duracao[a] for l in model.l for a in model.a) + sum(model.Inclui[p,t,i,j] * model.DistanciaTempo[i,j

		] for i in model.i for j in model.j) <= 480 model.r5 = pyo.Constraint(model.p, model.t,rule=rule_r5)
$RestLink_{mltp}$	$\sum_{i} Inclui_{ptil} \geq Serviu_{ptla} \quad \forall\ p,t,l,a$	def rule_r6(model,p,t,l,a): #RestLink return sum(model.Inclui[p,t,i,l] for i in model.i) >= model.Serviu[p,t,l,a] model.r6 = pyo.Constraint(model.p, model.t, model.l, model.a, rule=rule_r6)
$RestLigaTrator_{pt}$	$\sum_{l}\sum_{a}(Serviu_{ptla}) \leq big_{M^*}LigaTrator_{pt} \quad \forall\ p,t$	def rule_r7(model,p,t): #RestLigaTrator return sum(model.Serviu[p,t,l,a] for l in model.l for a in model.a) <= (10**9)*model.LigarTrator[p,t] model.r7 = pyo.Constraint(model.p, model.t,rule=rule_r7)
$RestMaqPorLote_{lp}$	$\sum_{t}\sum_{a}(Serviu_{ptla}) \leq 1 \quad \forall\ p,t$	def rule_r8(model,l,p): #RestMaqPorLote

		return sum(model.Serviu[p,t,l,a] for t in model.t for a in model.a) <= 1 model.r8 = pyo.Constraint(model.l, model.p, rule=rule_r8)
$RestMaq_{pm}$	$\sum_{t}(Conectado_{ptm}) \leq 1 \quad \forall\ p,m$	def rule_r9(model,p,m): #RestMaq return sum(model.Conectado[p,t,m] for t in model.t) <= 1 model.r9 = pyo.Constraint(model.p, model.m,rule=rule_r9)
$RestTrator_{pt}$	$\sum_{m}(Conectado_{ptm}) \leq 1 \quad \forall\ p,t$	def rule_r10(model,p,t): #RestTrator return sum(model.Conectado[p,t,m] for m in model.m) <= 1 model.r10 = pyo.Constraint(model.p, model.t, rule=rule_r10)
$RestMaqConectada_{ptm}$	$\sum_{m}(Conectado_{ptm}) \leq 1 \quad \forall\ p,t,$m	def rule_r11(model,p,t,m): #RestMaqConectada

		return sum(model.Serviu[p,t,l,a]*model.AtvMaq[a,m] for l in model.l for a in model.a) <= big_M * model.Conectado[p,t,m] model.r11 = pyo.Constraint(model.p, model.t, model.m, rule=rule_r11)

4.3.7. Resolução

Para inicializar o modelo é necessário acionar o *solver* desejado. Pyomo possui integrado em si os solver gratuitos GLPK e CBC sendo o método para acioná-los da seguinte forma

```
#GLPK
opt = pyo.SolverFactory('glpk')
opt.options['tmlim'] = 7200 #duration
opt.options['mipgap'] = 0.1 #gap
result = opt.solve(model)
result.write()
print("GLPK:",value(model.objective))

#CBC
solver = pyo.SolverFactory('cbc')
solver.options['tmlim'] = 7200 #duration
solver.options['mipgap'] = 0.1 #gap
result = solver.solve(model,tee = False)
```

```
result.write()
print("CBC:",value(model.objective))


#HiGHS
filename = 'model.mps'


h.setOptionValue('presolve', 'on')
h.setOptionValue('parallel', 'on')


h.readModel(filename)
h.run()


print('Model ', filename, ' has status ', h.getModelStatus())
print('HiGHS = ', info.objective_function_value)
```

Nenhum dos dois *solvers* nativos (GLPK e CBC) foram capazes de resolver o modelo utilizado por Azevedo (2022) com as 15 demandas em tempo hábil (testes feitos em períodos de 2 a 3 horas). Contudo com problemas de até 3 demandas totais foi possível validar com o mesmo resultado tanto no *AIMMS* quanto no *Google Colab* com soluções iguais utilizando também o HiGHS.

3 demandas: Limpar, Subsolar e Levantar Canteiro no lote 1

AIMMS:

- Tempo de resolução: 0,02 segundos
- Resultado apresentado: 2478 (ótimo)

GLPK:

- Tempo de resolução: 0,05 segundos
- Resultado apresentado: 2478 (ótimo)

CBC:

- Tempo de resolução: 0,12 segundos
- Resultado apresentado: 2478 (ótimo)

HiGHS:

- Tempo de resolução: 0,65 segundos
- Resultado apresentado: 2478 (ótimo)

Rota gerada:

Rota otimizada para 3 demandas distintas no lote 1 tanto no *AIMMS* quanto no programa desenvolvido, a rota foi igual:

Tabela 17 - Rota gerada a partir do modelo AIMMS e Python com 3 demandas

p	t	i	j
1	TT02	Deposito	1
1	TT02	1	Deposito
3	TT02	Deposito	1
3	TT02	1	Deposito
4	TT02	Deposito	1
4	TT02	1	Deposito

15 demandas: Modelo proposto

AIMMS:

- Tempo de resolução: 18,39 segundos
- Resultado apresentado: 4466 (ótimo)

GLPK:

- Tempo de resolução: excedeu limite de tempo (7200 segundos, duas horas)
- Resultado apresentado: viável
- *Gap*: testado até 10%

CBC:

- Tempo de resolução: excedeu limite de tempo (7200 segundos, duas horas)
- Resultado apresentado: viável
- *Gap*: testado até 10%

HiGHS:

- Tempo de resolução: 4319,13 segundos (1 hora e 12 minutos)
- Resultado apresentado: 4466 (ótimo)

Rota gerada:

Rota otimizada para 3 demandas distintas no lote 1 tanto no *AIMMS* quanto no programa desenvolvido, a rota foi igual:

Tabela 18 - Rota gerada a partir do modelo AIMMS e Python com 15 demandas

p	t	i	j
1	TT02	Deposito	1
1	TT02	1	Deposito
1	TT02	3	4
1	TT02	4	3

2	TT02	Deposito	8
2	TT02	8	Deposito
3	TT02	Deposito	1
3	TT02	1	Deposito
3	TT02	3	5
3	TT02	5	3
3	TT02	6	7
3	TT02	7	6
4	TT02	Deposito	2
4	TT02	2	8
4	TT02	8	Deposito
5	TT02	Deposito	1
5	TT02	1	3
5	TT02	3	Deposito
5	TT02	7	8
5	TT02	8	7

Comparação:

Segue a Tabela 19 que compara testes e resultados encontrados com os solvers

Tabela 19 - Comparação de testes (autoria própria)

	AIMMS			GLPK			CBC			HiGHS		
Demanda	3	7	15	3	7	15	3	7	15	3	7	15
Tempo (s)	0,02	0,34	18,39	0,05	7200	7200	0,12	7200	7200	0,65	4,14	4319,13
Gap (%)	0	0	0	0	10	10	0	10	10	0	0	0
Solução	2478	2558	4466	2478	Viável	Viável	2478	Viável	Viável	2478	2558	4466

Com os testes feitos com o modelo utilizando dados de 3, 7 e 15 demandas é possível tirar conclusões pertinentes para a comparação entre *AIMMS* e código *Python* implantando em *Google Colab*.

Todos os *solvers* são comparáveis em eficiência em modelos pequenos. Sendo uma via de validação do problema modelado. Todos resolveram o problema em menos de um segundo, retornando o mesmo resultado. Já em modelos maiores (exemplificado por 7 demandas no caso) os *solver* GLPK e CBC já não conseguem retornar em tempo hábil (definido como duas horas) um resultado ótimo, mesmo considerando uma tolerância (*gap*) de 10% do valor ótimo.

O solver HiGHS soluciona problemas com 7 e 15 demandas testadas com resultado igual ao *AIMMS*, contudo em tempo muito superior. Assim sendo, HiGHS é o *solver* gratuito mais indicado integrado ao *Python* no *Google Colab* para resolver problemas de programação inteira. Contudo não é indicado para problemas grandes, assim como *AIMMS* segundo os achados de Azevedo (2022), problemas pouco maiores já excederiam o limite de tempo determinado nos testes.

Como uma alternativa gratuita ao *AIMMS,* código *Python* implantado em *Google Colab* utilizando *solver* HiGHS e biblioteca Pyomo, é recomendado para problemas de Programação Inteira.

5. CONCLUSÃO E RECOMENDAÇÕES

A tecnologia no âmbito do agronegócio tem se tornado mais presente na última década (EMBRAPA, 2011) com a busca, não apenas de maior qualidade de produtos, mas também de otimização nos processos relacionados. Grande parte dos custos da cadeia de suprimentos de alimentos orgânicos está no custeio do maquinário de plantio, assim encarecendo o produto final. Dessa forma faz-se necessária a redução de custos do processo de plantio para disponibilizar em escala maior produtos em valores competitivos, uma vez que produtos orgânicos vêm ganhando grande parte da demanda de mercado.

Azevedo (2022) desenvolveu um modelo de Programação Inteira na plataforma *AIMMS Developer* baseado no problema do Caixeiro Viajante para determinar o roteamento ideal de tratores de acordo com as demandas de lotes em uma fazenda de plantio de produtos orgânicos. O modelo foi validado contudo não retornou resultado ótimo em tempo hábil. Em modelos menores, com menos demandas, retornou resultados rapidamente.

Contudo *AIMMS Developer* é ofertado com uma licença paga (com licença acadêmica gratuita) e foi o objetivo deste trabalho comparar com uma alternativa completamente gratuita. A alternativa estudada foi o desenvolvimento do mesmo modelo na linguagem de programação *Python* por meio de um caderno *Google Colab*. Ambas soluções gratuitas e com muitas fontes de apoio para desenvolvimento. Foi possível estudar que a programação *Python* em caderno *Google Colab* é uma alternativa gratuita viável, principalmente com o uso de *solver* HiGHS (também gratuito).

Devido ao armazenamento em nuvem do Google Colab, não é possível fazer instalações locais em máquinas, assim não possibilitando o uso do solver CPLEX para uma comparação direta com *AIMMS Developer,* sendo esse o solver usado na ferramenta. CPLEX oferece uma versão gratuita do *solver* para uso em modelos Python que pode ser carregado através nuvem, contudo possui um limite pequeno de variáveis (1000 variáveis) no modelo que não atende o modelo desenvolvido (que conta com 2700 variáveis). Dessa forma, recomenda-se fazer testes com outras plataformas de desenvolvimento gratuitas que possam ter uma interação mais

otimizada com arquivos locais de máquinas, assim permitindo o uso de outros *solvers proprietários como* CPLEX e *Gurobi*. Outras possíveis sofisticações do modelo seriam implantar escalas de tratoristas para operar mais de um trator por vez, priorização de demandas, dinamização das atividades necessárias, uma vez que a subsolação, por exemplo, não necessita ser feita com frequência.

Recomenda-se também fazer o teste do mesmo modelo na linguagem de programação Julia, focada em grandes computações numéricas, com o uso do pacote JuMP (pacote dedicado para programação mista). A linguagem *Julia* além de otimizada para grandes operações, também é gratuita e possui integração HiGHS.

Por fim, este trabalho é uma comparação de um mesmo modelo em duas formas de escrita, uma paga e outra gratuita. O *AIMMS Develope*r se mostrou mais potente e rápido na solução de problemas de grande porte. O *Google Colab* apresentou dificuldades na interface de *solvers* proprietários para auxílio de cálculos mais complexos, mas teve retorno positivo com o uso do solver HiGHS. Assim sendo, conclui-se que a ferramenta *AIMMS Developer* é mais eficiente para solução de problemas de grande porte, mas é possível fazer uso de ferramentas gratuitas para alcançar o mesmo resultado com uma dilatação no tempo de resposta.

REFERÊNCIA

ANDRÉ ANDRADE LONGARAY. **Introdução à pesquisa operacional.** [s.l.] Saraiva Educação S.A. p. 59 , 2017.

AZEVEDO, J. P. **Agronegócio no Brasil: qual a importância para o país e o seu papel**. Disponível em: <https://rehagro.com.br/blog/agronegocio-no-brasil-qual-o-seu-papel-e-importancia/#:~:text=O%20agroneg%C3%B3cio%20hoje%20%C3%A9%20respons%C3%A1vel>.

AZEVEDO, P. H. F. **CADEIA DE SUPRIMENTOS DE ALIMENTOS ORGÂNICOS: Uma abordagem matemática para o planejamento das rotas do maquinário**. Monografia—Universidade de Brasília: [s.n.].

BELFIORE, P.; FÁVERO, L.. **Pesquisa Operacional para cursos de engenharia**. Rio de Janeiro: Elsevier, 2013.

BOCK, F. **Mathematical programming solution of traveling salesman examplesRecent advances in Mathematical Programming,(RL Graves and P. Wolfe, eds.)**. [s.l: s.n.].

BRADLEY, S. P.; HAX, A. C.; MAGNANTI, T. L. **Applied mathematical programming.** [s.l.] Reading, Mass. Addison-Wesley, 1992.

Capítulo 2 - Tecnologia da Informação no agronegócio. In: **Estudo do mercado brasileiro de software para o agronegócio**. [s.l.] Embrapa Agricultura Digital, 2011.

CEPEA. **PIB DO AGRONEGÓCIO BRASILEIRO**. 2021. Disponível em: https://www.cepea.esalq.usp.br/br/pib-do-agronegocio-brasileiro.aspx. Acesso em: 13 jan. 2021.

CHIAVENATO, Idalberto. **Introdução à teoria geral da administração**. 7. ed. Rio de Janeiro: Elsevier, 2004.

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. 2020. 90 f., il. Trabalho de Conclusão de Curso (Bacharelado em Administração)—Universidade de Brasília, Brasília, 2020

DIMÉNY, I.; KOLTAI, T. **Supporting Operations Management Decisions with LP Parametric Analyses Using AIMMS. Periodica Polytechnica Social and Management Sciences,** v. 28, n. 2, p. 91–100, 8 jun. 2020.

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

Google Colab: saiba o que é essa ferramenta e como usar! | Insights para te ajudar na carreira em tecnologia | Blog da Trybe. Disponível em: <https://blog.betrybe.com/carreira/google-colab/>. Acesso em: 28 nov. 2022.

HART, W. E.; WATSON, J. P.; WOODRUFF, D. L. **Pyomo: modeling and solving mathematical programs in Python. Mathematical Programming Computation, p.** 3–219, 8 out. 2011.

HUANGFU, Q.; HALL,. Parallelizing the dual revised simplex method. **Mathematical Programming Computation**, v. 10, n. 1, p. 119–142, 2018.

IGNÁCIO, A. A. V.; FERREIRA FILHO, V. J. M.. **Seção de Software: o uso de software de modelagem AIMMS na solução de problemas de programação matemática**. Pesquisa Operacional, v. 24, n. Pesqui. Oper., 2004 24(1), jan. 2004.

JOHANNES BISSCHOP. **Aimms Optimization Modeling.** [s.l.] Lulu.com, 2006.

O que há de novo no Python. Disponível em: <https://docs.python.org/pt-br/3/whatsnew/index.html>. Acesso em: 14 jan. 2023.

pandas - **Python Data Analysis Library**. Disponível em: <https://pandas.pydata.org/about/index.html>. Acesso em: 25 jan. 2023.

PRADO, D. **Programação Linear.** [s.l.] Falconi Editora, 2016.

PYTHON SOFTWARE FOUNDATION. **Python Language Site: Documentation,** 2020. Página de documentação. Disponível em: <https://www.python.org/doc/>. Acesso em: 29 de nov. de 2022.

RECH, V.; GONÇALVES, R. B.; VIEIRA, G. B. B. **Estudo comparativo dos custos de produção de uvas pelos métodos orgânico e convencional**. Artigo—Custos e @gronegócio: [s.n.].

RINALDI, M.; HE, Z. Chapter six - **decision support systems to manage irrigation in Agriculture***Present address: Consiglio per la ricerca e la sperimentazione in agricoltura, cereal research centre, S.S. 673km 25,200, 71122 foggia, italy. In: SPARKS, D. L. (Ed.). [s.l.] Academic Press, 2014. v. 123p. 229–279.

SANTOS, H. G.; TOFFOLO, T. A. M. **Mixed Integer Linear Programming with Python.** p. 14, 10 nov. 2020.

SANTOS, J. A. L. **Análise de risco quanto aos custos de projetos de obra civil residencial utilizando modelagem computacional via Python**. Universidade Federal dos Vales do Jequitinhonha e Mucuri: [s.n.].

SILVA, E.L; MENEZES, E.M. **Metodologia da Pesquisa e Elaboração de Dissertação**. Florianópolis: UFSC, 2005.

What is integer programming? Disponível em: <https://www.ibm.com/docs/en/icos/12.8.0.0?topic=problem-what-is-integer-programming>.

APÊNDICE

Apêndice A - Dados de entrada demanda em Python

Tabela 2 - Dados de entrada demanda no Python (autoria própria)

	Limpar	Subsolar	Incorporar	Levantar Canteiro	Pulverização	Adubar
Depósito	0	0	0	0	0	0
1	1	1	1	0	0	0
2	0	0	0	1	0	0
3	1	1	1	0	0	0
4	0	1	0	0	0	0
5	0	0	1	0	0	0
6	0	0	0	0	1	0
7	1	0	1	0	0	0
8	1	0	0	1	0	1

Apêndice B - Entradas dos conjuntos do problema em python

Tabela 4 - Entrada dos conjuntos dos índices para python

Conjunto	Código

Nós	`Nós = ["Deposito", "1", "2", "3", "4", "5", "6", "7", "8"]`
	
	`Lotes = Nós[1:9]`
	`l = len(Lotes)`
	
	`Deposito = Nós[0:1]`
	`o = len(Deposito)`

Atividades	`Atividades = ["Limpar", "Subsolar", "Levantar Canteiro", "Adubar", "Incorporar", "Pulverização"]` `a = len(Atividades)`
Duração	`duracao_a = [20, 20, 20, 20, 20, 20]`
Tratores	`Tratores = ["TT02", "TT03", "TT07", "TT08"]` `t = len(Tratores)`
Custo	`CustoLigar_t = [10, 15, 20, 30]`

Máquinas	`Máquinas = ["Triton", "Subsolador", "Rotoencanteirador", "Adubadeira", "Carreta Bio"]`
	
	`m = len(Máquinas)`

Apêndice C - Variáveis em Pyomo

Tabela 14 - Comparação da declaração da variável no AIMMS e Python

Variável no modelo	Variável em Python
$Inclui_{ptij}$	`model.Inclui=pyo.Var(model.p,model.t,model.i,model.j, within=pyo.Binary)`
$Serviu_{ptla}$	`model.Serviu=pyo.Var(model.p,model.t,model.l,model.a, within=pyo.Binary)`
$LigaTrator_{ptj}$	`model.LigarTrator=pyo.Var(model.p,model.t, within=pyo.Binary)`
$Conectado_{ptm}$	`model.Conectado=pyo.Var(model.p,model.t,model.m, within=pyo.Binary)`

Apêndice D - Parâmetros em Pyomo

Tabela 15 - Comparação da declaração dos parâmetros no AIMMS e Python

Parâmetro no modelo	Parâmetro em Python
$Demanda_{ia}$	`model.Demanda = pyo.Param(model.i, model.a, initialize = lambda model, i, a: Demanda_la[i-1][a-1])`
$AtividadeMaq_{am}$	`model.AtvMaq = pyo.Param(model.a, model.m, initialize = lambda model, a, m: atvmaq[a-1][m-1])`
$Duração_{a}$	`model.Duracao = pyo.Param(model.a, initialize = lambda model, a: duracao_a[a-1])`
$Distância_{ij}$	`model.Distancia = pyo.Param(model.i, model.j,initialize=lambda model, i, j: cost_matrix[i-1][j-1])`
$DistanciaTempo_{ij}$	`model.DistanciaTempo = pyo.Param(model.i, model.j,initialize=lambda model, i, j: disttemp[i-1][j-1])`
$Turno$	`model.turno = pyo.Param(initialize=480)`
big_{M}	`model.bigM = pyo.Param(initialize=10**9)`

$CustoLigar_t$	`model.CustoLigar = pyo.Param(model.t, initialize = lambda model, t: CustoLigar_t[t-1])`

Apêndice E - Restrições em Pyomo

Tabela 16 - Comparação da declaração dos restrição no AIMMS e Python

Restrição	Python

$RestEquilibrio_{pti}$	`def rule_r1(model,p,t,i): #RestEquilibrio_pti`
	`return sum(model.Inclui[p,t,j,i] for j in model.j) == sum(model.Inclui[p,t,i,j] for j in model.j)`
	`model.r1 = pyo.Constraint(model.p, model.t, model.i,rule=rule_r1)`

$RestIncluiDemanda_l$	`def rule_r2(model,l): #RestIncluiDemanda`
	`return sum(model.Inclui[p,t,i,l] for p in model.p for t in model.t for i in model.i) == sum(model.Demanda[l,a] for a in model.a)`
	`model.r2 = pyo.Constraint(model.l,rule=rule_r2)`

$RestServiuDemanda_{la}$	`def rule_r3(model,l,a): #RestServiuDemanda`
	`return sum(model.Serviu[p,t,l,a] for p in model.p for t in model.t) == model.Demanda[l,a]`
	`model.r3 = pyo.Constraint(model.l, model.a,rule=rule_r3)`

$RestNoFinal_{tp}$	def rule_r4(model,p,t): #RestNoFinal return sum(model.Inclui[p,t,l,o] for l in model.l for o in model.o) == model.LigarTrator[p,t] model.r4 = pyo.Constraint(model.p, model.t, rule=rule_r4)
$RestTurno_{pt}$	def rule_r5(model,p,t): #RestTurno return sum(model.Serviu[p,t,l,a]*model.Duracao[a] for l in model.l for a in model.a) + sum(model.Inclui[p,t,i,j] * model.DistanciaTempo[i,j] for i in model.i for j in model.j) <= 480 model.r5 = pyo.Constraint(model.p, model.t,rule=rule_r5)
$RestLink_{mltp}$	def rule_r6(model,p,t,l,a): #RestLink return sum(model.Inclui[p,t,i,l] for i in model.i) >= model.Serviu[p,t,l,a] model.r6 = pyo.Constraint(model.p, model.t, model.l, model.a, rule=rule_r6)
$RestLigaTrator_{pt}$	def rule_r7(model,p,t): #RestLigaTrator return sum(model.Serviu[p,t,l,a] for l in model.l for a in model.a) <= (10**9)*model.LigarTrator[p,t] model.r7 = pyo.Constraint(model.p, model.t,rule=rule_r7)
$RestMaqPorLote_{lp}$	def rule_r8(model,l,p): #RestMaqPorLote

	return sum(model.Serviu[p,t,l,a] for t in model.t for a in model.a) <= 1 model.r8 = pyo.Constraint(model.l, model.p, rule=rule_r8)
$RestMaq_{pm}$	def rule_r9(model,p,m): #RestMaq return sum(model.Conectado[p,t,m] for t in model.t) <= 1 model.r9 = pyo.Constraint(model.p, model.m,rule=rule_r9)
$RestTrator_{pt}$	def rule_r10(model,p,t): #RestTrator return sum(model.Conectado[p,t,m] for m in model.m) <= 1 model.r10 = pyo.Constraint(model.p, model.t, rule=rule_r10)
$RestMaqConectada_{ptm}$	def rule_r11(model,p,t,m): #RestMaqConectada return sum(model.Serviu[p,t,l,a]*model.AtvMaq[a,m] for l in model.l for a in model.a) <= big_M * model.Conectado[p,t,m] model.r11 = pyo.Constraint(model.p, model.t, model.m, rule=rule_r11)

Apêndice F - Modelo em linguagem *Python*

```
#@title Instalações e Importações

#Pyomo
!pip install -q pyomo
from pyomo.environ import *
import pyomo.environ as pyo

#Solvers
!apt-get install -y -qq glpk-utils #GLPK
!apt-get install -y -qq coinor-cbc #CBC
!pip install highspy #HiGHS
import highspy
h = highspy.Highs()

#numpy
import numpy as np
from numpy.lib.shape_base import array_split

#drivemount para inserir dados via google drive
from re import A
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
pd.set_option('display.max_rows', 1000)
pd.set_option('display.max_columns', 1000)
pd.set_option('display.width', 1000)

#@title Parâmetros

#Distância em metros entre nó i e j
pathdist ="/content/drive/MyDrive/Colab Notebooks/Dados - Distancia.csv"
df = pd.read_csv(pathdist)
df.describe
cost_matrix = np.genfromtxt(pathdist, delimiter =",")

n = len(cost_matrix)

#Distância em metros/tempo
pathdisttemp ="/content/drive/MyDrive/Colab Notebooks/Dados - DistanciaTempo.csv"
df = pd.read_csv(pathdisttemp)
df.describe
disttemp = np.genfromtxt(pathdisttemp, delimiter =",")

#Atividades a de maquinário m
pathatv = "/content/drive/MyDrive/Colab Notebooks/Dados - AtvMaq.csv"
df = pd.read_csv(pathatv)
df.describe
atvmaq = np.genfromtxt(pathatv, delimiter =",")

#Demandas no lote l de uma Atividade a
pathdem = "/content/drive/MyDrive/Colab Notebooks/Dados - Demandas.csv"
df = pd.read_csv(pathdem)
df.describe
Demanda_la = np.genfromtxt(pathdem, delimiter =",")

#Nós
Nós = ["Deposito", "1", "2", "3", "4", "5", "6", "7", "8"]

#Máquinas
```

```
Máquinas = ["Triton", "Subsolador", "Rotoencanteirador",  "Adubadeira", "Carreta
Bio"]
m = len(Máquinas)

#Atividades
Atividades = ["Limpar", "Subsolar", "Levantar Canteiro",  "Adubar", "Incorporar",
"Pulverização"]
a = len(Atividades)

#Tratores
Tratores = ["TT02", "TT03", "TT07", "TT08"]
t = len(Tratores)

#Custo Ligar Trator t
CustoLigar_t = [10, 15, 20, 30]

#Duração de cada atividade em minutos
duracao_a = [20, 20, 20, 20, 20, 20]

#Horas de trabalho disponíveis
turno = 480

#Período
Periodo = [1, 2, 3, 4, 5]
p = len(Periodo)

#Lotes
Lotes = Nós[1:9]
l = len(Lotes)

#Deposito
Deposito = Nós[0:1]
o = len(Deposito)

big_M = 10**9

'''
#impressão das tabelas pode ser omitida
print("• Nós:")
print(np.array(Nós,list))

print(" ")
print("• Lotes:")
print(np.array(Lotes,list))

print(" ")
print("• Depósito:")
print(np.array(Deposito,list))

print(" ")
print("• Distância entre os nós:")

tabcost = pd.DataFrame(np.array(cost_matrix,list), columns =
[np.array(Nós,list)], index=[np.array(Nós,list)])
print(tabcost)

print(" ")
print("• Tempo entre os nós:")
tabdist = pd.DataFrame(np.array(disttemp,list), columns = [np.array(Nós,list)],
index=[np.array(Nós,list)])
print(tabdist)

print(" ")
print("• Tratores")
print(np.array(Tratores,list))

print(" ")
```

```
print("• Custo de ligar um trator (t)")
print(CustoLigar_t)

print(" ")
print("• Atividades")
print(np.array(Atividades,list))

print(" ")
print("• Máquinas:")
print(np.array(Máquinas,list))

print(" ")
print("• Atividade (a) de Maquinário (m):")
tabatvmaq = pd.DataFrame(np.array(atvmaq,list), columns = [Máquinas],
index=[Atividades])
print(tabatvmaq)

print(" ")
print("• Demandas em um Lote (l) de uma Atividade (a)")
tabdemand = pd.DataFrame(np.array(Demanda_la,list), columns = [Atividades], index
= [Nós])
print(tabdemand)

print(" ")
print("• bigM")
print(np.array(big_M,list))
'''

#@title Modelo
model = pyo.ConcreteModel()

#Índices
model.t = pyo.RangeSet(t)
model.m = pyo.RangeSet(m)
model.a = pyo.RangeSet(a)
model.p = pyo.RangeSet(p)
model.o = pyo.RangeSet(o)
model.j = pyo.RangeSet(n)
model.i = pyo.RangeSet(n)
model.l = pyo.RangeSet(2,n)

model.U = pyo.RangeSet(2,n)

#Parâmetros
model.Distancia = pyo.Param(model.i, model.j,initialize=lambda model, i, j:
cost_matrix[i-1][j-1])
model.DistanciaTempo = pyo.Param(model.i, model.j,initialize=lambda model, i, j:
disttemp[i-1][j-1])
model.CustoLigar = pyo.Param(model.t, initialize = lambda model, t:
CustoLigar_t[t-1])
model.AtvMaq = pyo.Param(model.a, model.m, initialize = lambda model, a, m:
atvmaq[a-1][m-1])
model.Demanda = pyo.Param(model.i, model.a, initialize = lambda model, i, a:
Demanda_la[i-1][a-1])
model.bigM = pyo.Param(initialize=10**9)
model.Duracao = pyo.Param(model.a, initialize = lambda model, a: duracao_a[a-1])
model.turno = pyo.Param(initialize=480)

#Variáveis
model.Inclui=pyo.Var(model.p,model.t,model.i,model.j, within=pyo.Binary)
model.Serviu=pyo.Var(model.p,model.t,model.l,model.a, within=pyo.Binary)
model.LigarTrator=pyo.Var(model.p,model.t, within=pyo.Binary)
model.Conectado=pyo.Var(model.p,model.t,model.m, within=pyo.Binary)

model.x=pyo.Var(model.i,model.j,within=pyo.Binary)
model.u=pyo.Var(model.i,within=NonNegativeIntegers,bounds=(0,n-1))
```

```
#Objetivo
def func_obj(model):
  return sum(model.Inclui[p,t,i,j]*model.Distancia[i,j] for p in model.p for t in
model.t for i in model.i for j in model.j) +
sum(model.LigarTrator[p,t]*model.CustoLigar[t] for p in model.p for t in model.t)

model.objective = pyo.Objective(rule=func_obj,sense=pyo.minimize)

#Restrições
def rule_r1(model,t,i,p): #RestEquilibrio_pti
    return sum(model.Inclui[p,t,j,i] for j in model.j) ==
sum(model.Inclui[p,t,i,j] for j in model.j)

model.r1 = pyo.Constraint(model.t, model.i, model.p,rule=rule_r1)

def rule_r2(model,l): #RestIncluiDemanda
    return sum(model.Inclui[p,t,i,l] for p in model.p for t in model.t for i in
model.i) == sum(model.Demanda[l,a] for a in model.a)

model.r2 = pyo.Constraint(model.l,rule=rule_r2)

def rule_r3(model,l,a): #RestServiuDemanda
    return sum(model.Serviu[p,t,l,a] for p in model.p for t in model.t) ==
model.Demanda[l,a]

model.r3 = pyo.Constraint(model.l, model.a,rule=rule_r3)

def rule_r4(model,p,t): #RestNoFinal
    return sum(model.Inclui[p,t,i,o] for i in model.i for o in model.o) ==
model.LigarTrator[p,t]

model.r4 = pyo.Constraint(model.p, model.t, rule=rule_r4)

def rule_r5(model,p,t): #RestTurno
    return sum(model.Serviu[p,t,l,a]*model.Duracao[a] for l in model.l for a in
model.a) + sum(model.Inclui[p,t,i,j] * model.DistanciaTempo[i,j] for i in model.i
for j in model.j) <= 480

model.r5 = pyo.Constraint(model.p, model.t,rule=rule_r5)

def rule_r6(model,p,t,l,a): #RestLink
    return sum(model.Inclui[p,t,i,l] for i in model.i) >= model.Serviu[p,t,l,a]

model.r6 = pyo.Constraint(model.p, model.t, model.l, model.a, rule=rule_r6)

def rule_r7(model,p,t): #RestLigaTrator
    return sum(model.Serviu[p,t,l,a] for l in model.l for a in model.a) <=
(10**9)*model.LigarTrator[p,t]

model.r7 = pyo.Constraint(model.p, model.t,rule=rule_r7)

def rule_r8(model,l,p): #RestMaqPorLote
    return sum(model.Serviu[p,t,l,a] for t in model.t for a in model.a) <= 1

model.r8 = pyo.Constraint(model.l, model.p, rule=rule_r8)

def rule_r9(model,p,m): #RestMaq
    return sum(model.Conectado[p,t,m] for t in model.t) <= 1

model.r9 = pyo.Constraint(model.p, model.m,rule=rule_r9)

def rule_r10(model,p,t): #RestTrator
    return sum(model.Conectado[p,t,m] for m in model.m) <= 1

model.r10 = pyo.Constraint(model.p, model.t, rule=rule_r10)

def rule_r11(model,p,t,m): #RestMaqConectada
```

```
    return sum(model.Serviu[p,t,l,a]*model.AtvMaq[a,m] for l in model.l for a in
model.a) <= (10**9) * model.Conectado[p,t,m]

model.r11 = pyo.Constraint(model.p, model.t, model.m, rule=rule_r11)

#modelo para HiGHS
model.write('model.mps','mps')
filename = 'model.mps'

#@title GLPK
opt = pyo.SolverFactory('glpk')
opt.options['tmlim'] = 7200 #Tempo
opt.options['mipgap'] = 0 #Gap
result = opt.solve(model)
result.write()
print("GLPK:",value(model.objective))

#@title CBC
solver = pyo.SolverFactory('cbc')
solver.options['tmlim'] = 7200 #Tempo
solver.options['mipgap'] = 0.1 #Gap
result = solver.solve(model,tee = False)
result.write()

#@title HiGHS
h.setOptionValue('presolve', 'on')
h.setOptionValue('parallel', 'on')
#h.setOptionValue('time_limit', '7200')
#h.setOptionValue('mip_feasibility_tolerance', '10e-1')
h.setOptionValue('solution_file', 'HiGHS.sol')
h.setOptionValue('write_model_file', 'HiGHS.mps')
solution = h.getSolution()
basis = h.getBasis()
info = h.getInfo()
model_status = h.getModelStatus()

h.readModel(filename)
h.run()
print('Model ', filename, ' has status ', h.getModelStatus())
print('HiGHS = ', info.objective_function_value)

print(solution)
print(basis)
print(info)

h.setOptionValue('write_solution_style',1)
h.setOptionValue('write_solution_to_file',True)

#Soluções comparadas
print("CBC:",value(model.objective))
print('Model ', filename, ' has status ', h.getModelStatus())
print('HiGHS = ', info.objective_function_value)

#Lista das variáveis escolhidas
List = list(model.Inclui.keys())
for i in List:
    if model.Inclui[i]() != 0:
        print(i,'--', model.Inclui[i]())
print(value(model.objective))
```