Universidade de Brasília

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

Departamento de Administração

Wilson de Brito Ferreira

PROBLEMA DE ALOCAÇÃO DE FROTA: modelagem, simulação e comparação de algoritmos para resolver problemas MILP usando computação paralela.

Brasília - DF

2023

Wilson de Brito Ferreira

PROBLEMA DE ALOCAÇÃO DE FROTA: modelagem, simulação e comparação de algoritmos para resolver problemas MILP usando computação paralela.

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

Orientador: Professor Doutor Victor Rafael Rezende Celestino

Brasília - DF

2023

Wilson de Brito Ferreira

PROBLEMA DE ALOCAÇÃO DE FROTA: modelagem, simulação e comparação de algoritmos para resolver problemas MILP usando computação paralela.

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

Wilson de Brito Ferreira

Dr. Victor Rafael Rezende Celestino

Professor-Orientador

Profa. Dra. Silvia Araújo dos Reis

Professora-Examinadora

Prof. Dr. Kayo Gonçalves e Silva

Professor-Examinador

Brasília, julho de 2023.

Dedicado a meu pai, Romildo de Brito Ferreira, in memoriam.

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus, que me concedeu concluir esta etapa de minha jornada.

Agradeço a meus pais, Teresinha e Romildo; a minha irmã, Keyla; minha amiga companheira de estudos Asinete; aos meus colegas estudantes, professores e funcionários da Universidade de Brasília.

Agradecimento especial ao Professor Dr. Victor Rafael Rezende Celestino, sempre diligente, paciente e objetivo nas orientações.

Gratidão aos professores Silvia e Kayo pela boa vontade em participar da banca avaliadora deste trabalho.

Agradeço ao Professor Dr. Samuel Xavier de Souza e ao Núcleo de Processamento de Alto Desempenho da UFRN pela disponibilização de acesso ao ambiente computacional paralelo.

Thomson's Rule for First-Time Telescope Makers:

"It is faster to make a four-inch mirror then a six-inch mirror than to make a six-inch mirror."

Programming Pearls, Communications of the ACM, September 1985.

Resumo

Este trabalho, construído sobre pesquisas anteriores, investigou o emprego de paradigmas de programação paralela, aliado à execução de programas em múltiplos processadores e unidades de processamento gráfico - GPUs - para a resolução de problemas de alocação de frota. O objetivo foi modelar o Problema de Alocação de Frota com estruturas da linguagem Python de forma a possibilitar a resolução do modelo de programação linear mista resultante com recursos de programação paralela a nível de software e hardware. Para a consecução do objetivo foram empregadas abordagens a nível de software, dentro de um ambiente de programação centrado na linguagem Python, e hardware, com uso de equipamentos com processamento paralelo em CPU. Programas Python foram adaptados para o ambiente de computação paralela do NPAD/UFRN, tanto na forma interativa do Jupyter Notebook quanto para execução não assistida em linha de comando e por intermédio de arquivos de comandos para o interpretador bash. O desempenho na solução do Problema de Alocação de Frota foi avaliado para os solvers CBC e HiGHS. Os resultados encontrados corroboram trabalho anterior, trazem perspectivas quanto ao uso dos recursos de programação paralela para a solução de problemas de programação linear mista em geral e propostas para novos estudos sobre o tema.

Palavras chaves: Pesquisa Operacional, Problema de Alocação de Frota, Aviação Regional, Programação Linear Inteira Mista, Programação Paralela, Hórus, Python, HiGHS.

Abstract

This work, built on previous research, investigated the use of parallel programming paradigms, combined with the execution of programs on multiple processors and graphic processing units - GPUs - to solve fleet assignment problems. The objective was to model the Fleet Assignment Problem with Python language structures in order to enable the resolution of the resulting mixed linear programming model with parallel programming resources at the software and hardware level. To achieve the objective, software-level approaches were used, within a programming environment centered on the Python language, and hardware, with the use of equipment with parallel processing on the CPU. Python programs were adapted for the NPAD/UFRN parallel computing environment, both in the interactive form of the Jupyter Notebook and for unattended execution in the command line and through command files for the bash interpreter. The performance in solving the Fleet Assignment Problem was evaluated for the CBC and HiGHS. The results found corroborate previous work, bring perspectives on the use of parallel programming resources to solve mixed linear programming problems in general and proposals for further studies on the subject.

Keywords: Operations Research, Fleet Assignment Problem, Regional Aviation, Mixed Integer Linear Programming, Parallel Programming, Horus, Python, HiGHS.

Sumário

1 INTRODUÇÃO

1.1 Contextualização

Este trabalho, baseando-se em modelo de Pesquisa Operacional para o Problema de Alocação de Frota no contexto da aviação regional nacional, elaborado em estudo anterior (FELIPE, 2022), visa analisar alternativas para a solução do modelo com recursos de computação paralela.

Segundo dados divulgados em julho de 2022 pela Agência Nacional de Aviação Civil - ANAC - o mercado de transporte aéreo brasileiro doméstico apresentou aumento no número de voos, quantidade de passageiros e aproveitamento das aeronaves no ano de 2021 em relação a 2020, marcado pelos efeitos da pandemia de COVID 19 (BRASIL, 2021).

Em vista deste cenário de recuperação econômica, a solução do Problema de Alocação de Frota é uma ferramenta auxiliar na tomada de decisão das empresas aéreas para um melhor aproveitamento das aeronaves, com consequente maximização do lucro operacional.

1.2 Formulação do problema

O transporte regional de passageiros, atividade econômica regular empregando aeronaves com capacidade inferior a cem assentos e efetuada em ligações de menor demanda dos aeroportos (BETTINI, 2007), é um importante componente do setor de transporte aéreo de passageiros no Brasil (ANAC, 2021).

Do ponto de vista da exploração comercial viável destas ligações, surge a necessidade de otimizar o emprego das aeronaves disponíveis a cada empresa, de modo a maximizar o lucro obtido das tarifas cobradas. Do faturamento obtido da operação de transporte de passageiros deve ser descontado o custo de operação, específico para cada aeronave e dependente da extensão do trecho voado. Dado um conjunto de aeronaves representativas do mercado aéreo nacional, cabe determinar quais podem ser empregadas visando esta maximização e atendendo a demanda das rotas consideradas.

1.3 Objetivo Geral

Modelar o Problema de Alocação de Frota com estruturas da linguagem Python de forma a possibilitar a resolução do modelo de programação linear mista resultante com recursos de programação paralela.

1.4 Objetivos Específicos

- Estudar e aperfeiçoar o método existente para a formulação e solução do Problema de Alocação de Frota;
- Executar experimentos com diferentes configurações de software e hardware:
  - Programação em Python, primariamente no ambiente Jupyter Notebook, usando o pacote PuLP;
  - Investigação de desempenho de solvers para programas lineares comerciais e de Software Livre;
  - Uso de ambientes de execução paralela baseados em CPU e GPU.
- Procurar novas alternativas de métodos e recursos de computação paralela na busca da resolução do problema.

1.5 Justificativa

A complexidade inerente ao modelo anteriormente desenvolvido (FELIPE, 2022), aliada à necessidade de aplicar o modelo como ferramenta de auxílio à tomada de decisão, tornam desejável a modificação do modelo para melhor adequação a um ambiente de execução paralela, o qual pode trazer agilidade à geração de soluções ótimas e, também, viabilizar a solução do modelo quando os parâmetros o tornam mais complexo, tal como quando considerado o aumento do número de aeronaves.

Conforme ressaltado por Bettini (2007), as dimensões continentais e número de centros urbanos do Brasil exigem o modal de transporte aéreo regional de passageiros como alternativa às opções de movimentação terrestre. O transporte aéreo regional, segundo o autor, fomenta atividades econômicas e auxilia a integração regional, tendo acompanhado a expansão das fronteiras agrícolas.

Tendo em vista a disponibilidade de GPUs de alto desempenho a custos de aquisição, operação e manutenção muito inferiores a instalações de processamento paralelo baseadas em processadores de uso geral, seu uso torna-se vantajoso às

operações comerciais e às aplicações de pesquisa operacional. Além da contribuição acadêmica, cumpre-se o requisito para a aquisição do Bacharelado em Administração na Universidade de Brasília.

2 REFERENCIAL TEÓRICO

2.1 Pesquisa Operacional

Winston (2004) define a Pesquisa Operacional como uma abordagem científica para o processo decisório aplicada à busca do melhor projeto e operação de um sistema sujeito a restrições de recursos. Segundo Arenales et al (2007), a Pesquisa Operacional tem suas origens durante o desenvolvimento inicial das operações de RADAR no Reino Unido, a partir de 1938.

Durante a Segunda Guerra Mundial, os métodos da Pesquisa Operacional foram aplicados a problemas de escolha de aeronaves para missões, manutenção e inspeção de aeronaves, e combate de submarinos. Posteriormente, nos Estados Unidos, em 1947, o matemático George Dantzig criou o método SIMPLEX para resolução de problemas de programação linear, com base em estudos anteriores dos matemáticos Kantorovich e Koopmans, laureados com o Prêmio Nobel de Economia em 1975 (DANTZIG, 2002; GASS, ASSAD, 2005) .

Winston (2004) sugere uma sequência de passos necessários ao uso da Pesquisa Operacional nas organizações:

Passo 1: Formulação do Problema. A primeira tarefa do pesquisador deve ser a definição do problema enfrentado pela organização, incluindo os objetivos buscados e a estrutura organizacional;

Passo 2: Observação do Sistema. A seguir, o pesquisador deve colher dados representativos que permitam estimar os parâmetros que farão parte do modelo matemático do problema, a ser construído nos passos 3 e 4 a seguir;

Passo 3: Formulação de um Modelo Matemático do Problema. É a definição de uma representação matemática do problema estudado;

Passo 4: Verificação do Modelo e Uso para Predição. Neste estágio o pesquisador efetua a validação do modelo proposto, verificando se os valores obtidos a partir dos previsto pelo modelo são coerentes com o problema real;

Passo 5: Seleção de uma Alternativa Adequada. Havendo alternativas de solução obtidas do modelo, o pesquisador irá selecionar, de acordo com a realidade da organização, aquelas que melhor atendem os objetivos organizacionais;

Passo 6: Apresentação dos Resultados e Conclusão do Estudo para a Organização. O pesquisador deve apresentar os resultados do estudo aos responsáveis pelas decisões da organização. Caso as recomendações resultantes do estudo não sejam acatadas, a pesquisa pode ser reiniciada a partir dos passos 1, 2 ou 3;

Passo 7: Implementação e Recomendação de Avaliação. Caso a organização aceite e decida implantar as recomendações do pesquisador, este deve auxiliar na implementação. Deve ocorrer monitoramento constante para garantir que os objetivos da organização estão sendo atingidos.

Taha (2008) elenca a programação linear, programação dinâmica, programação inteira, otimização de redes e outras técnicas como ferramentas utilizadas para a resolução dos modelos matemáticos. Dependendo da complexidade dos modelos gerados, pode ser necessária a utilização de métodos heurísticos além da redução da complexidade através da eliminação de detalhes do problema.

Em 1947, o matemático norte-americano George Dantzig definiu de forma geral o problema da programação linear e desenvolveu o método Simplex, uma das Ferramentas mais utilizadas para resolução de problemas de programação linear. Segundo Dantzig, um problema de programação linear pode ser modelado como a maximização ou minimização do produto **cx**, onde **c** é um vetor linha de dimensão 1 por n e **x**, um vetor coluna de dimensão n por 1, constituído por elementos não negativos. As restrições do modelo são representadas por **Ax** = **b** onde **A** é uma matriz de dimensões m por n e **b** um vetor coluna m x 1 (GASS, ASSAD, 2004).

2.2 Programação Inteira

Segundo Hillier e Lieberman (2006), muitos problemas reais possuem variáveis de decisão que são, por natureza, números inteiros. Para os autores, se todos os valores das variáveis de decisão são inteiros, trata-se de um problema de programação inteira. Entretanto, se algumas variáveis puderem ser números reais, o problema é denominado de programação inteira mista.

Um caso particular da programação inteira é a que envolve variáveis de decisão que permitem a escolha de determinadas alternativas. Essas variáveis podem ser representadas pelos valores 0 e 1 e são ditas variáveis binárias. Problemas de programação inteira apresentando apenas variáveis binárias podem ser denominadas de Programação Inteira Binária. Como destacado pelos autores (HILLIER, LIEBERMAN, 2006), o problema de alocação de frotas é um problema deste tipo, pois é necessário escolher um tipo específico de aeronave para cada trecho de voo atendido pela empresa para maximizar o lucro e cumprir a programação de voos.

A dificuldade de solução dos Problemas Inteiros Binários decorrem da quantidade exponencial de possíveis soluções a serem avaliadas. Há necessidade de testar as restrições para cada caso. Num problema de programação inteira, o número de variáveis inteiras, a sua caracterização como binárias ou variáveis inteiras genéricas e qualquer característica especial do problema tornam a resolução do problema inteiro mais complexa que da programação linear (HILLIER, LIEBERMAN, 2006). Embora seja possível usar os algoritmos Simplex e dual Simplex em problemas de programação inteira, não é possível garantir que os resultados arredondados serão soluções viáveis do problema.

Um método considerado eficiente para o ataque de problemas de programação inteira é o método de Branch-and-Bound ou ramificação e avaliação progressiva, o qual pode ser visto como um procedimento de enumeração inteligente dentro do espaço factível de soluções do problema. Este método consiste na ramificação ou subdivisão do conjunto de soluções viáveis, a limitação e a avaliação, incluindo eventual eliminação, de cada uma dessas possíveis soluções, repetindo-se o processo até alcançar a solução ótima.

2.3 Problema de Alocação de Frota - Fleet Assignment Problem

De acordo com Hillier e Lieberman (2006), o Problema de Alocação de Frota (FAP - Fleet Assignment Problem) consiste na escolha, dentre um rol de aeronaves disponíveis, daquelas que atenderão às rotas da malha viária de forma a maximizar o lucro de operação. Os autores destacam que o modelo do problema deve considerar o custo de operação das aeronaves quando operando em rotas que não preenchem a lotação total e, ao mesmo tempo, atender à demanda de passageiros em cada rota.

O modelo do Problema de Alocação de Frota como inicialmente implementado por Abara (1989) ignora a interdependência dos trechos das rotas aéreas na geração de receita, os quais podem ser compartilhados entre diferentes rotas, bem como o fenômeno de recaptura de passageiros entre os diferentes trechos (DUMAS; AITHNARD; SOUMIS, 2009). Estes autores propõem um cálculo iterativo da função objetivo levando em conta as perdas após uma alocação. Objetivam, com esta modificação, integrar ao modelo de Alocação de Frota a característica de rede presente no problema real. Uma consequência notada pelos autores é o aumento da complexidade computacional.

Unal et al. (2021) observam que o problema mais geral de gerenciamento de frota é dividido em problemas de agendamento de voo, alocação de frota, roteamento de aeronaves e alocação de tripulação, pois todos esses problemas são complexos e de larga escala, estando a resolução de um modelo que os abarque simultaneamente além das capacidades computacionais presentes. Os autores analisam a resolução do problema de alocação de frota como o da maximização de receita ou minimização do custo, considerando uma alocação de aeronaves para determinadas rotas e seguindo uma lógica de redução do número de pares de aeronaves associadas às rotas. O artigo explora a integração dos problemas de alocação de frota - fleet assignment problem - e roteamento de aeronaves - aircraft routing problem - os quais têm sido tradicionalmente abordados de forma separada devido à complexidade já mencionada.

Os dois problemas ,de alocação de frota e de roteamento de aeronaves, são combinados dando origem ao problema de agendamento de frota - fleet scheduling problem - tendo como meta minimizar o custo total ou maximizar a receita total dos pares aeronave e rota dentro de um período de planejamento definido. Uma conclusão retirada da análise do modelo do problema de agendamento de frota é que não é viável a

utilização de grandes aeronaves para pequenas rotas, pois seus custos operacionais excedem a receita de transporte de passageiros nestes trechos, sendo mais efetiva a utilização de pequenas aeronaves com maior frequência de voos para atender a demanda destas rotas.

2.4 Tendências na Solução de Problemas MILP (Mixed Integer Linear Programming) por Computação Paralela

Shinano et al. (2011) afirma que o procedimento de Branch and Bound evita a enumeração de todas as soluções potenciais de um problema, reduzindo a quantidade de casos a analisar na busca da solução ótima. Os autores analisaram uma extensão ao pacote SCIP do Python, denominada ParaSCIP, capaz de realizar a divisão do problema em máquinas de arquitetura paralela. Segundo os autores, a extensão paralela tem capacidade de utilizar até 10.000 núcleos de processamento simultaneamente.

O artigo estabelece que a extensão aloca dinamicamente subtarefas de resolução para cada nó de processamento de forma a garantir um aproveitamento ótimo dos recursos computacionais. Outro recurso importante é que a extensão evita a solução de sub problemas inviáveis. Esta característica, bem como as de armazenar o estado atual das soluções parciais e reiniciar o processamento quando encontrado um excesso de tempo de execução, implementam o balanceamento de carga da extensão.

Entretanto, para evitar a saturação dos canais de entrada e saída, apenas as informações dos estados iniciais criados para o particionamento do problema são armazenados em caso de reinicialização da solução.

Os autores do artigo utilizaram até 2048 núcleos de processamento para caracterizar o desempenho da extensão num problema exemplo derivado da estrutura de uma empresa alemã de transportes públicos. O problema exemplo foi resolvido em tempo hábil, validando a utilidade da extensão ParaSCIP na solução de problemas MILP. Ainda assim, os autores notaram o aumento do tempo de solução nas fases de abertura e redução da árvore de decisão, mesmo com o aumento do número de processadores.

Do ponto de vista do hardware usado para a solução de modelos de pesquisa operacional, Koch, Ralphs e Shinano (2012) pontuaram que, devido à integração de

números crescentes de núcleos de processamento nas CPUs dos principais fabricantes, haveria disponibilidade de sistemas de computação com milhões de núcleos. Esta tendência se verifica nos sistemas citados como os mais poderosos na lista TOP500 (2022), alguns dos quais se aproximam da dezena de milhões de núcleos.

Koch, Ralphs e Shinano (2012) afirmaram que o simples aumento de disponibilidade de núcleos não representa garantia direta de ganhos de performance na solução de problemas de otimização linear inteira. Outra limitação, de ordem prática, reside na eficiência energética de tais sistemas. Em 2012, o sistema de maior desempenho consumia 10 MW de potência. Em 2022, esta marca foi elevada para cerca de 21 MW (TOP500, 2022).

Além disso, a abordagem de subdivisão de problemas do algoritmo Branch and Bound não se beneficia de forma proporcional ao número de núcleos de processamento presentes, vez que ainda há a difícil tarefa de escolher estas subtarefas. Outra causa de redução do desempenho efetivo, citada pelos autores, é o consumo de processamento na transmissão e recepção de informação entre as operações de processamento, denominada *communication overhead.*

A abordagem de solução de problemas complexos por meio de unidades de processamento de uso geral, CPUs, tem sido complementada pela evolução do desempenho e adaptação de uso de processadores de uso gráfico, Graphical Processing Units - GPUs - para ambientes de computação geral. Boyer e El Baz (2013) concluem que é observada a tendência de uso de unidades de processamento gráfico por fabricantes de sistemas de computação para resolução de problemas computacionais em geral e com aplicação específica para problemas da pesquisa operacional. À época do artigo, as GPUs contavam com centenas de unidades de computação, sendo relativamente baratas e com o melhor eficiência energética que as CPUs tradicionais. Prosseguem fazendo uma apresentação focada na arquitetura CUDA da fabricante de GPUs NVidia, a qual combina software e hardware para permitir a implementação e a solução paralela de problemas utilizando GPUs.

Boyer e El Baz (2013) afirmam que as implementações de algoritmos Branch and Bound utilizando GPUs têm sido abordagens híbridas em que parte dos cálculos é realizada em CPUs tradicionais incluindo, obrigatoriamente, a eliminação de nós inviáveis. Observam que os ganhos devidos à computação em GPU são maiores quanto mais complexo o problema a ser resolvido indicando uma tendência no sentido de maior

eficiência computacional considerando a evolução das gerações de GPUs, as quais têm incorporado cada vez mais núcleos de computação e memória RAM dedicada.

3 MÉTODOS E TÉCNICAS DE PESQUISA

O presente trabalho utiliza o procedimento da Design Science Research (GOECKS et al., 2021) para a compreensão detalhada do problema de Fleet Assignment e elaboração de programas na linguagem de programação Python de forma a representar o problema e encontrar possíveis soluções ótimas para a alocação de aeronaves nas rotas de transporte aéreo de passageiros em território nacional.

Python é uma linguagem de programação desenvolvida nos anos 1990 pelo pesquisador holandês Guido van Rossum. Inicialmente concebida como uma linguagem para scripts, listas de instruções a serem interpretadas de forma não interativa. Evoluiu para um interpretador interativo de comandos (PYTHON, 2023).

Graças à filosofia que valoriza a compreensão dos algoritmos, Python conseguiu uma grande comunidade de usuários, a qual contribuiu para o uso e expansão da linguagem através de coleções de funções específicas denominadas pacotes. Funcionalmente, os pacotes são semelhantes às bibliotecas de funções de linguagens como C ou Fortran: permitem expandir as funcionalidades da linguagem.

Design Science Research é uma metodologia que busca a criação de artefatos para solucionar problemas ou contribuir com conhecimento inédito (GOECKS et al., 2021). Envolve o estudo e compreensão do problema seguido da aplicação de conhecimento. Posteriormente, é criado um artefato, uma ferramenta para a compreensão e solução do problema, que deverá ser validado experimentalmente. Tal procedimento tem obtido boa aceitação, visto que permite integrar teoria e prática na solução de problemas específicos.

Com base em atividades prévias relativas ao estudo do problema (FELIPE, 2022), buscar-se-á a identificação de artefatos, na forma de programas na linguagem Python, e a evolução dos contextos de solução no que tange à complexidade e migração da solução para um ambiente computacional paralelo visando solução viável para uso estratégico da informação no ambiente empresarial.

3.1 Tipologia e descrição geral dos métodos de pesquisa

Esta pesquisa é uma modelagem baseada no método Design Science Research.

Dresch, Lacerda e Antunes Jr. (2015) propõem doze etapas para a condução de uma pesquisa embasada nos princípios da Design Science Research, sumariamente:

3.1.1 Identificação do problema

O pesquisador deve identificar um problema relevante para pesquisa que represente adição ao conhecimento científico com vistas a preencher lacuna no entendimento, resolver um problema prático ou problemas assemelhados. Nesta fase, realiza-se a definição do problema, conduzindo à formalização de uma questão de pesquisa. Neste estudo, o aspecto principal foi o uso da computação paralela para reduzir o tempo de solução do modelo FAP.

3.1.2 Compreensão

Nesta etapa, o pesquisador deve obter o todos os detalhes possíveis sobre o problema. Os autores sugerem o emprego do pensamento sistêmico, de forma a integrar os aspectos de origem e efeitos do problema. Os resultados desta fase são a contextualização do problema e a definição do artefato a ser gerado para a solução do problema.

3.1.3 Revisão sistemática da literatura

De modo a executar adequadamente a etapa de compreensão, o pesquisador empreende consulta a bases de conhecimento em busca de problemas semelhantes e informações que assegurem a efetividade do artefato a ser desenvolvido.

3.1.4 Identificação dos artefatos e configuração das classes de problemas

Após a revisão sistemática da etapa anterior, o pesquisador terá uma compreensão de artefatos e problemas já desenvolvidos para a resolução de problemas semelhantes. A existência de artefatos já desenvolvidos permite ao pesquisador avançar na sugestão de novas possíveis soluções. Esta etapa garante que a pesquisa tenha relevância como uma nova contribuição ao conhecimento sobre o tema. Também, é possível definir os parâmetros de desempenho dos artefatos necessários à solução do problema. O artefato escolhido foi o modelo desenvolvido no trabalho de Felipe (2022).

3.1.5 Proposição de artefatos para resolver um problema específico

O pesquisador deve propor artefatos que atinjam o desempenho definido anteriormente levando em consideração aspectos práticos quanto à possibilidade de realização e adequação dos artefatos a problemas semelhantes. O conhecimento do contexto do problema permite evoluir a solução proposta dentro dos limites percebidos no passo anterior. Além dos programas em linguagem de programação Python, outros artefatos desenvolvidos neste trabalho são os arquivos de comandos para o gerenciador de processos do ambiente do supercomputador.

3.1.6 Projeto do artefato selecionado

Nesta etapa, um artefato será selecionado, dentro do conjunto de propostas que atendam às soluções do problema, e será formalmente projetado. Todas os componentes do artefato devem ser projetados, bem como descritos, de forma a atender a performance definida no estágio precedente e permitir a avaliação de desempenho.

3.1.7 Desenvolvimento do artefato

De posse do projeto do artefato, é realizada a implementação por meio de algoritmos de programação, protótipos, modelos ou outros meios adequados. Esta fase gera um artefato funcional e os procedimentos, ou heurística, de construção.

3.1.8 Avaliação do artefato

O artefato será utilizado e os resultados confrontados com os observados no estágio de compreensão do problema. As limitações do artefato, incluindo possíveis falhas no cumprimento das expectativas, deverão ser notadas pelo pesquisador. Esta fase foi realizada com as medidas de tempos de solução para variações do modelo em função do número de aeronaves escolhidas para alocação.

3.1.9 Esclarecimento do aprendizado obtido

De posse dos resultados da avaliação do artefato, o pesquisador deve destacar os resultados condizentes com a solução do problema bem como as limitações do artefato de modo a contribuir para a solução de problemas assemelhados.

3.1.10 Conclusões

A conclusão da pesquisa deve apresentar as decisões tomadas durante o estudo e os resultados obtidos. Também, são destacadas as limitações do estudo e sugeridos tópicos para pesquisas subsequentes. Dresch, Lacerda e Antunes Jr. (2015) ressaltam que, após esta e a etapa prévia, o pesquisador pode ter novas percepções sobre o problema, levando ao reinício do processo.

3.1.11 Generalização para uma classe de problemas

Nesta etapa o pesquisador generaliza o artefato gerado de forma que este possa ser aplicado à resolução de outros problemas, permitindo o reuso do conhecimento, conforme apresentado nos apêndices.

3.1.12 Comunicação dos resultados

É a etapa final do processo da Design Science Research, em que o conhecimento produzido é disseminado em periódicos, congressos e outros meios, alcançando o maior número de interessados.

3.4 Caracterização e descrição dos instrumentos de pesquisa

A descrição do modelo na linguagem Python utiliza o pacote PuLP, o qual permite traduzir a formulação matemática do modelo do FAP, consistindo na função objetivo e restrições das variáveis de decisão, para um formato geral de representação para programas de computador, denominado MPS (COIN-OR, 2009). PuLP permite salvar em armazenamento o modelo gerado por meio do método writeMPS(), o qual pode ser configurado para gravar o arquivo com nome definido, como ilustrado no Apêndice C. Com este recurso, é possível gerar o arquivo do modelo e resolvê-lo em linha de comando acionando diretamente o solver, de forma mais conveniente que a execução interativa do ambiente Jupyter Notebook. Jupyter Notebook é uma ferramenta de desenvolvimento compatível com as linguagens de programação Julia, Python e R. A codificação das instruções de programa é agrupada em blocos, ditos células. Cada célula pode ser executada em separado, permitindo a depuração do código de forma conveniente ao usuário. O ambiente também permite exportar os programas e as saídas de execução dos blocos em um arquivo, também denominado Jupyter Notebook. Via de regra, o arquivo .mps gerado é salvo na pasta de arquivos temporários do sistema operacional com um nome gerado automaticamente e, quando invocado o solver, este recebe implicitamente o arquivo como entrada de dados, explicitamente as diretivas de

controle adicionadas na lista de parâmetros do método getSolver() - ou do método solver() - e gera um arquivo de saída, também de natureza efêmera, com o mesmo nome do arquivo de entrada e terminação .sol. Estes arquivos temporários podem ser armazenados para uso ou análise posterior.

Seguindo os resultados do trabalho anterior de Felipe (2022), os dados coletados foram inseridos no modelo matemático do algoritmo do Problema de Alocação de Frota; a seguir, este modelo foi codificado na linguagem de programação Python e resolvido com o auxílio do pacote PuLP. Os recursos de processamento utilizados foram os ambientes Google Colab, nas versões gratuita e Pro; nós de processamentos do supercomputador do NPAD/UFRN; e um computador pessoal. A seguir, este modelo foi alterado em relação à distância mínima das rotas e demandas semanais.

3.4.1 - Modelo Matemático

O modelo, aperfeiçoado por Felipe (2022), visa maximar o lucro, Z, resultante da diferença entre a receita obtida da venda de passagens e os custos de operação para cada rota, resultando na função objetivo:

$$max\ Z\ = \sum_{h \in H} \sum_{j \in J} \sum_{a \in A} (T_{h,j} * P_{h,j,a} - C_{h,j,a} * D_{h,j} * S_a * F_{h,j,a})$$

Onde:

h : hub de origem dentro do conjunto de todos os hubs, H;

j : destino escolhido dentre os possíveis, J;

a : aeronave dentre as disponíveis na amostra A;

$T_{h,j}$: Tarifa de passagem entre o hub h e o destino $j,\ T \in \mathbb{R}$;

$P_{h,j,a}$: Número de passageiros transportados pela aeronave a do hub h ao destino $j,\ P_{h,j,a} \in \mathbb{Z}$;

$C_{h,j,a}$: CASK calculado da aeronave a para o trecho do hub h ao destino $j,\ C_{h,j,a} \in \mathbb{R}$;

$D_{h,j}$: Distância do trecho entre o hub h e o destino $j,\ D \in \mathbb{R}$;

S_a : Número total de assentos da aeronave $a,\ S_a \in \mathbb{Z}$;

$F_{h,j,a}$: Fluxo total da aeronave a do hub h ao destino $j,\ F_{h,j,a} \in \mathbb{Z}$.

A função objetivo calcula, para cada aeronave a e para cada trecho entre o hub h e destino j, a diferença entre receita, obtida do produto entre a tarifa do trecho e o

número de passageiros transportados por aeronave, e o custo de operação, produto entre o CASK específico da aeronave, a distância do trecho, a oferta total de assentos na aeronave e a quantidade total de passageiros transportados. Segundo Felipe (2022), o parâmetro CASK - Cost per Available Seat Kilometer - é a média de custo por assento e por quilômetro voado para cada aeronave.

O produto $S_a * F_{h,j,a}$ representa a oferta de assentos no trecho analisado para cada aeronave, enquanto $C_{h,j,a} * D_{h,j}$ é o custo de operação da aeronave no trecho.

Além da função objetivo, é necessário acrescentar as restrições dependentes do problema:

Restrição 1:

Para cada rota, a quantidade de passageiros transportados, considerando-se todas as aeronaves, deve atender à demanda daquela rota:

$$\sum_{a \in A} P_{h,j,a} = Q_{h,j} \ , \forall\, h \in H, \forall\, j \in J$$

Onde $Q_{h,j}$ é a demanda total da rota entre o hub h e o destino j, $Q_{h,j} \in \mathbb{Z}$.

Restrição 2:

$$P_{h,j,a} \leq S_a * F_{h,j,a} \ , \forall\, h \in H, \forall\, j \in J, \forall\, a \in A$$

Esta restrição impõe que, para cada rota, a oferta de assentos, produto $S_a * F_{h,j,a}$, tem que suprir a demanda de assentos, considerando todas as aeronaves disponíveis.

Restrição 3:

$$P_{h,j,a} \geq LF * S_a * F_{h,j,a} \ , \forall\, h \in H, \forall\, j \in J, \forall\, a \in A$$

Entretanto, complementando a restrição anterior, a oferta de assentos para cada rota não pode ser infinitamente superior à demanda. Para modular a restrição, introduz-se o fator LF, load factor, para limitar a oferta máxima de assentos em cada rota. Dessa forma, o valor de oferta de assentos para cada rota é limitado a $1/LF$ vezes a demanda da rota.

Restrição 4:

$$F_{h,j,a} \leq M * BIN1_{h,j,a} \ , \forall\, h \in H, \forall\, j \in J, \forall\, a \in A$$

Segundo esta restrição, cada aeronave a pode oferecer, no máximo, M vezes a capacidade de assentos necessários a cada fluxo total de passageiros por rota. M foi fixado em 1000 no modelo. $BIN1_{h,j,a}$ é uma variável binária associada a cada tipo de aeronave e rota. $BIN1_{h,j,a} \in \{0,1\}$.

Restrição 5:

$$R_a \geq D_j * BIN1_{h,j,a}, \forall h \in H, \forall j \in J, \forall a \in A$$

Para cada aeronave, esta só é viável para determinada rota se a distância da rota, D_j, for inferior ao alcance da aeronave em kilômetros, $R_a \in \mathbb{R}$.

Restrição 6:

$$F_{h,j,a} \leq M * BIN2_a, \forall h \in H, \forall j \in J, \forall a \in A$$

Esta restrição impede que seja incluído o fluxo de uma aeronave não incluída na simulação. A variável binária $BIN2_a \in \{0,1\}$ sinaliza a inclusão da aeronave.

Restrição 7:

$$\sum_{a \in A} BIN2_a \leq K$$

Esta última restrição é necessária para garantir que se utilize apenas o número de aeronaves escolhida para cada simulação, $K \in \mathbb{Z}$.

3.5 Procedimentos de coleta e de análise de dados

3.5.1 - Procedimento de coleta de dados

Os dados sobre a malha viária regional brasileira, incluindo os custos diretos, receitas de tarifas, estrutura da rede de trajetos, e oferta e demanda de assentos para os trechos a serem analisados, são de natureza secundária pois foram obtidos da base de dados do Sistema Hórus-SAC (UFSC, 2022), na forma de planilhas do programa Excel e formato CSV - comma-separated values - das quais são extraídos por intermédio de recursos da linguagem Python, por meio de uma sequência de comandos separados em células dentro do ambiente Jupyter Notebook. Este ambiente é um interpretador interativo de comandos da linguagem Python e aceita, também, comandos do sistema Linux subjacente. As instruções para uso do ambiente estão disponíveis na página de tutoriais do NPAD/UFRN.

Para acessar o ambiente, após a configuração inicial, é necessário criar um arquivo de comandos para informar ao programa gerenciador de carga slurm as características do ambiente de computação desejado, dentro das restrições de prioridade e disponibilidade de recursos. Para tanto, cria-se um arquivo de texto, exemplificado no Apêndice D, cujos comandos serão executados no conjunto de nós do supercomputador. A seguir, executa-se o comando sbatch seguido do nome do arquivo de texto. Será atribuído um número identificador à tarefa e criado um arquivo de log, ou registro de atividades, no mesmo diretório. Dependendo do nível de utilização dos recursos e da prioridade de execução dada ao usuário pela equipe do NPAD/UFRN, a tarefa contida pode ser executada ou negada, além de estar condicionada a uma fila de espera. O estado da tarefa pode ser acompanhado pelo comando squeue. Uma característica importante a ressaltar é que as tarefas têm um tempo máximo de execução limitado a 48 horas.

Neste trabalho, tendo em vista que o modelo foi verificado em estudos anteriores, buscou-se a redução do tempo de solução do problema de alocação de frota pela seleção de programas que fizessem uso dos recursos do processamento paralelo.

Por meio de adaptações ao código desenvolvido no trabalho de Felipe (2022), foi verificada a solução do problema dentro do ambiente interativo do Jupyter Notebook e, também, por intermédio de scripts de comandos para gerar arquivos contendo representações do problema para diferentes valores do parâmetro K no formato .mps. Estes arquivos de modelo foram resolvidos com o uso das versões de linha de comando dos solvers CBC (COIN-OR, 2009) e HiGHS (HUANGFU, HALL, 2018).Antes de serem executados no supercomputador, estes scripts Python e arquivos de comandos para o interpretador bash foram testados no ambiente online Google Colab e em um computador pessoal com sistema operacional Linux.

Em todos os códigos foram inseridos comandos para registro do tempo gasto na solução do modelo. Estes tempos foram usados para confecção de gráficos, tabelas e planilhas de dados permitindo melhor compreensão da forma de uso e das vantagens de cada programa analisado.

3.5.2 - Procedimento de análise de dados

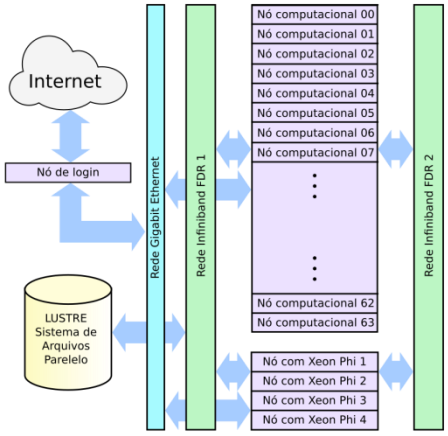
O modelo de programação linear inteira mista - MILP - gerado foi resolvido e avaliado por intermédio de algoritmos determinísticos e/ou heurísticos adaptados para o uso de recursos de computação paralela com foco na resolução das variações do problema em relação ao número de tipos de aeronaves a serem escolhidas, parâmetro K do modelo. O foco do trabalho foi a análise de viabilidade da resolução do Problema de Alocação de Frota em um ambiente de computação paralela, explorando o paralelismo com vistas à redução do tempo de computação das soluções, incluindo processadores e softwares com suporte à execução paralela, visto que a complexidade do problema torna o tempo de processamento excessivo em relação ao disponibilizado pelo ambiente Google Colab (FELIPE, 2022).

4 RESULTADOS EXPERIMENTAIS E ANÁLISE

A Figura 1 é uma diagrama representando a estrutura do supercomputador disponibilizado pelo NPAD/UFRN. O acesso ao sistema se dá por intermédio do nó de login, o qual é um computador que pode ser acessado por conexão segura, via protocolo SSH - Secure Socket Shell - permitindo a execução remota de comandos e a transferência de arquivos. A partir do nó de login as tarefas de solução do modelo FAP são transferidas, executadas e monitoradas nos nós computacionais.

A execução do código foi testada em sessão remota em conjuntos de nós computacionais do supercomputador do Núcleo de Processamento de Alto Desempenho - NPAD - da Universidade Federal do Rio Grande do Norte, dentro do ambiente Jupyter Notebook. Cada nó compreende dois processadores Intel Xeon E5-2698 v3, totalizando 32 núcleos computacionais ("cores"), 64 threads com uso do recurso Intel Hyper-Threading Technology, com 128 GB de memória RAM DDR4 2133 (NPAD, 2023) e executando Python versão 3.10.5-gnu8 sob ambiente Linux. Neste trabalho foram usados ambientes paralelos de um nó - 32 núcleos - e quatro nós - 128 núcleos.

Figura 1: Recursos de hardware do NPAD/UFRN.

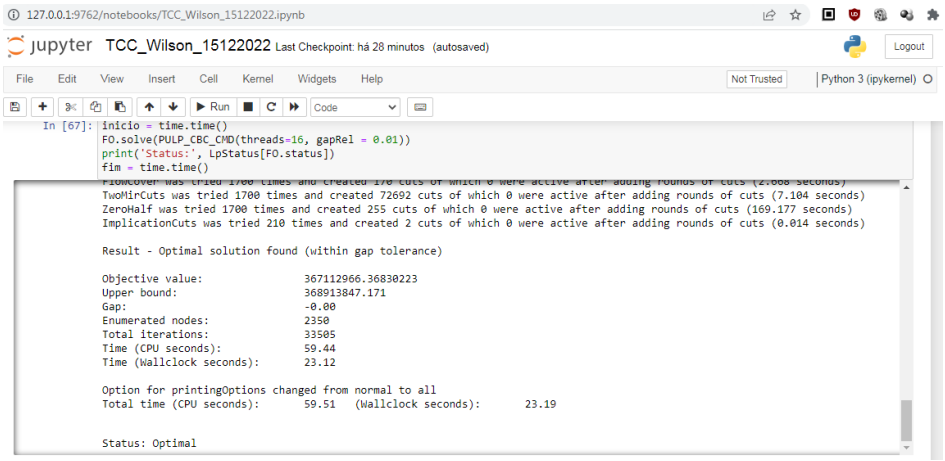


Fonte: NPAD, 2023.

Os solvers testados foram o Gurobi (GUROBI, 2023), o CBC, versão 2.10.3, do pacote PuLP, e versões do solver HiGHS (HUANGFU, HALL, 2018).

A Figura 2 ilustra a execução interativa da tarefa de solução do modelo dentro do ambiente Jupyter Notebook, acessado remotamente numa máquina conectada ao ambiente do supercomputador. Os dados da sessão de cálculo trafegam criptografados, dentro de um encapsulamento do protocolo SSH ("túnel"). No exemplo, o solver CBC foi acionado com 16 processos paralelos e com um relaxamento de 1% em relação à solução exata. O modelo FAP é o de Felipe (2022), com K = 2.

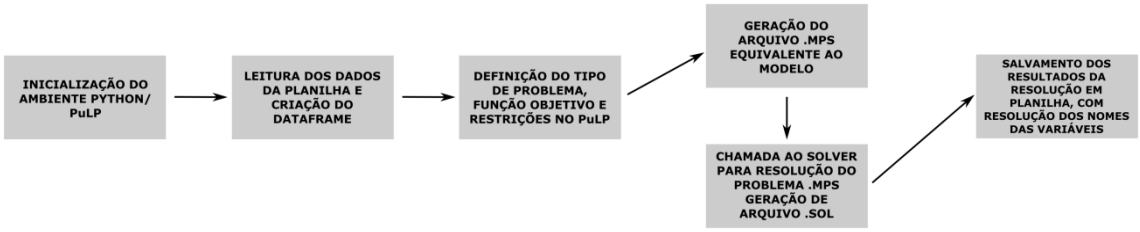
Figura 2: Execução do Jupyter Notebook no NPAD/UFRN.



O pacote PuLP apresenta um mecanismo para construir um modelo de programação linear e resolvê-lo por meio de um programa externo, denominado solver, que implementa o algoritmo de solução do problema de programação linear. Alguns solvers, tais como os do projeto COIN-OR, permitem selecionar o algoritmo de solução.

Após a seleção do solver e configuração opcional de parâmetros, é gerado um arquivo temporário no formato .mps, contendo informações dos valores das variáveis de decisão e as restrições do modelo. Este arquivo é resolvido pelo solver, com os resultados armazenados em um arquivo temporário de extensão .sol. A Figura 3 ilustra este processo.

Figura 3: Fluxo de trabalho no ambiente Python/PuLP para resolução de problemas de pesquisa operacional.



O formato .mps tem limitações quanto à representação numérica dos valores, por usar notação científica com doze casas decimais e expoentes de dois dígitos, e não armazena os nomes das variáveis, cabendo ao ambiente PuLP restaurar esta informação por meio das variáveis internas do modelo. É um formato textual reconhecido por vários solvers e permite comunicar a essência das relações entre as variáveis do modelo.

Exemplos de solvers disponíveis por meio do PuLP são: Gurobi, CPLEX, estes dois comerciais; GLPK, um pacote de Software Livre; os solvers do projeto COIN-OR, também de Software Livre, CLP e CBC, e o solver HiGHS, software livre distribuído sob a licença MIT e desenvolvido por pesquisadores da Universidade de Edimburgo.

No ambiente Python online do Google Colab Pro, estão disponíveis os solvers Gurobi , os do projeto COIN-OR, o solver HiGHS e outros. Dentro do Google Colab, na versão gratuita, o processador é limitado a duas linhas de execução - threads - e 12 GB de RAM, assim como o tempo de execução dentro do ambiente Jupyter Notebook. Uma forma de reduzir estas restrições é assinar o serviço Google Colab Pro, onde é possível configurar o ambiente para uma máquina contendo um processador de 12 threads, 80 GB de memória RAM e uma GPU de alto desempenho. Entretanto devido à política de cota de execução do serviço Google Colab Pro, e à complexidade do modelo gerado, este também é inadequado para as dimensões do problema, visto que as unidades de execução são rapidamente exauridas na situação em que não há relaxamento da solução.

No ambiente do supercomputador do NPAD/UFRN, estão disponíveis o solver Gurobi e os solvers do projeto COIN-OR. Embora disponível, o solver Gurobi é limitado a 200 variáveis de decisão, sendo inadequado para o problema em apreço. O acesso a outros solvers no ambiente do cluster depende da disponibilidade de um pacote instalável por meio do programa pip, ou de artifícios, como o uso de executáveis pré-compilados estaticamente. É possível executar um programa Python dentro de uma sessão remota do Jupyter Notebook, segundo instruções disponibilizadas pela instituição, ou adaptando o programa para execução em linha de comando, vide Apêndice C. A execução das tarefas é intermediada pelo gerenciador de carga de trabalho slurm (SLURM, 2021), o qual interpreta um arquivo texto de configuração com as características do conjunto de nós computacionais e tarefas, como exemplificado no Apêndice D, e transfere a execução para os nós do supercomputador.

Devido às limitações de licença do solver Gurobi, as tentativas iniciais de solução foram feitas com o solver COIN-OR CBC, configurando a execução em múltiplas threads, K = 2 e com relaxamentos de 1% ou 0,1%, no ambiente Google Colab.

No supercomputador do NPAD/UFRN, com relaxamento de 0,1% e utilizando números variáveis de threads, de 4 A 12, verificou-se que o solver CBC produziu resultados de função objetivo semelhantes em tempos semelhantes, indicando não haver ganho de performance no paralelismo implementado por este programa. Esse resultado corrobora observações dos artigos de Perumalla & Alam (2021) e Kock et al (2012) de que não há grande ganho de performance na implementação do algoritmo branch and cut em máquinas paralelas, pois a subdivisão de tarefas é sequencial. Contradiz, entretanto, o observado por ferramentas de monitoração de carga dos processadores quando da execução do programa Python por linha de comando ou no ambiente Jupyter Notebook. Esta monitoração só foi possível no ambiente de execução local com o emprego dos comandos top e xosview do Linux. A utilização de um monitor de desempenho, tal como o xosview, permite acompanhar visualmente o uso dos núcleos do processador, indicando a execução paralela do programa. A Figura 4 apresenta uma janela do programa xosview com a indicação gráfica do uso dos processadores ao longo do tempo. No Google Colab gratuito, e no supercomputador, não foi encontrado um recurso semelhante para avaliar se todos os núcleos de computação foram acionados. No Google Colab Pro, é possível inferir que, devido ao aumento da taxa de consumo de unidades de computação quando se usam múltiplas threads, ocorre paralelismo. Teste

posteriores revelaram que algumas versões do solver CBC não aceitam a ativação do paralelismo e não geram erro de invocação, o que pode ter comprometido estes resultados preliminares.

Figura 4: Monitor de recursos xosview acompanhando a execução em múltiplas threads em um processador AMD Ryzen 6800H.

Tendo em vista a limitação de tempo imposta pelo ambiente Google Colab para execução do código de resolução do modelo do problema sem relaxamento, o código em Python foi adaptado para execução em linha de comando (Apêndice B) em um equipamento com processador Intel Core i7-2670QM, 16 GB de memória RAM DDR3 1600, sob ambiente Debian Linux kernel 5.10.140-1 executando Python versão 3.9.2-3. O solver utilizado foi o CBC MILP Solver do pacote Linux coinor-cbc, versão 2.10.5+ds1-3, com 8 threads.

Em um primeiro teste, o processamento foi interrompido após cerca de uma semana de execução contínua, devido à exaustão da memória RAM, inicialmente de 8 GB. Posteriormente, foi iniciada nova execução, com o dobro de RAM, sendo que o processamento dessa segunda sessão se estendeu por 37 dias, 14 horas e 16 minutos, convergindo a um valor da função objetivo de US$ 368.661.174,96. O resultado obtido para K = 2, selecionando as aeronaves "Jato_1" e "Turboelice_2", com lucro de US$ 368.661.174,96 é coerente com os valores obtidos por Felipe (2022).

Considerando apenas o número de núcleos disponíveis no cluster do NPAD/UFRN, sem consideração pela diferença de frequência e admitindo ganhos de performance lineares em relação ao número de CPUs, a solução do problema tratado em Felipe (2022), com solver CBC, em quatro nós do supercomputador do NPAD, usando 64 threads, poderia ocorrer em 4 dias e 17 horas, além do tempo limite de execução das tarefas no cluster, de 2 dias. Considerando a razão de frequências de 3,6/2,2 entre os

processadores do cluster e o utilizado no experimento, a estimativa seria de 2 dias, 21 horas.

Foram feitas novas tentativas, no mesmo computador local, com processador Core i7-2670QM, 16 GB de memória RAM DDR3 1600, usando o solver CBC via linha de comando, mas com relaxamento de 1%. Estes testes exibiram o caráter não-determinístico da solução relaxada, em que diferentes árvores de decisão levam a valores diversos do lucro obtido:

Figura 5: Variação dos valores da função objetivo para múltiplas execuções do solver CBC com relaxamento.

```
user@debian:~/Documents/24DEZ2022$ grep 'K = ' TCC_Wilson_06FEB2023.py
        f'{DISTANCIA[j]}, CASK = {x:,.6f}')
CASK = DADOS4['Cask']
#CASK = CASK.values.tolist()
K = 6
user@debian:~/Documents/24DEZ2022$ grep -i 'relgap = ' TCC_Wilson_06FEB2023.py
user@debian:~/Documents/24DEZ2022$ grep -i 'gaprel = ' TCC_Wilson_06FEB2023.py
FO.solve(PULP_CBC_CMD(threads=8, gapRel = 0.01))
user@debian:~/Documents/24DEZ2022$ awk '{print $7}' lucros.txt  | wc -l
33
user@debian:~/Documents/24DEZ2022$ awk '{print $7}' lucros.txt
368625582.41
368647420.44
368631637.69
368643057.47
368637805.35
368641986.17
368640033.06
368647069.62
368641024.46
368630930.64
368641618.88
368626214.30
368647722.43
368639877.80
368616256.07
368639278.39
368619308.68
368636864.41
368623589.54
368646428.16
368633682.63
368640651.49
368645740.94
368642831.86
368631234.59
368627630.49
368634020.31
368643832.20
368623086.54
368630313.77
368646030.64
368635135.50
368648751.66
user@debian:~/Documents/24DEZ2022$
```

Para as análises posteriores, o modelo matemático foi modificado para os casos em que a distância das rotas é maior que 250 km, com demanda semanal mínima de 27 passageiros:

Figura 6: Modificações no modelo FAP.

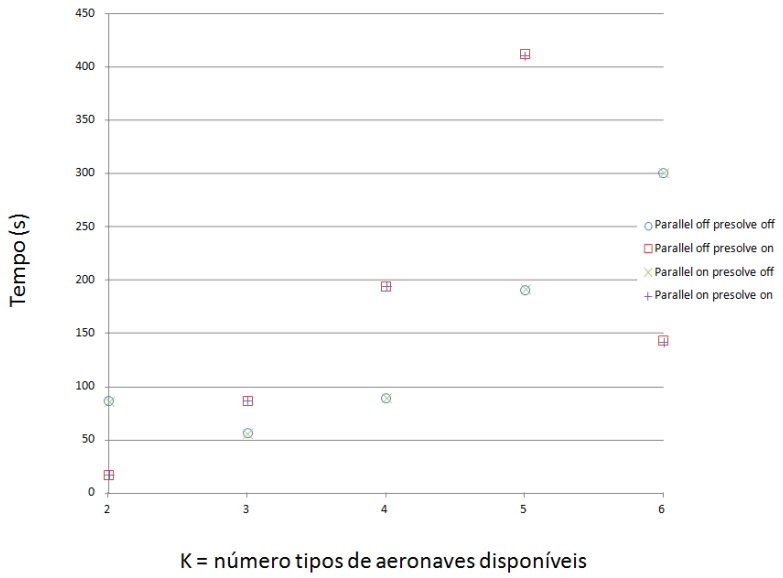
```
# Exclude distances shorter than 250 km
Data_BRRegAv = Data_BRRegAv[Data_BRRegAv.Distance >= 250]

# Exclude Demand lower than 27 (3 flights of 9 seats Turboprop_1)
Data_BRRegAv = Data_BRRegAv[Data_BRRegAv.Demand >= 27]
```

Com estas alterações, o número de variáveis de decisão foi reduzido de 12786 para 10248 e o número de restrições de 22720 para 18208. Visto que estas modificações afetam diretamente os dados importados da planilha de oferta e demanda de passageiros da malha viária, não há necessidade de reformular o modelo em relação à função objetivo e conjunto de restrições.

Ademais, o solver HiGHS foi escolhido para caracterização de uso, em parte pelo excelente desempenho aparente na solução de problemas de programação linear, inclusive com implementação paralela do algoritmo dual simplex (HIGHS, 2023). Este programa aceita opções de linha de comando, tornando-se prático executá-lo por meio de loops do interpretador de comandos (Apêndice F). Todavia, observou-se que, para problemas de programação inteira mista, o solver utiliza o algoritmo Interior Point Method por padrão, o qual, por enquanto, não utiliza recursos de paralelismo para solução dos problemas (HIGHS, 2023). Este fato foi verificado experimentalmente, com a execução do solver para os modelos gerados pelo código do Apêndice C, para K = 2,..,6, por meio de um loop (Apêndice F):

Figura 7: Tempos de execução do solver HiGHS em função do parâmetro K e de opções de linha de comando.



Como indicado na Figura 7, os tempos de execução do solver independem do valor da opção --parallel no caso MILP, mas são afetados pela opção --presolve, a qual foi prejudicial à performance do solver, exceto para K = 6. A operação presolve tem como objetivo eliminar linhas, colunas e elementos não-nulos, visto que o solver explora a esparsidade do modelo (HALL, 2019).

Foram obtidos os seguintes valores de alocação de aeronaves e lucro operacional para o modelo revisado deste estudo:

Tabela 1: Resultados obtidos com o solver HiGHS no cluster NPAD/UFRN.

K	Aeronaves escolhidas	Lucro (US$)	Tempo de cálculo (s)
2	Turboelice_2, Jato_2	36454800,78	181,59
3	Turboelice_2, Jato_1, Jato_3	36991057,42	154,99
4	Turboelice_2, Turboelice_3, Jato_1, Jato_3	37308115,45	210,36
5	Turboelice_2, Turboelice_3, Jato_1, Jato_2, Jato_3	37370216,21	136,60
6	Turboelice_1,Turboelice_2, Turboelice_3, Jato_1, Jato_2, Jato_3	37390389,31	90,85

Os valores de tempos de cálculo da Tabela 1, onde o modelo foi resolvido dentro do ambiente Python/PuLP (Apêndice G), não são coerentes com os valores correspondentes da Figura 7. Esta discrepância ocorre porque o sentido de otimização padrão do solver HiGHS é a minimização, executada quando da execução em linha de comando. No ambiente Python/PuLP, o sentido é ajustado para a minimização do oposto da função objetivo, ou seja, max(Z) = min(-Z), de forma a obter a maximização buscada.

A seguir, o solver CBC foi utilizado para resolver o problema, numa configuração de 64 threads (Apêndice H):

Tabela 2: Resultados obtidos com o solver CBC no cluster NPAD/UFRN.

K	Aeronaves escolhidas	Lucro (US$)	Tempo de cálculo (s)
2	Turboelice_2, Jato_2	36456029,44	20353,21
3	-	-	Mais de 48 h
4	Turboelice_2, Turboelice_3, Jato_1, Jato_3	37308935,64	72,60
5	Turboelice_2, Turboelice_3, Jato_1, Jato_2, Jato_3	37370497,50	42,59
6	Turboelice_1,Turboelice_2, Turboelice_3, Jato_1, Jato_2, Jato_3	37391668,65	21,66

Os resultados da Tabela 2 acima foram obtidos dentro do ambiente Python virtual do Jupyter Notebook no supercomputador do NPAD/UFRN. Destaca-se que, para o solver CBC sem relaxamento, os tempos de resolução do problema para K = 4, 5 e 6 são menores que os observados para o solver HiGHS. Estes tempos de processamento não podem ser comparados diretamente aos do problema abordado por Felipe (2022), diante da diferença de número de variáveis e restrições e por não usar o relaxamento da solução. Para K = 3, com 64 e 128 threads, a solução não foi alcançada dentro do limite de 48 horas de processamento para as tarefas no supercomputador.

Como discutido anteriormente em Felipe (2022), o intervalo de tolerância dado como parâmetro de relaxamento para o solver CBC, parâmetro GapRel, está relacionado à qualidade da solução obtida: quanto menor o intervalo de relaxamento mais próximo da solução real deve estar a solução calculada. Entretanto, para a situação limite de relaxamento nulo, o tempo de computação da solução pode ser excessivo. Como

verificado experimentalmente para o modelo FAP anterior sem restrições de alcance mínimo e demanda mínima semanal, para K = 2, o tempo de solução chegou a mais de 37 dias para o solver CBC operando com 8 threads. Em relação ao tempo esperado para diferentes arquiteturas e características de processadores, há que se considerar o número total de processos que podem ser lançados sem queda do desempenho devido à intercomunicação de processos. Outro fator a considerar é a diferença de frequência dos processadores em uso. No caso do cluster do NPAD, os processadores Intel Xeon E5-2698 v3 operam a 3,6 GHz, enquanto o processador usado no teste anteriormente referido opera a 2,2 GHz. Outras características que afetam o desempenho são os diferentes tipos e níveis de memória interna do processador e memória RAM do sistema.

Tendo em vista que o solver HiGHS não utiliza processamento paralelo para a solução de problemas MILP, buscou-se uma alternativa para compreender o comportamento esperado de um programa capaz de aproveitar o recurso. Gonçalves-e-Silva (2013) aponta duas medidas de desempenho de aplicações com suporte a processamento paralelo: o Speedup e a Eficiência. A primeira é definida como a razão entre o tempo de execução do programa de forma serial, T_s, e o tempo de execução paralela, T_p; é uma medida de quão mais veloz é o processamento da versão paralela do programa. A Eficiência mede o aproveitamento dos recursos de processamento; é definida como a razão entre o Speedup obtido e o número de processadores usado. Como modelo de programa capaz de aproveitar os recursos de programação paralela, foi escolhido o compactador pbzip2, o qual acrescenta ao popular compactador de arquivo bzip2 o suporte a processamento paralelo e permite controle conveniente do número de processos usados para a tarefa de compressão (PANKRATIUS, JANNESARI e TICHY, 2009) através da opção -p seguida do número de processos de compressão. Variando o argumento de -p de 1 a 128, num subcluster de 4 nós do supercomputador, e executando-se 33 vezes a tarefa de compressão de quatro arquivos de texto de 133, 541, 901 e 1,7 GB. O compressor bzip2, por padrão, segmenta o arquivo a comprimir em trechos de 900 KB. Os tempos de compressão foram armazenados e calculada a média aritmética e o desvio padrão das amostras. Considerando o desvio padrão inferior a 5% em todos os valores, a média aritmética foi utilizada para os gráficos de Speedup ($S = T_s/T_p$) e Eficiência (S/n).

As Figuras 8, 9, 10 e 11, a seguir, apresentam os resultados de Speedup e eficiência do processamento paralelo na compressão dos arquivos de texto de 133, 541,

901 e 1,7 GB, respectivamente. Foram testadas configurações de 2 nós, com até 32 threads de compressão, e 4 nós, com até 128 threads.

Figura 8: Speedup da compressão de arquivos de texto com pbzip2 em função do número de processos para 2 nós do supercomputador.

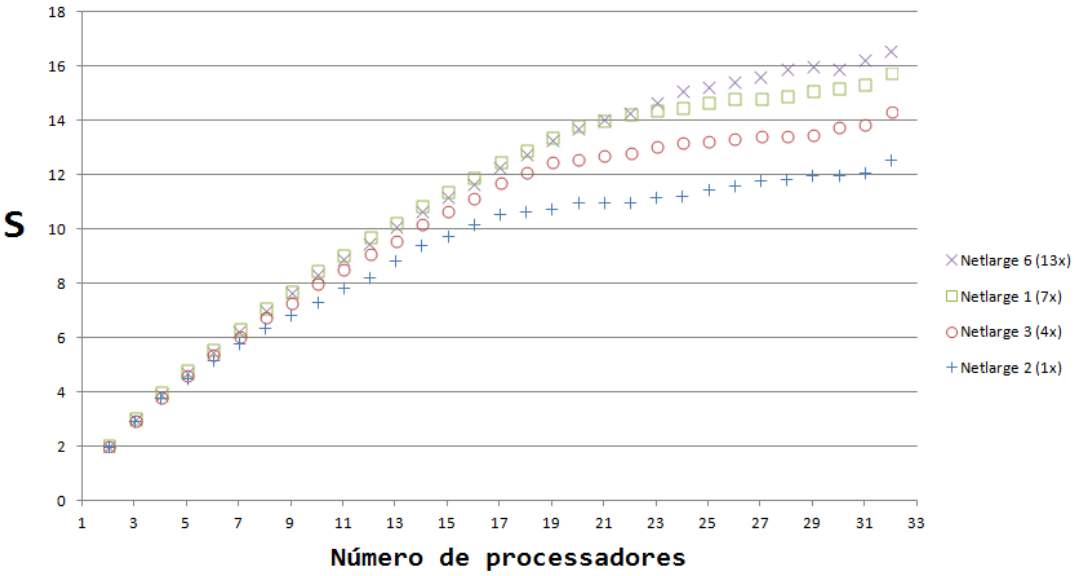
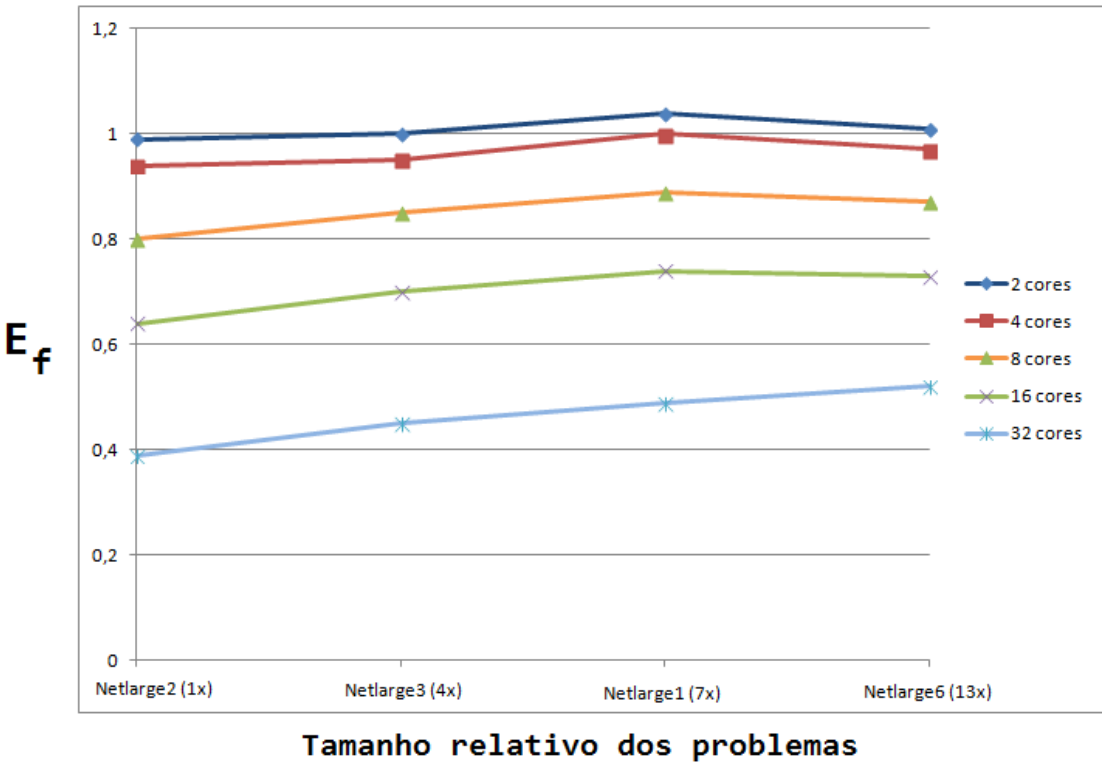


Figura 9: Eficiência do processo de compressão em relação ao número de threads e tamanho relativo dos problemas para 2 nós do supercomputador.



Para a configuração de dois nós e até 32 threads, de acordo com a Figura 8, observa-se a relação de aumento linear do Speedup até 8 threads. Até este ponto, o ganho de desempenho do processamento paralelo é semelhante para os quatro arquivos, os quais guardam uma relação de tamanho de 1, 4, 7 e 13 vezes, aproximadamente. A partir de 9 threads, ocorre divergência nos ganhos de tempo de compressão do arquivo em favor dos arquivos maiores. Em relação à eficiência, vista na Figura 9, é inversamente proporcional ao número de threads usadas na compressão e proporcional ao tamanho relativo dos arquivos.

Para os resultados das Figuras 10 e 11, a seguir, o procedimento foi repetido em uma configuração com 4 nós e até 128 threads como parâmetro para o programa pbzip2.

Figura 10: Speedup da compressão de arquivos de texto com pbzip2 em função do número de processos para 4 nós do supercomputador.

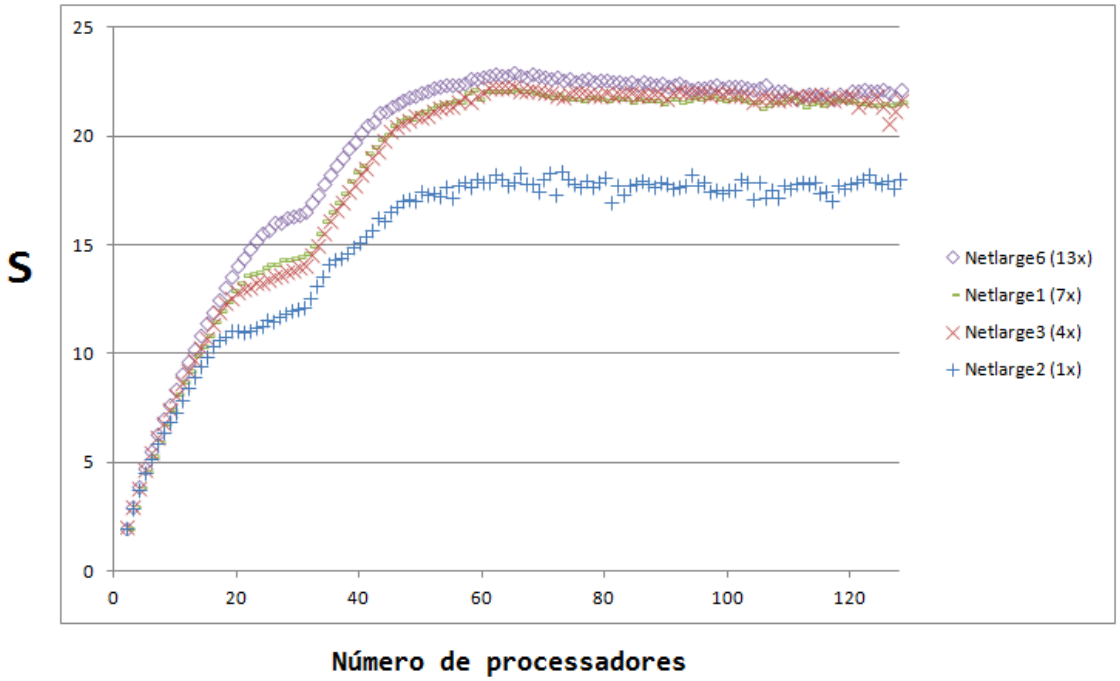
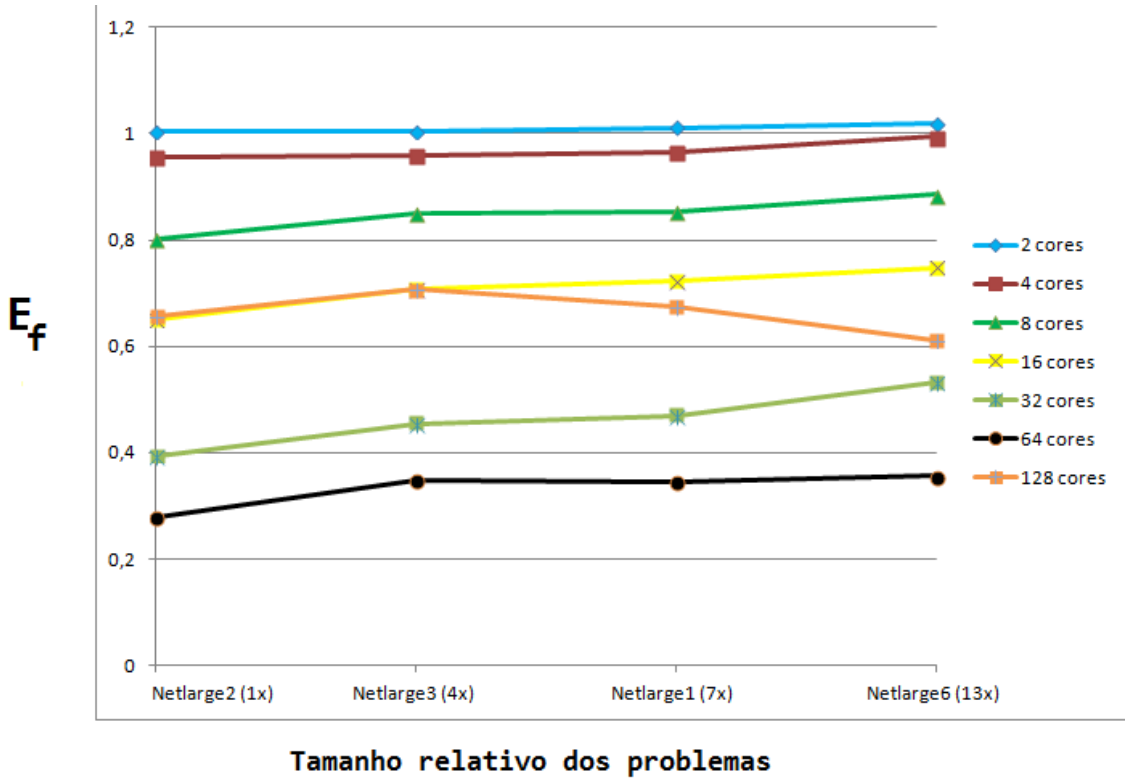


Figura 11: Eficiência do processo de compressão em relação ao número de threads e tamanho relativo dos problemas para 4 nós do supercomputador.



Nos gráficos das Figuras 10 e 11 observa-se, respectivamente, o aumento do desempenho da velocidade de compressão, mas com pontos de redução dos ganhos próximos a 16, 32 e até próximo de 64 threads, quando ocorre uma estagnação. O efeito é mais pronunciado para o problema menor. A eficiência, entretanto, apresenta um comportamento mais complexo: a eficiência para 128 threads é superior às para 32 e 64 threads e semelhante à para 16 threads em relação aos três problemas menores.

Para o solver CBC não foi possível coletar estas informações devido ao tempo necessário para resolver o problema MILP para todos os valores de K, mas seria importante caracterizar este comportamento em estudo posterior.

5 CONCLUSÕES

O objetivo geral do presente trabalho foi alcançado: seguindo a metodologia Design Science Research, foram cumpridas as etapas elencadas e gerados artefatos para aplicação à solução do problema de Fleet Allocation na malha viária regional brasileira nas rotas superiores a 250 km e com demanda semanal superior a 27 passageiros. Estes artefatos foram validados, obtendo-se conhecimento sobre a operação e uso de um ambiente de processamento paralelo, bem como o uso dos programas de computador necessários à extração de informação relevantes dos modelos. Além da linguagem Python, os artefatos fazem uso dos recursos de um interpretador de comandos do sistema Linux. Foram obtidos conhecimentos sobre o uso do processamento paralelo para viabilizar a análise de problemas complexos e sobre as limitações destes recursos dos pontos de vista de hardware e software.

Ao longo do estudo, algumas dificuldades foram observadas:

Nem sempre a documentação dos solvers é objetiva e clara em relação à sintaxe das opções disponíveis, cabendo a pesquisa nos sites dos projetos e em mecanismos de busca. Mesmo de posse dessa informação, a dinâmica do ciclo de vida dos programas pode torná-las obsoletas ou mesmo incorretas entre duas versões da aplicação.

O desempenho de serviços de computação em nuvem, como o Google Colab, é variável, dependendo de fatores alheios ao controle do usuário, mesmo na opção paga. Uma das consequências é a variação dos tempos de processamento dos problemas.

O uso de uma ambiente de computação paralela distribuída exige conhecimento de detalhes da arquitetura dos equipamentos, para minimizar a penalidade devida à comunicação de resultados entre os processadores e fluxo de dados entre nós.

O processo de solução de um problema num ambiente multiprocessado pode ser otimizado pelo teste do algoritmo e da implementação em uma máquina local, visto que o processamento no supercomputador é realizado em lote e obedece a fila e políticas de prioridade de alocação de recursos, não sendo conveniente para execução interativa.

Seria útil disponibilizar uma interface de monitoramento, semelhante ao comando Linux top, para verificar em tempo real o uso dos núcleos dos nós de processamento.

A complexidade do problema FAP analisado não é linear em relação ao número de tipos de aeronaves disponíveis, K. Os resultados de tempo de execução do solver CBC sem relaxamento sugerem que, mesmo quando usadas múltiplas threads de execução, os tempos de processamento deste solver não são adequados para a resolução deste problema.

O solver CBC não está instalado no ambiente do cluster. As tentativas para usar uma versão compilada a partir dos fontes falharam por falta da biblioteca libcholmod.so.3. As versões estáticas, disponíveis no site do projeto COIN-OR, bem como versões pré-compiladas extraídas de pacotes .rpm (via rpm2cpio), também não funcionaram, por incompatibilidade com a biblioteca libbz2. A variável de ambiente LD_PRELOAD não é exportada pelo script de comandos para o slurm e os compiladores GCC e Intel disponíveis no nó de login não geram executáveis estáticos. Ademais, as versões mais recentes do solver CBC, quando compiladas dos fontes, não aceitam o parâmetro threads; a última versão com suporte é de maio de 2022. A instalação de versões nativas à distribuição Linux do cluster permitiria executar os programas como scripts Python, sem necessidade de interação com o usuário, permitindo avaliar o desempenho do PuLP em conjunto com o solver.

Outros solvers deverão ser analisados em trabalhos futuros, bem como outros meios, independentes de Python e PuLP, para construir e resolver o modelo FAP. Uma questão em aberto é o desempenho do solver MIP com suporte a processamento paralelo do projeto HiGHS, a ser lançado até o final de 2023. Também, seria importante avaliar o impacto das contribuições dos estudantes da UFRN ao código-fonte do HiGHS.

REFERÊNCIAS

ABARA, J. Applying Integer Linear Programming to the Fleet Assignment Problem. **INTERFACES** 19: 4 (pp. 20-28) July-August 1989.

Ajayi, T., Thomas, C., Schaefer, A. The Gap Function: Evaluating Integer Programming Models over Multiple Right-Hand Sides. **Operations Research** Vol. 70, No. 2. 2021. https://doi.org/10.1287/opre.2020.2003

APIOLA, M., & SUTINEN, E. . Design science research for learning software engineering and computational thinking: Four cases. **Computer Applications in Engineering Education**, 29(1), 83–101. January 2021. https://doi.org/10.1002/cae.22291

ARENALES, M., ARMENTANO, V., MORABITO, R., YANASSE, H. **Pesquisa Operacional**. Rio de Janeiro: Elsevier, 2011.

BETTINI, H. F. Um Retrato da Aviação Regional no Brasil. **Revista de Literatura dos Transportes**. Vol. 1. Num. 1. pp. 46-65. 2007.

BOYER, V., & EL BAZ, D. Recent advances on GPU computing in operations research. Proceedings - **IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum**, IPDPSW 2013, 1778–1787. 20-24 may 2013. https://doi.org/10.1109/IPDPSW.2013.45

BOYER, V., EL BAZ, D., & SALAZAR-AGUILAR, M. A. GPU computing applied to linear and mixed-integer programming. In **Advances in GPU Research and Practice** (pp. 247–271). 2017. https://doi.org/10.1016/B978-0-12-803738-6.00010-0

BRASIL. Agência Nacional de Aviação Civil - ANAC. Anuário do Transporte Aéreo 2021. Sumário Executivo. Disponível em: https://www.gov.br/anac/pt-br/assuntos/dados-e-estatisticas/mercado-de-transporte-aereo/anuario-do-transporte-aereo/2021.zip Acesso em: 18 jan. 2023.

COIN-OR. How to import and export models in PuLP. 2009. Disponível em https://coin-or.github.io/pulp/guides/how_to_export_models.html Acesso em 05 fev. 2023.

Dantzig, George B. Linear Programming . **Operations Research**. Vol. 50, No. 1. 2002. pp. 42-47. https://doi.org/10.1287/opre.50.1.42.17798

DRESCH, A., LACERDA, D.P., ANTUNES JR., J.A.V. Proposal for the conduct of design science research. In: **Design Science Research - A Method for Science and Technology Advancement** (pp. 117–127). Springer Cham. 2015. https://doi.org/10.1007/978-3-319-07374-3

DUMAS, J., AITHNARD, F., & SOUMIS, F. Improving the objective function of the fleet assignment problem. **Transportation Research Part B: Methodological**, 43(4), 466–475. May 2009. https://doi.org/10.1016/j.trb.2008.08.005

FELIPE, J. R. **Modelagem do Problema da Alocação de Frota Aplicado à Aviação Regional A Partir da Redução da Complexidade Computacional**. 2022. 102 f. Trabalho de Conclusão de Curso (Graduação em Engenharia Aeronáutica). Faculdade UnB Gama - FGA. Universidade de Brasília. Brasília.

Gass, S. I., Assad, A. A. An Annotated Timeline of Operations Research: An Informal History. Boston: Springer Science. 2005.

GOECKS, L. S., SOUZA, M. de, LIBRELATO, T. P., & TRENTO, L. R. Design Science Research in practice: review of applications in Industrial Engineering. **Gestão & Produção**, 28(4). 2021. https://doi.org/10.1590/1806-9649-2021v28e5811

Gonçalves-e-Silva, K. **Análise de Escalabilidade de uma Implementação Paralela do Simulated Annealing Acoplado**. Orientador: Prof. Dr. Samuel Xavier de Souza. Co-orientador: Prof. Dr. Daniel Aloise. 2013. Dissertação (Mestrado). Programa de Pós-Graduação em Engenharia Elétrica e de Computação. Centro de Tecnologia. Universidade Federal do Rio Grande do Norte. Natal, RN, 2013. Disponível em: https://repositorio.ufrn.br/bitstream/123456789/15471/1/KayoGS_DISSERT.pdf Acesso em 12 jun. 2023.

GOOGLE. Google Colaboratory. 2023. Disponível em: https://colab.research.google.com/?utm_source=scs-index . Acesso em 05 fev. 2023.

GUROBI. Gurobi Optimizer. Disponível em https://www.gurobi.com/solutions/gurobi-optimizer/ Acesso em 23 jan. 2023.

Hall, J. HiGHS: A High-performance Linear Optimizer . **lecture**. in INFORMS Annual Meeting. Seattle. 21 October 2019. Disponível em https://www.maths.ed.ac.uk/hall/INFORMS-HiGHS19/INFORMS-HiGHS19.pdf Acesso em 09 jul. 2023.

HIGHS. HiGHS Documentation. Disponível em https://ergo-code.github.io/HiGHS/dev/ . Acesso em 10 jul. 2023.

Hillier, F. S., Lieberman, G. J. Introdução à Pesquisa Operacional. 8ª Edição. São Paulo: McGraw-Hill, 2006.

Huangfu, Q., Hall, J. A. J. Parallelizing the dual revised simplex method. Mathematical Programming Computation. 10 (1). 2018. pp. 119-142. DOI: https://link.springer.com/article/10.1007/s12532-017-0130-5

LEURQUIN, P., AVELAR, M. M. Os desafios jurídicos e econômicos da aviação regional no Brasil. **Revista Brasileira de Políticas Públicas**. v. 6, n. 2. pp. 204-220. 2016. https://doi.org/10.5102/rbpp.v6i2.3979

Pankratius, V., Jannesari, A., Tichy, W. F. Parallelizing Bzip2: A Case Study in Multicore Software Engineering. **IEEE Software**. Vol. 26, Issue 6, 2009. Disponível em: https://ieeexplore-ieee-org.ez54.periodicos.capes.gov.br/stamp/stamp.jsp?tp=&arnumber=5287014 Acesso: 01 jul. 2023.

PYTHON. General Python FAQ. Disponível em https://docs.python.org/3/faq/general.html Acesso em 10 jan. 2023.

KOCH, T., RALPHS, T., & SHINANO, Y. Could we use a million cores to solve an integer program? **Mathematical Methods of Operations Research**, 76(1), 67–93. 2012. https://doi.org/10.1007/s00186-012-0390-9

NVIDIA. GeForce RTX 4090 - Beyond Fast. Disponível em https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/rtx-4090/ Acesso em: 23 jan. 2023.

PERUMALLA, K., & ALAM, M. Design Considerations for GPU-based Mixed Integer Programming on Parallel Computing Platforms. **ACM International Conference Proceeding Series**. August 2021. https://doi.org/10.1145/3458744.3473366

SHERALI,H.D., BISH, E.K., ZHU, X. Airline fleet assignment concepts, models, and algorithms. **European Journal of Operational Research**. Volume 172, Issue 1, pp. 1 - 30. July 2006. https://doi.org/10.1016/j.ejor.2005.01.056

SHINANO, Y., ACHTERBERG, T., BERTHOLD, T., HEINZ, S., & KOCH, T. ParaSCIP: A Parallel Extension of SCIP. In **Competence in High Performance Computing 2010** (pp. 135–148). 2011. https://doi.org/10.1007/978-3-642-24025-6_12

SLURM. Slurm Workload Manager. 2021. Disponível em https://slurm.schedmd.com/overview.html . Acesso em 10 fevereiro 2023.

Taha, H. A. Pesquisa Operacional. 8ª Edição. São Paulo: Pearson. 2008.

TOP500. NOVEMBER 2022 Release. Disponível em: https://top500.org/lists/top500/2022/11/ Acesso em 23 jan. 2023.

UFRN. Universidade Federal do Rio Grande do Norte. Núcleo de Processamento de Alto Desempenho. Informações de Hardware do Supercomputador da UFRN. Disponível em: http://npad.ufrn.br/hardware.php Acesso em: 23 jan. 2023.

UFSC. Universidade Federal de Santa Catarina. Laboratório de Transportes e Logística. Sistema Hórus - SAC. Matriz Origem Destino. 2022. Disponível em: https://horus.labtrans.ufsc.br/gerencial/?auth=s#MatrizOd Acesso em: 12 fev. 2023.

UNAL, Y. Z., SEVKLI, M., UYSAL, O., & TURKYILMAZ, A. A new approach to fleet assignment and aircraft routing problems. **Transportation Research Procedia**, 59, 67–75. 2021. https://doi.org/10.1016/j.trpro.2021.11.098

UNB. Universidade de Brasília. Conselho de Ensino, Pesquisa e Extensão. Resolução nº 128/2021 de 10 dez. 2021. Disponível em

https://deg.unb.br/images/legislacao/resolucao_cepe_128_2021.pdf Acesso em: 14 fev. 2023.

WINSTON, W.L. **Operations Research** - Applications and Algorithms. 4th Edition. Belmont: Brooks/Cole -Thomson Learning, 2004.

APÊNDICES

APÊNDICE A - CÓDIGO EM PYTHON PARA EXECUÇÃO NO AMBIENTE JUPYTER NOTEBOOK NO GOOGLE COLAB E NO SUPERCOMPUTADOR DO NPAD/UFRN - VERSÃO INICIAL.

```
%reset
!pip install --upgrade pulp
!pip install pandas
!pip install numpy
!pip install gurobipy

from pulp import *
import pandas as pd
import numpy as np
from pandas import ExcelWriter
import time
import gurobipy as gp

#inicio = time.time()
#Function para calcular o preço da passagem
def Tarifa(dist):
    return np.exp(-0.73953 * np.log(dist) + 4.44446)*dist

#Function para calcular o CASK TF e CASK TP
def Cask(dist, seats, tipo):
    if tipo == 'TF':
        return np.exp(1.728912 - 0.40453 * np.log(dist) - 0.35671 * np.log(seats))
    elif tipo == 'TP':
        return np.exp(-0.481549 - 0.099212 * np.log(dist) - 0.356708 * np.log(seats))

# Leitura dos dados sobre aeronaves

DADOS3 = pd.read_excel('Dados_TCC_2022.xlsx',sheet_name = 'Aeronaves')
DADOS3.dropna(inplace=True)
DADOS3.reset_index(drop=True, inplace=True)
DADOS3.drop_duplicates(inplace=True)

AERONAVE = DADOS3['AERONAVE'].astype(str)
N_ANV = len(AERONAVE)

ASSENTOS = DADOS3['ASSENTOS'].astype(int)

ALCANCE = DADOS3['ALCANCE'].astype(int)

TIPO = DADOS3['TIPO'].astype(str)

#Leitura dos dados para construção da malha

DADOS1 = pd.read_excel('Dados_TCC_2022.xlsx',sheet_name = 'Dados')
DADOS1.dropna(inplace=True)
DADOS1.reset_index(drop=True, inplace=True)

HUB = DADOS1['Hub'].astype(int)
HUBN = HUB.drop_duplicates()
numero_HUB = len(HUBN)
#HUB = HUB.values.tolist()

SPOKE = DADOS1['Spoke'].astype(int)
```

```
SPOKEN = SPOKE.drop_duplicates()
numero_SPOKE = len(SPOKEN)
#SPOKE = SPOKE.values.tolist()

DEMANDA = round(DADOS1['Demanda']).astype(int)
#DEMANDA = DEMANDA.values.tolist()

DISTANCIA = round(DADOS1['Distancia']).astype(int)
# #DISTANCIA = DISTANCIA.values.tolist()

# # Definição do número de vertices (edges)
N = len(DISTANCIA)

# Cálculo de TARIFA por rota

DADOS2 = pd.DataFrame(columns=['Passagem'])
for i in range(N):
    x = Tarifa(DISTANCIA[i])
    DADOS2 = DADOS2.append({'Passagem': x}, ignore_index=True)

    print((
          f"Hub({HUB[i]})-Spoke({SPOKE[i]}): Distancia = "
          f"{DISTANCIA[i]}, Passagem = {DADOS2.Passagem[i]:,.2f}"
          ))

DADOS2.reset_index(drop=True, inplace=True)
PASSAGEM = DADOS2['Passagem']
#PASSAGEM = PASSAGEM.values.tolist()


# Cálculo do CASK por aeronave e rota

DADOS4 = pd.DataFrame(columns=['Cask'])
for i in range(N_ANV):
    print(f'CASK da Aeronave {AERONAVE[i]}')
    for j in range(N):
        if DISTANCIA[j] > ALCANCE[i]:
            x = 10
        else:
            x = Cask(DISTANCIA[j], ASSENTOS[i], TIPO[i])
        DADOS4 = DADOS4.append({'Cask': x}, ignore_index=True)
        print(f'Hub({HUB[j]})-Spoke({SPOKE[j]}): Distancia = '
          f'{DISTANCIA[j]}, CASK = {x:,.6f}')

DADOS4.reset_index(drop=True, inplace=True)
CASK = DADOS4['Cask']
#CASK = CASK.values.tolist()

# Cálculo do Load Factor por distância
# Estabelecendo o load factor por aeronave na rota
# A oferta (Fluxo * ASSENTOS) não pode superar demasiadamente a demanda atendida (PAS
SAGEIROS)
# Para rotas longas (DISTANCIA > 1300km), aceita-
se oferta 4 vezes maior que demanda (LF = 0.25)
# Para rotas curtas (DISTANCIA <= 1300), aceita-
se oferta 2 vezes maior que demanda (LF = 0.50)

LF = []
for j in range(N):
    if DISTANCIA[j] <= 1300:
        LF.append(0.50)
```

```
    else:
        LF.append(0.25)

M = 1000
K = 2

FO = LpProblem('Malha_PAN',LpMaximize)

FLUXO = LpVariable.dicts('Fluxos',[(a,j,HUB[j],SPOKE[j]) for j in range(N) for a in r
ange(N_ANV)],0,None,cat='Integer')
PASSAGEIROS = LpVariable.dicts('PASSAGEIROS',[(a,j,HUB[j],SPOKE[j]) for j in range(N)
 for a in range(N_ANV)],0,None,cat='Integer')
BINARIO1 = LpVariable.dicts('Bin1',[(a,j) for j in range(N) for a in range(N_ANV)],0,
None,cat='Binary')
BINARIO2 = LpVariable.dicts('Bin2',[(a) for a in range(N_ANV)],0,None,cat='Binary')

#Função Objetivo (4260 termos - 8520 colunas do tableau)
FO += lpSum(PASSAGEM[j]*PASSAGEIROS[(a,j,HUB[j],SPOKE[j])] -
 CASK[((N*a)+j)]*ASSENTOS[a]*DISTANCIA[j]*FLUXO[(a,j,HUB[j],SPOKE[j])] for a in range
(N_ANV) for j in range(N))

# R.1 - Num de pax transportado igual a demanda
# A demanda de cada rota pode ser atendida por mais de um tipo de aeronave
# TESTAR: restrição de <=
# 710 restrições - restrição igualdade resulta no dobro de desigualdades, i.e., 1420

for j in range(N):
    FO += lpSum(PASSAGEIROS[(a,j,HUB[j],SPOKE[j])] for a in range(N_ANV)) -
 DEMANDA[j] == 0


# R.2 -
 Oferta de assentos por Fluxo maior ou igual à demanda expressa em PASSAGEIROS (R.1)
# A oferta (Fluxo * ASSENTOS) pode vir de mais de um tipo de aeronave para atender à
demanda
# Em cada rota, a oferta precisa ser maior que a demanda atendida (PASSAGEIROS) para
cada aeronave
# 4260 restrições

for a in range(N_ANV):
    for j in range(N):
        FO += PASSAGEIROS[(a,j,HUB[j],SPOKE[j])] -
 ASSENTOS[a]*FLUXO[(a,j,HUB[j],SPOKE[j])] <= 0


# R.2.1 - Demanda atendida por Fluxo de cada aeronave na rota (NOVA)
# A oferta (Fluxo * ASSENTOS) não pode superar demasiadamente a demanda atendida (PAS
SAGEIROS)
# A partir do load factor (LF) por rota
# Estabelece um limite máximo que de uma oferta 1/LF vezes maior que a demanda
# 4260 restrições

for a in range(N_ANV):
    for j in range(N):
        FO += PASSAGEIROS[(a,j,HUB[j],SPOKE[j])] -
 LF[j]*ASSENTOS[a]*FLUXO[(a,j,HUB[j],SPOKE[j])] >= 0

# R.3 - Definindo o Fluxo máximo de cada modelo por rota
# Um modelo de aeronave só pode atender uma demanda de M vezes a capacidade de ASSENT
OS
# 4260 restrições
```

```
for a in range(N_ANV):
    for j in range(N):
        FO += FLUXO[(a,j,HUB[j],SPOKE[j])] - M*BINARIO1[(a,j)] <= 0


# R.4 - Número máximo de modelos de aeronaves por rota (EXCLUIDA. REDUNDANTE com R.7)
# Esta restrição não é necessária, pois o modelo não vai incluir mais do que K por R.
7
#for j in range(N):
#    FO += lpSum(BINARIO1[(a,j)] for a in range(N_ANV)) - K <= 0

# R.5 - Alcance de aeronave (MODIFICADA)
#Anteriormente, estava excluindo TP quando existia DISTANCIA > ALCANCE
# 4260 restrições

for a in range(N_ANV):
    for j in range(N):
        FO += ALCANCE[a] - DISTANCIA[j]*BINARIO1[(a,j)] >= 0

# R.6 - Ajustando o Fluxo somente para os modelos incluídos (MODIFICADA)
# Impede que haja fluxo de modelo de aeronave não incluído
# 4260 restrições

for a in range(N_ANV):
    for j in range(N):
        FO += FLUXO[(a,j,HUB[j],SPOKE[j])] - M*BINARIO2[(a)] <= 0

# R.7 - Num Max de modelos de aeronave que podem ser incluídos por rota
# 710 restrições

for j in range(N):
    FO += lpSum(BINARIO2[(a)] for a in range(N_ANV)) - K <= 0


inicio = time.time()
# GUROBIPY:
#solver = GUROBI(timeLimit = 300)
#solver.buildSolverModel(FO)
#solver.callSolver(FO)
#solver.findSolutionValues(FO)
##############################
# PuLP:
#solver = getSolver('GUROBI')
#FO.solve(GUROBI())
FO.solve(PULP_CBC_CMD(threads=16, gapRel = 0.01))
#solver = getSolver(solver=CBC, path='/home/vrrcelestino/PPA/16DEZ2022/COIN-
OR/dist/bin/cbc', keepFiles=1, mip=1, msg=0, fracGap=0.001, maxSeconds=900, threads=1
0)
#FO.solve()
print('Status:', LpStatus[FO.status])
fim = time.time()

fim - inicio

lucro = value(FO.objective)
print('\n')
print('O lucro total é de US$ %.2f' % lucro)

#Alocando Dados finais em uma planilha Excel, para melhor visualização.
lista_index = []
```

```
lista_profit = []
lista_origem = []
lista_destino = []
lista_passageiros = []
lista_aero = []
lista_fluxo = []

for j in range(N):
    for a in range(N_ANV):
        profit = PASSAGEM[j]*PASSAGEIROS[(a,j,HUB[j],SPOKE[j])].varValue -
 CASK[((N*a)+j)]*ASSENTOS[a]*DISTANCIA[j]*FLUXO[(a,j,HUB[j],SPOKE[j])].varValue
        if profit != 0:
            lista_profit.append(profit)
            lista_index.append(j)
            lista_origem.append(HUB[j])
            lista_destino.append(SPOKE[j])
            lista_passageiros.append(PASSAGEIROS[(a,j,HUB[j],SPOKE[j])].varValue)
            lista_aero.append(AERONAVE[a])
            lista_fluxo.append(FLUXO[(a,j,HUB[j],SPOKE[j])].varValue)



Excel_1 = {'INDEX':lista_index,'ORIGEM': lista_origem, 'DESTINO':lista_destino,'PASSA
GEIROS':lista_passageiros,'AERONAVE':lista_aero, 'FLUXO':lista_fluxo, 'LUCRO ROTA':li
sta_profit}
DF_EXCEL = pd.DataFrame(Excel_1, columns=['ORIGEM','DESTINO','PASSAGEIROS','AERONAVE'
,'FLUXO','LUCRO ROTA'])
Excel_2 = pd.DataFrame({'Característica':['Lucro (US$)','Status'],'Valor':['%.2f' % l
ucro, LpStatus[FO.status]]})

with ExcelWriter("Resultados_TCC_Wilson_modelo_k2_gap0.01_09122022.xlsx") as writer:
    #DATA_HUB.to_excel(writer, sheet_name = 'Rota Hub-Hub', index=False)
    #DADOS1.to_excel(writer, sheet_name = 'Rotas', index=False)
    #MALHA_DEM_AJ.to_excel(writer, sheet_name = 'Malha aju com zeros', index=False)
    #MALHA_DEM_AJ_NOZEROS.to_excel(writer, sheet_name = 'Malha aju sem zeros', index=
False)
    DADOS1.to_excel(writer, sheet_name = 'MALHA DEFINITIVA', index=False)
    DF_EXCEL.to_excel(writer, sheet_name = 'RESULTADOS', index=False)
    Excel_2.to_excel(writer, sheet_name='Config Result Simulação', index=False)
```

APÊNDICE B - CÓDIGO EM PYTHON MODIFICADO PARA EXECUÇÃO EM LINHA DE COMANDO.

```
from pulp import *
import pandas as pd
import numpy as np
from pandas import ExcelWriter
import time

def Tarifa(dist):
    return np.exp(-0.73953 * np.log(dist) + 4.44446)*dist

#Function para calcular o CASK TF e CASK TP
def Cask(dist, seats, tipo):
    if tipo == 'TF':
        return np.exp(1.728912 - 0.40453 * np.log(dist) - 0.35671 * np.log(seats))
    elif tipo == 'TP':
        return np.exp(-0.481549 - 0.099212 * np.log(dist) - 0.356708 * np.log(seats))

# Leitura dos dados sobre aeronaves

DADOS3 = pd.read_excel('Dados_TCC_2022.xlsx',sheet_name = 'Aeronaves')
DADOS3.dropna(inplace=True)
DADOS3.reset_index(drop=True, inplace=True)
DADOS3.drop_duplicates(inplace=True)

AERONAVE = DADOS3['AERONAVE'].astype(str)
N_ANV = len(AERONAVE)

ASSENTOS = DADOS3['ASSENTOS'].astype(int)

ALCANCE = DADOS3['ALCANCE'].astype(int)

TIPO = DADOS3['TIPO'].astype(str)


#DADOS3


#Leitura dos dados para construção da malha

DADOS1 = pd.read_excel('Dados_TCC_2022.xlsx',sheet_name = 'Dados')
DADOS1.dropna(inplace=True)
DADOS1.reset_index(drop=True, inplace=True)

HUB = DADOS1['Hub'].astype(int)
HUBN = HUB.drop_duplicates()
numero_HUB = len(HUBN)
#HUB = HUB.values.tolist()

SPOKE = DADOS1['Spoke'].astype(int)
SPOKEN = SPOKE.drop_duplicates()
numero_SPOKE = len(SPOKEN)
#SPOKE = SPOKE.values.tolist()

DEMANDA = round(DADOS1['Demanda']).astype(int)
#DEMANDA = DEMANDA.values.tolist()

DISTANCIA = round(DADOS1['Distancia']).astype(int)
```

```
# #DISTANCIA = DISTANCIA.values.tolist()

# # Definição do número de vertices (edges)
N = len(DISTANCIA)


# Cálculo de TARIFA por rota

DADOS2 = pd.DataFrame(columns=['Passagem'])
for i in range(N):
    x = Tarifa(DISTANCIA[i])
    DADOS2 = DADOS2.append({'Passagem': x}, ignore_index=True)

    print((
          f"Hub({HUB[i]})-Spoke({SPOKE[i]}): Distancia = "
          f"{DISTANCIA[i]}, Passagem = {DADOS2.Passagem[i]:,.2f}"
          ))

DADOS2.reset_index(drop=True, inplace=True)
PASSAGEM = DADOS2['Passagem']
#PASSAGEM = PASSAGEM.values.tolist()

# Cálculo do CASK por aeronave e rota

DADOS4 = pd.DataFrame(columns=['Cask'])
for i in range(N_ANV):
    print(f'CASK da Aeronave {AERONAVE[i]}')
    for j in range(N):
        if DISTANCIA[j] > ALCANCE[i]:
            x = 10
        else:
            x = Cask(DISTANCIA[j], ASSENTOS[i], TIPO[i])
        DADOS4 = DADOS4.append({'Cask': x}, ignore_index=True)
        print(f'Hub({HUB[j]})-Spoke({SPOKE[j]}): Distancia = '
          f'{DISTANCIA[j]}, CASK = {x:,.6f}')

DADOS4.reset_index(drop=True, inplace=True)
CASK = DADOS4['Cask']
#CASK = CASK.values.tolist()


# Cálculo do Load Factor por distância
# Estabelecendo o load factor por aeronave na rota
# A oferta (Fluxo * ASSENTOS) não pode superar demasiadamente a demanda atendida (PAS
SAGEIROS)
# Para rotas longas (DISTANCIA > 1300km), aceita-
se oferta 4 vezes maior que demanda (LF = 0.25)
# Para rotas curtas (DISTANCIA <= 1300), aceita-
se oferta 2 vezes maior que demanda (LF = 0.50)

LF = []
for j in range(N):
    if DISTANCIA[j] <= 1300:
        LF.append(0.50)
    else:
        LF.append(0.25)


# In[15]:
```

```
M = 1000
K = 2

FO = LpProblem('Malha_PAN',LpMaximize)

FLUXO = LpVariable.dicts('Fluxos',[(a,j,HUB[j],SPOKE[j]) for j in range(N) for a in r
ange(N_ANV)],0,None,cat='Integer')
PASSAGEIROS = LpVariable.dicts('PASSAGEIROS',[(a,j,HUB[j],SPOKE[j]) for j in range(N)
 for a in range(N_ANV)],0,None,cat='Integer')
BINARIO1 = LpVariable.dicts('Bin1',[(a,j) for j in range(N) for a in range(N_ANV)],0,
None,cat='Binary')
BINARIO2 = LpVariable.dicts('Bin2',[(a) for a in range(N_ANV)],0,None,cat='Binary')


#Função Objetivo (4260 termos - 8520 colunas do tableau)
FO += lpSum(PASSAGEM[j]*PASSAGEIROS[(a,j,HUB[j],SPOKE[j])] -
 CASK[((N*a)+j)]*ASSENTOS[a]*DISTANCIA[j]*FLUXO[(a,j,HUB[j],SPOKE[j])] for a in range
(N_ANV) for j in range(N))



# R.1 - Num de pax transportado igual a demanda
# A demanda de cada rota pode ser atendida por mais de um tipo de aeronave
# TESTAR: restrição de <=
# 710 restrições - restrição igualdade resulta no dobro de desigualdades, i.e., 1420

for j in range(N):
    FO += lpSum(PASSAGEIROS[(a,j,HUB[j],SPOKE[j])] for a in range(N_ANV)) -
 DEMANDA[j] == 0



# R.2 -
 Oferta de assentos por Fluxo maior ou igual à demanda expressa em PASSAGEIROS (R.1)
# A oferta (Fluxo * ASSENTOS) pode vir de mais de um tipo de aeronave para atender à
demanda
# Em cada rota, a oferta precisa ser maior que a demanda atendida (PASSAGEIROS) para
cada aeronave
# 4260 restrições

for a in range(N_ANV):
    for j in range(N):
        FO += PASSAGEIROS[(a,j,HUB[j],SPOKE[j])] -
 ASSENTOS[a]*FLUXO[(a,j,HUB[j],SPOKE[j])] <= 0



# R.2.1 - Demanda atendida por Fluxo de cada aeronave na rota (NOVA)
# A oferta (Fluxo * ASSENTOS) não pode superar demasiadamente a demanda atendida (PAS
SAGEIROS)
# A partir do load factor (LF) por rota
# Estabelece um limite máximo que de uma oferta 1/LF vezes maior que a demanda
# 4260 restrições

for a in range(N_ANV):
    for j in range(N):
        FO += PASSAGEIROS[(a,j,HUB[j],SPOKE[j])] -
 LF[j]*ASSENTOS[a]*FLUXO[(a,j,HUB[j],SPOKE[j])] >= 0
```

```
# R.3 - Definindo o Fluxo máximo de cada modelo por rota
# Um modelo de aeronave só pode atender uma demanda de M vezes a capacidade de ASSENT
OS
# 4260 restrições

for a in range(N_ANV):
    for j in range(N):
        FO += FLUXO[(a,j,HUB[j],SPOKE[j])] - M*BINARIO1[(a,j)] <= 0



# R.4 - Número máximo de modelos de aeronaves por rota (EXCLUIDA. REDUNDANTE com R.7)
# Esta restrição não é necessária, pois o modelo não vai incluir mais do que K por R.
7
#for j in range(N):
#    FO += lpSum(BINARIO1[(a,j)] for a in range(N_ANV)) - K <= 0


# R.5 - Alcance de aeronave (MODIFICADA)
#Anteriormente, estava excluindo TP quando existia DISTANCIA > ALCANCE
# 4260 restrições

for a in range(N_ANV):
    for j in range(N):
        FO += ALCANCE[a] - DISTANCIA[j]*BINARIO1[(a,j)] >= 0



# R.6 - Ajustando o Fluxo somente para os modelos incluídos (MODIFICADA)
# Impede que haja fluxo de modelo de aeronave não incluído
# 4260 restrições

for a in range(N_ANV):
    for j in range(N):
        FO += FLUXO[(a,j,HUB[j],SPOKE[j])] - M*BINARIO2[(a)] <= 0



# R.7 - Num Max de modelos de aeronave que podem ser incluídos por rota
# 710 restrições

for j in range(N):
    FO += lpSum(BINARIO2[(a)] for a in range(N_ANV)) - K <= 0


# Total de restrições = 2 * 710 + 5 * 4260 = 22.720
#
# (Obs.: não contou dobrado a restrição de igualdade <= e >=)


inicio = time.time()
# GUROBIPY:
#solver = GUROBI(timeLimit = 300)
#solver.buildSolverModel(FO)
#solver.callSolver(FO)
#solver.findSolutionValues(FO)
##############################
# PuLP:
#solver = getSolver('GUROBI')
#FO.solve(GUROBI())
FO.solve(PULP_CBC_CMD(threads=8, gapRel = None))
```

```
#solver = getSolver(solver=CBC, path='/home/vrrcelestino/PPA/16DEZ2022/COIN-
OR/dist/bin/cbc', keepFiles=1, mip=1, msg=0, fracGap=0.001, maxSeconds=900, threads=1
0)
#FO.solve()
print('Status:', LpStatus[FO.status])
fim = time.time()

fim - inicio

lucro = value(FO.objective)
print('\n')
print('O lucro total é de US$ %.2f' % lucro)


#Alocando Dados finais em uma planilha Excel, para melhor visualização.
lista_index = []
lista_profit = []
lista_origem = []
lista_destino = []
lista_passageiros = []
lista_aero = []
lista_fluxo = []

for j in range(N):
    for a in range(N_ANV):
        profit = PASSAGEM[j]*PASSAGEIROS[(a,j,HUB[j],SPOKE[j])].varValue -
 CASK[((N*a)+j)]*ASSENTOS[a]*DISTANCIA[j]*FLUXO[(a,j,HUB[j],SPOKE[j])].varValue
        if profit != 0:
            lista_profit.append(profit)
            lista_index.append(j)
            lista_origem.append(HUB[j])
            lista_destino.append(SPOKE[j])
            lista_passageiros.append(PASSAGEIROS[(a,j,HUB[j],SPOKE[j])].varValue)
            lista_aero.append(AERONAVE[a])
            lista_fluxo.append(FLUXO[(a,j,HUB[j],SPOKE[j])].varValue)



Excel_1 = {'INDEX':lista_index,'ORIGEM': lista_origem, 'DESTINO':lista_destino,'PASSA
GEIROS':lista_passageiros,'AERONAVE':lista_aero, 'FLUXO':lista_fluxo, 'LUCRO ROTA':li
sta_profit}
DF_EXCEL = pd.DataFrame(Excel_1, columns=['ORIGEM','DESTINO','PASSAGEIROS','AERONAVE'
,'FLUXO','LUCRO ROTA'])
Excel_2 = pd.DataFrame({'Característica':['Lucro (US$)','Status'],'Valor':['%.2f' % l
ucro, LpStatus[FO.status]]})

with ExcelWriter("Resultados_TCC_Wilson_modelo_k2_gapNone_8threads_24122022.xlsx") as
 writer:
    #DATA_HUB.to_excel(writer, sheet_name = 'Rota Hub-Hub', index=False)
    #DADOS1.to_excel(writer, sheet_name = 'Rotas', index=False)
    #MALHA_DEM_AJ.to_excel(writer, sheet_name = 'Malha aju com zeros', index=False)
    #MALHA_DEM_AJ_NOZEROS.to_excel(writer, sheet_name = 'Malha aju sem zeros', index=
False)
    DADOS1.to_excel(writer, sheet_name = 'MALHA DEFINITIVA', index=False)
    DF_EXCEL.to_excel(writer, sheet_name = 'RESULTADOS', index=False)
    Excel_2.to_excel(writer, sheet_name='Config Result Simulação', index=False)
```

APÊNDICE C - CÓDIGO EM PYTHON MODIFICADO PARA GERAÇÃO DOS MODELOS EM FORMATO MPS EM FUNÇÃO DO PARÂMETRO K

```
# Fleet Assignment Problem (FAP) - Brazilian Regional Aviation
This notebook applies a version of FAP to optimize the Brazilian regional avation
potential network, modeled and solved as a Mixed Integer Linear Programming (MILP)
problem.
## Section 1 - Initialization
from pulp import *
import pandas as pd
import numpy as np
Data_BRRegAv_complete = pd.read_csv('SAC_Horus_2019.csv',delimiter=';')
Data_BRRegAv = Data_BRRegAv_complete[Data_BRRegAv_complete['Demand_Selected_Routes']
!= '0']
Data_BRRegAv.reset_index(drop=True, inplace=True)
Data_BRRegAv = Data_BRRegAv.iloc[:,3:]
Data_BRRegAv = Data_BRRegAv.drop(['Annual_Demand_2019'], axis=1)
Data_BRRegAv.columns = ['Hub','Spoke','Distance','Demand']
Data_BRRegAv['Distance'] =
Data_BRRegAv.loc[:,'Distance'].str.replace(',','.').astype(float)
Data_BRRegAv['Demand'] =
Data_BRRegAv.loc[:,'Demand'].str.replace(',','.').astype(float)
#Transform distance into integer
Data_BRRegAv.Distance = round(Data_BRRegAv.Distance).astype(int)
# Transform annual into integer weekly demand
# 52 weeks - two ways demand
Data_BRRegAv.Demand = round(Data_BRRegAv.Demand / 52 / 2).astype(int)
# Exclude distances shorter than 250 km
Data_BRRegAv = Data_BRRegAv[Data_BRRegAv.Distance >= 250]
# Exclude Demand lower than 27 (3 flights of 9 seats Turboprop_1)
Data_BRRegAv = Data_BRRegAv[Data_BRRegAv.Demand >= 27]
#Function to calculate ticket fare
def ticket_fare(dist):
    return np.exp(-0.73953 * np.log(dist) + 4.44446)*dist
#Function to calculate TF CASK and TP CASK
def cask_calc(dist, seats, type):
    if type == 'TF':
        return np.exp(1.728912 - 0.40453 * np.log(dist) - 0.35671 * np.log(seats))
    elif type == 'TP':
        return np.exp(-0.481549 - 0.099212 * np.log(dist) - 0.356708 * np.log(seats))
# Identify Hub-Spoke data and number of edges in the network
data1 = Data_BRRegAv.loc[:,['Hub','Spoke','Distance','Demand']]
data1.reset_index(drop=True, inplace=True)
# Transforming pandas dataframe into numpy array
data1 = np.array(list(data1.values), dtype=float)
# Manipulating data1 as numpy array
hub = data1[:,0].astype(int)
hubn = np.unique(hub)
N_HUB = len(hubn)
spoke = data1[:,1].astype(int)
spoken = np.unique(spoke)
N_SPOKE = len(spoken)
distance = data1[:,2].astype(int)
demand = data1[:,3].astype(int)
# Definition of number of edges
N = len(distance)
# Ticket fare calculation by route (edge)
#  This code calculates the fare for a given route. It takes the
```

```
#  distance as an input, calculates the ticket fare for
#  that given route. It then stores the fare in a list called fare
#  and can be used for further calculations.
fare_list = [ticket_fare(d) for d in distance]
data2 = pd.DataFrame(columns=['Fare'])
data2['Fare'] = fare_list
data2.reset_index(drop=True, inplace=True)
data2 = np.array(list(data2.values), dtype=float)
fare = data2[:,0]
# Identify aircraft data
data3 = pd.DataFrame()
data3['AIRCRAFT'] = ['Turboprop_1','Turboprop_2', 'Turboprop_3',
                     'Turbojet_1','Turbojet_2','Turbojet_3']
data3['SEATS'] = [9, 48, 75, 105, 130, 145]
data3['RANGE'] = [1900, 1300, 1600, 5000, 5000, 6800]
data3['TYPE'] = ['TP','TP','TP','TF','TF','TF']
data3.reset_index(drop=True, inplace=True)
data3 = np.array(list(data3.values)) #, dtype=float)
# Manipulating data3 as numpy array
aircraft = data3[:,0].astype(str)
N_AC = len(aircraft)
seats = data3[:,1].astype(int)
ac_range = data3[:,2].astype(int)
ac_type = data3[:,3].astype(str)
# CASK (cost per seat-km) calculation by aircraft and route
#  This code creates a list named "cask" and assigns a cask value to
#  each element in the range of N_ANV and N. The cask value is determined
#  based on the distance, number of seats, and the type of aircraft.
#  Then, the cask values are stored in the dataframe "DADOS4" and the
#  list "CASK" is created from the values in the dataframe.
cask_list = list()
for i in range(N_AC):
    for j in range(N):
        if distance[j] > ac_range[i]:
            x = 10
        else:
            x = cask_calc(distance[j], seats[i], ac_type[i])
        cask_list.append(x)
data4 = pd.DataFrame(columns=['Cask'])
data4['Cask'] = cask_list
data4.reset_index(drop=True, inplace=True)
data4 = np.array(list(data4.values), dtype=float)
data4 = data4.reshape(N_AC,N)
cask = data4
# Aircraft Load Factor calculation by distance
# Supply (flux * seats) should not be much larger than demand (passengers)
# For longer routes (distance > 1300km), supply can be 4 times demand (LF = 0.25)
# For shorter routes (distance <= 1300), supply can be 2 times demand (LF = 0.50)
LF = []
for j in range(N):
    if distance[j] <= 1300:
        LF.append(0.50)
    else:
        LF.append(0.25)
# Constants definition
# M is maximum flux of one aircraft (AC) type by route
# M can be reasonably limited by 24 flights/day times 7 days a week (M = 168)
# M = 100 means that an AC with 9 seats can serve up to 900 demand
# K is the maximum number of different AC types accepted
M = 100
#Asks a value for the parameter K:
```

```
K = int(input('Valor de K?\n'))
# Problem declaration
prob = LpProblem('FAP_BRRegAv',LpMaximize)
# Number of flights of one AC type in each route
flux = LpVariable.dicts('Flux',[(a,j,hub[j],spoke[j]) for a in range(N_AC) \
                for j in range(N)],0,None,LpInteger)
#Number of passengers transported in each route
passengers = LpVariable.dicts('Passenger',[(a,j,hub[j],spoke[j]) \
                for a in range(N_AC) for j in range(N)],0,None,LpInteger)
#Used to define the maximum flux and limit range of one aircraft type
binary1 = LpVariable.dicts('Bin1',[(a,j) for a in range(N_AC) \
                for j in range(N)],0,1,LpBinary)
#Used to limit the maximum number of types allowed by route and in the model
binary2 = LpVariable.dicts('Bin2',[(a) for a in range(N_AC)],0,1,
                LpBinary)
#Objective Function
prob += lpSum(fare[j]*passengers[(a,j,hub[j],spoke[j])] - cask[a,j]* \
            seats[a]*distance[j]*flux[(a,j,hub[j],spoke[j])] \
            for a in range(N_AC) for j in range(N))
# Constraint C.1
# Number of transported passengers (PAX) equal demand
# Demand in each route can be supplied by more than one AC type
for j in range(N):
    prob += lpSum(passengers[(a,j,hub[j],spoke[j])] \
                for a in range(N_AC)) - demand[j] == 0
# Constraint C.2
# Sum of seats supply of fluxes of all AC type serving each route
# must be greater or equal to demand in passengers of the route considered
# Supply (flux * seats) can have more than one AC type to match demand
# In each route, the supply of each AC type must be larger than demand
# in passengers transported by the AC type considered
for j in range(N):
    for a in range(N_AC):
        prob += passengers[(a,j,hub[j],spoke[j])] - \
                seats[a]*flux[(a,j,hub[j],spoke[j])] <= 0
# Constraint C.3
# Demand served by AC type flux in each route
# Supply (flux * seats) can not be much larger than demand served in passengers
# From the acceptable load factor (LF) in each route, a supply maximum limit
# of 1/LF times demand is established
for j in range(N):
    for a in range(N_AC):
        prob += passengers[(a,j,hub[j],spoke[j])] - \
                LF[j]*seats[a]*flux[(a,j,hub[j],spoke[j])] >= 0
# Constraint C.4
# Defining the maximum flux of one aircraft type
# One AC type can only serve demand of M times its seat capacity
for a in range(N_AC):
    for j in range(N):
        prob += flux[(a,j,hub[j],spoke[j])] - M*binary1[(a,j)] <= 0
# Constraint C.5
# On AC type can only serve routes within its range
for a in range(N_AC):
    for j in range(N):
        prob += ac_range[a] - distance[j]*binary1[(a,j)] >= 0
# Constraint C.6
# Limit flux by route of maximum number of different AC types
for a in range(N_AC):
    for j in range(N):
        prob += flux[(a,j,hub[j],spoke[j])] - M*binary2[(a)] <= 0
# Constraint C.7
```

```
# Maximum number of different AC types allowed in the model
for j in range(N):
    prob += lpSum(binary2[(a)] for a in range(N_AC)) - K <= 0
# Saves the .mps file of the model built:
model_name = input('Nome do modelo?\n')
prob.writeMPS(model_name + '.mps')
```

APÊNDICE D - EXEMPLO DE ARQUIVO DE CONFIGURAÇÃO PARA EXECUÇÃO DE TAREFAS NO AMBIENTE DE COMPUTAÇÃO PARALELA DO NPAD

```
#!/bin/bash
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=32       #Quantidade de núcleos
#SBATCH --exclusive            #Se cpus-per-task=32, deixe apenas 1 # no início dessa
linha
#SBATCH --time=1-12:00          #Tempo máximo do job no formato DIAS-HORAS:MINUTOS
#SBATCH --hint=compute_bound

###############
# 28JUN2023 wbf
# Teste de escalabilidade do pbzip2 em 4 nós do cluster.
# Criando diretórios, 01...128, para organizar os resultados:
for i in $(seq -w 1 128);do
        mkdir $i;
        cd $i;
        # Executando pbzip2 em um arquivo texto de 1700 MB, compactando e
descompactando 33 vezes, com 'p' threads e salvando os tempos de compressão:
        for j in $(seq -w 1 33);do
                time -p (/home/wdbferreira/ETC/28ABR2023/pbzip2-1.1.13/pbzip2 -p$i
../../../23MAI2023/netlarge6.mps) > log_$j 2>&1 ;
                # Descomprimindo para reiniciar o processo:
                /home/wdbferreira/ETC/28ABR2023/pbzip2-1.1.13/pbzip2 -d
../../../23MAI2023/netlarge6.mps.bz2;
        done;
        # Retornando ao diretório raiz dos testes:
        cd ..;
done
```

APÊNDICE E - EXEMPLO DE ARQUIVO DE COMANDOS PARA EXTRAÇÃO DOS TEMPOS DE EXECUÇÃO DOS PROGRAMAS E CRIAÇÃO DE PLANILHAS

```
#!/bin/bash
# Cria um loop usando os nomes dos diretórios, de 001 a 128, como índices;
# Para cada arquivo de log de execução, filtra os valores de tempo de execução;
# Isola o valor numérico em cada linha filtrada;
# substitui o ponto decimal por vírgula e acrescenta um espaço em branco ao final do
valor;
# Concatena os tempos de execução em um arquivo temporário - cada um destes arquivos
será
# uma coluna do arquivo .csv.
for i in ???; do
        grep real "$i"/log* | awk '{print $2}' | sed 's/\./,/g' | sed 's/$/ /g' >
./tempos_"$i"_cores.txt;
done
# Cria um cabeçalho, com as quantidades de threads,  para o arquivo .csv onde serão
# gravados os tempos de execução para cada número de threads de execução:
for i in ???; do
        echo -n "$i; " ;
done > tempos_netlarge6_pbzip2_128.csv
# Inclui um caractere de retorno de carro ao final deste cabeçalho:
echo -e '\r' >> tempos_netlarge6_pbzip2_128.csv
# Organiza os arquivos temporários como colunas do arquivo .csv:
paste -d ';' ./tempos*txt >> tempos_netlarge6_pbzip2_128.csv
# Remove os arquivos temporários:
rm tempos*txt
```

APÊNDICE F - EXEMPLO DE ARQUIVO DE COMANDOS PARA EXECUÇÃO REPETIDA DO SOLVER HIGHS

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=32      #Quantidade de núcleos
#SBATCH --exclusive            #Se cpus-per-task=32, deixe apenas 1 # no início dessa
linha
#SBATCH --time=1-12:00          #Tempo máximo do job no formato DIAS-HORAS:MINUTOS
#SBATCH --hint=compute_bound

# 20JUN2023
# wbf
# Executa o solver HiGHS sobre cada modelo FAP.
# parallel on
# presolve off

time -p for i in 2 3 4 5 6;do
        mkdir $i
        cd $i
        for j in $(seq -w 1 33);do
                ../HiGHSstatic.v1.5.3/bin/highs --presolve off --parallel on --
model_file ../K_"$i".mps > highs_log_"$j"
        done
        cd ..
done
```

APÊNDICE G - CÓDIGO EM PYTHON PARA EXECUÇÃO NO AMBIENTE JUPYTER NOTEBOOK NO SUPERCOMPUTADOR DO NPAD/UFRN - VERSÃO HIGHS FINAL.

```
%reset

!pip install --upgrade pulp

from pulp import *
from time import *
import pandas as pd
import numpy as np

# Initial model data loading.
Data_BRRegAv_complete = pd.read_csv('SAC_Horus_2019.csv',delimiter=';')
Data_BRRegAv = Data_BRRegAv_complete[Data_BRRegAv_complete['Demand_Selected_Routes']
!= '0']
Data_BRRegAv.reset_index(drop=True, inplace=True)
Data_BRRegAv = Data_BRRegAv.iloc[:,3:]
Data_BRRegAv = Data_BRRegAv.drop(['Annual_Demand_2019'], axis=1)
Data_BRRegAv.columns = ['Hub','Spoke','Distance','Demand']
Data_BRRegAv['Distance'] =
Data_BRRegAv.loc[:,'Distance'].str.replace(',','.').astype(float)
Data_BRRegAv['Demand'] =
Data_BRRegAv.loc[:,'Demand'].str.replace(',','.').astype(float)
Data_BRRegAv.Distance = round(Data_BRRegAv.Distance).astype(int)
Data_BRRegAv.Demand = round(Data_BRRegAv.Demand / 52 / 2).astype(int)

Data_BRRegAv = Data_BRRegAv[Data_BRRegAv.Distance >= 250]

Data_BRRegAv = Data_BRRegAv[Data_BRRegAv.Demand >= 27]

#Function to calculate ticket fare
def ticket_fare(dist):
    return np.exp(-0.73953 * np.log(dist) + 4.44446)*dist

#Function to calculate TF CASK and TP CASK
def cask_calc(dist, seats, type):
    if type == 'TF':
        return np.exp(1.728912 - 0.40453 * np.log(dist) - 0.35671 * np.log(seats))
    elif type == 'TP':
        return np.exp(-0.481549 - 0.099212 * np.log(dist) - 0.356708 * np.log(seats))

# Identify Hub-Spoke data and number of edges in the network

data1 = Data_BRRegAv.loc[:,['Hub','Spoke','Distance','Demand']]

data1.reset_index(drop=True, inplace=True)

# Transforming pandas dataframe into numpy array

data1 = np.array(list(data1.values), dtype=float)

# Manipulating data1 as numpy array

hub = data1[:,0].astype(int)
hubn = np.unique(hub)
N_HUB = len(hubn)

spoke = data1[:,1].astype(int)
spoken = np.unique(spoke)
```

```
N_SPOKE = len(spoken)

distance = data1[:,2].astype(int)

demand = data1[:,3].astype(int)

# Definition of number of edges
N = len(distance)

# Ticket fare calculation by route (edge)

#  This code calculates the fare for a given route. It takes the
#  distance as an input, calculates the ticket fare for
#  that given route. It then stores the fare in a list called fare
#  and can be used for further calculations.

fare_list = [ticket_fare(d) for d in distance]
data2 = pd.DataFrame(columns=['Fare'])

data2['Fare'] = fare_list

data2.reset_index(drop=True, inplace=True)

data2 = np.array(list(data2.values), dtype=float)

fare = data2[:,0]

# Identify aircraft data

data3 = pd.DataFrame()
data3['AIRCRAFT'] = ['Turboprop_1','Turboprop_2', 'Turboprop_3',
                     'Turbojet_1','Turbojet_2','Turbojet_3']
data3['SEATS'] = [9, 48, 75, 105, 130, 145]
data3['RANGE'] = [1900, 1300, 1600, 5000, 5000, 6800]
data3['TYPE'] = ['TP','TP','TP','TF','TF','TF']

data3.reset_index(drop=True, inplace=True)

data3 = np.array(list(data3.values)) #, dtype=float)

# Manipulating data3 as numpy array

aircraft = data3[:,0].astype(str)
N_AC = len(aircraft)

seats = data3[:,1].astype(int)

ac_range = data3[:,2].astype(int)

ac_type = data3[:,3].astype(str)

# CASK (cost per seat-km) calculation by aircraft and route

#  This code creates a list named "cask" and assigns a cask value to
#  each element in the range of N_ANV and N. The cask value is determined
#  based on the distance, number of seats, and the type of aircraft.
#  Then, the cask values are stored in the dataframe "DADOS4" and the
#  list "CASK" is created from the values in the dataframe.

cask_list = list()
```

```
for i in range(N_AC):
    for j in range(N):
        if distance[j] > ac_range[i]:
            x = 10
        else:
            x = cask_calc(distance[j], seats[i], ac_type[i])
        cask_list.append(x)

data4 = pd.DataFrame(columns=['Cask'])

data4['Cask'] = cask_list

data4.reset_index(drop=True, inplace=True)

data4 = np.array(list(data4.values), dtype=float)

data4 = data4.reshape(N_AC,N)

cask = data4

# Aircraft Load Factor calculation by distance

# Supply (flux * seats) should not be much larger than demand (passengers)
# For longer routes (distance > 1300km), supply can be 4 times demand (LF = 0.25)
# For shorter routes (distance <= 1300), supply can be 2 times demand (LF = 0.50)

LF = []
for j in range(N):
    if distance[j] <= 1300:
        LF.append(0.50)
    else:
        LF.append(0.25)

# Constants definition
# M is maximum flux of one aircraft (AC) type by route
# M can be reasonably limited by 24 flights/day times 7 days a week (M = 168)
# M = 100 means that an AC with 9 seats can serve up to 900 demand
# K is the maximum number of different AC types accepted

M = 100

# Asks for a value for the parameter K:

K = int(input('K parameter value?\n'))

# Problem declaration

prob = LpProblem('FAP_BRRegAv',LpMaximize)

# Number of flights of one AC type in each route

flux = LpVariable.dicts('Flux',[(a,j,hub[j],spoke[j]) for a in range(N_AC) \
                for j in range(N)],0,None,LpInteger)

#Number of passengers transported in each route

passengers = LpVariable.dicts('Passenger',[(a,j,hub[j],spoke[j]) \
                for a in range(N_AC) for j in range(N)],0,None,LpInteger)
#Used to define the maximum flux and limit range of one aircraft type

binary1 = LpVariable.dicts('Bin1',[(a,j) for a in range(N_AC) \
```

```
                for j in range(N)],0,1,LpBinary)
#Used to limit the maximum number of types allowed by route and in the model

binary2 = LpVariable.dicts('Bin2',[(a) for a in range(N_AC)],0,1,
                LpBinary)
#Objective Function

prob += lpSum(fare[j]*passengers[(a,j,hub[j],spoke[j])] - cask[a,j]* \
            seats[a]*distance[j]*flux[(a,j,hub[j],spoke[j])] \
            for a in range(N_AC) for j in range(N))

# Constraint C.1
# Number of transported passengers (PAX) equal demand
# Demand in each route can be supplied by more than one AC type

for j in range(N):
    prob += lpSum(passengers[(a,j,hub[j],spoke[j])] \
                for a in range(N_AC)) - demand[j] == 0

# Constraint C.2
# Sum of seats supply of fluxes of all AC type serving each route
# must be greater or equal to demand in passengers of the route considered
# Supply (flux * seats) can have more than one AC type to match demand
# In each route, the supply of each AC type must be larger than demand
# in passengers transported by the AC type considered

for j in range(N):
    for a in range(N_AC):
        prob += passengers[(a,j,hub[j],spoke[j])] - \
                seats[a]*flux[(a,j,hub[j],spoke[j])] <= 0

# Constraint C.3
# Demand served by AC type flux in each route
# Supply (flux * seats) can not be much larger than demand served in passengers
# From the acceptable load factor (LF) in each route, a supply maximum limit
# of 1/LF times demand is established

for j in range(N):
    for a in range(N_AC):
        prob += passengers[(a,j,hub[j],spoke[j])] - \
                LF[j]*seats[a]*flux[(a,j,hub[j],spoke[j])] >= 0

# Constraint C.4
# Defining the maximum flux of one aircraft type
# One AC type can only serve demand of M times its seat capacity

for a in range(N_AC):
    for j in range(N):
        prob += flux[(a,j,hub[j],spoke[j])] - M*binary1[(a,j)] <= 0

# Constraint C.5
# On AC type can only serve routes within its range

for a in range(N_AC):
    for j in range(N):
        prob += ac_range[a] - distance[j]*binary1[(a,j)] >= 0
# Constraint C.6
# Limit flux by route of maximum number of different AC types

for a in range(N_AC):
    for j in range(N):
```

```
        prob += flux[(a,j,hub[j],spoke[j])] - M*binary2[(a)] <= 0

# Constraint C.7
# Maximum number of different AC types allowed in the model

for j in range(N):
    prob += lpSum(binary2[(a)] for a in range(N_AC)) - K <= 0

print('Number of decision variables: ', prob.numVariables(), '\n')
print('Number of constraints: ', prob.numConstraints(), '\n')

# Downloading and unpacking the HiGHS solver (comment out if done)

!mkdir ./HiGHSstatic.v1.5.3.x86_64
%cd ./HiGHSstatic.v1.5.3.x86_64
!wget -c
https://github.com/JuliaBinaryWrappers/HiGHSstatic_jll.jl/releases/download/HiGHSstat
ic-v1.5.3%2B0/HiGHSstatic.v1.5.3.x86_64-linux-gnu-cxx11.tar.gz
!tar -xzf HiGHSstatic.v1.5.3.x86_64-linux-gnu-cxx11.tar.gz
%cd ..

# Path to the solver executable:
import os
os.environ['PATH'] = "/home/wdbferreira/PPA/bin:/home/wdbferreira/.local/bin:\
/home/wdbferreira/bin:/opt/npad/bin:/usr/condabin:/usr/local/bin:/usr/bin:/usr/local/
sbin:/usr/sbin:\
/home/wdbferreira/.local/bin:/home/wdbferreira/.local/bin:\
/home/vrrcelestino/PPA/10JUL2023/HiGHSstatic.v1.5.3.x86_64/bin"
path_to_highs =
r'/home/vrrcelestino/PPA/10JUL2023/HiGHSstatic.v1.5.3.x86_64/bin/highs'

# Options to the PuLP framework
# mip : if False, problem will be forcibly dealt with as LP;
# keepFiles : if True, stores .mps file. Default: False;
# msg : echos HiGHS messages as stored in HiGHS.log;
# threads : should means max number of threads to launch. Not working here.
# Command line options to the HiGHS solver:
# options : Other options to highs binnary (presolve on/off, parallel on/off,
random_seed value).
solver = HiGHS_CMD(path = path_to_highs, mip=True, options=['--parallel on --presolve
on --random_seed 0'])

# Solve and keeps track of execution time

pulp_initial_time = time()
result = prob.solve(solver)
solver_dict = solver.toDict()
print('Status:', LpStatus[prob.status])
pulp_end_time = time()
print('PuLP HiGHS execution time: %.2fs' % (pulp_end_time - pulp_initial_time))
print('\n')

# Print number of variables
print('Problem has',prob.numVariables(),'variables')

# Print number of constraints
print('Problem has',prob.numConstraints(),'constraints')
for v in prob.variables():
    if v.varValue != 0: # and v.varValue != M:
        if 'Bin2' in v.name:
            print(f'{v.name}: {v.varValue}')
```

```
profit = value(prob.objective)
print('\n')
print('The total profit is US$ %.2f' % profit)

#Write results in CSV file for better visualization

list_index = []
list_origin = []
list_destination = []
list_distance = []
list_demand = []
list_aircraft = []
list_passengers = []
list_flux = []
list_profit = []

for j in range(N):
    for a in range(N_AC):
        profit = fare[j]*passengers[(a,j,hub[j],spoke[j])].varValue - \
         cask[a,j]*seats[a]*distance[j]*flux[(a,j,hub[j],spoke[j])].varValue
        if profit != 0:
            list_index.append(j)
            list_origin.append(hub[j])
            list_destination.append(spoke[j])
            list_distance.append(distance[j])
            list_demand.append(demand[j])
            list_aircraft.append(aircraft[a])
            list_passengers.append(passengers[(a,j,hub[j],spoke[j])].varValue)
            list_flux.append(flux[(a,j,hub[j],spoke[j])].varValue)
            list_profit.append(profit)

results = {'INDEX':list_index,'ORIGIN': list_origin,
           'DESTINATION':list_destination, 'DISTANCE': list_distance,
           'DEMAND': list_demand, 'AIRCRAFT':list_aircraft,
           'PASSENGERS':list_passengers,
           'FLUX':list_flux,
           'ROUTE PROFIT':list_profit}

Results = pd.DataFrame(results, columns=['ORIGIN','DESTINATION', 'DISTANCE',
                                         'DEMAND', 'AIRCRAFT', 'PASSENGERS',
                                          'FLUX', 'ROUTE PROFIT'])


# Set an output CSV filename for each K

CSV_name = 'PuLP_K' + str(K) + '_HiGHS.csv'

# Save CSV

Results.to_csv(CSV_name)
```

APÊNDICE H - CÓDIGO EM PYTHON PARA EXECUÇÃO NO AMBIENTE JUPYTER NOTEBOOK NO SUPERCOMPUTADOR DO NPAD/UFRN - VERSÃO CBC FINAL.

```
%reset

!pip install --upgrade pulp

from pulp import *
from time import *
import pandas as pd
import numpy as np

# Initial model data loading.
Data_BRRegAv_complete = pd.read_csv('SAC_Horus_2019.csv',delimiter=';')
Data_BRRegAv = Data_BRRegAv_complete[Data_BRRegAv_complete['Demand_Selected_Routes']
!= '0']
Data_BRRegAv.reset_index(drop=True, inplace=True)
Data_BRRegAv = Data_BRRegAv.iloc[:,3:]
Data_BRRegAv = Data_BRRegAv.drop(['Annual_Demand_2019'], axis=1)
Data_BRRegAv.columns = ['Hub','Spoke','Distance','Demand']
Data_BRRegAv['Distance'] =
Data_BRRegAv.loc[:,'Distance'].str.replace(',','.').astype(float)
Data_BRRegAv['Demand'] =
Data_BRRegAv.loc[:,'Demand'].str.replace(',','.').astype(float)
Data_BRRegAv.Distance = round(Data_BRRegAv.Distance).astype(int)
Data_BRRegAv.Demand = round(Data_BRRegAv.Demand / 52 / 2).astype(int)

Data_BRRegAv = Data_BRRegAv[Data_BRRegAv.Distance >= 250]

Data_BRRegAv = Data_BRRegAv[Data_BRRegAv.Demand >= 27]

#Function to calculate ticket fare
def ticket_fare(dist):
    return np.exp(-0.73953 * np.log(dist) + 4.44446)*dist

#Function to calculate TF CASK and TP CASK
def cask_calc(dist, seats, type):
    if type == 'TF':
        return np.exp(1.728912 - 0.40453 * np.log(dist) - 0.35671 * np.log(seats))
    elif type == 'TP':
        return np.exp(-0.481549 - 0.099212 * np.log(dist) - 0.356708 * np.log(seats))

# Identify Hub-Spoke data and number of edges in the network

data1 = Data_BRRegAv.loc[:,['Hub','Spoke','Distance','Demand']]

data1.reset_index(drop=True, inplace=True)

# Transforming pandas dataframe into numpy array

data1 = np.array(list(data1.values), dtype=float)

# Manipulating data1 as numpy array

hub = data1[:,0].astype(int)
hubn = np.unique(hub)
N_HUB = len(hubn)

spoke = data1[:,1].astype(int)
spoken = np.unique(spoke)
```

```
N_SPOKE = len(spoken)

distance = data1[:,2].astype(int)

demand = data1[:,3].astype(int)

# Definition of number of edges
N = len(distance)

# Ticket fare calculation by route (edge)

#  This code calculates the fare for a given route. It takes the
#  distance as an input, calculates the ticket fare for
#  that given route. It then stores the fare in a list called fare
#  and can be used for further calculations.

fare_list = [ticket_fare(d) for d in distance]
data2 = pd.DataFrame(columns=['Fare'])

data2['Fare'] = fare_list

data2.reset_index(drop=True, inplace=True)

data2 = np.array(list(data2.values), dtype=float)

fare = data2[:,0]

# Identify aircraft data

data3 = pd.DataFrame()
data3['AIRCRAFT'] = ['Turboprop_1','Turboprop_2', 'Turboprop_3',
                     'Turbojet_1','Turbojet_2','Turbojet_3']
data3['SEATS'] = [9, 48, 75, 105, 130, 145]
data3['RANGE'] = [1900, 1300, 1600, 5000, 5000, 6800]
data3['TYPE'] = ['TP','TP','TP','TF','TF','TF']

data3.reset_index(drop=True, inplace=True)

data3 = np.array(list(data3.values)) #, dtype=float)

# Manipulating data3 as numpy array

aircraft = data3[:,0].astype(str)
N_AC = len(aircraft)

seats = data3[:,1].astype(int)

ac_range = data3[:,2].astype(int)

ac_type = data3[:,3].astype(str)

# CASK (cost per seat-km) calculation by aircraft and route

#  This code creates a list named "cask" and assigns a cask value to
#  each element in the range of N_ANV and N. The cask value is determined
#  based on the distance, number of seats, and the type of aircraft.
#  Then, the cask values are stored in the dataframe "DADOS4" and the
#  list "CASK" is created from the values in the dataframe.

cask_list = list()
```

```
for i in range(N_AC):
    for j in range(N):
        if distance[j] > ac_range[i]:
            x = 10
        else:
            x = cask_calc(distance[j], seats[i], ac_type[i])
        cask_list.append(x)

data4 = pd.DataFrame(columns=['Cask'])

data4['Cask'] = cask_list

data4.reset_index(drop=True, inplace=True)

data4 = np.array(list(data4.values), dtype=float)

data4 = data4.reshape(N_AC,N)

cask = data4

# Aircraft Load Factor calculation by distance

# Supply (flux * seats) should not be much larger than demand (passengers)
# For longer routes (distance > 1300km), supply can be 4 times demand (LF = 0.25)
# For shorter routes (distance <= 1300), supply can be 2 times demand (LF = 0.50)

LF = []
for j in range(N):
    if distance[j] <= 1300:
        LF.append(0.50)
    else:
        LF.append(0.25)

# Constants definition
# M is maximum flux of one aircraft (AC) type by route
# M can be reasonably limited by 24 flights/day times 7 days a week (M = 168)
# M = 100 means that an AC with 9 seats can serve up to 900 demand
# K is the maximum number of different AC types accepted

M = 100

# Asks for a value for the parameter K:

K = int(input('K parameter value?\n'))

# Problem declaration

prob = LpProblem('FAP_BRRegAv',LpMaximize)

# Number of flights of one AC type in each route

flux = LpVariable.dicts('Flux',[(a,j,hub[j],spoke[j]) for a in range(N_AC) \
                for j in range(N)],0,None,LpInteger)

#Number of passengers transported in each route

passengers = LpVariable.dicts('Passenger',[(a,j,hub[j],spoke[j]) \
                for a in range(N_AC) for j in range(N)],0,None,LpInteger)
#Used to define the maximum flux and limit range of one aircraft type

binary1 = LpVariable.dicts('Bin1',[(a,j) for a in range(N_AC) \
```

```
                for j in range(N)],0,1,LpBinary)
#Used to limit the maximum number of types allowed by route and in the model

binary2 = LpVariable.dicts('Bin2',[(a) for a in range(N_AC)],0,1,
                LpBinary)
#Objective Function

prob += lpSum(fare[j]*passengers[(a,j,hub[j],spoke[j])] - cask[a,j]* \
            seats[a]*distance[j]*flux[(a,j,hub[j],spoke[j])] \
            for a in range(N_AC) for j in range(N))

# Constraint C.1
# Number of transported passengers (PAX) equal demand
# Demand in each route can be supplied by more than one AC type

for j in range(N):
    prob += lpSum(passengers[(a,j,hub[j],spoke[j])] \
                for a in range(N_AC)) - demand[j] == 0

# Constraint C.2
# Sum of seats supply of fluxes of all AC type serving each route
# must be greater or equal to demand in passengers of the route considered
# Supply (flux * seats) can have more than one AC type to match demand
# In each route, the supply of each AC type must be larger than demand
# in passengers transported by the AC type considered

for j in range(N):
    for a in range(N_AC):
        prob += passengers[(a,j,hub[j],spoke[j])] - \
                seats[a]*flux[(a,j,hub[j],spoke[j])] <= 0

# Constraint C.3
# Demand served by AC type flux in each route
# Supply (flux * seats) can not be much larger than demand served in passengers
# From the acceptable load factor (LF) in each route, a supply maximum limit
# of 1/LF times demand is established

for j in range(N):
    for a in range(N_AC):
        prob += passengers[(a,j,hub[j],spoke[j])] - \
                LF[j]*seats[a]*flux[(a,j,hub[j],spoke[j])] >= 0

# Constraint C.4
# Defining the maximum flux of one aircraft type
# One AC type can only serve demand of M times its seat capacity

for a in range(N_AC):
    for j in range(N):
        prob += flux[(a,j,hub[j],spoke[j])] - M*binary1[(a,j)] <= 0

# Constraint C.5
# On AC type can only serve routes within its range

for a in range(N_AC):
    for j in range(N):
        prob += ac_range[a] - distance[j]*binary1[(a,j)] >= 0
# Constraint C.6
# Limit flux by route of maximum number of different AC types

for a in range(N_AC):
    for j in range(N):
```

```
        prob += flux[(a,j,hub[j],spoke[j])] - M*binary2[(a)] <= 0

# Constraint C.7
# Maximum number of different AC types allowed in the model

for j in range(N):
    prob += lpSum(binary2[(a)] for a in range(N_AC)) - K <= 0

print('Number of decision variables: ', prob.numVariables(), '\n')
print('Number of constraints: ', prob.numConstraints(), '\n')

# Options to the PuLP framework
# mip : if False, problem will be forcibly dealt with as LP;
# keepFiles : if True, writes .mps and .sol files. Uses name defined in LpProblem
statement as base. Default: False;
# msg : outputs CBC solver's messages;
# threads : sets maximum number of threads to launch;
# gapRel : "None" or a fraction, float number. Gap tolerance to set a stop to the
solver.
# Ex.:
# solver=PULP_CBC_CMD(threads=128, msg=False, gapRel = 0.001)
solver=PULP_CBC_CMD(threads=64, msg=False, mip=True, gapRel = None)

# Solve and keeps track of execution time

cbc_init_time = time()
result = prob.solve(solver)
solver_dict = solver.toDict()
print('Status:', LpStatus[prob.status])
cbc_end_time = time()
print('PuLP CBC execution time: %.2fs' % (cbc_end_time - cbc_init_time))

# Print number of variables
print('Problem has',prob.numVariables(),'variables')

# Print number of constraints
print('Problem has',prob.numConstraints(),'constraints')
for v in prob.variables():
    if v.varValue != 0: # and v.varValue != M:
        if 'Bin2' in v.name:
            print(f'{v.name}: {v.varValue}')


profit = value(prob.objective)
print('\n')
print('The total profit is US$ %.2f' % profit)

#Write results in CSV file for better visualization

list_index = []
list_origin = []
list_destination = []
list_distance = []
list_demand = []
list_aircraft = []
list_passengers = []
list_flux = []
list_profit = []

for j in range(N):
    for a in range(N_AC):
```

```
        profit = fare[j]*passengers[(a,j,hub[j],spoke[j])].varValue - \
         cask[a,j]*seats[a]*distance[j]*flux[(a,j,hub[j],spoke[j])].varValue
        if profit != 0:
            list_index.append(j)
            list_origin.append(hub[j])
            list_destination.append(spoke[j])
            list_distance.append(distance[j])
            list_demand.append(demand[j])
            list_aircraft.append(aircraft[a])
            list_passengers.append(passengers[(a,j,hub[j],spoke[j])].varValue)
            list_flux.append(flux[(a,j,hub[j],spoke[j])].varValue)
            list_profit.append(profit)

results = {'INDEX':list_index,'ORIGIN': list_origin,
           'DESTINATION':list_destination, 'DISTANCE': list_distance,
           'DEMAND': list_demand, 'AIRCRAFT':list_aircraft,
           'PASSENGERS':list_passengers,
           'FLUX':list_flux,
           'ROUTE PROFIT':list_profit}

Results = pd.DataFrame(results, columns=['ORIGIN','DESTINATION', 'DISTANCE',
                                         'DEMAND', 'AIRCRAFT', 'PASSENGERS',
                                          'FLUX', 'ROUTE PROFIT'])


# Set an output CSV filename for each K

CSV_name = 'PuLP_K' + str(K) + '_CBC.csv'

# Save CSV

Results.to_csv(CSV_name)
```